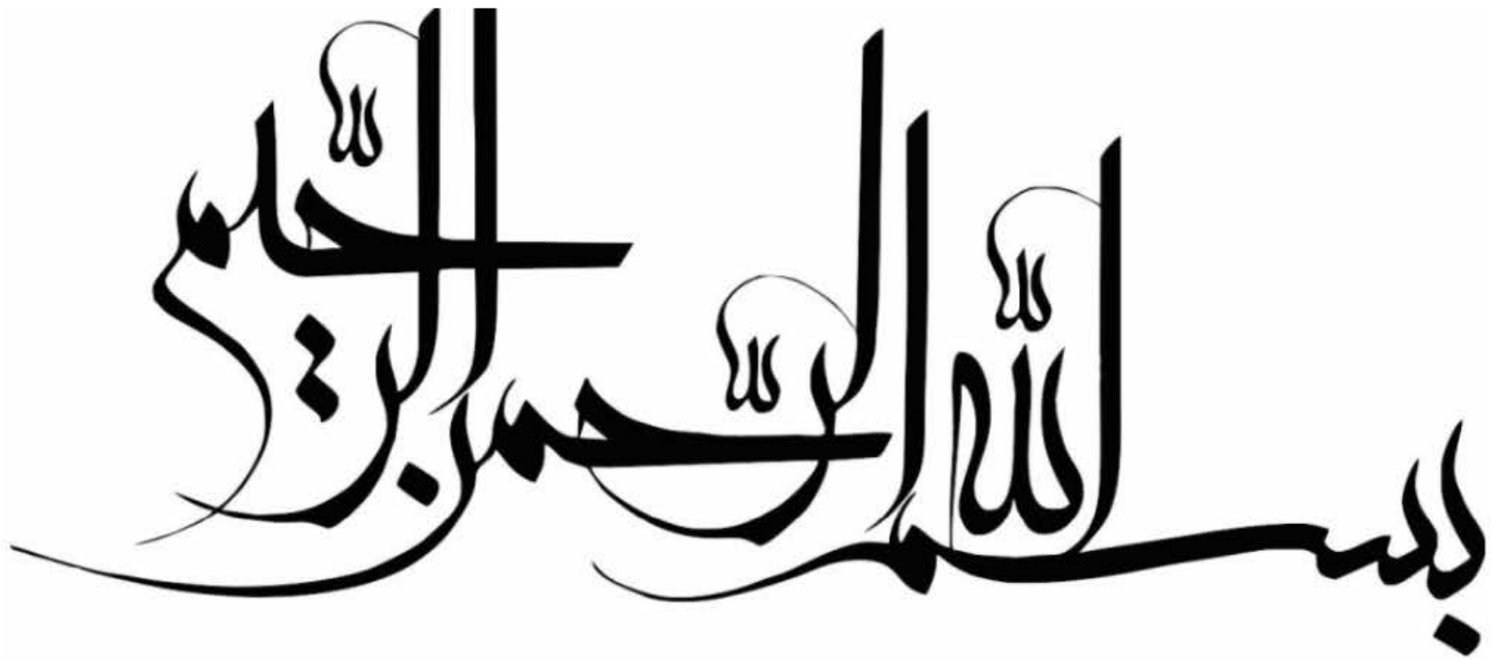

بسم الله الرحمن الرحيم

مهندسی نرم افزار شیٔ گرا

یک متدولوژی حاکم یکنواخت

تالیف:

دیوید کونگ

ترجمه:

دکتر بهمن زمانی دکتر افسانه فاطمی

دانشگاه اصفهان

۱۳۹۵

سرشناسه : کونگ، چن ـ هو
Kung, Chen-Ho
عنوان و نام پدیدآور : مهندسی نرم‌افزار شئ‌گرا : یک متدولوژی چابک یکنواخت/تالیف دیوید کونگ؛ ترجمه بهمن زمانی، افسانه فاطمی.
مشخصات نشر : اصفهان: دانشگاه اصفهان، ۱۳۹۴.
مشخصات ظاهری : ۴۵۰ ص.: مصور، جدول، نمودار.
فروست : انتشارات دانشگاه اصفهان؛ ۶۴۷.
شابک : ۹۷۸-۶۰۰-۱۱۰-۱۱۲-۰
وضعیت فهرست نویسی : فیپا
یادداشت : عنوان اصلی: Object-Oriented Software Engineering: An Agile Unified Methodology
موضوع : نرم‌افزار -- مهندسی
موضوع : برنامه‌نویسی شئ‌گرا
شناسه افزوده : زمانی، بهمن، ۱۳۴۴ -، مترجم
شناسه افزوده : فاطمی، افسانه، ۱۳۵۲ -، مترجم
شناسه افزوده : دانشگاه اصفهان
رده بندی کنگره : QA۷۵۸/۷۶/ک۸۸م۹ ۱۳۹۴
رده بندی دیویی : ۰۰۵/۱۱۷
شماره کتابشناسی ملی : ۴۱۸۰۶۱۳

انتشارات دانشگاه اصفهان

عنوان کتاب: مهندسی نرم‌افزار شئ‌گرا: یک متدولوژی چابک یکنواخت
ترجمه: دکتر بهمن زمانی، دکتر افسانه فاطمی
ناشر: دانشگاه اصفهان
نوبت چاپ: چاپ اول ـ بهار ۱۳۹۵
شمارگان: ۱۰۰۰ جلد
چاپ و صحافی: چاپخانه دانشگاه اصفهان
قیمت: ۲۵۰/۰۰۰ ریال

کلیه حقوق نشر برای ناشر محفوظ است

مراکز فروش کتاب:
اصـفهان: میدان آزادی ـ خیابان دانشگاه ـ فروشگاه کتاب دانشگاه اصفهان
تـلفن: ۳۷۹۳۲۱۷۷ ـ ۰۳۱ پُست الکـترونیـکی: entesharat@ui.ac.ir
تهران: میدان انقلاب ـ خیابان نصرت ـ خیابان دکتر قریب، نرسیده به خیابان فرصت ـ پلاک ۷
مـؤسسه کـتابیران، مـرکز پخش کتاب‌های دانشگاهی ـ تلفـن: ۱۵ ـ ۶۶۵۶۶۵۱۰ ـ ۰۲۱

فهرست مطالب

پیشگفتار مترجمان ع

پیشگفتار ف

قسمت اول مقدمه و مهندسی سیستم ۱

فصل ۱: مقدمه ۳

۱-۱ مهندسی نرم‌افزار چیست؟ ۴

۱-۲ چرا مهندسی نرم‌افزار؟ ۴

۱-۳ فعالیت‌های چرخه‌ی حیات نرم‌افزار ۵

۱-۳-۱ فرایند توسعه‌ی نرم‌افزار ۶

۱-۳-۲ تضمین کیفیت نرم‌افزار ۱۱

۱-۳-۳ مدیریت پروژه‌ی نرم‌افزاری ۱۱

۱-۴ مهندسی نرم‌افزار شیءگرا ۱۳

۱-۴-۱ زبان‌های مدل‌سازی و طراحی شیءگرا ۱۴

۱-۴-۲ فرایندهای توسعه‌ی شیءگرا ۱۴

۱-۴-۳ متدولوژی‌های توسعه‌ی شیءگرا ۱۴

۱-۴-۴ آیا روش‌های شیءگرا جایگزین روش‌های سنتی می‌شوند؟ ۱۵

۱-۵ مهندسی نرم‌افزار و علم کامپیوتر ۱۵

خلاصه ۱۶

برای مطالعه‌ی بیش‌تر ۱۷

سوال‌های مروری فصل ۱۷
تمرین‌ها ۱۷

فصل ۲: فرایند و متدولوژی نرم‌افزار ۱۹
۲-۱ چالش‌های توسعه‌ی سیستم ۲۰
۲-۲ فرایند نرم‌افزار ۲۱
۲-۳ محاسن و مشکلات فرایند آبشاری ۲۲
۲-۴ توسعه‌ی نرم‌افزار یک مسئله‌ی بدرفتار است. ۲۳
۲-۵ مدل‌های فرایند نرم‌افزار ۲۵
۲-۵-۱ فرایند نمونه‌سازی ۲۵
۲-۵-۲ فرایند تکاملی ۲۵
۲-۵-۳ فرایند حلزونی ۲۶
۲-۵-۴ فرایند یکنواخت ۲۶
۲-۵-۵ فرایند نرم‌افزار شخصی ۲۹
۲-۵-۶ فرایند نرم‌افزار تیمی ۳۳
۲-۵-۷ فرایندهای چابک ۳۴
۲-۶ متدولوژی توسعه‌ی نرم‌افزار ۴۱
۲-۶-۱ تفاوت بین فرایند و متدولوژی ۴۲
۲-۶-۲ فواید متدولوژی ۴۳
۲-۶-۳ متدولوژی‌های ساختمند ۴۳
۲-۶-۴ متدولوژی‌های کلاسیک شئ‌گرا ۴۴
۲-۷ روش‌های چابک ۴۴
۲-۷-۱ روش توسعه‌ی پویای سیستم‌ها ۴۶
۲-۷-۲ اسکرام ۴۶
۲-۷-۳ توسعه‌ی خصیصه-رانده ۴۷
۲-۷-۴ برنامه‌نویسی افراطی ۴۹
۲-۷-۵ چابک یا برنامه-رانده ۴۹
۲-۸ مروری بر فرایند و متدولوژی این کتاب ۵۰
خلاصه ۵۶
برای مطالعه‌ی بیش‌تر ۵۷
سوال‌های مروری فصل ۵۷
تمرین‌ها ۵۸

فصل ۳: مهندسی سیستم ۵۹
۳-۱ سیستم چیست؟ ۶۰

۲-۳ مهندسی سیستم چیست؟ ۶۱
۳-۳ تعریف نیازمندی‌های سیستم ۶۴
۳-۳-۱ شناسایی نیازهای کسب‌وکار ۶۴
۳-۳-۲ تعریف نیازمندی‌های سیستم ۶۶
۴-۳ طراحی معماری سیستم ۶۶
۳-۴-۱ تجزیه‌ی سیستم ۶۷
۳-۴-۲ تخصیص نیازمندی‌ها ۷۰
۳-۴-۳ نمودارهای طراحی معماری ۷۱
۳-۴-۴ تبیین وظایف و واسط زیرسیستم‌ها ۷۴
۵-۳ توسعه‌ی زیرسیستم‌ها ۷۸
۳-۵-۱ نمودار زمینه‌ی شئ‌گرا ۷۸
۳-۵-۲ فواید نمودار زمینه‌ی شئ‌گرا ۷۸
۳-۵-۳ همکاری تیم‌های مهندسی ۷۹
۶-۳ یکپارچه‌سازی، آزمون و استقرار سیستم ۸۰
۷-۳ مدیریت پیکربندی سیستم ۸۰
خلاصه ۸۲
برای مطالعه‌ی بیش‌تر ۸۲
سوال‌های مروری فصل ۸۳
تمرین‌ها ۸۳

قسمت دوم تحلیل و طراحی معماری ۸۵

فصل ۴: استخراج نیازمندی‌های نرم‌افزار ۸۷
۴-۱ استخراج نیازمندی‌ها چیست؟ ۸۸
۴-۲ اهمیت استخراج نیازمندی‌ها ۹۰
۴-۳ چالش‌های استخراج نیازمندی‌ها ۹۰
۴-۴ انواع نیازمندی ۹۲
۴-۵ گام‌های استخراج نیازمندی‌ها ۹۳
۴-۵-۱ جمع‌آوری اطلاعات ۹۴
۴-۵-۲ ساخت مدل‌های تحلیل ۹۸
۴-۵-۳ استنتاج نیازمندی‌ها و قیدها ۹۹
۴-۵-۴ استانداردهای توصیف نیازمندی‌ها ۱۰۴
۴-۵-۵ انجام مطالعه‌ی امکان‌پذیری ۱۰۴
۴-۵-۶ مرور شرح نیازمندی‌ها ۱۰۴

۴-۶ اعمال قوانین چابکی ۱۰۶
۴-۷ مدیریت نیازمندی‌ها و ابزارهای آن ۱۰۸
خلاصه ۱۰۸
برای مطالعه‌ی بیش‌تر ۱۰۹
سوال‌های مروری فصل ۱۱۰
تمرین‌ها ۱۱۰

فصل ۵: مدل‌سازی دامنه ۱۱۳
۵-۱ مدل‌سازی دامنه چیست؟ ۱۱۴
۵-۲ چرا مدل‌سازی دامنه؟ ۱۱۴
۵-۳ شئ‌گرایی و نمودار کلاس ۱۱۵
۵-۳-۱ تعاریف توسعه‌ای و ذهنی ۱۱۵
۵-۳-۲ کلاس و شئ ۱۱۷
۵-۳-۳ شئ و صفت ۱۱۸
۵-۳-۴ انجمن ۱۱۹
۵-۳-۵ تعدد و نقش ۱۲۰
۵-۳-۶ تجمیع ۱۲۲
۵-۳-۷ ارث‌بری ۱۲۲
۵-۳-۸ ارث‌بری وچندریختی ۱۲۴
۵-۳-۹ کلاس انجمنی ۱۲۴
۵-۴ گام‌های مدل‌سازی دامنه ۱۲۶
۵-۴-۱ جمع‌آوری اطلاعات دامنه‌ی کاربرد ۱۲۷
۵-۴-۲ طوفان فکری ۱۲۸
۵-۴-۳ دسته‌بندی نتایج طوفان فکری ۱۲۹
۵-۴-۴ تصویرسازی از مدل دامنه ۱۳۴
۵-۴-۵ فهرست مروری مدل دامنه ۱۳۸
۵-۵ جمع‌بندی ۱۳۹
۵-۶ رهنمون‌هایی برای مدل‌سازی دامنه ۱۴۲
۵-۷ به‌کارگیری قوانین چابکی ۱۴۵
۵-۸ ابزارهای پشتیبان مدل‌سازی دامنه ۱۴۵
خلاصه ۱۴۷
برای مطالعه‌ی بیش‌تر ۱۴۷
سوال‌های مروری فصل ۱۴۸
تمرین‌ها ۱۴۹

فصل ۶: طراحی معماری ۱۵۱
۶-۱ طراحی معماری چیست؟ ۱۵۲
۶-۲ اهمیت طراحی معماری ۱۵۲
۶-۳ فرایند طراحی معماری ۱۵۳
۶-۳-۱ تبیین اهداف طراحی معماری ۱۵۴
۶-۳-۲ تعیین نوع سیستم ۱۵۵
۶-۳-۳ استفاده از سبک‌های معماری ۱۵۹
۶-۳-۴ انجام طراحی معماری لازم ۱۶۹
۶-۳-۵ تعیین واسط‌ها و عملیات زیرسیستم ۱۷۰
۶-۳-۶ بازبینی طراحی معماری ۱۷۰
۶-۴ سبک معماری و نمودار بسته ۱۷۰
۶-۵ اعمال قوانین طراحی نرم‌افزار ۱۷۳
۶-۵-۱ قوانین طراحی نرم‌افزار کدامند؟ ۱۷۳
۶-۵-۲ طراحی برای تغییر ۱۷۳
۶-۵-۳ جداسازی دغدغه‌ها ۱۷۵
۶-۵-۴ پنهان‌سازی اطلاعات ۱۷۵
۶-۵-۵ چسبندگی زیاد ۱۷۶
۶-۵-۶ جفت‌شدگی کم ۱۷۷
۶-۵-۷ ساده و احمقانه فرض کن ۱۷۸
۶-۶ رهنمون‌هایی برای طراحی معماری ۱۷۸
۶-۷ طراحی معماری و الگوهای طراحی ۱۷۹
۶-۸ به‌کارگیری قوانین چابکی ۱۸۰
خلاصه ۱۸۰
برای مطالعه‌ی بیش‌تر ۱۸۱
سوال‌های مروری فصل ۱۸۱
تمرین‌ها ۱۸۲

قسمت سوم مدل‌سازی و طراحی سیستم‌های تعاملی ۱۸۳

فصل ۷: استنتاج مورد کاربردها از نیازمندی‌ها ۱۸۵
۷-۱ کنشگر چیست؟ ۱۸۶
۷-۲ مورد کاربرد چیست؟ ۱۸۶
۷-۳ فرایند کسب‌وکاری، عمل، و کنش ۱۸۷
۷-۴ گام‌های استنتاج مورد کاربردها از نیازمندی‌ها ۱۸۹
۷-۴-۱ شناسایی مورد کاربردها ۱۹۰

۷-۴-۲ تعیین قلمرو مورد کاربردها ۱۹۸
۷-۴-۳ مصورسازی زمینه‌ی مورد کاربرد ۲۰۱
۷-۴-۴ بازبینی مشخصات مورد کاربردها ۲۰۶
۷-۴-۵ تخصیص مورد کاربردها به تکرارها ۲۰۷
۷-۵ رهنمون‌هایی برای استنتاج مورد کاربردها ۲۰۸
۷-۶ به‌کاربردن اصول چابکی ۲۱۲
۷-۷ پشتیبانی ابزاری برای مدل‌سازی مورد کاربردها ۲۱۴
خلاصه ۲۱۴
برای مطالعه‌ی بیش‌تر ۲۱۶
سوال‌های مروری فصل ۲۱۶
تمرین‌ها ۲۱۷

فصل ۸: مدل‌سازی تعامل کنشگر-سیستم ۲۱۹
۸-۱ مدل‌سازی تعامل کنشگر-سیستم چیست؟ ۲۲۰
۸-۲ اهمیت مدل‌سازی تعامل کنشگر-سیستم ۲۲۰
۸-۳ گام‌های مدل‌سازی تعامل کنشگر-سیستم ۲۲۱
۸-۳-۱ ایجاد یک جدول دو ستونی ۲۲۲
۸-۳-۲ تعیین گام‌های تعامل کنشگر-سیستم ۲۲۲
۸-۳-۳ بازبینی مشخصات تعامل کنشگر-سیستم ۲۲۳
۸-۴ تبیین جریان‌های جایگزین ۲۲۴
۸-۵ استفاده از پیش‌نمونه‌های واسط کاربر ۲۲۴
۸-۶ رسیدگی به استثناها را نشان ندهید ۲۲۷
۸-۷ پیش‌شرط و پس‌شرط برای مورد کاربردها ۲۲۸
۸-۸ شمول مورد کاربردهای دیگر ۲۲۹
۸-۹ ادامه دادن با مورد کاربردهای دیگر ۲۲۹
۸-۱۰ مشکلات رایج ۲۳۱
۸-۱۱ اعمال اصول چابکی ۲۳۲
خلاصه ۲۳۳
برای مطالعه‌ی بیش‌تر ۲۳۴
سوال‌های مروری فصل ۲۳۴
تمرین‌ها ۲۳۵

فصل ۹: مدل‌سازی تعامل شئ ۲۳۷
۹-۱ مدل‌سازی تعامل شئ چیست؟ ۲۳۷
۹-۲ نمودار توالی UML ۲۳۹

۹-۲-۱ انگاره‌ها و نمادگان ۲۳۹
۹-۲-۲ نمایش نمونه‌های یک کلاس ۲۳۹
۹-۲-۳ چند نمودار توالی نمونه ۲۴۱
۹-۲-۴ استفاده از نمودار توالی در تحلیل و طراحی ۲۴۳
۹-۲-۵ استفاده‌ی درست از نمادگان ۲۴۴
۹-۳ گام‌های مدل‌سازی تعامل شئ ۲۴۶
۹-۳-۱ جمع‌آوری اطلاعات درباره‌ی فرایندهای کسب‌وکار ۲۴۶
۹-۳-۲ شناسایی گام‌های غیربدیهی ۲۴۸
۹-۳-۳ نوشتن سناریو برای گام‌های غیربدیهی ۲۴۹
۹-۳-۴ ساخت جداول سناریو ۲۵۲
۹-۳-۵ چگونه سناریو بنویسیم؟ ۲۵۲
۹-۳-۶ استنتاج نمودارهای توالی از جداول سناریو ۲۵۷
۹-۳-۷ فهرست بررسی جهت بازبینی مدل‌سازی تعامل شئ ۲۶۶
۹-۴ اعمال اصول چابکی ۲۶۷
۹-۵ پشتیبانی ابزاری برای مدل‌سازی تعامل شئ ۲۷۰
خلاصه ۲۷۰
برای مطالعه‌ی بیش‌تر ۲۷۱
سوال‌های مروری فصل ۲۷۱
تمرین‌ها ۲۷۲

فصل ۱۰: اعمال الگوهای واگذاری مسئولیت ۲۷۳
۱۰-۱ الگوهای طراحی چیستند؟ ۲۷۴
۱۰-۲ چرا الگوهای طراحی؟ ۲۷۵
۱۰-۳ الگوهای خاص موقعیت و واگذاری- مسئولیت ۲۷۵
۱۰-۴ تبیین الگو ۲۷۶
۱۰-۵ الگوی کنترلگر ۲۷۶
۱۰-۵-۱ مثال انگیزشی ۲۷۸
۱۰-۵-۲ کنترلگر چیست؟ ۲۷۹
۱۰-۵-۳ اعمال الگوی کنترلگر ۲۸۰
۱۰-۵-۴ گونه‌های کنترلگر ۲۸۳
۱۰-۵-۵ ردگیری وضعیت مورد کاربرد ۲۸۴
۱۰-۵-۶ کنترلگر متورم ۲۸۵
۱۰-۵-۷ مقایسه‌ی طراحی‌های مختلف ۲۸۵
۱۰-۵-۸ چه موقع باید الگوی کنترلگر را به‌کار برد؟ ۲۸۷
۱۰-۵-۹ رهنمون‌هایی برای استفاده از کنترلگر ۲۸۸

۱۰-۶ الگوی خبره ۲۸۹
۱۰-۶-۱ خبره‌ی اطلاعات ۲۸۹
۱۰-۶-۲ اعمال الگوی خبره ۲۹۰
۱۰-۶-۳ الگوی خبره شامل بیش از یک شئ ۲۹۰
۱۰-۶-۴ چه موقع الگوی خبره به‌کار می‌رود؟ ۲۹۲
۱۰-۶-۵ رهنمون‌هایی برای استفاده از خبره ۲۹۲
۱۰-۷ الگوی ایجادکننده ۲۹۲
۱۰-۷-۱ ایجادکننده چیست؟ ۲۹۳
۱۰-۷-۲ اعمال الگوی ایجادکننده ۲۹۴
۱۰-۷-۳ فواید الگوی ایجادکننده ۲۹۶
۱۰-۷-۴ چه موقع الگوی ایجادکننده به‌کار می‌رود؟ ۲۹۶
خلاصه ۲۹۶
برای مطالعه‌ی بیش‌تر ۲۹۶
سوال‌های مروری فصل ۲۹۷
تمرین‌ها ۲۹۸

فصل ۱۱: استنتاج نمودار کلاس طراحی ۲۹۹
۱۱-۱ نمودار کلاس طراحی چیست؟ ۳۰۱
۱۱-۲ فواید نمودار کلاس طراحی ۳۰۱
۱۱-۳ گام‌های استنتاج یك نمودار کلاس طراحی ۳۰۲
۱۱-۳-۱ شناسایی کلاس‌ها ۳۰۲
۱۱-۳-۲ شناسایی متدها ۳۰۴
۱۱-۳-۳ شناسایی صفت‌ها ۳۰۵
۱۱-۳-۴ روابط بین کلاس‌ها ۳۰۷
۱۱-۳-۵ شناسایی روابط ۳۰۹
۱۱-۳-۶ فهرست بررسی برای بازبینی نمودار کلاس طراحی ۳۱۲
۱۱-۴ سازمان‌دهی کلاس‌ها با نمودار بسته ۳۱۲
۱۱-۵ اعمال اصول چابکی ۳۱۴
۱۱-۶ پشتیبانی ابزاری از نمودار کلاس طراحی ۳۱۵
خلاصه ۳۱۶
برای مطالعه‌ی بیش‌تر ۳۱۶
سوال‌های مروری فصل ۳۱۶
تمرین‌ها ۳۱۶

فصل ۱۲: طراحی واسط کاربر ۳۱۷
۱۲-۱ طراحی واسط کاربر چیست؟ ۳۱۸

۱۲-۲ چرا طراحی واسط کاربر مهم است؟ ۳۱۹
۱۲-۳ ویجت‌های واسط گرافیکی کاربر ۳۲۰
۱۲-۳-۱ ویجت‌های دربرگیرنده ۳۲۰
۱۲-۳-۲ ویجت‌های ورودی، خروجی و نمایش اطلاعات ۳۲۲
۱۲-۳-۳ رهنمون‌هایی برای استفاده از ویجت‌های GUI ۳۲۳
۱۲-۴ فرایند طراحی واسط کاربر ۳۲۵
۱۲-۴-۱ مطالعه‌ی موردی: طراحی واسط کاربر برای یك نرم‌افزار ویرایشگر نمودار ۳۲۶
۱۲-۴-۲ شناسایی نمایش‌های اصلی سیستم ۳۲۶
۱۲-۴-۳ تولید یک طراحی چیدمان اولیه ۳۲۸
۱۲-۴-۴ تبیین رفتار تعاملی ۳۳۰
۱۲-۴-۵ ساخت یک پیش‌نمونه ۳۳۱
۱۲-۴-۶ ارزیابی طراحی واسط کاربر با کاربران ۳۳۳
۱۲-۴-۷ فهرست بررسی بازبینی طراحی واسط کاربر ۳۳۴
۱۲-۵ طراحی قابلیت‌های پشتیبانی از کاربران ۳۳۵
۱۲-۶ رهنمون‌هایی برای طراحی واسط کاربر ۳۳۶
۱۲-۷ اعمال اصول چابکی ۳۳۸
۱۲-۸ پشتیبانی ابزاری برای طراحی واسط کاربر ۳۴۰
خلاصه ۳۴۰
برای مطالعه‌ی بیش‌تر ۳۴۰
سوال‌های مروری فصل ۳۴۱
تمرین‌ها ۳۴۱

پیوست‌ها ۳۴۳

فصل آ: فرایند شخصی نرم‌افزار: تخمین، برنامه‌ریزی و تضمین کیفیت ۳۴۵
آ-۱ تخمین تلاش در PSP ۳۴۵
آ-۲ تضمین کیفیت نرم‌افزار در PSP ۳۴۷
آ-۳ طراحی و کیفیت ۳۴۷

فصل ب: تکنولوژی‌های جاوا ۳۴۹
ب-۱ شروع با اتصال پایگاه‌داده ۳۴۹
ب-۱-۱ اتصال پایگاه‌داده چیست؟ ۳۴۹
ب-۱-۲ تنظیم منابع داده ۳۵۰
ب-۱-۳ دسترسی به پایگاه‌داده‌ها از یک برنامه ۳۵۰

ب-۲ آغاز یادگیری Swing ۳۵۲
ب-۲-۱ ایجاد پنجره‌ی اصلی با JFrame ۳۵۲
ب-۲-۲ استفاده از مدیر چیدمان (Layout) برای مرتب‌سازی مولفه‌ها ۳۵۴
ب-۲-۳ پردازش رویدادهای دکمه‌ها با Action Listener ۳۵۵
ب-۲-۴ پیاده‌سازی قابلیت‌های رسم ۳۵۶
ب-۳ شروع به‌کار با صفحات سرور جاوا ۳۵۷
ب-۳-۱ صفحات سرور جاوا چه هستند؟ ۳۵۷
ب-۳-۲ جریان کار JSP ۳۵۷
ب-۳-۳ نصب یک سرویس‌دهنده‌ی وب با JSP Container ۳۵۸
ب-۳-۴ استفاده از صفحات سرور جاوا ۳۵۸

فصل پ: ابزارهای نرم‌افزاری ۳۶۳

پ-۱ NetBeans ۳۶۳
پ-۲ استفاده از JUnit ۳۶۴
پ-۳ اجرای JUINT در NetBeans ۳۶۸
پ-۴ ابزار پوشش Emma ۳۶۸
پ-۵ ابزار پوشش Cobertura ۳۶۹
پ-۶ آزمون برنامه‌های کاربردی تحت وب با HttpUnit ۳۷۱
پ-۶-۱ پیکربندی یک IDE برای استفاده از HttpUnit ۳۷۱
پ-۶-۲ پیاده‌سازی موردهای آزمون در HttpUnit ۳۷۱
پ-۷ استفاده از CVS و SubVersion در NetBeans ۳۷۲
پ-۷-۱ ایجاد یک مخزن راه‌دور CVS ۳۷۳
پ-۷-۲ تنظیم Subversion در NetBeans ۳۷۴
پ-۷-۳ به امانت گرفتن فایل‌ها از یک مخزن ۳۷۶
پ-۷-۴ ویرایش منابع و مشاهده‌ی تغییرات ۳۷۸
پ-۷-۵ مشاهده‌ی وضعیت فایل ۳۷۸
پ-۷-۶ مقایسه‌ی نسخه‌های مروری فایل ۳۷۹
پ-۷-۷ ادغام تغییرات از مخزن ۳۷۹
پ-۷-۸ حل برخوردها ۳۷۹
پ-۷-۹ به‌روزرسانی کپی‌های محلی ۳۸۰
پ-۷-۱۰ اعمال فایل‌های محلی بر مخزن ۳۸۰
پ-۷-۱۱ وارد کردن فایل‌ها به یک مخزن ۳۸۰

فصل ت: شرح پروژه‌ها ۳۸۱

ت-۱ سیستم کرایه‌ی خودرو ۳۸۱

ت-۲ سیستم ملی خدمات نمایشگاهی ۳۸۲
ت-۳ سیستم مدیریت تحصیل در خارج‌ازکشور ۳۸۳
ت-۴ ویرایشگر نمودار کلاس UML ۳۸۵
ت-۵ شبیه‌ساز ارتباطات رادیویی ۳۸۷
ت-۶ محیط آزمون حالت اشیا ۳۸۹

مراجع ۳۹۱

واژه‌نامه انگلیسی به فارسی ۳۹۸

واژه‌نامه فارسی به انگلیسی ۴۰۹

فهرست شکل‌ها

۱-۱ سه شاخه از فعالیت‌های چرخه‌ی حیات ۶
۱-۲ مدل آبشاری در توسعه نرم‌افزار ۷
۱-۳ طراحی سیستم با استفاده از نمودار بلوکی ۸
۱-۴ یک معماری nتایی برای مدیریت حساب‌ها ۱۰
۱-۵ SQA در چرخه‌ی حیات سنتی ۱۲

۲-۱ ویژگی‌های مسائل خوش‌رفتار و بدرفتار ۲۴
۲-۲ فرایند حلزونی ۲۷
۲-۳ شرح فرایند یکنواخت ۲۸
۲-۴ فعالیت‌های PSP به‌وسیله‌ی اسکریپت‌ها توصیف می‌شوند. ۳۰
۲-۵ یاداشت ثبت زمان PSP ۳۱
۲-۶ یاداشت ثبت خرابی PSP ۳۲
۲-۷ خلاصه انواع خرابی در PSP ۳۲
۲-۸ فرم خلاصه برنامه‌ی پروژه PSP0 . . . . . ۳۳
۲-۹ فرایند نرم‌افزار تیمی ۳۴
۲-۱۰ اسکریپت توسعه‌ی TSP ۳۵
۲-۱۱ تفاوت متدولوژی و فرایند نرم‌افزاری ۴۲
۲-۱۲ خلاصه‌ای از چند روش چابک ۴۵
۲-۱۳ فرایند روش توسعه‌ی پویای سیستم ۴۷
۲-۱۴ فعالیت‌های توسعه‌ی اسکرام ۴۸
۲-۱۵ فرایند آبشاری و فرایند چابک ۵۱
۲-۱۶ نمای کلی از متدولوژی یکنواخت چابک ۵۲
۲-۱۷ نمودار مورد کاربرد برای سیستم کتابخانه ۵۳

۲-۱۸ یک مدل دامنه‌ی نمونه برای سیستم کتابخانه ۵۳
۲-۱۹ نمونه ای از یک مورد کاربرد گسترش‌یافته ۵۴
۲-۲۰ مدل‌سازی تعامل اشیا برای امانت مدرک ۵۴
۲-۲۱ نمودار کلاس طراحی با ترتیب پیاده‌سازی ۵۶

۳-۱ فرایند مهندسی سیستم ۶۳
۳-۲ تجزیه‌ی سیستم به‌صورت سلسله‌مراتبی از زیرسیستم‌ها ۶۹
۳-۳ خوشه‌های کارکردی و زیرسیستم‌های کارکردی ۷۰
۳-۴ جدول ردیابی‌پذیری نیازمندی‌ها به زیرسیستم‌ها ۷۲
۳-۵ نمودار بلوکی برای یک سیستم مدیریت بار در فرودگاه ۷۲
۳-۶ نمودار بلوکی که پالایش سیستم را نشان می‌دهد ۷۳
۳-۷ مدل‌سازی سیستم با استفاده از یک نمودار مولفه‌ی استریوتایپ‌دار ۷۴
۳-۸ ارتباط بین نمودارهای SysML و UML ۷۵
۳-۹ نمودار تعریف بلوک SysML برای ABHS ۷۶
۳-۱۰ نمودار داخل بلوک SysML برای زیرسیستم پایانه از ABHS ۷۶
۳-۱۱ مدل‌سازی جریان مواد ۷۷
۳-۱۲ نمایشی از واسط سخت‌افزار-نرم‌افزار ۷۷
۳-۱۳ یک مدل دامنه‌ی زمینه برای نرم‌افزار کنترل ABHS ۷۹
۳-۱۴ کنترل تغییر پیکربندی ۸۲

۴-۱ مشکلات مربوط به نیازمندی‌ها در توسعه‌ی نرم‌افزار ۹۱
۴-۲ استخراج نیازمندی‌های نرم‌افزار ۹۴
۴-۳ مدل‌سازی کاربردهای موجود با نمودارهای UML ۹۹
۴-۴ نمودار فعالیت برای یک فرایند کسب‌وکار موجود ۱۰۰
۴-۵ استاندارد IEEE Std 830-1998 SRS ۱۰۵
۴-۶ فعالیت‌های مدیریت نیازمندی‌ها ۱۰۹

۵-۱ چند انگاره و نماد پراستفاده در نمودار کلاس ۱۱۶
۵-۲ نمایش کلاس‌ها از دو نمای فشرده و گسترده ۱۱۷
۵-۳ نمونه‌هایی از یک انجمن ۱۲۰
۵-۴ نمادهایی برای بیان شکل‌های مختلف تعدد ۱۲۲
۵-۵ نمایش روابط و کلاس تجمیع ۱۲۳
۵-۶ نمایش جدولی اشیای یک کلاس انجمنی ۱۲۶
۵-۷ گام‌های مدل‌سازی دامنه ۱۲۷
۵-۸ نتیجه‌ی طوفان فکری از یک قطعه شرح SAMS ۱۳۰
۵-۹ قوانین دسته‌بندی برای دسته‌بندی نتایج طوفان فکری ۱۳۱
۵-۱۰ کدهای دسته‌بندی ۱۳۱

۵-۱۱ دسته‌بندی مفاهیم طوفان فکری برای SAMS ۱۳۳
۵-۱۲ قوانین تبدیل: از نتیجه‌ی کلاس‌بندی‌شده به نمودار کلاس ۱۳۵
۵-۱۳ تبدیل نتیجه‌ی کلاس‌بندی به یک نمودار کلاس ۱۳۶
۵-۱۴ کاربردهای نادرست اجتماع ۱۳۶
۵-۱۵ توصیف نادرست روابط ۱۳۷
۵-۱۶ قطعه و قطعه‌ی مرکب ۱۳۸
۵-۱۷ زیرکلاس، روابط را از کلاس مافوق به ارث می‌برد. ۱۳۹
۵-۱۸ توصیف جزیی CRS، با عبارات خط‌کشی شده ۱۴۰
۵-۱۹ کلاس‌بندی مفاهیم کاربرد CRS ۱۴۱
۵-۲۰ تصویرسازی مدل دامنه‌ی اجاره‌ی خودرو ۱۴۲
۵-۲۱ شناسایی دستی و کلاس‌بندی دستی ۱۴۶
۵-۲۲ نمودار کلاس تولیدشده برای مدل دامنه ۱۴۷

۶-۱ فرایند طراحی معماری ۱۵۴
۶-۲ چهار نوع مختلف سیستم و رفتار ۱۵۶
۶-۳ نگاشت انواع سیستم به سبک‌های معماری ۱۶۱
۶-۴ معماری چندلایه ۱۶۲
۶-۵ به‌کاربردن سبک معماری مشتری-خادم ۱۶۳
۶-۶ معماری برنامه‌ی‌اصلی و زیربرنامه‌ها ۱۶۴
۶-۷ به‌کاربردن سبک طراحی معماری برنامه‌ی اصلی و زیربرنامه‌ها ۱۶۶
۶-۸ معماری سیستم رویداد-رانده ۱۶۷
۶-۹ نمایش معماری سیستم رویداد-رانده ۱۶۷
۶-۱۰ نمایش سبک معماری چارچوب ماندگاری ۱۶۸
۶-۱۱ نمودار بسته برای یک سیستم اطلاعاتی کتابخانه ۱۷۲

۷-۱ مثالی از مورد کاربردها، اعمال و کنش‌ها ۱۹۰
۷-۲ گام‌های مدل‌سازی مورد کاربردها ۱۹۱
۷-۳ شناسایی مورد کاربردها برای LIS ۱۹۲
۷-۴ شناسایی مورد کاربرد برای SAMS ۱۹۴
۷-۵ زیرسیستم‌هایی که توسط کنشگرها شناسایی شده‌اند. ۱۹۷
۷-۶ زیرسیستم‌های منتج از افراز ارتباطی ۱۹۷
۷-۷ جدول نمونه‌ی ردیابی‌پذیری نیازمندی-مورد کاربرد ۱۹۹
۷-۸ نمودار مورد کاربرد برای مثال ATM ۲۰۲
۷-۹ انگاره‌ها و نمادگان در نمودار مورد کاربرد ۲۰۳
۷-۱۰ نمودار مورد کاربرد برای SAMS ۲۰۳
۷-۱۱ (الف) استفاده از ارث‌بری کنشگرها (ب) جهت ساده‌سازی نمودارهای مورد کاربرد ۲۰۴
۷-۱۲ نمایش فرایندهای جایگزین با استفاده از ارث‌بری مورد کاربردها ۲۰۵

۷-۱۳ رابطه‌ی گسترش بین مورد کاربردها ۲۰۵
۷-۱۴ نمایش رابطه‌ی شمول مورد کاربرد ۲۰۶
۷-۱۵ تخصیص مورد کاربردها به تکرارها ۲۰۸
۷-۱۶ نمودار مورد کاربرد کلاً پیچیده که باید ازآن پرهیز نمود. ۲۱۰
۷-۱۷ افراز مبتنی بر نقش باعث کاهش پیچیدگی و افزایش چسبندگی می‌شود. ۲۱۱
۷-۱۸ نمودارهای مورد کاربرد که نادرست رسم شده‌اند. ۲۱۲
۷-۱۹ شناسایی مورد کاربردها به صورت دستی ۲۱۵
۷-۲۰ استنتاج خودکار مورد کاربردها ۲۱۵

۸-۱ مورد کاربرد گسترده برای سیستم اطلاعات کتابخانه ۲۲۱
۸-۲ ایجاد یک مورد کاربرد گسترده ۲۲۲
۸-۳ استفاده مجدد اطلاعات از مدل دامنه ۲۲۳
۸-۴ تبیین جریان‌های جایگزین در مورد کاربردهای گسترده ۲۲۴
۸-۵ یک مورد کاربرد گسترده و پیش‌نمونه‌های واسط کاربر ۲۲۵
۸-۶ مورد کاربرد گسترده با ارجاعاتی به نمایش واسط کاربر ۲۲۵
۸-۷ نمایش ۱ از پیش‌نمونه‌ی واسط کاربر ۲۲۶
۸-۸ نمایش ۲ از پیش‌نمونه‌ی واسط کاربر ۲۲۶
۸-۹ پیش‌شرط مورد کاربرد برای وضعیت یک شئ ۲۲۸
۸-۱۰ مورد کاربرد گسترده‌ای که شامل مورد کاربرد دیگری است ۲۲۹
۸-۱۱ مشخص کردن جریان‌های جایگزین به مورد کاربردهای دیگر ۲۳۰
۸-۱۲ انتقال وجه از طریق چک، مورد کاربرد دیگری را ادامه می‌دهد. ۲۳۰
۸-۱۳ تبیین مورد کاربرد گسترده مشکل‌دار ۲۳۱
۸-۱۴ تعامل کنشگر سیستم که بدون دلیل پیچیده شده است ۲۳۲

۹-۱ انگاره‌ها و نمادگان نمودار توالی ۲۴۰
۹-۲ نمونه‌های مختلف یک شئ از یک کلاس ۲۴۱
۹-۳ نمودار توالی برای سناریوی Login ۲۴۲
۹-۴ نمودار توالی برای مورد کاربرد دریافت مدرک ۲۴۳
۹-۵ تفاوت میان تحلیل و طراحی ۲۴۵
۹-۶ استفاده‌های درست و نادرست از نمادگان ۲۴۵
۹-۷ مروری بر مدل‌سازی تعامل شئ ۲۴۷
۹-۸ مورد کاربرد گسترده خروج مدرک ۲۴۸
۹-۹ سناریوی گام ۴ از شکل ۹-۸ . . . . . ۲۵۱
۹-۱۰ سناریوی علامت‌گذاری شده‌ی یک گام غیربدیهی ۲۵۳
۹-۱۱ جدول سناریو برای مورد کاربردی خروج مدرک ۲۵۴
۹-۱۲ کاربرگ برای توسعه سناریو ۲۵۵
۹-۱۳ واگذاری وظایف به اشیا ۲۵۷

۹-۱۴ تکمیل اشیای درخواست‌کننده ۲۵۷
۹-۱۵ تکمیل توصیف سناریو ۲۵۸
۹-۱۶ از جدول سناریو به نمودار توالی ۲۶۰
۹-۱۷ جدول سناریوی Login همراه با شماره ردیف ۲۶۱
۹-۱۸ یک نمودار توالی غیررسمی دارای حلقه ۲۶۱
۹-۱۹ تصمیم‌گیری درباره‌ی نام، پارامترها، و متغیرهای برگشتی توابع ۲۶۳
۹-۲۰ اشیایی که از یک متغیر کنترلی به‌عنوان واسط استفاده می‌کنند. ۲۶۵
۹-۲۱ بهبودیافته: جفت‌شدگی کنترلی حذف شده است. ۲۶۶
۹-۲۲ نمودار توالی خروج مدرک که در آن پیام‌ها به‌طور رسمی مشخص شده‌اند. ۲۶۷
۹-۲۳ مورد کاربرد گسترده با دو گام غیربدیهی ۲۶۹
۹-۲۴ توصیف یک سناریوی ساده شده ۲۷۰

۱۰-۱ الگوی تک ۲۷۵
۱۰-۲ تبیین الگوی تک ۲۷۷
۱۰-۳ یک نمودار توالی متداول برای خروج مدرک ۲۸۰
۱۰-۴ نمایش الگوی کنترلگر ۲۸۱
۱۰-۵ توصیف کنترلگر بر اساس قالب الگو ۲۸۲
۱۰-۶ اعمال الگوی کنترلگر ۲۸۳
۱۰-۷ نمودار انتقال وضعیت برای مورد کاربرد Reserve Flight ۲۸۵
۱۰-۸ یک کنترلگر متورم که همه کار را خودش انجام می‌دهد. ۲۸۶
۱۰-۹ نمایش الگوی خبره ۲۹۰
۱۰-۱۰ یک نمودار توالی Login ۲۹۱
۱۰-۱۱ اعمال الگوی خبره ۲۹۱
۱۰-۱۲ نمایش الگوی ایجادکننده ۲۹۴
۱۰-۱۳ چه کسی باید یک شئ Chapter ایجاد کند؟ ۲۹۴
۱۰-۱۴ چه کسی باید مسئول ایجاد شئ Position باشد؟ ۲۹۵
۱۰-۱۵ چه کسی باید شئ DBMgr را ایجاد کند؟ ۲۹۵
۱۰-۱۶ چه کسی باید شئ Loan را ایجاد کند؟ ۲۹۵
۱۰-۱۷ خلاصه ای از الگوهای GRASP ۲۹۷

۱۱-۱ شناسایی کلاس‌ها از روی نمودارهای توالی ۳۰۳
۱۱-۲ DCD که کلاس‌هایی را شناسایی کرده است. ۳۰۴
۱۱-۳ شناسایی متدهای یك کلاس ۳۰۵
۱۱-۴ کلاس‌ها و متدهایی که برای آن‌ها شناسایی شده است ۳۰۵
۱۱-۵ DCD که متدهای آن پر شده‌اند ۳۰۷
۱۱-۶ شناسایی صفت‌ها با استفاده از قوانین ۱ تا ۳ ۳۰۸
۱۱-۷ شناسایی صفت‌ها با استفاده از قوانین ۴ تا ۵ ۳۰۸

۱۱-۸ یک DCD که خصیصه‌ها در آن شناسایی شده‌اند ۳۰۹
۱۱-۹ شناسایی روابط ۳۱۱
۱۱-۱۰ DCD حاوی روابط و وابستگی call ۳۱۲
۱۱-۱ ابسته‌ها و زیرسیستم‌های کارکردی ۳۱۴
۱۱-۲ ابسته‌بندی کلاس‌ها بر طبق یک سبک معماری ۳۱۵

۱۲-۱ یك نرم‌افزار حمل‌ونقل که کار کاربر را آسان می‌کند. ۳۲۱
۱۲-۲ ویجت‌هایی برای ورودی کاربر و نمایش نتایج ۳۲۴
۱۲-۳ فرایند طراحی واسط کاربر ۳۲۵
۱۲-۴ مورد کاربردی گسترده «ویرایش نمودار حالت» ۳۲۷
۱۲-۵ نمایش‌های اصلی سیستم برای مورد کاربردی ویرایش نمودار حالت ۳۲۸
۱۲-۶ نمایش‌های اصلی سیستم برای مورد کاربردی ویرایش نمودار حالت ۳۲۹
۱۲-۷ طراحی پنجره‌ی اصلی برای یک ویرایشگر نمودار حالت ۳۳۰
۱۲-۸ طراحی یک جعبه‌ی گفت‌وگو برای ویرایش یک حالت ۳۳۱
۱۲-۹ یک نمودار حالت واسط کاربری که تقریباً کامل شده است. ۳۳۲

ب-۱ Java DataBase Connectivity (JDBC) ۳۵۰
ب-۲ دسترسی به یک پایگاه‌داده از یک برنامه‌ی جاوا ۳۵۱
ب-۳ پیاده‌سازی ناحیه‌ی رسم ۳۵۶
ب-۴ پیاده‌سازی زیرکلاس‌های الگوی حالت ۳۵۷
ب-۵ شرح جریان کاری JSP ۳۵۸
ب-۶ یک فرم و Java bean متناظر آن ۳۵۹
ب-۷ یک صفحه‌ی JSP برای پردازش یک درخواست و نمایش خروجی ۳۶۰
ب-۸ ساختار کشویی برای یک فایل war. ۳۶۱

پ-۱ نواحی کاری معمول در NetBeans ۳۶۴
پ-۲ کلاس TestCase برای آزمودن Purge ۳۶۵
پ-۳ واسط کاربری اجراکننده‌های آزمون JUnit ۳۶۷
پ-۴ گزارش نمونه‌ای از پوشش Cobertura ۳۷۰
پ-۵ مشاهده‌ی نتایج تست واحد در NetBeans ۳۷۰
پ-۶ آزمون یک صفحه‌ی سرور با استفاده از HttpUnit ۳۷۲
پ-۷ دستورات پراستفاده CVS ۳۷۳
پ-۸ مشخص کردن مسیر فایل اجرایی Subversion Client ۳۷۵
پ-۹ به امانت گرفتن فایل از یک مخزن ۳۷۶
پ-۱۰ آیکون‌های عملیاتی در پنجره‌ی نسخه‌بندی ۳۷۸
پ-۱۱ وارد کردن فایل‌ها به یک مخزن راه دور CVS ۳۸۰

ت-۱ جریان کار ابزار ۳۹۰

ت-۲ یک واسط کاربری گرافیکی برای ویرایش مدل آزمون ClassBench ۳۹۰

پیش‌گفتار مترجمان

بی‌شک مهندسی نرم‌افزار علم روز دنیاست. به گوشه‌وکنار و به دستگاه‌هایی که با آن‌ها سروکار دارید نگاه کنید. عملکرد بسیاری از وسایل زندگی ما توسط نرم‌افزاری کنترل می‌شود که درون آن‌ها قرار داده شده است. برخی نرم‌افزارها کنترل دستگاه‌هایی را در اختیار می‌گیرند که عملکردی حساس دارند. سیستم کنترل خودکار هواپیما و سیستم کنترل علایم حیاتی بیمار در اتاق مراقبت‌های ویژه، نمونه‌هایی از این نرم‌افزارها هستند که هرگونه اشکال در عملکرد آن‌ها می‌تواند منجر به یک فاجعه گردد. افزون بر این، بیشتر امور در سازمان‌ها به کمک نرم‌افزارها انجام می‌شود. نرم‌افزار، به‌ویژه وقتی بزرگ و پیچیده باشد و با امور حیاتی سروکار داشته باشد، باید به‌دست مهندسان نرم‌افزار و در قالب پروژه‌های نرم‌افزاری نوشته شود. چنان‌چه نرم‌افزار توسط افراد غیرمتخصص ساخته شود، احتمال شکست پروژه و یا تولید نرم‌افزار بی‌کیفیت زیاد است. با کمال تأسف باید اذعان کرد که فهرست پروژه‌های نرم‌افزاری شکست‌خورده، در دنیا و نیز در کشور ما، هم‌چنان طولانی است. یکی از مهم‌ترین دلایل شکست پروژه‌های نرم‌افزاری، نداشتن متدولوژی و یا به‌کارگیری نادرست گام‌های آن می‌باشد. آموزش مهندسی نرم‌افزار، به‌ویژه وقتی با تاکید بر یک متدولوژی مناسب باشد، می‌تواند موجب تربیت مهندسانی شود که با مدیریت درست پروژه‌ها، آن‌ها را به موفقیت برسانند.

کتابی که پیش رو دارید، با ریزبینی و دقت زیاد، اصول مهندسی نرم‌افزار را بر اساس فناوری شیءگرایی تعریف نموده و سعی در آموزش یک متدولوژی نوین در توسعه‌ی نرم‌افزار دارد. این متدولوژی که آموزه‌های متدولوژی فرایند یکنواخت و روش‌های چابک را درکنار هم به‌کار گرفته است، «متدولوژی چابک یکنواخت» نام‌گذاری شده است. نویسنده حاصل سال‌ها تلاش در تدریس اصول مهندسی نرم‌افزار را با تجارب آموخته از صنعت، درهم آمیخته و در اختیار خواننده قرار داده است. نتیجه، گرچه کتابی طولانی و مفصل شده است، ولی به عقیده‌ی مترجمان می‌تواند مرجع بسیار مفیدی برای کسانی باشد که قصد دارند اصول مهندسی نرم‌افزار را همراه با یک متدولوژی نوین فراگیرند. مخاطبان این کتاب در درجه‌ی اول دانشجویان دوره‌های کارشناسی و کارشناسی ارشد رشته‌های مرتبط با کامپیوتر (اعم از نرم‌افزار، سخت‌افزار، فناوری اطلاعات، و علوم کامپیوتر) و در درجه‌ی دوم مهندسان شاغل در صنعت تولید نرم‌افزار و نیز مدیران این حرفه می‌باشند.

با توجه به منابع متعدد و مفیدی که نویسنده‌ی کتاب در اختیار استادان محترم دانشگاه‌ها گذاشته است، این کتاب را می‌توان به‌عنوان مرجع درس‌های «مهندسی نرم‌افزار» و «تحلیل و طراحی شیءگرا» استفاده نمود. البته بخش‌های انتخابی از کتاب می‌تواند در دروس تخصصی‌تری همچون «آزمون نرم‌افزار» نیز استفاده شود که در این رابطه، خواننده را به پیش‌گفتار نویسنده ارجاع می‌دهیم. با توجه به حجم کتاب، تصمیم مترجمان بر آن شد که ترجمه کتاب در دو جلد در اختیار خوانندگان

قرار گیرد. جلد اول شامل فصل‌های اول تا دوازدهم می‌باشد که بیشتر به مباحث تحلیل و طراحی شئ‌گرا می‌پردازد. جلد دوم به فصل‌های سیزدهم تا بیست‌وچهارم اختصاص خواهد داشت که مباحثی چون الگوهای طراحی، اطمینان از کیفیت، آزمون، مدیریت پیکربندی، و مدیریت پروژه‌های نرم‌افزاری را دربرمی‌گیرد.

ذکر این نکته را ضروری می‌دانیم که امر ترجمه، کاری تخصصی و بس دشوار است و اگر مترجم دچار وسواس شود ممکن است هیچ‌گاه کار به نتیجه نرسد. از همین رو، در حالی که مترجمان سعی کرده‌اند در ترجمه و انتخاب واژگان فارسی نهایت دقت را بنمایند ولی گاهی صلاح را در این دیده‌اند که به منظور فهم بهتر مطلب واژه‌ی کاملاً فارسی را کنار گذاشته و از واژگان ملموس‌تر و یا حتی از خود کلمه‌ی انگلیسی استفاده نمایند (مثلاً در مورد واژه‌ی «متدولوژی» که برای آن معادل‌های فارسی «روش‌شناسی» یا «شیوه‌شناسی» پیشنهاد شده است، یا در مورد واژه‌ی «کلاس» که معادل فارسی «رده» دارد، از واژه‌ی اصلی استفاده شده است). امید است اهل ادب و پارسی‌دوستان این امر را بر ما خرده نگیرند و بدانند که انتخاب‌ها با آگاهی صورت پذیرفته است.

نکته‌ی دیگر این‌که، نویسنده‌ی کتاب برای تمام شکل‌ها، جدول‌ها، نمودارها و حتی قطعات کد، از واژه‌ی «شکل» استفاده نموده است که ما نیز همین روال را برگزیده و به همه‌ی موارد فوق شکل اطلاق نموده‌ایم. این انتخاب نیز از روی آگاهی و به‌منظور سازگاری با کتاب اصلی انجام شده است.

در پایان، بر خود لازم می‌دانیم از همکاری و زحمات تمامی دوستان و دانشجویانی که در ترجمه‌ی این کتاب ما را یاری رسانده‌اند تشکر نماییم. به طور خاص از افراد زیر سپاسگزاریم.

- آقای مهندس یوسف امیری، که با پیشنهادهای ارزشمند خود در انتخاب واژگان فارسی کمک شایانی به ما نمودند و البته در برخی موارد به دلیل نو بودن واژه ترجیح داده‌ایم که از معادل‌های قدیمی‌تر ولی ملموس‌تر استفاده کنیم.
- آقای دکتر هومان نیک‌مهر، که قالب حروف‌چینی آماده‌ای را تحت بسته‌ی زی‌پرشین در اختیار ما قرار دادند که کار حروف‌چینی را بسیار سرعت بخشید.
- خانم مهندس زهرا رئیس‌جعفری، که حروف‌چینی کتاب را با دقت و نظمی ستودنی انجام داده و کتابی زیبا تولید نمودند.
- آقای مهندس محمدرضا تاج‌وری، که زحمت طراحی جلد را بر عهده داشتند.
- دانشجویان گرامی رشته‌ی مهندسی کامپیوتر دانشگاه اصفهان، که در تایپ و حروف‌چینی اولیه‌ی متون دست‌نویس و نیز بازخوانی متن نهایی ما را یاری رساندند.
- داوران گرامی که زحمت داوری کتاب را تقبل نموده و با پیشنهادهای ارزنده‌شان موجب بهبود کیفیت ترجمه شدند.

و سخن آخر این‌که، بی‌شک ترجمه‌ی کتابی با این حجم نمی‌تواند عاری از اشکال باشد، گرچه مترجمان تمام سعی خود را در ارائه‌ی کتابی با کیفیت به‌کار بسته‌اند. لذا، از شما خواننده‌ی گرامی تقاضا داریم با مراجعه به وبگاه کتاب به‌نشانی bahmanzamani.com/books ضمن آگاهی از آخرین وضعیت کتاب، هر گونه ایراد، اظهارنظر یا پیشنهاد را از طریق این وبگاه با ما در میان بگذارید.

بهمن زمانی - افسانه فاطمی

بهار ۱۳۹۵

پیش‌گفتار

پیش‌زمینه

امروزه کامپیوترها در همه‌ی بخش‌های جامعه‌ی ما به‌طور گسترده‌ای مورد استفاده قرار می‌گیرند و با اجرای نرم‌افزارهای کاربردی، کارهای گوناگونی انجام می‌دهند. به این ترتیب، بازار کار برای مهندسین نرم‌افزار به‌سرعت درحال توسعه است. مجله‌ی پول[۱] در شماره‌ی مارس ۲۰۰۶ خود، مهندس نرم‌افزار را به‌عنوان اولین شغل از ۵۰ شغل برتر آمریکا رده‌بندی کرده است. اداره‌ی آمار کار[۲]، پیش‌بینی کرده است که بین سال‌های ۲۰۱۰ و ۲۰۲۰، تعداد کل فرصت‌های شغلی در زمینه‌ی مهندس نرم‌افزار برنامه‌نویس و تحلیل‌گر سیستم، به ترتیب از ۵۲۰۸۰۰ به ۶۶۴۵۰۰ (۲۷٫۶% رشد) و از ۵۴۴۴۰۰ به ۶۶۴۸۰۰ (۲۲٫۱۰% رشد) فرصت شغلی افزایش یابد. برای این‌که بتوان وظایف یک مهندس نرم‌افزار یا تحلیل‌گر سیستم را انجام داد، تحصیل در رشته‌ی مهندسی نرم‌افزار ضروری است. اما، بنابر اطلاعات منتشرشده توسط BLS، «جوایز و مدارک دریافتی به تفکیک رشته‌ی تحصیلی بین سال‌های ۲۰۰۵ و ۲۰۰۶»، در سال ۲۰۰۶ تنها ۱۶۰ مدرک کارشناسی و ۶۰۰ مدرک کارشناسی‌ارشد در رشته‌ی مهندسی نرم‌افزار، و ۱۰۲۸۹ مدرک کارشناسی و ۴۵۱۲ مدرک کارشناسی ارشد در رشته‌ی علوم‌کامپیوتر صادر شده است. بدین ترتیب، به‌خصوص برای دانش‌آموختگان رشته‌ی مهندسی نرم‌افزار، شکاف قابل‌توجهی بین عرضه و تقاضا وجود دارد.

بسیاری از مردم، قلمرو و فایده‌ی مهندسی نرم‌افزار را، به‌عنوان یک حرفه، نمی‌دانند و این رشته اغلب به‌خوبی شناخته نمی‌شود. بسیاری از اصحاب رسانه، مهندسی نرم‌افزار را به‌عنوان نوشتن برنامه‌های جاوا معرفی می‌کنند. برخی از دانشجویان فکر می‌کنند که مهندسی نرم‌افزار شامل هر چیزی می‌شود که به نرم‌افزار مربوط است. برخی دیگر، همان‌گونه که داستان زیر نشان می‌دهد، فکر می‌کنند که مهندسی نرم‌افزار، رسم نمودارهای UML است. سال‌ها پیش، پس از اولین جلسه‌ی درس مهندسی نرم‌افزار شیءگرا[۳]، یک دانشجو به من گفت «استاد! این درس برای من بسیار ساده است، چون من قبلاً نمودارهای UML زیادی رسم کرده‌ام.» در پایان ترم، همان دانشجو دوباره نزد من آمد و گفت «استاد! باید به شما بگویم که ما برای این درس سخت کار کردیم و چیزهای زیادی در مورد طراحی شیءگرا یاد گرفتیم. طراحی شیءگرا، آن‌طور که من فکر می‌کردم، فقط رسم نمودارهای UML نیست.» پس مهندسی نرم‌افزار چیست؟ مهندسی نرم‌افزار رشته‌ای است که حیطه‌های مختلفی چون پژوهش، آموزش، و کاربرد فرایندهای مهندسی، متدولوژی‌ها، تضمین کیفیت، و مدیریت پروژه را شامل می‌شود تا باعث افزایش بهره‌وری[۴] و

---

[۱] Money Magazine
[۲] Bureau of Labor Statistics (BLS)
[۳] Object-Oriented Software Engineering (OOSE)
[۴] Productivity

کیفیت نرم‌افزار، و در عین حال کاهش هزینه و زمان تحویل محصول به بازار شود. OOSE شاخه‌ای از مهندسی نرم‌افزار است که جهان را به‌عنوان مجموعه‌ای از اشیای مرتبط و در تعامل با هم می‌بیند. ظهور زبان برنامه‌نویسی C++ در دهه‌ی ۱۹۸۰، به‌عنوان شروع دوران OOSE تلقی می‌شود. از آن زمان به بعد، تولید نرم‌افزار رشد جهانی غیرقابل‌پیش‌بینی خود را آغاز کرد و به‌دنبال آن، با ایجاد و پذیرش جهانی زبان مدل‌سازی یکنواخت[۱] و فرایند یکنواخت[۲]، شتاب بیشتری گرفت. به بیان دقیق‌تر می‌توان گفت فرایند نرم‌افزار، مراحل ایجاد نرم‌افزار و آنچه باید در هر مرحله انجام شود را توصیف می‌کند. فرایند، توضیحی درباره‌ی چگونگی انجام فعالیت‌های هر مرحله (با جزییات) ارائه نمی‌کند. یک زبان مدل‌سازی مانند UML، نمادها، نحو، و معانی لازم برای ارتباط و مستندسازی ایده‌های تحلیل و طراحی را تعریف می‌نماید. زبان UML و فرایند UP هم خوبند و هم لازم، ولی کافی نیستند، زیرا چگونگی تولید ایده‌های تحلیل و طراحی لازم برای رسم نمودارهای بامعنی UML را نشان نمی‌دهند.

انگیزه

برای پرکردن شکاف مطرح شده در بند آخر، ما به یک متدولوژی یا یک «کتاب آشپزی»[۳] نیاز داریم. برخلاف فرایند، متدولوژی عبارت است از توصیف جزیی از گام‌ها[۴] و رویه‌ها[۵] یا چگونگی انجام فعالیت‌ها تا حدی که یک فرد مبتدی بتواند با دنبال‌کردن آن، محصول نرم‌افزاری مطلوب را تولید و تحویل نماید. بدون داشتن متدولوژی، ممکن است یک مهندس نرم‌افزار مبتدی مجبور شود سال‌ها صرف آموزش ضمن خدمت کند تا مهارت‌های طراحی، پیاده‌سازی، و آزمون شیءگرا را فرا گیرد.

از دیگر انگیزه‌های نگارش این کتاب همچنین می‌توان به پیدایش تمایل به فرایندهای چابک[۶]، الگوهای طراحی[۷]، و توسعه‌ی آزمون‌رانده[۸] اشاره نمود. فرایندهای چابک بر روی کار تیمی، طراحی برای تغییر، استقرار سریع افزایش‌های کوچک از سیستم نرم‌افزاری، و توسعه در کنار مشتری و کاربران تاکید می‌کند. الگوهای طراحی، راه‌حل‌های مؤثری در طراحی هستند که برای مسائل متداول طراحی ارائه شده‌اند. این الگوها بازاستفاده[۹] از نرم‌افزار را تشویق کرده و ارتباط تیمی را تقویت می‌کنند. توسعه‌ی آزمون‌رانده، طرفدار نرم‌افزار آزمون‌پذیر است و تولید اسکریپت‌های آزمون[۱۰] قبل از پیاده‌سازی یک برنامه را ضروری می‌داند، تا بتوان بلافاصله و به‌دفعات برنامه را آزمایش نمود.

به‌عنوان مثال، احداث یک پارک تفریحی (شهربازی) را در نظر بگیرید. فرایند کلی کار شامل این مراحل است: برنامه‌ریزی[۱۱]، تأیید عمومی، تحلیل و طراحی، تأمین منابع مالی، تهیه نقشه‌های ساختمانی، ساخت، تأمین تجهیزات، نصب تجهیزات، افتتاح آزمایشی، و افتتاح عمومی. اما، تنها دانستن فرایند کلی کافی نیست. تیم توسعه باید بداند که در هر مرحله چه فعالیت‌هایی می‌بایست انجام شود. مثلاً، فعالیت‌های مرحله برنامه‌ریزی شامل توسعه یک مدل مفهومی اولیه، مطالعه‌ی امکان‌پذیری، و تولید یک برنامه کلان[۱۲] می‌باشد. تیم مسئول احداث پارک همچنین باید بداند که چگونه این فعالیت‌ها را انجام دهد. فعالیت‌های تحلیل و طراحی عبارتند از: «جمع‌آوری نیازمندی‌ها» از ذی‌نفعان[۱۳]، بررسی محل ایجاد پارک، طراحی نقشه‌ی پارک، طراحی نوع سرگرمی برای قسمت‌های مختلف پارک، ایجاد مدل‌هایی جهت مطالعه‌ی نقشه و فضاسازی پارک، و تولید طرح کلان. مجدداً، تیم مذکور باید بداند چگونه این فعالیت‌ها را انجام دهد تا بتواند طرح کلان را تولید کند. برخلاف فرایند که مراحل فعالیت‌ها را توصیف می‌کند، متدولوژی جزییات گام‌ها و رویه‌ها، یا چگونگی انجام فعالیت‌ها را نیز به‌تفصیل

---

[۱] Unified Modeling Language (UML)
[۲] Unified Process (UP)
[۳] Cookbook
[۴] Steps
[۵] Procedures
[۶] Agile Processes
[۷] Design Patterns
[۸] Test-Driven Development (TDD)
[۹] Reuse
[۱۰] Test Scripts
[۱۱] Planning
[۱۲] Master Plan
[۱۳] Stakeholders

بیان می‌کند.

ساخت یک شهربازی یک پروژه‌ی چند لایه است و میلیاردها دلار هزینه دربردارد. سرمایه‌گذار می‌خواهد پارک هرچه زودتر درآمد داشته باشد، ولی با فرایند گفته‌شده در بالا، سرمایه‌گذار مجبور است تا زمانی که کل پارک کامل شود، صبر کند. همچنین، به‌دلیل محدودیت‌هایی که توسط این فرایند نوعی تحمیل می‌شود، وقتی طرح کلان نهایی شد، به‌سادگی امکان تغییر در آن وجود ندارد. اگر هم پارک انتظارات ذی‌نفعان را برآورده نسازد، انجام تغییرات پس از کامل شدن آن بسیار هزینه‌بر خواهد بود.

فرایندهای چابک قصد دارند این مشکلات را حل کنند. با استفاده از یک فرایند چابک، فهرستی از نیازمندی‌های اولیه پارک خیلی سریع جمع‌آوری و اجازه داده می‌شود که این فهرست در حین فرایند توسعه تکمیل شود. پس از آن، فهرست وسایل تفریح و سرگرمی از روی نیازمندی‌ها استنتاج شده و به‌دقت در خوشه‌هایی از تسهیلات، گروه‌بندی می‌شود. آنگاه برنامه‌ای برای توسعه و تحویل خوشه‌ها در فواصل زمانی نسبتاً کوتاه تهیه می‌شود، بدین معنی که استقرار سریع از افزایش‌های کوچک انجام می‌شود. بنابراین، به‌جای یک طرح کلان نهایی شده، این فرایند خوشه‌ها را یکی‌یکی طراحی نموده و تحویل می‌دهد. به‌تدریج که خوشه‌ها نصب و عملیاتی شدند، بازخورد گرفته می‌شود و با همکاری ذی‌نفعان، تغییرات موردنظر در نیازمندی‌ها، برنامه‌ی توسعه، بودجه، و زمان‌بندی اعمال می‌شود؛ این معنای توسعه‌ی مشترک[۱] است. علاوه بر این، به‌کارگیری الگوهای طراحی معماری باعث بهبود کیفیت و نیز امکان تطبیق پارک با نیازمندی‌های تغییریابنده می‌گردد؛ این معنای طراحی برای تغییر است. تأکید بر روی کار گروهی به این دلیل است که همکاری و همکوشی[۲] مؤثر بین تیم‌ها و اعضای تیم تضمین می‌کند که تسهیلات در زمان مقرر و بدون ایراد ساخته و تحویل خواهند شد. فرایند چابک چند مزیت دارد. سرمایه‌گذار می‌تواند خیلی زودتر به سود برسد، زیرا تجهیزات در کوتاه‌ترین زمان ممکن و مطلوب عملیاتی می‌شوند. به‌علاوه، از آن‌جا که در هر زمان فقط تعداد کمی از تجهیزات ساخته و نصب می‌شوند، ایرادها را می‌توان تصحیح کرد و تغییرات را به سادگی اعمال نمود.

به‌طور خلاصه، این کتاب با محوریت یک متدولوژی چابک یکنواخت تهیه شده‌است که UML، الگوهای طراحی، آزمون نرم‌افزارهای شیءگرا، و توسعه‌ی آزمون‌رانده را نیز در بر می‌گیرد. متدولوژی ارائه‌شده در این کتاب، «متدولوژی یکنواخت» نامیده می‌شود زیرا از UML به‌عنوان زبان مدل‌سازی استفاده می‌کند و از یک فرایند چابک یکنواخت پیروی می‌نماید. این بدان مفهوم نیست که متدولوژی‌های دیگر را متحد می‌کند یا این‌که به‌عنوان یک متدولوژی «یکنواخت» برای تمام پروژه‌ها باید استفاده شود.

مخاطبان کتاب

این کتاب برای دانشجویان رشته‌های علوم کامپیوتر یا مهندسی نرم‌افزار، و نیز برای کسانی که حرفه‌شان توسعه‌ی نرم‌افزار است، نوشته شده است. به‌طور خاص، قصد بر آن بوده که کتاب به‌عنوان منبع اولیه برای دروس سال‌های آخر مقطع کارشناسی، دروس مقدماتی مقطع کارشناسی ارشد، و نیز دوره‌های آموزش تخصصی در صنعت فناوری اطلاعات[۳] مورد استفاده قرار گیرد. مطالب این کتاب در طی یک دهه‌ی اخیر از دروسی که در دانشگاه‌ها و شرکت‌های متعدد، اعم از محلی و بین‌المللی، تدریس شده‌اند و نیز به‌کارگیری این مطالب در پروژه‌های با سرمایه‌گذاری صنعتی و پروژه‌هایی که توسط مهندسان نرم‌افزار در شرکت‌های مختلف اجرا شده‌اند، تهیه و تکمیل شده است. پروژه‌های مذکور به مولف این امکان را داده است تا از نزدیک مشاهده کند که چگونه دانشجویان و مهندسان نرم‌افزار، فرایند یکنواخت، UML، الگوهای طراحی، و توسعه‌ی آزمون‌رانده را به‌کار می‌گیرند و با چه مشکلاتی مواجه می‌شوند. بازخوردی که ایشان می‌دادند موجب بهبود مستمر این کتاب گردیده است.

---

[۱] Joint Development

[۲] Collaboration

[۳] Information Technology (IT)

ساختار کتاب

این کتاب ۲۴ فصل دارد که به هشت قسمت تقسیم گردیده است:

- قسمت ۱) مقدمه و مهندسی سیستم: این قسمت شامل سه فصل اول است و مروری بر فعالیت‌های چرخه‌ی حیات نرم‌افزار ارائه می‌دهد. به‌خصوص، مدل‌های فرایند نرم‌افزار، مفهوم متدولوژی، تفاوت بین فرایند و متدولوژی، و مهندسی نرم‌افزار در این قسمت پوشش داده می‌شوند.
- قسمت ۲) تحلیل و طراحی معماری: این قسمت فعالیت‌های برنامه‌ریزی را ارائه می‌دهد که شامل جمع‌آوری نیازمندی‌ها، مدل‌سازی قلمرو، و طراحی معماری می‌باشد.
- قسمت ۳) مدل‌سازی و طراحی سیستم‌های تعاملی: این قسمت با مدل‌سازی و طراحی سیستم‌های تعاملی سروکار دارد و شامل شش فصل می‌شود. این فصول چگونگی شناسایی مورد کاربردها [۱] از نیازمندی‌ها، چگونگی مدل‌سازی و طراحی تعاملات کنشگر-سیستم و رفتار تعاملی اشیا، چگونگی اعمال الگوهای واگذاری مسئولیت، چگونگی استنتاج یک نمودار کلاس طراحی [۲] که بتواند به‌عنوان نقشه‌ی طراحی استفاده شود، و نهایتاً چگونگی طراحی واسط کاربر را معرفی می‌نمایند.
- قسمت ۴) مدل‌سازی و طراحی گونه‌های دیگری از سیستم‌ها: این قسمت شامل سه فصل می‌باشد، که هر کدام درباره‌ی مدل‌سازی و طراحی یک گونه از سیستم‌ها بحث می‌کنند. فصل ۱۳ مدل‌سازی و طراحی سیستم‌های رویداد- رانده، فصل ۱۴ مدل‌سازی و طراحی سیستم‌های تبدیلی، و فصل ۱۵ مدل‌سازی و طراحی سیستم‌های مبتنی بر قواعد کسب‌وکار را بیان می‌کند.
- قسمت ۵) اعمال الگوهای خاص موقعیت: این قسمت شامل دو فصل می‌شود و چگونگی اعمال الگوهای خاص موقعیت را ارائه می‌دهد. از یک مطالعه موردی، که طراحی یک ویراستار نمودار حالت است، استفاده‌شده تا فرایند درک شود.
- قسمت ۶) پیاده‌سازی و تضمین کیفیت: این قسمت شامل سه فصل می‌باشد، که ملاحظات پیاده‌سازی، فعالیت‌ها و مفاهیم تضمین کیفیت نرم‌افزار و آزمون نرم‌افزار را معرفی می‌کند.
- قسمت ۷) نگهداری و مدیریت پیکربندی: این قسمت متشکل از دو فصل است که نگهداری نرم‌افزار و مدیریت پیکربندی نرم‌افزار را پوشش می‌دهند.
- قسمت ۸) مدیریت پروژه و امنیت نرم‌افزار: آخرین قسمت کتاب شامل دو فصل می‌شود. یکی از فصل‌ها مدیریت پروژه نرم‌افزار را معرفی می‌کند، فصل دیگر امنیت نرم‌افزار، یعنی فعالیت‌های چرخه‌ی حیات که مربوط به مدل‌سازی و طراحی سیستم‌های نرم‌افزاری امن [۳] می‌باشند، را پوشش می‌دهد.

مطالب این کتاب می‌تواند ملزومات چند درس مهندسی نرم‌افزاری را پاسخ دهد. به‌عنوان مثال:

۱. قسمت ۱ تا ۳ و مباحث منتخبی از قسمت ۶ تا ۸ ترکیب مناسبی برای یک درس با عنوان «مهندسی نرم‌افزار شئ-گرا» یا یک درس با عنوان «مقدمه‌ای بر مهندسی نرم‌افزار» خواهند بود. این درس می‌تواند درس سال آخری برای مقطع کارشناسی یا یک درس مقدماتی برای مقطع کارشناسی ارشد باشد.
۲. قسمت ۲، قسمت ۵ و بخش‌های منتخبی از سایر فصول می‌تواند یک درس در مورد «الگوهای طراحی نرم‌افزاری» را تشکیل دهد. توصیه می‌شود که درس OOSE که در بالا به آن اشاره شد پیش‌نیاز این درس باشد. اما، خیلی

---

[۱] Use Cases
[۲] Design Class Diagram
[۳] Secure

از دانشجویان خارجی ممکن است درس OOSE را نگذرانده باشند؛ در این حالت، مروری بر متدولوژی ارائه‌شده در قسمت‌های ۲ و ۳ لازم است. مرور متدولوژی چارچوبی برای اعمال الگوها فراهم می‌کند. این مرور می‌تواند دو تا چهار هفته طول بکشد.

۳. قسمت ۶ و قسمت ۷ را می‌توان به روش‌های مختلف تدریس نمود. این قسمت‌ها می‌توانند یک درس با عنوان تضمین کیفیت، آزمون و نگهداری را تشکیل دهند، یا در قالب دو درس تضمین کیفیت نرم‌افزار و آزمون و نگهداری نرم‌افزار تدریس شوند. یا این‌که می‌توان این فصل‌ها را به‌عنوان سه درس تضمین کیفیت نرم‌افزار، آزمون نرم‌افزار، و نگهداری نرم‌افزار تدریس نمود.
۴. فصل‌های ۱۳ الی ۱۵، ۱۹ و ۲۰ به اضافه‌ی الگوهای منتخبی از فصول دیگر می‌توانند یک درس درباره‌ی مدل‌سازی، طراحی، وارسی و اعتبارسنجی سیستم‌های پیچیده تشکیل دهند.
۵. قسمت ۱، قسمت‌های ۶ تا ۸ و فصول منتخبی از دیگر قسمت‌ها می‌توانند یک درس با موضوع مدیریت پروژه‌های نرم‌افزاری را شکل بدهند.
۶. در نهایت، قسمت ۱، قسمت‌های ۲ و فصل ۲۴ به اضافه مباحث و الگوهای منتخبی از سایر فصول می‌توانند محتوای یک درس با عنوان «مقدمه‌ای بر امنیت نرم‌افزار» باشند. مدرس می‌تواند با استفاده از مطالب اضافی این درس را تکمیل نماید.

مطالب تکمیلی متعددی برای تدریس این کتاب در نشانی http://www.mhhe.com/kung وجود دارد. این مطالب شامل اسلایدهای پاورپوینت تدریس، نرم‌افزار تولید آزمونک و امتحان، بانک‌های اطلاعاتی از نمونه سؤالات امتحانی، طرح درس و سرفصل‌های نمونه، دستور کار آزمایشگاه و ابزارهای نرم‌افزاری می‌باشند. مدرسینی که تا کنون دروس پیش‌تر نام برده را تدریس ننموده‌اند خواهند دید که این ابزارهای آموزشی در کاهش زمان و تلاش لازم برای آماده‌سازی مطالب تدریس بسیار مفید هستند.

قدردانی

این‌جانب مایلم از تمامی دانشجویانم که به‌طور مداوم با سؤال، بازخورد، ابراز علاقه و تلاش برای اعمال متدولوژی در پروژه‌های واقعی محرک من بوده‌اند تشکر نمایم. دانشجویانم همچنین متن کتاب و مطالب آموزشی را خوانده و مورد استفاده قرار داده و بازخوردها و پیشنهادهای سازنده و باارزشی ارائه نموده‌اند. برخی از دانشجویانم در طراحی و پیاده‌سازی ابزارهایی که پشتیبان متدولوژی ارائه‌شده در این کتاب می‌باشند نیز مشارکت داشته‌اند. بعضی از ایشان در آزمایش‌هایی که مؤثر بودن متدولوژی را ارزیابی می‌کرده است درگیر بوده‌اند. بسیاری از آن‌ها پس از پایان تحصیلات خود استفاده از متدولوژی را در صنعت نیز ادامه داده و تجارب ارزشمندشان را با من به اشتراک گذاشته‌اند.

من مایلم از دخترم ژاکلین کونگ به خاطر ویرایش نگارش اولیه‌ی تعدادی از فصول سپاسگزاری نمایم. از «راگو سیرینی واسان» ناشر سراسری گروه مهندسی و علوم کامپیوتر انتشارات مک‌گراهیل، به خاطر پیشنهادهای سازنده و با ارزش وی و راهنمایی‌هایش در طی فرایند چاپ تشکر می‌نمایم. همچنین مایلم از داوران به خاطر توضیحات و توصیه‌هایشان قدردانی نمایم. این توصیه‌ها و توضیحات به طرز چشمگیری سازمان‌دهی، ارائه، و بسیاری از جنبه‌های دیگر کتاب را بهبود بخشید.

در حین فرایند طولانی نگارش، همسرم «سیندی کونگ» و بسیاری از همکارانم که در مؤسسات علمی و در صنعت با آن‌ها همکاری می‌کردم، همواره مشوق من بودند و حمایت‌شان را از من دریغ نمی‌کردند. همه آن‌ها نیز شایسته سپاسگزاری هستند.

اول

مقدمه و مهندسی سیستم

فصل اول مقدمه

فصل دوم فرایند و متدولوژی نرم‌افزار

فصل سوم مهندسی سیستم

۱ مقدمه

نکات کلیدی

- هدف مهندسی نرم‌افزار، بهبود بهره‌وری[۱] و کیفیت[۲] نرم‌افزار ، در عین کاهش هزینه[۳] و زمان تا تحویل محصول به بازار[۴] می‌باشد[۵].
- مهندسی نرم‌افزار متشکل از سه مسیر متعامل از فعالیت‌های چرخه حیات می‌باشد: توسعه‌ی نرم‌افزار، تضمین کیفیت نرم‌افزار، و مدیریت پروژه نرم‌افزار.
- مهندسی نرم‌افزار شیءگرا (OO) نوع خاصی از مهندسی نرم‌افزار است که جهان و سیستم‌ها را متشکل از اشیایی که با یکدیگر در تعامل هستند، می‌بیند.

کامپیوترها در تمام بخش‌های جامعه استفاده می‌شوند. امروزه، به سختی می‌توان بیمارستان، مدرسه، فروشگاه، بانک، کارخانه یا سازمانی را در کشور یافت که متکی بر کامپیوتر نباشد. حتی گوشی‌های تلفن همراه، خودروها، و تلویزیون‌ها نیز مبتنی بر سکوهای کامپیوتری هستند.

نیروی محرکه‌ای که در پس این استفاده‌ی روزافزون از کامپیوترها قرار گرفته، اقتصاد بازار است. اما این نرم‌افزار است که باعث می‌شود کامپیوترها آن‌گونه که ما می‌خواهیم، کار کنند. نرم‌افزارها یا برنامه‌های کامپیوتری متشکل از هزاران یا میلیون‌ها دستورالعمل هستند که کامپوتر را قادر می‌سازند محاسبات پیچیده را انجام دهد و عملیات دستگاه‌های سخت‌افزاری را کنترل نماید. در سال‌های اخیر، تقاضا برای نرم‌افزارهای کامپیوتری به‌شدت افزایش یافته است. در سال ۲۰۰۹ میلادی، تولید نرم‌افزار در دنیا به مرز ۹۸۵/۷۰ میلیارد دلار رسید، که رشد سالانه‌ای معادل ۶/۸۰ درصد را از ۱۹۹۹ تا ۲۰۰۹ میلادی نشان می‌دهد. اداره‌ی آمار مشاغل آمریکا می‌گوید که انتظار می‌رود از ۲۰۱۰ تا ۲۰۲۰ میلادی تعداد کل فرصت‌های شغلی برای مهندس نرم‌افزار برنامه‌ساز و تحلیل‌گر سیستم به‌ترتیب از ۵۲۰۸۰۰ به ۶۶۴۵۰۰ (۲۷/۶% رشد) و از ۵۴۴۴۰۰ به ۶۶۴۸۰۰ (۲۲/۱% رشد) برسد. برای این که فردی قادر باشد امور مربوط به شغل مهندس نرم‌افزار برنامه‌ساز یا تحلیل‌گر سیستم را انجام دهد، داشتن آموزش در مهندسی نرم‌افزار اکیداً توصیه می‌شود.

---

[۱] Productivity
[۲] Quality
[۳] Cost
[۴] Time to Market
[۵] مترجم: منظور بهبود بهره‌وری درتوسعه نرم‌افزار و بهبود کیفیت نرم‌افزار تولید شده است.

۱-۱ مهندسی نرم‌افزار چیست؟

سیستم‌های نرم‌افزاری محصولات فکری پیچیده‌ای هستند. توسعه‌ی نرم‌افزار باید تضمین کند که سیستم نرم‌افزاری نیازمندی‌های موردنظر را برآورده می‌سازد، هزینه‌ی توسعه از بودجه‌ی پیش‌بینی‌شده تجاوز نمی‌نماید، و سیستم طبق زمان‌بندی تحویل می‌گردد. برای رسیدن به این اهداف، اصطلاح مهندسی نرم‌افزار در اجلاسی از ناتو در سال ۱۹۶۸ پیشنهاد گردید تا نیاز به یک رویکرد مهندسی در تولید نرم‌افزار را تبلیغ نماید. از آن موقع به بعد، مهندسی نرم‌افزار به یک نظام مهندسی و/یا رشته‌ی تحصیلی[۱] تبدیل شده و پیشرفت‌های قابل‌ملاحظه‌ای نیز داشته است. تلاش‌هایی که در این رشته صورت گرفته منجر به تعریف زیر شده است.

تعریف ۱-۱

مهندسی نرم‌افزار به‌عنوان یک نظام مهندسی، بر روی پژوهش، آموزش، و اعمال روش‌ها و فرایندهای مهندسی تمرکز دارد تا به طرز قابل‌توجهی بهره‌وری و کیفیت نرم‌افزار را افزایش دهد، در حالی که هزینه و زمان تحویل را کاهش می‌دهد.

به‌عبارت دیگر، اهداف کلی مهندسی نرم‌افزار عبارتند از: افزایش قابل توجه در بهره‌وری و کیفیت نرم‌افزار، در عین کاهش هزینه‌های تولید و عملیاتی کردن و زمان تا تحویل محصول به بازار. این اهداف در عبارت PQCT (یادآوری: Productivity، Quality، Cost، Time to market) خلاصه می‌شوند. پژوهش، آموزش، و کاربست فرایندها و روش‌های[۲] مهندسی نرم‌افزار وسیله‌ای برای دستیابی به این اهداف است. این روش‌ها و فرایندها در سه مجموعه فعالیت طبقه‌بندی می‌شوند: توسعه، تضمین کیفیت، و مدیریت پروژه. فعالیت‌های توسعه، یک مفهوم اولیه از سیستم را به یک سیستم عملیاتی تبدیل می‌کنند. فعالیت‌های تضمین کیفیت اطمینان می‌دهند که فعالیت‌های توسعه به‌درستی انجام شده‌اند و نیز دست‌ساخته‌های[۳] تولیدشده در این فعالیت‌ها درست هستند. همه‌ی این‌ها تضمین می‌کنند که سیستم نرم‌افزاری مطلوب، تولید و تحویل گردیده است. فعالیت‌های مدیریت پروژه، برای پروژه برنامه‌ریزی می‌کنند، فعالیت‌های توسعه و تضمین کیفیت را زمان‌بندی کرده و منابع مورد نیاز را به آن‌ها تخصیص می‌دهند و اطمینان می‌دهند که سیستم به‌موقع و در محدوده‌ی بودجه‌ی موردنظر ساخته و تحویل خواهد شد.

۱-۲ چرا مهندسی نرم‌افزار؟

نخست این‌که، نرم‌افزار در تمام بخش‌های جامعه‌ی ما در حال گسترش است. شرکت‌ها برای اجرا و توسعه‌ی کسب‌وکار خود متکی به نرم‌افزار هستند. سیستم‌های نرم‌افزاری در حال بزرگ‌تر و پیچیده‌ترشدن هستند. امروزه خیلی عادی است که سیستمی تهیه شود که میلیون‌ها خط کد منبع داشته باشد. برای خیلی از سیستم‌های جاسازی‌شده[۴] هزینه‌ی نرم‌افزار به ۹۰ الی ۹۵ درصد کل هزینه‌ی سیستم افزایش پیدا کرده است؛ در حالی که در دو دهه‌ی پیش، این هزینه ۵ الی ۱۰ درصد بوده است. برخی از سیستم‌های جاسازی‌شده از مدارهای مجتمع خاص-کاربرد[۵] و سفت‌افزار[۶] استفاده می‌کنند. در این مدارهای مجتمع، نرم‌افزار در واقع درون یک سخت‌افزار حک شده است. جایگزینی این مدارها بسیار پرهزینه است؛ بنابراین، کیفیت نرم‌افزار مسئله‌ای بحرانی[۷] است. همه‌ی این موارد، نشان‌دهنده‌ی لزوم یک روش مهندسی نرم‌افزار برای توسعه‌ی سیستم می‌باشد.

---

[۱] Discipline
[۲] Methods
[۳] Artifacts
[۴] Embedded
[۵] Application-Specific Integrated Circuits (ASIC)
[۶] Firmware
[۷] Critical

دوم این‌که، مهندسی نرم‌افزار از کار گروهی، که برای توسعه‌ی سیستم‌های بزرگ لازم است، حمایت می‌کند. طراحی، پیاده‌سازی، و آزمون سیستم‌های نرم‌افزاری بزرگ نیازمند تلاش قابل‌ملاحظه‌ای می‌باشد. یک مهندس نرم‌افزار نوعی می‌تواند به‌طور میانگین ۵۰ الی ۱۰۰ خط کد منبع[۱] در روز تولید کند. این شامل زمان لازم برای انجام تحلیل، طراحی، پیاده‌سازی، یکپارچه‌سازی و آزمون می‌شود. بنابراین، یک سیستم کوچک که ۱۰۰۰۰ خط کد دارد، به یک مهندس نرم‌افزار نیاز دارد که بین ۱۰۰ تا ۲۰۰ روز، یا ۵ تا ۱۰ ماه کار کند. یک سیستم با اندازه‌ی متوسط که ۵۰۰۰۰۰ خط کد منبع دارد به یک مهندس نرم‌افزار نیاز دارد که ۵۰۰۰ الی۱۰۰۰۰ روز یا ۲۰ الی ۴۰ سال کار کند. برای بیشتر کسب‌وکارها پذیرفتنی نیست که تا این حد برای یک سیستم منتظر بمانند. بنابراین سیستم‌های نرم‌افزاری دنیای واقعی باید توسط یک یا چند تیم از مهندسان نرم‌افزار طراحی و پیاده‌سازی شوند. به‌عنوان مثال، یک سیستم نرم‌افزاری با اندازه‌ی متوسط به ۲۰ الی ۴۰ مهندس نرم‌افزار نیاز دارد که برای مدت یک سال کار کنند. وقتی چند مهندس نرم‌افزار با هم کار می‌کنند تا یک سیستم نرم‌افزاری را بسازند، مشکلات جدی در مفهوم‌سازی[۲]، ارتباط و هماهنگی پیش می‌آید.

مفهوم‌سازی عبارت است از فرایند مشاهده و طبقه‌بندی پدیده‌های دنیای واقعی، به‌منظور تشکیل یک مدل ذهنی که به فهم کاربردی که سیستم برای آن ساخته می‌شود کمک کند. مفهوم‌سازی، برای کار تیمی یک چالش است زیرا مهندسان نرم‌افزار به‌دلیل تفاوت‌های آموزشی، پیشینه‌ی فرهنگی، تجربه‌ی کاری، مفروضات و سایر موارد، ممکن است درک متفاوتی از دنیا داشته باشند. داستان قدیمی چهار مرد نابینا و فیل همین مشکل را نشان می‌دهد. چهار مرد نابینا می‌خواستند بدانند فیل شبیه چه چیزی است؛ ایشان توانستند فیل را لمس کنند. یکی از آن‌ها پای فیل را لمس کرد و گفت فیل باید شبیه به تنه درخت باشد. سه تای دیگر به ترتیب شکم، دم ، و گوش فیل را لمس کردند و گفتند فیل شبیه دیوار، طناب ، و بادبزن است. ما، به‌عنوان طراحان و توسعه‌دهندگان نرم‌افزار، نیز همانند آن چهار مرد نابینا هستیم که سعی می‌کنیم یک کاربرد را بفهمیم یا درک کنیم. اگر توسعه‌دهندگان درک متفاوتی از کاربرد داشته باشند، چگونه می‌توانند قطعات نرم‌افزاری را به‌گونه‌ای طراحی و پیاده‌سازی کنند که با یکدیگر کار کنند؟ مهندسی نرم‌افزار یک راه‌حل برای این مسئله ارائه می‌دهد. تکنیک‌های مدل‌سازی که در این کتاب ارائه می‌شوند، به مهندسان نرم‌افزار کمک می‌کنند فهم مشترکی از دامنه کاربرد و فرایندهای کسب‌وکاری کاربرد داشته باشند.

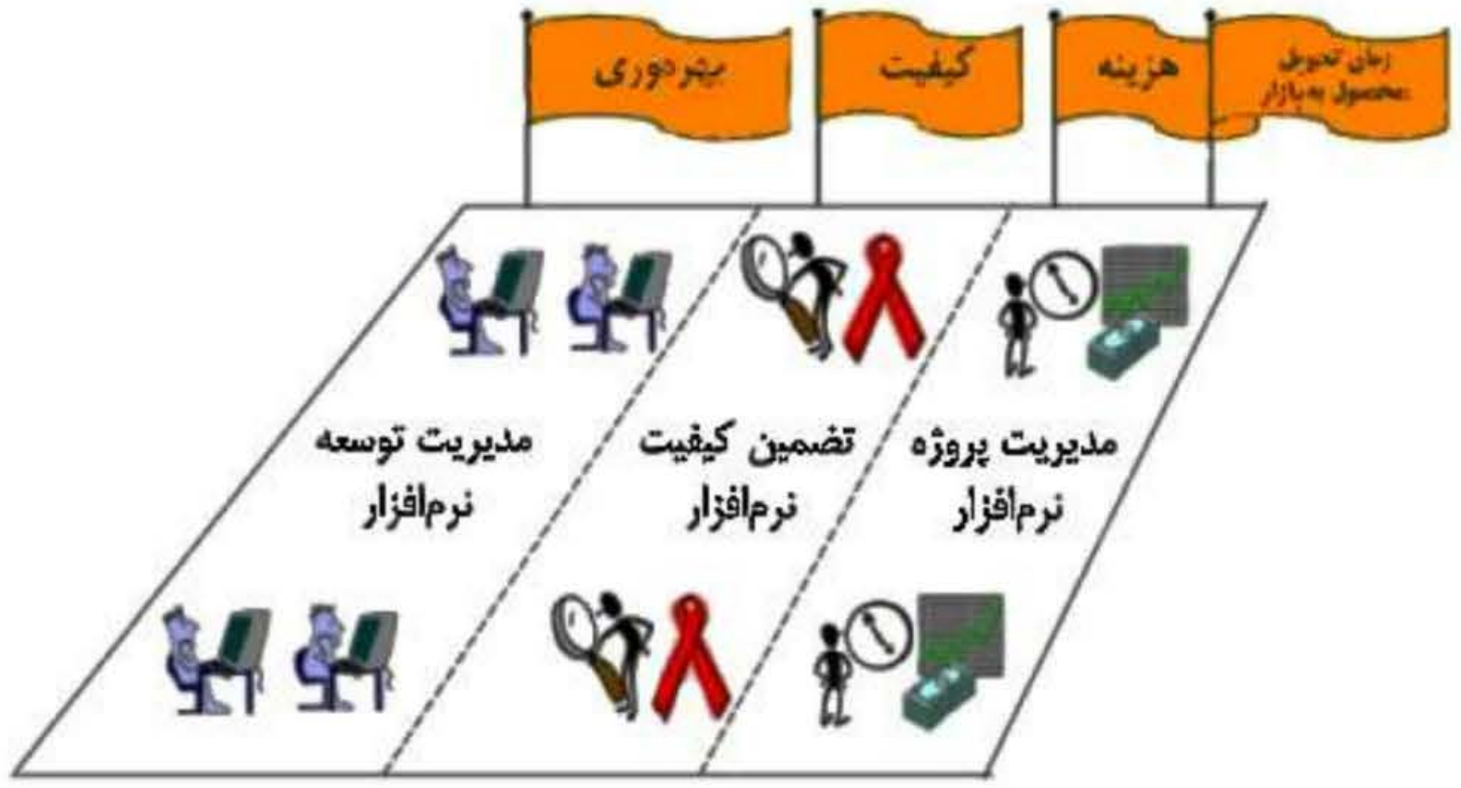
وقتی یک تیم از مهندسان نرم‌افزار با یکدیگر کار می‌کنند باید بتوانند درک خود و ایده‌های طراحی‌شان را با هم تبادل کنند. ولی، زبان طبیعی بسیار غیررسمی است و اغلب منجر به سوءتفاهم می‌شود. مهندسی نرم‌افزار زبان مدل‌سازی یکنواخت را در اختیار مهندسان نرم‌افزار گذاشته است تا ایده‌های خود را تبادل نمایند. آخرین نکته این است که وقتی تیم‌هایی از مهندسان نرم‌افزار با یکدیگر کار می‌کنند، چگونه می‌توانند همکوشی[۳] کرده و تلاش‌هایشان را هماهنگ نمایند؟ به‌عنوان مثال، چگونه کار را تقسیم می‌کنند و قسمت‌هایی از آن را به تیم‌ها یا اعضای تیم تخصیص می‌دهند؟ چگونه مولفه‌هایی[۴] که توسط تیم‌ها یا اعضای مختلف طراحی و پیاده‌سازی شده‌اند را یکپارچه می‌کنند؟ مهندسی نرم‌افزار برای این مشکل نیز یک راه‌حل ارائه می‌نماید. متدولوژی چابک یکنواخت، ارائه‌شده در این کتاب، برای مهندسان نرم‌افزار امکان همکوشی را به‌گونه‌ای فراهم می‌کند که همه آن‌را بفهمند و و از آن تبعیت کنند.

۱-۳ فعالیت‌های چرخه‌ی حیات نرم‌افزار

همان‌طور که در شکل ۱-۱ نشان داده شده است، مهندسی نرم‌افزار برروی سه شاخه از فعالیت‌ها تمرکز دارد. این فعالیت‌ها در حین چرخه‌ی حیات نرم‌افزار به‌طور هم‌زمان اتفاق می‌افتند.

---

[۱] Source Code
[۲] Conceptualization
[۳] Collaboration
[۴] Components



شکل ۱-۱: سه شاخه از فعالیت‌های چرخه‌ی حیات

۱. فرایند توسعه‌ی نرم‌افزار. فرایند توسعه‌ی نرم‌افزار، مفهوم اولیه‌ی سیستم را به سیستم عملیاتی تبدیل می‌کند که روی محیط مقصد اجرا می‌شود. این فرایند، نیازهای کسب وکار را شناسایی می‌کند، مطالعه امکان‌سنجی انجام می‌دهد، و نیازمندی‌ها یا قابلیت‌هایی که سیستم باید ارائه دهد را مدون می‌سازد. همچنین این فرایند، سیستم را طراحی، پیاده‌سازی، و آزمایش می‌کند و روی محیط مقصد مستقر می‌نماید.
۲. تضمین کیفیت نرم‌افزار. تضمین کیفیت نرم‌افزار[۱] اطمینان می‌دهد که فعالیت‌های توسعه‌ای به‌درستی انجام شده و دست‌ساخته‌های نرم‌افزاری که در اثر انجام فعالیت‌های توسعه‌ای تولید شده‌اند، نیازمندی‌های نرم‌افزاری و استانداردهای کیفی موردنظر را برآورده می‌سازند.
۳. مدیریت پروژه نرم‌افزاری. مدیریت پروژه نرم‌افزار بر فعالیت‌های توسعه و تضمین کیفیت نرم‌افزار نظارت و کنترل دارد. برخی فعالیت‌های مدیریت پروژه عبارتند از: تخمین تلاش، برنامه‌ریزی و زمان‌بندی پروژه، مدیریت ریسک ، و مدیریت عالی پروژه. این فعالیت‌ها، تضمین می‌کنند که سیستم نرم‌افزاری سرموقع و طبق بودجه تحویل می‌شود.

۱-۳-۱- فرایند توسعه‌ی نرم‌افزار

فرایند توسعه‌ی نرم‌افزار اغلب فرایند نرم‌افزار نیز نامیده می‌شود. نیاز به داشتن یک فرایند در ساختن یک خانه یا انجام هر کار مهم دیگر نیز حس می‌شود. فعالیت‌های اصلی که در ساختن یک خانه انجام می‌شوند عبارتند از: جمع‌آوری خواسته‌های مالک خانه، طراحی خانه، ساخت، بازرسی و تحویل خانه. به‌طور مشابه، فرایند نرم‌افزار نیز شامل تعدادی مرحله یا فعالیت می‌باشد که برای تولید سیستم نرم‌افزاری انجام می‌شوند. بعضی مواقع، سیستم نرم‌افزاری جزیی از یک سیستم بزرگ‌تر است. به‌عنوان مثال، سیستم‌های مخابراتی، سیستم‌های پردازش نامه، سیستم‌های کنترل فرایند صنعتی، و یا تجهیزات پزشکی از نرم‌افزار استفاده می‌کنند تا رویدادها[۲] را پردازش و سخت‌افزارها را کنترل نمایند. این سیستم‌ها را سیستم‌های جاسازی‌شده می‌نامند. در این حالت، فرایند نرم‌افزار جزیی از یک فرایند بزرگ‌تر است که فرایند توسعه‌ی سیستم، یا فرایند مهندسی سیستم نامیده می‌شود. مهندسی سیستم، به‌جای تنها سیستم نرم‌افزاری، کل سیستم را در نظر می‌گیرد.

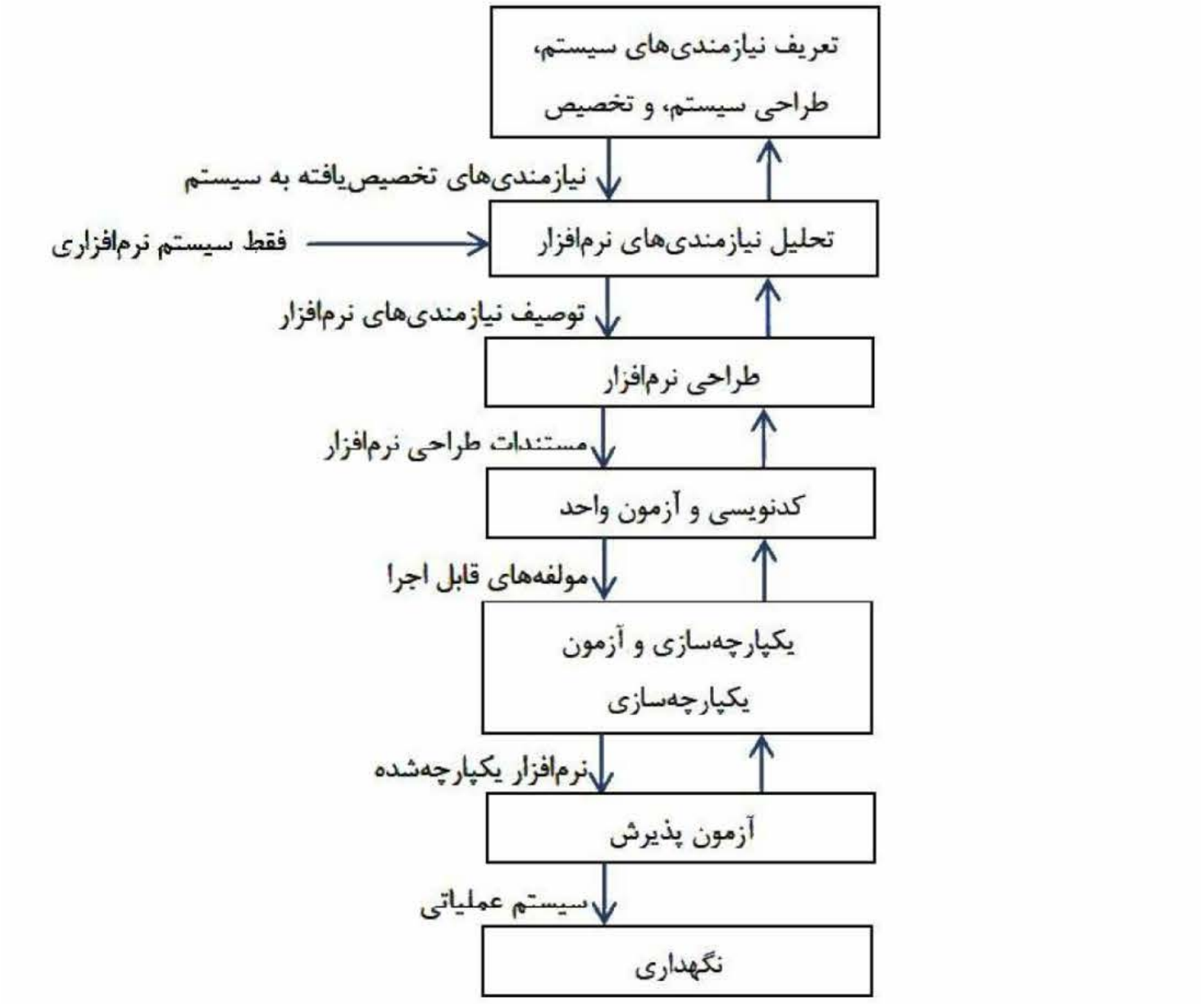
در طول تاریخچه‌ی مهندسی نرم‌افزار، مدل‌های فرایند نرم‌افزار متعددی معرفی شده‌اند که برخی از آن‌ها عبارتند از:

---

[۱] Software Quality Assurance (SQA)

[۲] Event

آبشاری[۱]، پیش‌نمونه‌سازی[۲]، تکاملی[۳]، حلزونی[۴]، یکنواخت[۵]، و چابک[۶]. شکل ۱-۲ فرایند سنتی آبشاری را نشان می‌دهد که از یک روش قدیمی مهندسی اقتباس شده است. این فرایند، فعالیت‌های مهندسی سیستم را در نظر گرفته است که عبارتند از: تعریف نیازمندی‌های سیستم، طراحی سیستم و تخصیص نیازمندی‌های سیستم. مراحل فرایند آبشاری در ادامه تشریح می‌شوند.



شکل ۱-۲: مدل آبشاری در توسعه نرم‌افزار

تعریف نیازمندی‌های سیستم، طراحی سیستم، و تخصیص

این فعالیت‌ها، همان فعالیت‌های مهندسی سیستم هستند که اغلب برای سیستم‌های جاسازی‌شده انجام می‌گیرند. هدف از تعریف نیازمندی‌های سیستم شناسایی قابلیت‌های کل سیستم و تدوین آن‌ها در قالب نیازمندی‌های سیستم می‌باشد. به‌عنوان مثال، توسعه‌ی یک سیستم ارتباطات رادیویی[۷] را در نظر بگیرید. این سیستم شبیه به یک شبکه تلفن همراه است؛ با این تفاوت که فقط یک ایستگاه اصلی توان بالا دارد که محدوده‌ای به مراتب وسیع‌تر از محدوده‌ی یک تلفن همراه در یک شبکه تلفن

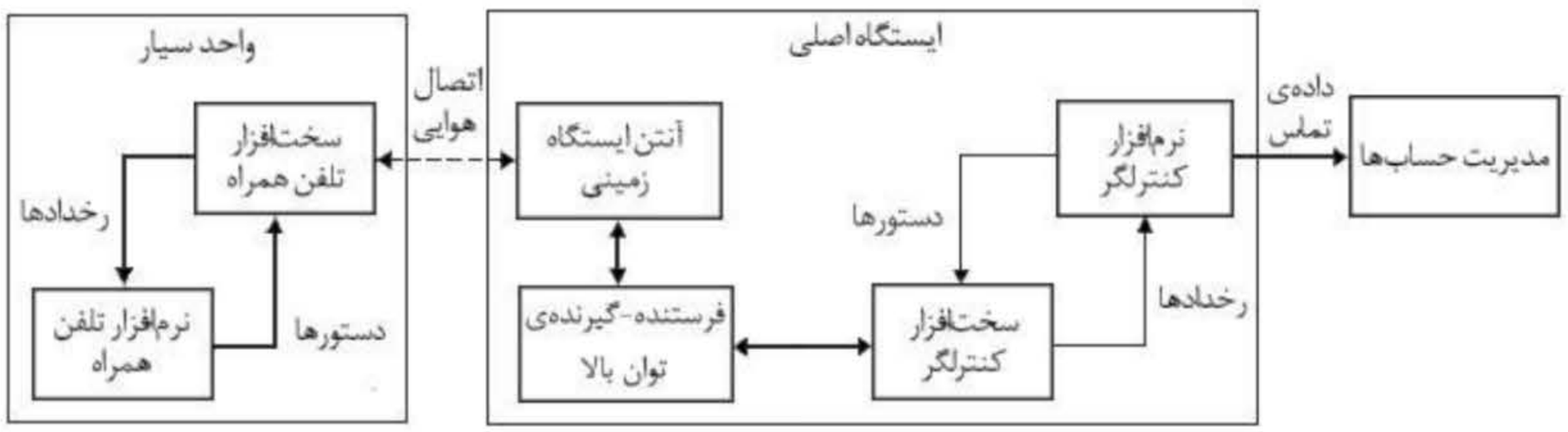
---

[۱] Waterfall
[۲] Prototyping
[۳] Evolutionary
[۴] Spiral
[۵] Unified
[۶] Agile
[۷] Radio Communication System (RCS)

همراه را سرویس می‌دهد. نیازمندی‌های سیستم، توانایی‌های کل سیستم را مشخص می‌کنند. در ادامه، چهار مورد از چندین نیازمندی شناسایی شده برای این سیستم آورده شده‌اند.

R1) RCS باید به مشترکین تلفن همراه اجازه دهد تا تماس‌هایی با دیگر مشترکین تلفن همراه یا تلفن ثابت برقرار نمایند.
R2) RCS باید به مشترکین تلفن همراه اجازه دهد که تماس‌های سایر مشترکین را پاسخ دهند.
R3) RCS باید امکان حسابداری تماس‌ها را فراهم کند تا تماس‌های تلفن همراه را ثبت نموده و بتواند صورت‌حساب مشترک را صادر نماید.
R4) RCS باید امکان مدیریت حساب‌های مشترکین را به مدیران مسئول حساب‌ها بدهد.

هدف از طراحی سیستم، تعیین زیرسیستم‌های اصلی سیستم کلی و مشخص کردن رابطه‌ی بین این زیرسیستم‌ها است. این موارد در نمودار طراحی معماری سیستم نشان داده می‌شود. شکل ۱-۳ با استفاده از نمودار بلوکی یک طراحی معماری را برای RCS نشان می‌دهد که شامل سه زیرسیستم است: واحد سیار، ایستگاه اصلی، و مدیریت حساب‌ها. واحد سیار شامل زیرسیستم سخت‌افزار تلفن همراه و زیرسیستم نرم‌افزار تلفن همراه است. ایستگاه زمینی شامل زیرسیستم‌های آنتن ایستگاه زمینی، فرستنده-گیرنده‌ی توان بالا، سخت‌افزار کنترلگر و نرم‌افزار کنترلگر می‌باشد. پیکان‌ها در شکل ۱-۳ رابطه‌ی بین زیرسیستم‌ها را نشان می‌دهند.



شکل ۱-۳: طراحی سیستم با استفاده از نمودار بلوکی

فعالیت تخصیص، نیازمندی‌های سیستم را به زیرسیستم‌ها واگذار می‌کند. نتیجه‌ی طراحی سیستم و تخصیص، باید زیرسیستم‌هایی باشند که نسبتاً مستقل بوده و به سادگی با هم تعامل داشته باشند. به‌علاوه زیرسیستم‌ها می‌توانند توسط تیم‌های مهندسی جداگانه، مانند تیم‌های مهندسی برق و الکترونیک، تیم‌های مهندسی مکانیک، و تیم‌های مهندسی نرم‌افزار تهیه شوند. تخصیص سیستم ممکن است نیازمندی‌های سیستم را به نیازمندی‌های سطح پایین‌تری بشکند، و آن‌ها را به زیرسیستم‌های مجزا واگذار نماید. به‌عنوان مثال، نیازمندی R1 واحد سیار و ایستگاه زمینی و نیز عملیات سخت‌افزاری و نرم‌افزاری را دخیل می‌کند. ارسال یک درخواست تماس به ایستگاه زمینی، وظیفه‌ی واحد سیار است. تفسیر این درخواست یک وظیفه‌ی سخت‌افزاری است. بررسی درخواست تماس و اطمینان از این‌که آن درخواست یک تماس مشترک-با-مشترک است یک وظیفه‌ی نرم‌افزاری است. بنابراین، این نیازمندی به موارد زیر تجزیه می‌شود:

R1.1) واحدهای سیار باید در موقع ارسال یک درخواست تماس شماره‌های شناسایی خودکار[1] را هم ضمیمه کنند.
R1.2) ایستگاه زمینی باید قبل از این که تماس را برقرار کند درخواست‌کننده و دریافت‌کننده‌ی تماس را با استفاده از شماره‌های ANI وارسی نماید.

[1] Automatic Number Identification (ANI)

R1.2.1) نرم‌افزار کنترلگر باید درخواست‌کننده و دریافت‌کننده‌ی تماس را وارسی نماید، و اگر نتیجه وارسی موفق بود، به سخت‌افزار کنترلگر دستور دهد که ارتباط را برقرار کند.

R1.2.2) سخت‌افزار کنترلگر باید به فرستنده-گیرنده‌ی توان بالا دستور دهد که تحت کنترل نرم‌افزار یک اتصال هوایی برقرار کند.

بعد از این که تجزیه فوق صورت گرفت، نیازمندی R1.1 به واحد سیار واگذار می‌شود. همچنین نیازمندی R1.2.1 به نرم‌افزار کنترلگر ایستگاه زمینی، و نیازمندی R1.2.2 به سخت‌افزار کنترلگر ایستگاه زمینی واگذار می‌شود. به‌همین‌ترتیب، نیازمندی R3 می‌تواند تجزیه شده و به زیرسیستم‌های مناسب منسوب شود. نیازمندی R4 به زیرسیستم مدیریت حساب‌ها واگذار می‌شود چون یک نیازمندی صرفاً نرم‌افزاری است.

تحلیل نیازمندی‌های نرم‌افزار

هدف از تحلیل نیازمندی‌های نرم‌افزار این است که نیازمندی‌های سیستم را که به سیستم نرم‌افزاری تخصیص داده شده‌اند پالایش[۱] کرده، و همچنین قابلیت‌های دیگری را برای سیستم نرم‌افزاری شناسایی کند. تمام این قابلیت‌ها و نیازمندی‌های پالایش‌شده، در قالب یک سند به نام بیانیه نیازمندی‌های نرم‌افزار[۲] تدوین می‌شوند. به‌عنوان مثال، نیازمندی R4 نمی‌گوید که یک مدیر حساب دارای مجوز، برای مدیریت حساب‌ها چه‌کار می‌تواند انجام دهد. لذا، این نیازمندی به‌صورت زیر پالایش می‌شود.

R4.1) RCS باید به مدیر حساب دارای مجوز، اجازه دهد تا برای مشترک حساب ایجاد کند.
R4.2) RCS باید به مدیر حساب دارای مجوز، اجازه‌ی فعال کردن حساب مشترک را بدهد.
R4.3) RCS باید به مدیر حساب دارای مجوز، اجازه‌ی غیرفعال کردن حساب مشترک را بدهد.
R4.4) RCS باید به مدیر حساب دارای مجوز، اجازه‌ی بستن حساب مشترک را بدهد.
R4.5) RCS باید به مدیر حساب دارای مجوز، اجازه‌ی حذف حساب مشترکی که قبلاً بسته شده را بدهد.

برای انجام اعمال فوق، مدیر حساب‌ها باید ابتدا عمل ورود به سیستم[۳] را انجام داده و پس از اتمام کارها، عمل خروج از سیستم[۴] را نیز انجام دهد تا مانع از دسترسی دیگران به حساب‌ها شود. این‌ها نمونه‌هایی از نیازمندی‌های نرم‌افزاری هستند که در حین مرحله‌ی تحلیل نیازمندی‌های نرم‌افزار شناسایی می‌شوند.

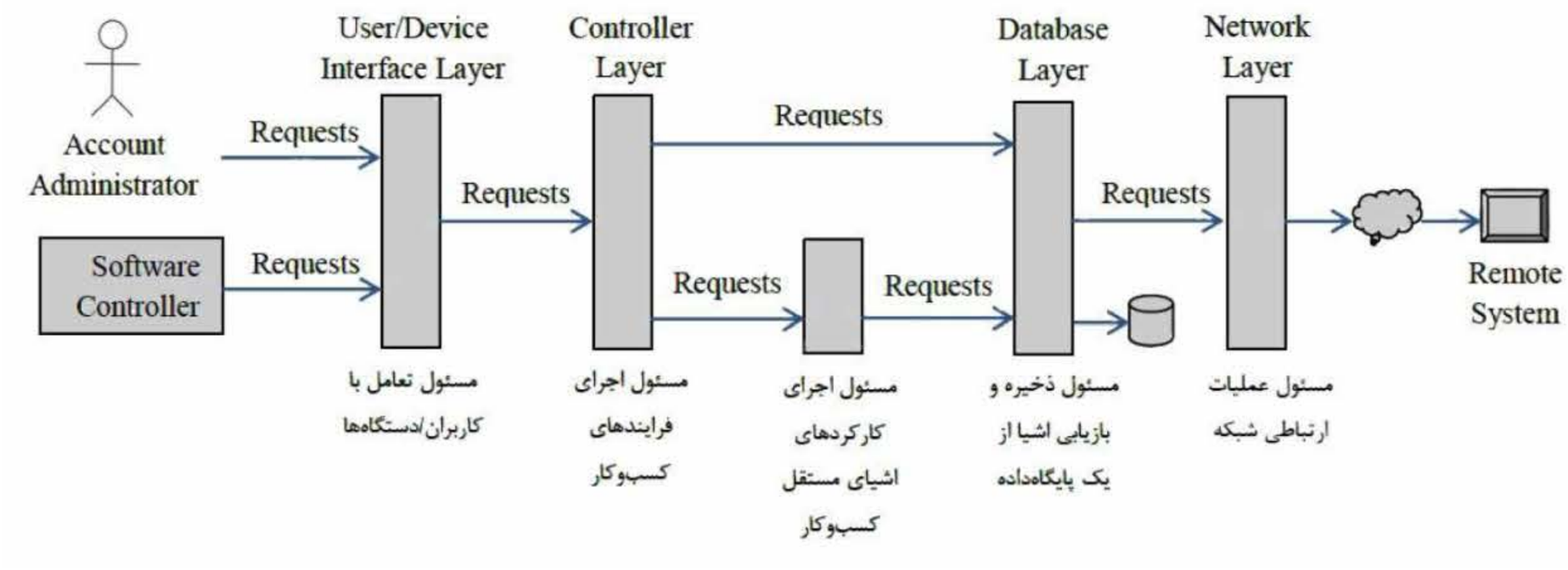
طراحی نرم‌افزار

طراحی نرم‌افزار، تعیین معماری نرم‌افزار یا ساختار کلی سیستم نرم‌افزاری را مشخص می‌کند. این معماری، زیرسیستم‌ها و رابطه‌ی بین آن‌ها، وظایف و رابط‌های این زیرسیستم‌ها، و چگونگی تعامل زیرسیستم‌ها با یکدیگر را مشخص می‌کند. یکی دیگر از فعالیت‌های مهم در طراحی نرم‌افزار، طراحی واسط کاربری می‌باشد. در این فعالیت، شکل ظاهری پنجره‌ها و جعبه‌های گفتگو[۵] طراحی شده و چگونگی تعامل سیستم با کاربران توصیف می‌شود. به‌علاوه، در طراحی نرم‌افزار الگوریتم‌های پردازش اطلاعات نیز مشخص می‌شوند.

---

[۱] Refine
[۲] Software Requirements Specific (SRS)
[۳] Login
[۴] Logout
[۵] Dialog Boxes

به‌عنوان مثالی از طراحی معماری برای RCS، شکل ۱-۴ یک طراحی nتایی (nلایه‌ای) برای سیستم مدیریت حساب‌ها را نشان می‌دهد. در این معماری، کلاس‌ها یا اشیای در لایه‌های جداگانه سازمان‌دهی شده‌اند. هر لایه مسئولیت واضح و مشخصی دارد و خدماتی[۱] از لایه‌ی بعدی درخواست می‌نماید. برای مثال، لایه‌ی واسط کاربر-دستگاه خدماتی را در اختیار مدیریت حساب‌ها و نیز نرم‌افزار کنترلگر قرار می‌دهد؛ و در عوض از خدمات لایه‌ی کنترلگر استفاده می‌کند. معماری لایه‌ای، پیاده‌سازی فرایندهای کسب‌وکاری را آسان می‌کند. به‌عنوان مثال، وقتی مدیریت حساب‌ها بخواهد یک حساب جدید ایجاد کند، واسط کاربری درخواست وی را به کنترلگر ایجاد حساب در لایه‌ی کنترلگر، ارجاع می‌دهد. سپس کنترلگر یک شئ حساب[۲] ایجاد کرده و از طریق لایه‌ی پایگاه‌داده آن را ذخیره می‌کند. این مثال نشان می‌دهد که پیاده‌سازی و استفاده از هر لایه نسبتاً آسان است. مزیت دیگری معماری nلایه‌ای این است که تغییر در یک لایه تاثیر کمی روی لایه‌های دیگر دارد، به این شرط که واسط‌های بین لایه‌ها بدون تغییر باقی بمانند.



شکل ۱-۴: یک معماری nتایی برای مدیریت حساب‌ها

پیاده‌سازی، آزمون، و نگهداری

در مرحله‌ی پیاده‌سازی و آزمون واحد[۳]، برنامه‌هایی نوشته می‌شود تا طراحی را پیاده‌سازی کند. این برنامه‌ها آزمایش می‌شوند و توسط نظیر[۴] مورد بازبینی قرار می‌گیرند تا از درستی آن‌ها و پیروی[۵] از استانداردهای کدنویسی[۶] اطمینان حاصل شود. در مرحله‌ی یکپارچه‌سازی،پیمانه‌های[۷] برنامه یکپارچه و سپس آزمایش می‌شوند، تا مطمئن شویم که با یکدیگر نیز به‌درستی کار می‌کنند. در مرحله‌ی آزمون پذیرش[۸]، موارد آزمونی[۹] طراحی و اجرا می‌شوند تا بررسی شود که نرم‌افزار درواقع نیازمندی‌های نرم‌افزاری را برآورده می‌سازد. پس از آن، سیستم نرم‌افزاری در محیط مقصد[۱۰] نصب شده و توسط کاربران آزمایش می‌شود. اینک نرم‌افزار به مرحله‌ی نگهداری وارد شده است. در مرحله‌ی نگهداری، به‌طور مداوم اصلاحات، بهبودها و ارتقاءها انجام می‌شوند تا زمانی که سیستم جای خود را به سیستم دیگری بدهد.

---

[۱] Services
[۲] Account
[۳] Unit Testing
[۴] Peer
[۵] Compliance
[۶] Coding Standards
[۷] Modules
[۸] Acceptance Testing
[۹] Test Cases
[۱۰] Target Environment

۱-۳-۲- تضمین کیفیت نرم‌افزار

SQA تضمین می‌کند که سیستم نرم‌افزاری که در دست ساخت است، نیازمندی‌های نرم‌افزاری و استانداردهای کیفی موردنظر را برآورده می‌سازد. وارسی[۱]، اعتبارسنجی[۲]، و آزمون همگی از راه‌های رسیدن به این اهداف می‌باشند. همان‌گونه که در شکل ۱-۵ مشاهده می‌شود، این فعالیت‌هادر طول چرخه‌ی حیات در جریانند. به‌طور مشخص، وارسی تضمین می‌کند که فعالیت‌های توسعه مطابق با روش‌ها و فرایندهای نرم‌افزاری انتخاب‌شده برای پروژه انجام شده‌اند. به‌علاوه، وارسی اطمینان می‌دهد که دست‌ساخته‌ها نرم‌افزاری تولید شده، با استانداردهای کیفی وفق[۳] دارند. به‌عنوان مثال، اگر پروژه‌ی RCS از روش آبشاری استفاده کند، وارسی اطمینان می‌دهد که مراحل آبشاری به‌درستی انجام شده‌اند. همچنین، وارسی بررسی می‌کند که تبیین نیازمندی‌های سیستم RCS، طراحی سیستم، تخصیص، تبیین نیازمندی‌های نرم‌افزار، طراحی نرم‌افزار، و سایر دست‌ساخته‌ها، تولید شده و استانداردهای کیفی لازم را برآورده می‌کنند. از سوی دیگر، اعتبارسنجی، درستی دست‌ساخته‌های نرم‌افزاری تولیدشده توسط فعالیت‌های توسعه را بررسی می‌کند. برای مثال، اعتبارسنجی در مورد RCS نیازمندی‌های سیستم و نرم‌افزار را بررسی می‌کند تا مطمئن شود که این نیازمندی‌ها، خواسته‌های واقعی کسب‌وکار را بیان می‌کنند. علاوه بر این، اعتبارسنجی بررسی می‌کند که آیا طراحی سیستم و طراحی نرم‌افزاری آن، نیازمندی‌ها و قیود[۴] را برآورده می‌سازند؟ و این‌که آیا سیستم نرم‌افزاری پیاده‌سازی‌شده، مشکلات کسب‌وکاری موردنظر را حل می‌کند؟

فعالیت‌های اعتبارسنجی به دو دسته‌ی اعتبارسنجی ایستا و پویا تقسیم می‌شوند. اعتبارسنجی ایستا، درستی دست‌ساخته‌های نرم‌افزاری را بررسی می‌کند، بدون این‌که نرم‌افزار را اجرا کند. در واقع، این نوع اعتبارسنجی برای دست‌ساخته‌هایی که غیرقابل‌اجرا هستند، مانند مستندات طراحی نرم‌افزار یا تبیین نیازمندی‌ها، قابل‌اعمال است. این کار مشابه وقتی است که مشخصات یک خودرو بررسی می‌شود تا معلوم شود آیا نیازهای یک خریدار خودرو را برآورده می‌سازد یا خیر. اعتبارسنجی پویا نرم‌افزار را اجرا می‌کند تا مطمئن شود که کار می‌کند و نتیجه‌ی درستی تولید می‌نماید. آزمون نرم‌افزار یک روش اعتبارسنجی پویا محسوب می‌شود که در آن، نرم‌افزار با استفاده از موارد آزمون اجرا می‌گردد و بررسی می‌شود که آیا نتایج آزمون با نتایج مورد انتظار تطبیق می‌کند؟ این کار مشابه وقتی است که با یک خودرو به‌طور آزمایشی رانندگی می‌شود تا اطمینان حاصل شود که نیازمندی‌های کارکردی و کارایی[۵] را برآورده می‌سازد.

در آزمون واحد، برای سیستم مدیریت حساب‌ها، موارد آزمونی تولید می‌شوند تا بررسی شود که هر کدام از کلاس‌ها در لایه‌های واسط، کنترلگر، اشیای کسب‌وکار، پایگاه‌داده و شبکه به درستی کارکرد[۶] آن کلاس را پیاده‌سازی نموده‌اند. آزمون یکپارچه‌سازی موارد آزمونی تولید می‌کند تا کلاس‌های مذکور را آزمایش نموده و مطمئن شود که با یکدیگر کار می‌کنند. آزمون پذیرش، موارد آزمونی را از نیازمندی‌های سیستم مدیریت حساب‌ها اجرا می‌کند تا تضمین شود که سیستم نیازمندی را برآورده می‌سازد.

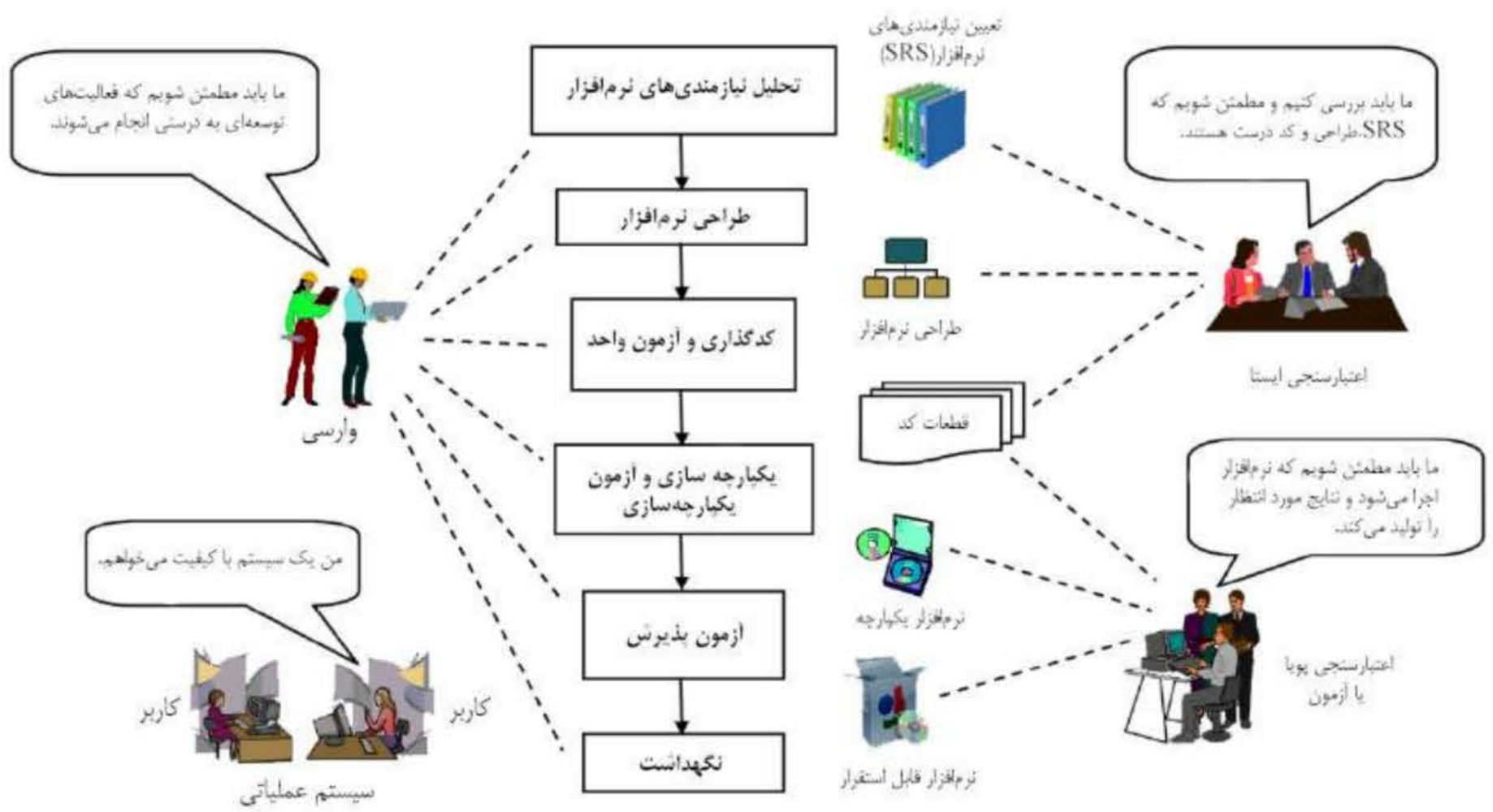
۱-۳-۳- مدیریت پروژه‌ی نرم‌افزاری

فعالیت‌های مربوط به مدیریت پروژه‌ی نرم‌افزاری[۷] تضمین می‌کنند که سیستم نرم‌افزاری در حال ساخت، به موقع و براساس محدودیت‌های بودجه‌ای تعیین‌شده، تحویل خواهد شد. برای دستیابی به این اهداف، فعالیت‌های مختلفی در مدیریت پروژه انجام می‌شود، که برخی از آن‌ها در ادامه معرفی شده‌اند.

- **تخمین تلاش:** تخمین تلاش، منابع انسانی و مدت زمان لازم برای انجام فعالیت‌های توسعه و SQA را مشخص

---

[۱] Verification
[۲] Validation
[۳] Conform
[۴] Constraints
[۵] Performance
[۶] Functionality
[۷] Software Project Management



شکل ۱-۵: SQA در چرخه‌ی حیات سنتی

می‌کند. برای مثال، چه تعداد برنامه‌ساز[۱] برای انجام هر یک از فعالیت‌های تحلیل، طراحی، پیاده‌سازی، و آزمون مورد نیاز است؟ مدت هر کدام از فعالیت‌های مذکور چقدر است؟ پاسخ این پرسش‌ها در برنامه‌ریزی پروژه و زمان‌بندی فعالیت‌ها مورد استفاده قرار می‌گیرند. فعالیت تخمین تلاش، این تخمین‌ها را از عوامل مختلفی چون اندازه و پیچیدگی تخمینی نرم‌افزار، به‌علاوه تاریخ تحویل مطلوب به‌دست می‌آورد.

- **برنامه‌ریزی و زمان‌بندی پروژه:** هدف از برنامه‌ریزی و زمان‌بندی پروژه، تولید یک برنامه‌ی کلی برای پروژه است. برنامه‌ی پروژه باعث هدایت تیم‌های پروژه در طی فرایند چرخه‌ی حیات می‌شود. اما به دلیل تغییرات در دنیای واقعی، این برنامه نیز باید اصلاح شود تا تغییرات در آن لحاظ گردند. برنامه‌ی پروژه، مواردی هم‌چون اهداف پروژه، فرایند رسیدن به این اهداف، نقاط عطف[۲] پروژه، زمان‌بندی فعالیت‌ها و تحویلی‌های پروژه[۳]، تیم‌های پروژه و چگونگی سازمان‌دهی این تیم‌ها، و در نهایت بودجه‌ی پروژه را مشخص می‌نماید. برنامه‌ی تضمین کیفیت نیز بخش دیگری از برنامه‌ی پروژه است. برنامه‌ی تضمین کیفیت، استانداردهای کیفی، فعالیت‌های SQA و زمان‌بندی این فعالیت‌ها را تعیین می‌کند. به‌علاوه، برنامه‌ی مدیریت ریسک نیز جزء دیگری از برنامه‌ی پروژه است که در ادامه توضیح داده می‌شود.
- **مدیریت ریسک:** رویدادهای مختلفی ممکن است یک پروژه را به مخاطره بیاندازند. برای مثال، یک مدیر یا فرد فنی کلیدی پروژه را ترک کند، یا پروژه از زمان‌بندی خیلی عقب بیافتد. به این موارد، اقلام ریسک[۴] می‌گویند. مدیریت ریسک سعی می‌کند تاثیر این رویدادها را از طریق برنامه‌ریزی مدیریت ریسک کاهش دهد. برنامه‌ی مدیریت ریسک، اقلام ریسک را شناسایی و آن‌ها را بر اساس احتمال وقوع و میزان خسارات وارده بر پروژه، اولویت‌بندی کرده و سپس

---

[۱] Developer
[۲] Milestone
[۳] Deliverables
[۴] Risk Items

راه‌چاره‌هایی برای حل‌وفصل[۱] ریسک تعیین می‌کند تا در صورت وقوع هرکدام از ریسک‌ها با آن مقابله کند.

- **سرپرستی پروژه**[۲] : سرپرستی پروژه یکی از وظایف جاری مدیریت پروژه است که فعالیت‌های مدیریتی را آن‌گونه که در برنامه‌ی پروژه تعیین شده انجام می‌دهد. دغدغه‌ی سرپرست پروژه، نظارت مداوم بر پیشرفت پروژه و اجرای کارهای لازم برای تطبیق پروژه با موقعیت جدید می‌باشد. از دیگر وظایف سرپرستی پروژه می‌توان به مدیریت روزانه‌ی فعالیت‌های پروژه، مانند هماهنگی تیم‌های پروژه یا اعضای تیم‌ها، زمان‌بندی و برگزاری جلسات، و حل‌وفصل مسائل روزمره اشاره نمود.
- **مدیریت پیکربندی نرم‌افزار**[۳] : در طول فرایند توسعه، دست‌ساخته‌های نرم‌افزاری متعددی تولید می‌شوند. از جمله‌ی این دست‌ساخته‌ها می‌توان به تبیین نیازمندی‌ها، طراحی نرم‌افزار، کُد، موارد آزمون، و راهنمای کاربر اشاره نمود. این دست‌ساخته‌ها، نرم‌افزار یا بخشی از آن را، در مراحل مختلف فرایند توسعه تشکیل می‌دهند. مثلاً تبیین نیازمندی‌ها همان نرم‌افزار است در شکل غیراجرایی‌اش؛ تبیین طراحی نیز پالایشی از تبیین نیازمندی‌ها می‌باشد؛ و کُد نیز پالایشی از طراحی است. این مستندات به یکدیگر وابسته‌اند. برای مثال، طراحی نرم‌افزار به نیازمندی‌ها وابسته است. اگر نیازمندی‌ها تغییر کنند، طراحی نیز به‌اجبار باید تغییر کند؛ و به همین صورت، این تغییر نیز به نوبه خود ممکن است تغییر در کُدی که طراحی را پیاده‌سازی کرده است لازم داشته باشد. بنابراین، مهندسی نرم‌افزار به مکانیزمی برای هماهنگی نیاز دارد؛ به گونه‌ای که تغییرات انجام‌شده با یکدیگر سازگار باشند. این چیزی است که مدیریت پیکربندی نرم‌افزار (SCM) نامیده می‌شود

۱-۴ مهندسی نرم‌افزار شیءگرا

مهندسی نرم‌افزار شیءگرا (OOSE) یک شکل خاص از مهندسی نرم‌افزار است. OOSE دنیا و سیستم‌ها را متشکل از اشیایی می‌بیند که به هم مرتبط بوده و با یکدیگر در تعامل هستند. برای پشتیبانی از توسعه‌ی نرم‌افزار شیءگرا، OOSE موارد زیر را فراهم کرده است:

۱. زبان‌های مدل‌سازی و طراحی OO را برای اعضای تیم فراهم کرده تا بتوانند ایده‌های تحلیل و طراحی خود را با هم تبادل کنند. زبان مدل‌سازی و طراحی، مفاهیم و نمادها و نیز قوانینی برای به‌کاربردن این نمادها تعریف می‌کند. زبان مدل‌سازی یکنواخت متداول‌ترین زبان مدل‌سازی و طراحی شیءگراست.
۲. فرایند توسعه‌ی نرم‌افزاری OO را برای هدایت تلاش توسعه فراهم کرده است. امروزه فرایند یکنواخت (UP) یک فرایند توسعه‌ی جاافتاده و معروف است؛ در حالی که فرایندهای چابک در سال‌های اخیر ظهور کرده‌اند.
۳. متدولوژی‌های توسعه‌ی نرم‌افزار OO را فراهم کرده است، که گام‌ها یا چگونگی انجام فعالیت‌های یک فرایند نرم‌افزاری را به‌تفصیل بیان می‌کنند.
۴. محیط‌ها و ابزارهای توسعه‌ی OO را فراهم کرده است که متدولوژی‌ها و فرایندهای توسعه را پشتیبانی می‌کنند. این محیط‌ها و ابزارها هم به‌صورت محصولات تجاری و هم به‌صورت نرم‌افزارهای رایگان عمومی[۴] موجود می‌باشند. به‌عنوان مثال، محیط توسعه‌ی یکپارچه[۵] با عنوان NetBeans یک نرم‌افزار متن‌باز رایگان است. این نرم‌افزار انبوهی از افزونه‌ها[۶] دارد که فعالیت‌های کل چرخه‌ی حیات توسعه‌ی نرم‌افزار را پشتیبانی می‌کنند.

---

[۱] Resolution
[۲] Project Administration
[۳] Software Configuration Management (SCM)
[۴] Public Domain Software
[۵] Integrated Development Environment (IDE)
[۶] Plugins

۱-۴-۱- زبان‌های مدل‌سازی و طراحی شئ‌گرا[۱]

گسترش سریع C++ در دهه‌ی ۱۹۸۰ میلادی، نیاز به یک متدولوژی توسعه برای هدایت فعالیت‌های توسعه‌ی نرم‌افزار به روش شئ‌گرا را آشکار ساخت. در این راستا، سه متدولوژی توسعه‌ی شئ‌گرا معرفی شدند که بیش از دیگر متدولوژی‌ها مطرح و به‌طور گسترده در صنعت نرم‌افزار استفاده شدند. این سه متدولوژی عبارتند از نمودار بوچ[۲]، روش مدل‌سازی شئ[۳]، و مهندسی مورد کاربرد[۴].

بلافاصله صنعت دریافت که یکپارچه‌سازی سیستم‌هایی که با استفاده از متدولوژی‌های مختلف طراحی و پیاده‌سازی شده‌اند یک چالش دست‌نیافتنی است، زیرا متدولوژی‌های مختلف، نمادها و مفاهیم مدل‌سازی متفاوتی را استفاده می‌کنند. برای حل این مشکل، مؤسسه‌ی «گروه مدیریت شئ»[۵] زبان مدل‌سازی یکنواخت UML را به‌عنوان یک استاندارد پذیرفت. UML خانواده‌ای از نمودارهاست که برای مدل‌سازی و طراحی جنبه‌های مختلف یک سیستم OO استفاده می‌شوند. نمودارهای UML در مرحله‌ی تحلیل نیازمندی‌ها استفاده می‌شود تا به تیم توسعه کمک کند که کسب‌وکار سیستم موجود را درک کنند. این نمودارها در مرحله‌ی طراحی به‌عنوان بخشی از مشخصات طراحی استفاده می‌شوند. در ادامه‌ی این کتاب، نمودارهای UML معرفی خواهند شد.

۱-۴-۲- فرایندهای توسعه‌ی شئ‌گرا

ماهیت ترتیبی فرایند آبشاری می‌گوید که تغییر نیازمندی‌ها سخت و پرهزینه است. این بدان علت است که هرگونه تغییر در نیازمندی‌ها بر روی طراحی و پیاده‌سازی تاثیر می‌گذارد، و آن‌ها نیز باید تغییر کنند. متأسفانه، در اکثر پروژه‌های دنیای واقعی تغییر در نیازمندی‌ها یک رویداد متداول است، و دلیل آن هم می‌تواند تغییر در شرایط بازار، پیشرفت فناوری، و یا سایر عوامل باشد. زمان طولانی توسعه در فرایند آبشاری باعث می‌شود که سیستم به محض تحویل، کهنه و تاریخ گذشته شود. زیرا قابلیت‌ها یا نیازمندی‌های سیستم مدت‌ها قبل شناسایی شده بوده‌اند. برای غلبه بر این مشکلات، چندین مدل فرایند نرم‌افزاری تاکنون پیشنهاد شده است. تمام این مدل‌ها به‌جای یک فرایند کاملاً ترتیبی، یک فرایند تکراری را برای فعالیت‌های توسعه برگزیده‌اند. نمونه‌هایی از این فرایندها عبارتند از: فرایند حلزونی، فرایند یکنواخت، و فرایندهای چابک.

۱-۴-۳- متدولوژی‌های توسعه‌ی شئ‌گرا

یک فرایند نرم‌افزاری مشخص می‌کند که «چه‌موقع چه‌کاری انجام شود» اما نمی‌گوید که «چگونه این کارها انجام شوند». یعنی، فرایند نرم‌افزار فعالیت‌های توسعه را تعریف می‌کند، ولی کاری به چگونگی انجام این فعالیت‌ها ندارد. UML یک زبان مدل‌سازی است که به مهندسان نرم‌افزار امکان می‌دهد ایده‌های تحلیل و طراحی خود را با استفاده از نمودار توصیف کنند، ولی UML در مورد تولید ایده‌های تحلیل وطراحی به مهندسان نرم‌افزار کمکی نمی‌کند. متدولوژی توسعه‌ی نرم‌افزار این شکاف را پر می‌کند؛ بدین‌صورت که گام‌ها و چگونگی اجرای گام‌ها برای انجام فعالیت‌های یک فرایند نرم‌افزاری را مشخص می‌کند. برخی از متدولوژی‌های متداول توسعه‌ی OO عبارت‌اند از: نمودار بوچ، روش مدل‌سازی شئ، و مهندسی مورد کاربرد. برخی متدولوژی‌های چابک عبارتند از: اسکرام[۶]، روش توسعه‌ی پویای سیستم[۷]، توسعه‌ی ویژگی‌رانده[۸]، کریستال شفاف[۹]،

---

[۱] Object-Oriented Modeling and Design Languages
[۲] Booch Diagram
[۳] Object Modeling Technique (OMT)
[۴] Use-Case Engineering
[۵] Object Management Group (OMG)
[۶] Scurm
[۷] Dynamic System Development Method (DSDM)
[۸] Feature-Driven Development (FDD)
[۹] Cristal Clear

برنامه‌نویسی افراطی[۱]، روش توسعه‌ی لاغر[۲].

۱-۴-۴- آیا روش‌های شیءگرا جایگزین روش‌های سنتی می‌شوند؟

پاسخ به این سؤال منفی است و دلایل آن نیز متعدد است. نخست، نگهداری سیستم‌های سنتی [۳] متعدد و موجود، ضروری است. دوم، هنوز سازمان‌های متعددی وجود دارند که از روش‌های سنتی استفاده می‌کنند. سوم، برای برخی پروژه‌ها مانند رایانش‌های علمی، ممکن است یک متدولوژی سنتی مناسب‌تر باشد. و آخر این‌که، یک سیستم ممکن است شامل اجزایی باشد که هم توسط روش‌های OO و هم توسط روش‌های سنتی تهیه شده باشد. بنابراین، روش‌های OO و سنتی تا سال‌های متمادی در کنار یکدیگر وجود خواهند داشت.

۱-۵ مهندسی نرم‌افزار و علم کامپیوتر

مهندسی نرم‌افزار و علم کامپیوتر با هم فرق دارند. رشته‌های مختلف علم کامپیوتر مانند داده‌ساختارها[۴] و الگوریتم‌ها، سیستم‌های پایگاه‌داده، هوش مصنوعی[۵]، و سیستم‌های عامل[۶]، بر روی بازدهی رایانشی، اشتراک منابع، دقت [۷]، بهینه‌سازی، و کارایی تأکید دارند. این ویژگی‌ها را می‌توان بلافاصله و به‌صورت کمّی اندازه‌گیری نمود. درواقع در چند دهه‌ی اخیر (از ۱۹۵۰ میلادی تاکنون) هدف از تمامی تلاش‌ها و منابع صرف‌شده در پژوهش‌های علم کامپیوتر، ارتقای جنبه‌های مذکور بوده است. بیش‌تر فصل‌های یک کتاب درسی علم کامپیوتر، درباره‌ی روش‌ها، الگوریتم‌ها، و تکنیک‌های بهبود این جنبه‌ها بوده است.

بر خلاف علم کامپیوتر، مهندسی نرم‌افزار بر روی PQCT تأکید دارد. برای مثال، هدف علم کامپیوتر اغلب به‌دست آوردن یک راه‌حل بهینه می‌باشد. ولی مهندسی نرم‌افزار ممکن است از یک راه‌حلِ به اندازه‌ی کافی خوب [۸] استفاده کند تا زمان و تلاش توسعه را کاهش دهد. تلاش‌ها و منابع صرف‌شده در واحدهای تحقیق و توسعه (R& D) مهندسی نرم‌افزار با این هدف است که PQCT نرم‌افزار را بهبود بخشند. بیش‌تر فصول یک کتاب درسی مهندسی نرم‌افزار درباره‌ی روش‌ها و تکنیک‌هایی است که این چهار جنبه را بهبود می‌بخشند. متأسفانه، تاثیر یک فرایند یا متدولوژی مهندسی نرم‌افزار را نمی‌توان بلافاصله اندازه‌گیری نمود. برای این‌که تاثیر معنی داشته باشد، باید در مدت زمانی طولانی مورد ارزیابی قرار گیرد. برای مثال، یک دهه طول کشید تا پژوهش‌گران دریابند که دستور پرش بدون‌شرط[۹] موجب ضرر است. بدین معنی که استفاده‌ی کنترل‌نشده از دستور go to منجر به تولید برنامه‌هایی با ساختار ضعیف می‌شود که فهمیدن و آزمودن آن برنامه‌ها بسیار دشوار است.

علم کامپیوتر فقط روی جنبه‌های فنی تأکید دارد. مهندسی نرم‌افزار باید موارد غیرفنی را در نظر گیرد. برای نمونه، مراحل اولیه‌ی فرایند توسعه بر روی شناسایی نیازهای کسب‌وکار و تدوین نیازمندی‌ها و قیود تمرکز دارد. این فعالیت‌ها نیاز به دانش، تجربه، و مهارت در خصوص ارتباط، رابطه با مشتری، و تحلیل کسب‌وکار دارد. همچنین، مهندسی نرم‌افزار نیاز به دانش و تجربه در زمینه‌ی مدیریت پروژه دارد. طراحی واسط کاربر باید عوامل انسانی از قبیل ترجیحات کاربر و چگونگی استفاده‌ی کاربران از سیستم را در نظر بگیرد. به‌علاوه، توسعه‌ی نرم‌افزار باید موارد سیاسی را هم درنظر بگیرد؛ زیرا سیستم ممکن است افراد زیادی را به‌راههای مختلف تحت‌تاثیر قرار دهد.

تشخیص تفاوت‌ها بین مهندسی نرم‌افزار و علم کامپیوتر می‌تواند به فهم و ارزش نهادن به فرایندها، متدولوژی‌ها و اصول[۱۰] مهندسی نرم‌افزار کمک نماید. به‌عنوان مثال طراحی یک سیستم نرم‌افزاری را درنظر بگیرید که باید به یک پایگاه‌داده دسترسی

---

[۱] Extreme Programing (XP)
[۲] Lean Development
[۳] Conventional
[۴] Data Structure
[۵] Artificial Intelligence
[۶] Operating Systems
[۷] Accuracy
[۸] Good-Enough
[۹] go to
[۱۰] Principles

پیدا کند. رشته‌های علم کامپیوتر ممکن است بر روی ذخیره و بازیابی کارآی داده‌ها تاکید کنند و طراحی‌ای را بپسندند که در آن اشیای کسب‌وکار مستقیما به پایگاه‌داده دسترسی پیدا می‌کنند. می‌گوییم چنین طراحی‌ای جفت‌شدگی محکمی[۱] بین اشیای کسب‌وکار و پایگاه‌داده ایجاد کرده است. مهندسی نرم‌افزار این طراحی را خوب نمی‌داند، مگر این‌که کارایی به‌عنوان یک دغدغه‌ی جدی درنظر گرفته شود. اگر پایگاه‌داده، یا سیستم مدیریت پایگاه‌داده[۲] تغییر کند، اشیای کسب‌وکار نیز باید عوض شوند. این کار ممکن است سخت و پرهزینه باشد. اگر پایگاه‌داده در محل دیگری باشد، در این‌صورت اشیای کسب‌وکار باید بدانند چگونه با پایگاه‌داده‌ی راه دور[۳] ارتباط برقرار کنند. این کار باعث ایجاد اشیای کسب‌وکار پیچیده‌ای می‌شود که آزمون و نگهداری آن‌ها دشوار خواهد بود.

باوجود تفاوت‌ها، مهندسی نرم‌افزار و علوم کامپیوتر کاملا به هم مرتبط هستند. نسبت علم کامپیوتر به مهندسی نرم‌افزار مانند فیزیک است به مهندسی برق و الکترونیک، یا شیمی به مهندسی شیمی. یعنی، علم کامپیوتر بنیان نظری و فناورانه‌ی مهندسی نرم‌افزار است. مهندسی نرم‌افزار نیز کاربرد علم کامپیوتر است، ولی مهندسی نرم‌افزار مباحث پژوهشی خاص خود را دارد که شامل پژوهش در زمینه‌هایی چون متدولوژی‌ها و فرایندهای نرم‌افزار، وارسی و اعتبارسنجی نرم‌افزار، و روش‌های آزمون می‌شود. هدف از پژوهش در مهندسی نرم‌افزار بهبود قابل توجهی در PQCT نرم‌افزار و دادوستد[۴] بین این عوامل چهارگانه است.

مهندسی نرم‌افزار محدوده‌ی وسیعی دارد. یک مهندس نرم‌افزار باید زمینه‌های وسیعی از علم کامپیوتر مانند سیستم‌های پایگاه‌داده، سیستم‌های عامل، داده‌ساختارها و الگوریتم‌ها، زبان‌های برنامه‌سازی، و ساخت مترجم[۵] را بداند. برای توسعه‌ی سیستم‌های جاسازی‌شده یک مهندس نرم‌افزار باید دانش اولیه‌ای از مدارهای الکترونیکی و چگونگی اتصال با دستگاه‌های سخت‌افزاری داشته باشد. نهایتاً، برای آن‌که یک مهندس نرم‌افزار دانش دامنه و تجارب طراحی را کسب نماید و یک معمار نرم‌افزار خوب بشود، زمان زیادی لازم است. این چالش‌ها و توانایی طراحی و پیاده‌سازی سیستم‌های پیچیده‌ی بزرگ که نیازهای واقعی را برآورده نمایند، باعث شده که مهندسی نرم‌افزار یک زمینه‌ی جذاب بشود. کاربردهای روزافزون کامپیوتر فرصت‌های گران‌بهایی را برای مهندسان نرم‌افزار و نیز پژوهش‌گران مهندسی نرم‌افزار ایجاد می‌نماید.

خلاصه

مهندسی نرم‌افزار به‌عنوان رشته‌ای تعریف می‌شود که فرایندها و متدولوژی‌های مهندسی را برای بهبود PQCT نرم‌افزار بررسی و به‌کار می‌بندد. در این فصل، در خصوص نیاز به مهندسی‌نرم‌افزار بحث شد و فعالیت‌های چرخه‌ی حیات نرم‌افزار تشریح گردیدند. مهندسی نرم‌افزار OO نوع خاصی از مهندسی نرم‌افزار است که دنیا و سیستم‌ها را متشکل از اشیایی می‌بیند که به هم مرتبط بوده و با هم در تعامل هستند. این فصل با بحثی در مورد تفاوت‌ها و رابطه‌ی بین علم کامپیوتر و مهندسی نرم‌افزار به پایان رسید. علم کامپیوتر بنیان مهندسی نرم‌افزار است. در حالی که علم کامپیوتر عمدتاً نگران بهینه‌سازی و کارایی است، مهندسی نرم‌افزار با PQCT نرم‌افزار سروکار دارد. دانستن این موارد به برنامه‌نویس کمک می‌کند تا مهندسی نرم‌افزار را بفهمد و دلیل تعریف فرایندها، متدولوژی‌ها، زبان‌های مدل‌سازی، الگوهای طراحی، و سایر موارد را درک کند. تمام این موارد به منظور بهبود PQCT نرم‌افزار طراحی شده‌اند.

---

[۱] Tight Coupling
[۲] Database Management System (DBMS)
[۳] Remote
[۴] Trade-off
[۵] Compiler

برای مطالعه‌ی بیش‌تر

کتاب‌های زیادی هستند که رشته‌ی مهندسی نرم‌افزار را به‌طور جامع پوشش می‌دهند. نمونه‌هایی از این کتاب‌ها عبارتند از [۱۲۳، ۱۲۵، ۱۳۹]. در مورد OOSE نیز کتاب‌های متعددی وجود دارند، از جمله [۴۰، ۱۳۳]. یک مقدمه‌ی عالی برروی تحلیل و طراحی OO در [۱۰۴] ارائه شده است. کتاب‌های متعددی نیز منتشر شده‌اند که UML و UP را شرح می‌دهند، به‌ویژه [۳۶، ۶۳، ۹۱]. وب‌گاه گروه مدیریت شئ (http://www.omg.org) مشخصات UML و مطالب مرتبط را منتشر می‌کند. در خصوص توسعه‌ی چابک نیز کتاب‌های زیادی وجود دارند، مانند [۲۱، ۵۰، ۵۲، ۱۱۹، ۱۲۴، ۱۴۰]. وب‌گاه متحدان چابک [۱] (http://www.agilealliance.org) یک منبع عالی است که مقالات و بحث‌های خوبی را روی موضوع چابکی منتشر می‌کند.

سوال‌های مروری فصل

۱. مهندسی نرم‌افزار چیست؟ و چرا مورد نیاز است؟
۲. فرایند توسعه‌ی نرم‌افزار چیست؟
۳. تضمین کیفیت نرم‌افزار چیست؟
۴. مدیریت پروژه‌ی نرم‌افزار چیست؟
۵. مدیریت پیکربندی نرم‌افزار چیست؟
۶. تفاوت‌های میان مهندسی نرم‌افزار شئ‌گرا با مهندسی نرم‌افزار سنتی در چیست؟
۷. چه تفاوت‌ها و رابطه‌هایی بین مهندسی نرم‌افزار و علم کامپیوتر وجود دارد؟ آیا می‌شود یکی را بدون دیگری داشت؟

تمرین‌ها

۱-۱ با جستجو در ادبیات [۲] چهار تعریف دیگر از مهندسی نرم‌افزار علاوه بر آن‌هایی که در این فصل ارائه شد پیدا کنید. در مورد شباهت‌ها و تفاوت‌های بین این تعاریف بحث کنید.

۱-۲ در یک متن کوتاه وظایف فرایند توسعه‌ی نرم‌افزار، تضمین کیفیت نرم‌افزار، مدیریت پروژه نرم‌افزار و مدیریت پیکربندی نرم‌افزار را تشریح کنید. در مورد این‌که در حین چرخه‌ی حیات توسعه‌ی نرم‌افزار چگونه این‌ها با هم کار می‌کنند بحث کنید. در مورد این‌که چگونه این‌ها PQCT نرم‌افزار را بهبود می‌بخشند، بحث کنید.

۱-۳ آیا بهینه‌سازی می‌تواند نقطه‌ی تمرکز مهندسی نرم‌افزار باشد؟ به اختصار توضیح دهید و جواب خودتان را با مثالی عملی توجیه کنید.

۱-۴ به انتخاب خودتان سه درس در زمینه‌ی علوم کامپیوتر شناسایی کنید. فایده‌ی این درس‌ها را در فعالیت‌های چرخه‌ی حیات نرم‌افزار نشان دهید.

۱-۵ وابستگی‌هایی درونی [۳] بین بهره‌وری، کیفیت و زمان تا تحویل محصول به بازار در نرم‌افزار وجود دارد. برای مثال، هر چه زمان و تلاش صرف شده برای کدگذاری بیشتر باشد باعث بهره‌وری بیشتر می‌شود. ولی در عوض این امر ممکن است باعث صرف زمان و تلاش کمتر برای تضمین کیفیت نرم‌افزار شود. چون کل زمان و تلاش

---

[۱] Agile Alliance
[۲] Literature
[۳] Interdependencies

ثابت است؛ و دست آخر کیفیت پایین ممکن است بهره‌وری را کاهش دهد، چون نیاز به دوباره‌کاری خواهیم داشت.

به انتخاب خودتان، سه جفت از این وابستگی‌های درونی را شناسایی کنید و در مورد اثرات کوتاه‌مدت و بلندمدت آن‌ها بر روی سازمان توسعه‌ی نرم‌افزار بحث کنید. مهندس نرم‌افزار چگونه باید «معماهای» [۱] منتج از این وابستگی‌های درونی را حل کنید.

۱-۶ تفاوت‌ها و رابطه‌های بین مهندسی نرم‌افزار OO با مهندسی نرم‌افزار سنتی چیست؟ در مورد این که آیا مهندسی نرم‌افزار OO می‌تواند جایگزین مهندسی نرم‌افزار سنتی شود بحث کنید.

---

[۱] Dilemma

۲

فرایند و متدولوژی نرم‌افزار

نکات کلیدی

- فرایند نرم‌افزار، مراحل فعالیت‌ها یا کارهایی که باید برای ساختن یک سیستم نرم‌افزاری انجام شوند را تعریف می‌کند.
- متدولوژی نرم‌افزار، گام‌ها یا چگونگی انجام فعالیت‌های فرایند نرم‌افزار را تعریف می‌کند. متدولوژی پیاده‌سازی یک فرایند است.
- توسعه‌ی نرم‌افزار نیازمند یک فرایند و یک متدولوژی است.

نوشتن برنامه‌هایی که چند هزار خط کد داشته باشند، شاید برای اکثر دانشجویان کارشناسی و کارشناسی ارشد یک تجربه‌ی جدید و یک چالش به حساب آید. ولی در صنعت نرم‌افزار، توسعه‌ی نرم‌افزار برای سیستم‌هایی که میلیون‌ها خط کد دارند، امری متداول است. تفاوت بین پروژه‌های برنامه‌نویسی دانشگاهی و پروژه‌های توسعه‌ی نرم‌افزاری در دنیای واقعی تنها محدود به تعداد خط کد نمی‌شود. توسعه‌ی نرم‌افزار در دنیای واقعی باید بر چالش‌های متعدد دیگری نیز غلبه نماید. برای غلبه بر این چالش‌ها، یک روش منضبط برای توسعه‌ی سیستم مورد نیاز است که معمولاً با عنوان فرایند نرم‌افزار[۱] از آن یاد می‌شود. فرایند آبشاری که در فصل ۱ معرفی شد یک چنین فرایند یا مدل فرایند[۲] است. بخشی از دلایل پذیرش فرایند آبشاری توسط بسیاری از سازمان‌های توسعه‌ی نرم‌افزار، سادگی آن است. ولی تجربه نشان می‌دهد که فرایند آبشاری با مشکلاتی همراه است. برای غلبه بر این مشکلات، چندین فرایند نرم‌افزاری دیگر پیشنهاد شده‌اند. پس از آن، صنعت نرم‌افزار دریافت که توسعه‌ی نرم‌افزار نه تنها به فرایند، بلکه به متدولوژی نیز نیاز دارد. درحالی‌که فرایند، فعالیت‌هایی که باید برای توسعه‌ی یک سیستم نرم‌افزاری اجرا شوند را تعریف می‌کند، متدولوژی، گام‌ها و چگونگی انجام این گام‌ها، که باید برای اجرای فعالیت‌های یک فرایند طی شوند را نشان می‌دهد. به عبارت دیگر، متدولوژی پیاده‌سازی یک فرایند است.

به‌طور خلاصه، این فصل مطالب زیر را ارائه می‌دهد:

- چالش‌های توسعه‌ی سیستم در دنیای واقعی
- مزایا[۳] و محدودیت‌های[۴] فرایند آبشاری

---

[۱] Software Process
[۲] Process Model
[۳] Merits
[۴] Limitations

- سایر مدل‌های شناخته شده‌ی فرایند نرم‌افزار
- متدولوژی نرم‌افزار و تفاوت‌های میان فرایند و متدولوژی
- نظریه مسائل بدرفتار[۱] و چگونگی ارتباط این نظریه با توسعه‌ی نرم‌افزار
- فرایندهای چالاک و روش‌های چابک
- مروری بر فرایند و متدولوژی ارائه شده در این کتاب.

۲-۱ چالش‌های توسعه‌ی سیستم

توسعه‌ی سیستم‌های نرم‌افزاری در آزمایشگاه با توسعه‌ی این سیستم‌ها در دنیای واقعی کاملاً متفاوت است. پروژه‌های دنیای واقعی بسیار بزرگ‌تر و پیچیده‌ترند. به جز این چالش‌های آشکار، تفاوت‌های دیگر نیز وجود دارند. فهمیدن برخی از این چالش‌ها برای مطالعه‌ی مهندسی نرم‌افزار مفید است. بدین معنی که، به فهم این‌که چرا علم کامپیوتر به تنهایی کافی نیست و چرا ما نیاز به فرایندها و متدولوژی‌های نرم‌افزار داریم، کمک می‌نماید. به‌علاوه، کمک می‌کند بدانیم چگونه فرایندها و متدولوژی‌های نرم‌افزار در غلبه بر این چالش‌ها به یاری ما می‌آیند. ابتدا، چالش‌های پروژه را برمی‌شماریم.

- **واقعیت ۱ در مورد پروژه.** بسیاری از پروژه‌های توسعه‌ی سیستم دوران توسعه‌ای طولانی از یک‌سال تا چندسال، دارند. در پروژه‌های دنیای واقعی باید محدودیت‌های زمان‌بندی و بودجه‌ای رعایت شوند.
- **چالش ۱ در مورد پروژه.** بدون دانش کاملی از آن‌چه مشتری می‌خواهد، و آن‌چه در چند سال آتی رخ خواهد داد، چگونه می‌توانیم کار پروژه را برنامه‌ریزی، زمان‌بندی، و مدیریت کنیم؟
- **واقعیت ۲ در مورد پروژه.** اکثر پروژه‌های توسعه‌ی سیستم نیاز به همکاری چندین تیم توسعه یا چندین بخش دارند. به‌عنوان مثال، بسیاری از سیستم‌های بزرگ جاسازی‌شده متشکل از اجزاء سخت‌افزاری و نرم‌افزاری می‌شوند و نیاز به همکاری بخش مهندسی نرم‌افزار با بخش‌های سخت‌افزاری دارند.
- **چالش ۲ در مورد پروژه.** چگونه می‌توان کار را تقسیم کرد و قسمت‌های به‌هم وابسته را به تیم‌ها و بخش‌ها واگذار نمود، و سپس اجزاء تولید شده توسط تیم‌ها و بخش‌های مختلف را به‌طور مناسب با هم یکپارچه نمود؟
- **واقعیت ۳ در مورد پروژه.** بخش‌ها و تیم‌های مختلف ممکن است فرایندها، روش‌ها و ابزارهای مختلفی را برای توسعه به‌کار ببرند. بخش‌ها و تیم‌های مختلف ممکن است در محل‌های مختلفی مستقر باشند.
- **چالش ۳درمورد پروژه.** چگونه مطمئن شویم که بین بخش‌ها و تیم‌ها ارتباط و هماهنگی مناسبی برقرار است

علاوه بر چالش‌های پروژه، چالش‌هایی نیز در مورد محصول وجود دارند. برخی از این چالش‌ها عبارتند از:

- **واقعیت ۱ در مورد سیستم.** اکثر سیستم‌های دنیای واقعی باید تعداد زیادی از نیازها و محدودیت‌ها را برآورده سازند که از آن جمله می‌توان به محدودیت‌های سخت سیستم‌های بلادرنگ اشاره نمود. به‌عنوان مثال، یک سیستم خدمات پستی، صدها نیازمندی و محدودیت دارد. در چنین سیستمی باید بیش از ۴۰۰۰۰ بسته پستی را در هر ساعت پویش[۲] و پردازش نمود که می‌شود ۱۲ بسته پستی در ثانیه.
- **چالش ۱ در مورد سیستم.** چگونه سیستم را توسعه دهیم که مطمئن شویم این نیازمندی‌ها و محدودیت‌ها برآورده شده‌اند؟

---

[۱] Wicked Problems

[۲] Scan

- **واقعیت ۲ در مورد سیستم.** نیازمندی‌ها و محدودیت‌ها ممکن است از یک زمان تا زمان دیگر تغییر کنند. به‌عنوان مثال، پروژه‌ی کوچکی را در نظر بگیرید که مولف کتاب قرارداد اجرای آن را بسته بود. در سه ماه اول، نیازمندی‌ها هر هفته تغییر داده می‌شد، زیرا درخواست‌های مشتری تغییر می‌کرد.
- **چالش ۲ در مورد سیستم.** چگونه فرایندها و محصولات را طراحی کنیم تا بر تغییراتی که اغلب ناگزیرند و تغییراتی که پیش‌بینی دقیق آن‌ها غیر ممکن است، غلبه کنیم؟
- **واقعیت ۳ در مورد سیستم.** سیستم در طول سال‌های سال تکامل پیدا خواهد کرد و چالشی برای نگهداری آن ایجاد خواهدکرد. به‌عنوان نمونه، تغییرات یا اصلاً مستند نشده یا خیلی ضعیف مستند می‌شوند، تغییرات در سیستم ممکن است اشکال‌هایی[۱] را اضافه کند یا ساختار سیستم را خراب کند. تمام این‌ها باعث می‌شود که فهم، آزمون، و نگهداری سیستم سخت و سخت‌تر شود.
- **چالش ۳ در مورد سیستم.** چگونه می‌توان سیستم را طراحی کرد بگونه‌ای که تغییرات را نسبتاً به‌سادگی و بدون تاثیر روی بقیه‌ی سیستم انجام داد؟
- **واقعیت ۴ در مورد سیستم.** سیستم ممکن است متشکل از اجزاء سخت‌افزاری و نرم‌افزاری باشد، اجزایی که از سایر شرکت‌ها خریداری شده‌اند را استفاده کند یا با استفاده از زبان‌های برنامه‌سازی مختلف ساخته شده باشد، و روی سکّوهای[۲] مختلف و ماشین‌هایی که در محل‌های مختلف قرار گرفته‌اند اجرا شود.
- **چالش ۴ در مورد سیستم.** چگونه می‌توان سیستم را طراحی کرد به‌گونه‌ای که ویژگی‌های خاص سخت‌افزار، سکو، و پیاده‌سازی را مخفی نماید و در نتیجه، تغییر در این قسمت‌ها بر روی بقیه‌ی سیستم تاثیر نگذارد؟

فهرست فوق کامل نیست. چالش‌های خیلی زیادی در دنیای واقعی وجود دارند که در این لیست نیامده‌اند. اما، همین لیست کافی است تا نشان دهد که یک رویکرد[۳] علمی برای مواجهه با مشکلات کفایت نمی‌کند. ما نیاز به یک روش مهندسی داریم. دو جزء مهم عبارتند از فرایند نرم‌افزار و متدولوژی نرم‌افزار.

۲-۲ فرایند نرم‌افزار

پیشرفت علوم کامپیوتر در دهه‌ی ۱۹۶۰ میلادی، به‌خصوص در رایانش غیرعددی و سیستم‌های پایگاه‌داده‌ی رابطه‌ای، منجر به گسترش سریع کاربردهای کامپیوتر در بخش‌های تجاری گردید. فرایندهای توسعه‌ی اقتضایی[۴] موجود، نمی‌توانست نیازهای این قبیل پروژه‌ها را برآورده سازد. لذا مفهوم فرایند توسعه‌ی نرم‌افزار یا فرایند نرم‌افزار به‌صورت زیر معرفی و تعریف شد.

تعریف ۲-۱

فرایند نرم‌افزار دنباله‌ای از مراحل فعالیت‌ها است که برای ساخت یک سیستم نرم‌افزاری انجام می‌شوند. هر مرحله تعدادی دست‌ساخته (Artifact) را تولید می‌کند که به‌عنوان ورودی برای دیگر مراحل محسوب می‌شود. هر مرحله، مجموعه‌ای از معیارهای ورود (Entrance Criteria) و مجموعه‌ای از معیارهای خروج (Exit Criteria) دارد.

---

[۱] Bug
[۲] Platforms
[۳] Approach
[۴] Adhoc

به‌عنوان مثال، فرایند آبشاری که در فصل ۱ معرفی شد، دنباله سخت‌گیرانه‌ای از فعالیت‌های توسعه را به نمایش می‌گذارد. مرحله‌ی نیازمندی‌ها، تبیین نیازمندی‌ها را تولید می‌کند، مرحله طراحی، طراحی را تولید می‌کند، و الی آخر. معمولاً، معیارهای ورود و خروج به ترتیب مصنوعات نرم‌افزاری که باید آماده باشند و یا تولید شوند را مشخص می‌کنند.

۲-۳ محاسن و مشکلات فرایند آبشاری

فرایند آبشاری سنتی دنباله‌ی مشخصی از فعالیت‌ها، مانند تحلیل نیازمندی‌ها، طراحی، پیاده‌سازی، آزمون و نگهداری را تعریف می‌نماید. این فرایند ده‌ها سال است که توسط شرکت‌های زیادی استفاده می‌شود. قبل از این‌که بخواهیم راجع به مشکلات فرایند آبشاری بحث کنیم، انصاف این است که اشاره‌ای به تعدادی از محاسن این فرایند نیز داشته باشیم. بدین‌سان معلوم می‌شود که چرا این فرایند هنوز در صنعت نرم‌افزار استفاده می‌شود. نخست، دنباله‌ی ساده و مشخص مراحل در فرایند آبشاری کار برنامه‌ریزی پروژه، زمان‌بندی پروژه و پیگیری[۱] وضعیت پروژه را بسیار ساده می‌کند. بدین شکل، این فرایند به‌صورت یک فرایند «قابل پیش‌بینی»[۲] تصور می‌شود. دوم، دنباله‌ی ثابت فعالیت‌های کارکردی امکان سازمان‌دهی پروژه را به‌صورت کارکرد-گرا[۳] فراهم می‌کند. در چنین سازمانی، تیم کارکردی در زمینه‌های کارکردی مختلف تخصص دارند، مانند تکمیل نیازمندی‌ها، طراحی و پیاده‌سازی، پروژه‌ها نیز به‌صورت پشت سر هم (خط لوله[۴]) توسط تیم‌های کارکردی اجرا می‌شوند. سومین حسن فرایند آبشاری این است که روش مرحله به مرحله برای برخی سیستم‌های جاسازی‌شده، طولانی، پیچیده و بزرگ مناسب است. به‌عنوان مثال‌هایی از چنین سیستم‌ها می‌توان به سیستم‌های مرتب‌سازی و هدایت نامه، سیستم‌های کنترل فرایند، و انواع مختلفی از تجهیزات کامپیوتری اشاره نمود. این قبیل سیستم‌ها باید به رخدادهای مختلفی که توسط سخت‌افزارها ایجاد می‌شوند پاسخ دهند، حجم زیادی از داده‌های ورودی را پردازش کنند، و رفتار دستگاه‌های سخت‌افزاری را کنترل نمایند. معمولاً قابلیت‌ها یا نیازمندی‌های چنین سیستم‌هایی مشترکاً توسط مشتری و سازنده‌ی تجهیزات تعریف می‌شوند. در خیلی از موارد، سیستمی که قرار است ساخته شود، فقط یک بهبود اساسی در عملکرد، کارایی و امنیت یک سیستم موجود ایجاد می‌کند. در این حالت، فروشنده تجربه و دانش خوبی از محصول و آن‌چه که مشتری می‌خواهد دارد، و بنابراین، تغییرات اساسی در نیازمندی‌ها به ندرت اتفاق می‌افتد. وقتی هم که سفارش خرید صادر شد، مشتری به محصول دسترسی ندارد تا وقتی که مرحله آزمون و پذیرش انجام شود، گرچه مشتری در بازبینی‌ها و نمایش پیش‌نمونه‌ها مشارکت دارد. روش مرحله‌ای امکان اجرای هر مرحله به‌صورت خشک و جدی را می‌دهد و بدین‌صورت تضمین می‌شود که سیستم به‌طور مطمئن اجرا شده و قیدهای زمانی و کارایی را نیز برآورده می‌سازد.

اما، فرایند آبشاری مشکلات جدی زیادی نیز دارد. نخست، دنباله‌ی ثابت و مشخصی از مراحل و نقاط عطف مربوط به آن‌ها، پاسخ دادن به تغییر نیازمندی‌ها را دشوار می‌سازد. درحالی‌که، تغییر در نیازمندی‌ها در بسیاری شرایط از قبیل افزایش رقبای تجاری، قوانین یا استانداردهای جدید، یا ناتوانی در شناسایی تمام نیازمندی‌ها، ضروری است. در خیلی از کاربردها، طولانی شدن مرحله‌ی توسعه پذیرفتنی نیست زیرا نیازمندی‌ها مدت‌ها قبل شناسایی شده‌اند و نیازهای کسب‌وکار به طرز فاحشی تغییر کرده‌اند. سوم، کاربر تا زمان تحویل سیستم امکان کار با آن و ارائه‌ی بازخورد را پیدا نمی‌کند. تجربه نشان داده است که گرفتن بازخوردهای زود هنگام از کاربر کمک شایانی به یافتن مصداق‌های درک نادرست[۵] نیازهای کسب‌وکار و نیز مشکلات در طراحی واسط کاربر می‌نماید. عیب دیگر این است که در طی زمان طولانی توسعه، مشتری نمی‌تواند از فواید

---

[۱] Tracking
[۲] Predictable
[۳] Function-Oriented
[۴] Piple line
[۵] Misconception

سیستم جدید برخوردار شود. نکته‌ی آخر این‌که، در صورت لغو پروژه مشتری مجبور است ریسک پایین بودن بازگشت سرمایه[۱] را بپذیرد. بدین معنی که مستندات طراحی و قسمت‌هایی از کد که آزمایش نیز شده‌اند، ارزش سرمایه‌گذاری نداشته‌اند.

۲-۴ توسعه‌ی نرم‌افزار یک مسئله‌ی بدرفتار است.

مشکلات فرایند آبشاری کاملاً با نظریه‌ی مسائل بدرفتار مرتبط است. این نظریه اولین بار توسط هورست ریتل[۲]، استاد فقید دانشگاه کالیفرنیا-برکلی [۳] مطالعه شده است. هر مسئله‌ی بدرفتار، تعدادی از ویژگی‌های مسئله‌ی بدرفتار (به اختصار ویژگی‌های بدرفتار) را از خود نشان می‌دهد. این ویژگی‌ها می‌گویند که حل کردن مسائل بدرفتار دشوار است. در مقابل، مسائل خوش‌رفتار[۴] مانند سیستم‌های حل معادلات ریاضی، توسعه‌ی برنامه‌های بازی شطرنج، ساخت مترجم، و تحقیق در عملیات، ویژگی‌های خوبی از خود نشان می‌دهند. شکل ۲-۱ ویژگی‌های این دو نوع مسئله را با هم مقایسه می‌کند.

متاسفانه، توسعه‌ی نرم‌افزار کاربردی عموماً یک مسئله‌ی بدرفتار است. مسئله‌ی خوش‌رفتاری چون حل یک سیستم معادلات، یک شکل (فرمول‌بندی) مشخص دارد. شرح مسئله و جواب را می‌توان از هم جدا کرد. سیستم معادلات همان شرح مسئله است، و انتساب مقدار به متغیرها نیز همان جواب است. برای بسیاری از پروژه‌های توسعه‌ی نرم‌افزار کاربردی، این مصداق ندارد. به‌عنوان مثال تبیین نیازمندی‌ها ممکن است نیازمندی‌های واقعی را مشخص نکند. به همین دلیل است که بسیاری از پروژه‌ها به دلیل عدم‌هم‌خوانی سیستم تحویل شده با انتظارات کاربر، شکست می‌خورند. برای کمک به شناسایی نیازمندی‌های واقعی اغلب یک نمونه ساخته می‌شود. در این موارد، نمونه، هم مشخصات است و هم پیاده‌سازی، زیرا نه تنها ویژگی‌ها را تعیین می‌کند بلکه چگونگی پیاده‌سازی ویژگی‌ها را هم مشخص می‌نماید.

تعداد گام‌هایی که در حل یک سیستم معادلات برداشته می‌شود متناهی است. هر گام، تنها شامل تعداد متناهی از حرکت‌های ممکن است. در مورد طراحی نرم‌افزار چنین چیزی وجود ندارد. تعداد انتخاب‌های طراحی ممکن فقط محدود به دانش و خلاقیت طراح است. فرایند حل یک سیستم معادلات وقتی پایان می‌پذیرد که یک حل پیدا شود. ولی پروژه‌های نرم‌افزاری به این دلیل متوقف می‌شوند که زمان، بودجه، یا صبر تیم تمام شده باشد. حل یک سیستم معادلات را می‌توان بلافاصله و به‌صورت عینی[۵] مورد ارزیابی قرار داد و درستی یا نادرستی آن را سنجید. ولی یک سیستم نرم‌افزاری فقط از نظر خوبی یا بدی می‌تواند ارزیابی شود و قضاوت نیز بسته به دانش، تجربه و تمایلات فردی کسی است که ارزیابی را انجام می‌دهد[۶]. در مقابل، بسیاری از سیستم‌های نرم‌افزاری نیاز به آزمایش دائمی دارند. به‌عنوان نمونه، یک حراج کامپیوتری در تاریخ ۶ می ۲۰۱۰ در یک لحظه ضرری در حدود ۱۰۰۰۰ امتیاز یا ۱۰% ارزش سهام شرکت را برای شرکت صنایع DowJones به بار آورد. سهام شرکت پررونق Procter & Gamble با کاهش تقریبی ۳۷% به پایین‌ترین حد خود در هفت سال رسید. علت این خسارات یک خطای تایپی بود که قبلاً باید جلوی آن گرفته می‌شد. این سیستم، قبل از وقوع این اتفاق، برای ده‌ها سال در حال استفاده بود. این شواهد نرم‌افزاری، و بسیاری دیگر، نشان می‌دهند که چرا سیستم‌های نرم‌افزاری نیاز به آزمون مادام‌العمر دارند. در حالی که حل یک مسئله‌ی خوش‌رفتار را می‌توان برای حل یک مسئله‌ی مشابه تطبیق داد، ولی هر پروژه‌ی توسعه‌ی نرم‌افزار کاربردی پروژه‌ای یکتا است. به همین خاطر است که نرم‌افزار کاربردی اصطلاحاً توسعه می‌یابد یا به‌صورت سفارشی ساخته می‌شود، نه این‌که تولید شود.

توسعه‌ی نرم‌افزار یک فرایند علمی نیست. به عبارت دیگر، بسیاری از تصمیم‌ها نه به‌صورت علمی بلکه به‌صورت سیاسی و اقتصادی گرفته می‌شوند. به‌عنوان مثال، ممکن است به‌جای یک الگوریتم بهینه (کاملاً خوب)، یک الگوریتم نسبتاً خوب

---

[۱] Return on Investment
[۲] Horst Rittle
[۳] University of California Berkeley
[۴] Tame Problems
[۵] Objective
[۶] Evaluator

ویژگی‌های مسائل خوش‌رفتار	ویژگی‌های مسائل بدرفتار
۱. یک مسئله خوش‌رفتار را می‌توان به‌طور کامل مشخص کرد.	۱. یک مسئله بدرفتار ساختار مشخصی ندارد.
۲. برای یک مسئله‌ی خوش‌رفتار، مشخصات مسئله و حل را می‌توان از هم جدا کرد.	۲. در مسائل بدرفتار، تبیین همان حل است و برعکس.
۳. برای مسائل خوش‌رفتار، قوانین توقف وجود دارند.	۳. برای مسائل بدرفتار، قانون توقفی وجود ندارد- همواره می‌توان حل را بهتر کرد.
۴. جواب یک مسئله خوش‌رفتار را می‌توان از نظر درستی یا نادرستی ارزیابی نمود.	۴. جواب یک مسئله‌ی بدرفتار را فقط می‌توان از نظر خوب یا بد بودن ارزیابی کرد و قضاوت نیز سلیقه‌ای است.
۵. برای هر قدم از فرایند حل مسئله، تعداد متناهی از حرکات ممکن وجود دارد.	۵. در هر قدم از فرایند حل مسئله تعداد نامتناهی انتخاب وجود دارد-همه چیز به اصول برمی‌گردد.
۶. یک زنجیره‌ی معینی از استدلال‌های علت و معلولی وجود دارد.	۶. استدلال علت و معلولی مبتنی بر فرض اولیه است، که منجر به اعمال مختلفی می‌شود، ولی نمی‌توان گفت کدام بهترین است.
۷. راه‌حل را می‌توان بلافاصله آزمایش کرد، و پس از آن، برای همیشه درست باقی خواهد ماند.	۷. حل را نمی‌توان بلافاصله آزمایش کرد و منوط به آزمایش مادام‌العمر است.
۸. راه‌حل را می‌توان برای حل مسائل مشابه نیز اتخاذ کرد.	۸. هر مسئله‌ی بدرفتار مسئله‌ای یکتا است.
۹. فرایند حل، یک فرایند علمی است.	۹. فرایند حل یک فرایند سیاست‌مدارانه است.
۱۰. اگر مسئله حل نشود، خیلی ساده دوباره سعی کنید.	۱۰. حل‌کننده‌ی مسئله حق ندارد اشتباه کند چون منجر به فاجعه می‌شود.

شکل ۲-۱: ویژگی‌های مسائل خوش‌رفتار و بدرفتار

انتخاب شود چون پیاده‌سازی، استفاده و نگهداری آن با صرفه‌تر است. برخی مواقع، انتخاب یک زبان برنامه‌سازی یا یک DBMS خاص یک تصمیم سیاسی است. در یکی از پروژه‌هایی که مولف مسئول توسعه‌ی آن بود، مشتری اعلام کرد که از یک محصول به‌خصوص نباید استفاده شود، زیرا وی تجربه‌ی بدی با فروشنده‌ی آن محصول داشته است. به‌عنوان آخرین و مهم‌ترین مطلب، شکست نرم‌افزار می‌تواند منجر به وقوع میلیون‌ها دلار خسارت به اموال و یا از دست رفتن جان انسان‌ها شود. بنابراین، توسعه‌دهندگان نرم‌افزار حق اشتباه کردن ندارند. مسائل خوش‌رفتار چنین وضعی ندارند، اگر نتیجه نادرست باشد، توسعه‌دهنده‌ی نرم‌افزار می‌تواند به سادگی دوباره سعی نماید.

۲-۵ مدل‌های فرایند نرم‌افزار

مشکلات فرایند آبشاری پژوهش‌گران و بنگاه‌های سرمایه‌گذاری را بر آن داشت تا به دنبال فرایند بهتری باشند که ویژگی‌های بدرفتار توسعه‌ی نرم‌افزار را در نظر گیرد. انبوهی از پول و تلاش در این راه صرف گردید. در نتیجه، تعداد زیادی مدل فرایند نرم‌افزار ارائه گردید. اغلب این مدل‌ها، به‌جای یک فرایند ترتیبی محض برای فعالیت‌ها، فرایندی تکراری را انتخاب کرده‌اند. در این بخش، برخی از این مدل‌های فرایند مرور خواهند شد.

۲-۵-۱- فرایند نمونه‌سازی

مدل فرایند نمونه‌سازی، عدم تطابق بین سیستم نرم‌افزاری جدید با انتظارات کاربر و چالش پیش روی تحویل قابلیت‌ها با محدودیت‌های زمانی و بودجه‌ای را قبول می‌کند. به‌عنوان راه‌حل، ابتدا یک نمونه از سیستم ساخته شده و برای جمع‌آوری و اعتبارسنجی نیازمندی‌ها از آن استفاده می‌شود. از این نمونه برای مطالعه‌ی امکان‌سنجی و اعتبارسنجی طراحی نیز استفاده می‌شود. نمونه می‌تواند خیلی ساده یا خیلی پیچیده باشد. در نمونه‌ی ساده، فقط ظاهر سیستم ساخته می‌شود. این نمونه در واقع شامل تعدادی عکس از صفحه است که نشان می‌دهد چگونه سیستم با کاربر تعامل می‌کند. در نمونه‌ی پیچیده، ممکن است بسیاری از وظایف سیستم نیز پیاده‌سازی شده باشند. نمونه‌ها عموماً به دو دسته‌ی «نمونه‌های دور ریختنی» و «نمونه‌های تکاملی» تقسیم می‌شوند. نمونه‌ی دور ریختنی خیلی سریع و ارزان ساخته می‌شود. فقط به اندازه‌ای که کار مربوطه انجام شود. نمونه‌ی دور ریختنی را می‌توان در آزمون واحد یا یکپارچه‌سازی نیز بازاستفاده نمود تا به‌عنوان یک پیاده‌سازی مرجع برای بررسی این‌که پیاده‌سازی، محصول درستی را تولید کرده است، استفاده شود. به‌علاوه، از نمونه می‌توان استفاده کرد تا قبل از تحویل سیستم، به کاربران آموزش داده شود.

۲-۵-۲- فرایند تکاملی

ساختن نمونه به اکتساب نیازمندی‌ها، اعتبارسنجی نیازمندی‌ها، مطالعه‌ی امکان‌سنجی، و اعتبارسنجی ایده‌های طراحی کمک می‌نماید. ولی، نمونه‌ای که دور ریخته شود نشان می‌دهد که تلاش زیادی به هدر رفته است. این در مواقعی اتفاق می‌افتد که برای مطالعه‌ی امکان‌سنجی و اعتبارسنجی طراحی سیستم‌های جاسازی‌شده بلادرنگ و بزرگ نیاز به ساخت نمونه‌های پیچیده باشد. هدف از مدل فرایند تکاملی این است که امکان تکامل یافتن نمونه را بدهد و بدین ترتیب مشکل فوق را حل کند. این مدل، به کاربر اجازه می‌دهد با یک نمونه‌ی اولیه که براساس مجموعه‌ای از نیازمندی‌های ابتدایی ساخته شده است کار کند. بازخوردی که از کاربران دریافت می‌شود در تکامل نمونه مورد استفاده قرار می‌گیرد. این فرایند تا زمانی که گسترش بیشتری نیاز نباشد ادامه می‌یابد. به همین دلیل، به آن مدل فرایند نمونه‌سازی تکاملی نیز گفته می‌شود. از آنجایی که نمونه از نوع دورریختنی نیست (قرار نیست دور ریخته شود)، باید کارکرد عملیاتی و استحکام[1] لازم در ساخت آن در نظر گفته شود.

فرایند تکاملی برای پروژه‌های از نوع اکتشافی، که در آن‌ها الگوریتم‌ها و نیازمندی‌های دقیق باید از طریق کارکردن با سیستم به دست آیند، مناسب‌ترین فرایند می‌باشد. بسیاری از پروژه‌های دنیای واقعی، مثل سیستم‌های هوشمند، نرم‌افزارهای پژوهشی که هدفشان کشف چیزهای ناشناخته مانند دنباله‌ی ژن‌ها است، یا سیستم‌های جاسازی‌شده که فعالانه با محیط در تعامل هستند، از این دسته می‌باشند. فرایند تکاملی برای پروژه‌هایی که زمان‌بندی پیشرفت آن‌ها باید قابل پیش‌بینی باشد، مناسب نیست.

---

[1] Robustness

۲-۵-۳- فرایند حلزونی

فرایند حلزونی که توسط آقای بری بوئم[۱] ارائه گردیده است به دلیل ویژگی منحصربه‌فردی که در مدیریت ریسک دارد، شناخته شده است. همان‌طور که از نام آن برمی‌آید و در شکل ۲-۲ نیز نشان داده شده است، فرایند توسعه حلزونی شکل است. در هر دور از شکل حلزونی، هدف این است که جنبه‌ی مشخصی از سیستم در حال ساخت، به‌عنوان‌مثال عملکرد، کارایی، یا کیفیت را ارتقا دهیم. از این نظر، هرچه مدل دورهای حلزونی را تکرار می‌کند، سیستم به مرور تکامل می‌یابد. هر دور از شکل حلزونی، به‌طور انتخابی برخی از گام‌های زیر را اجرا می‌کند:

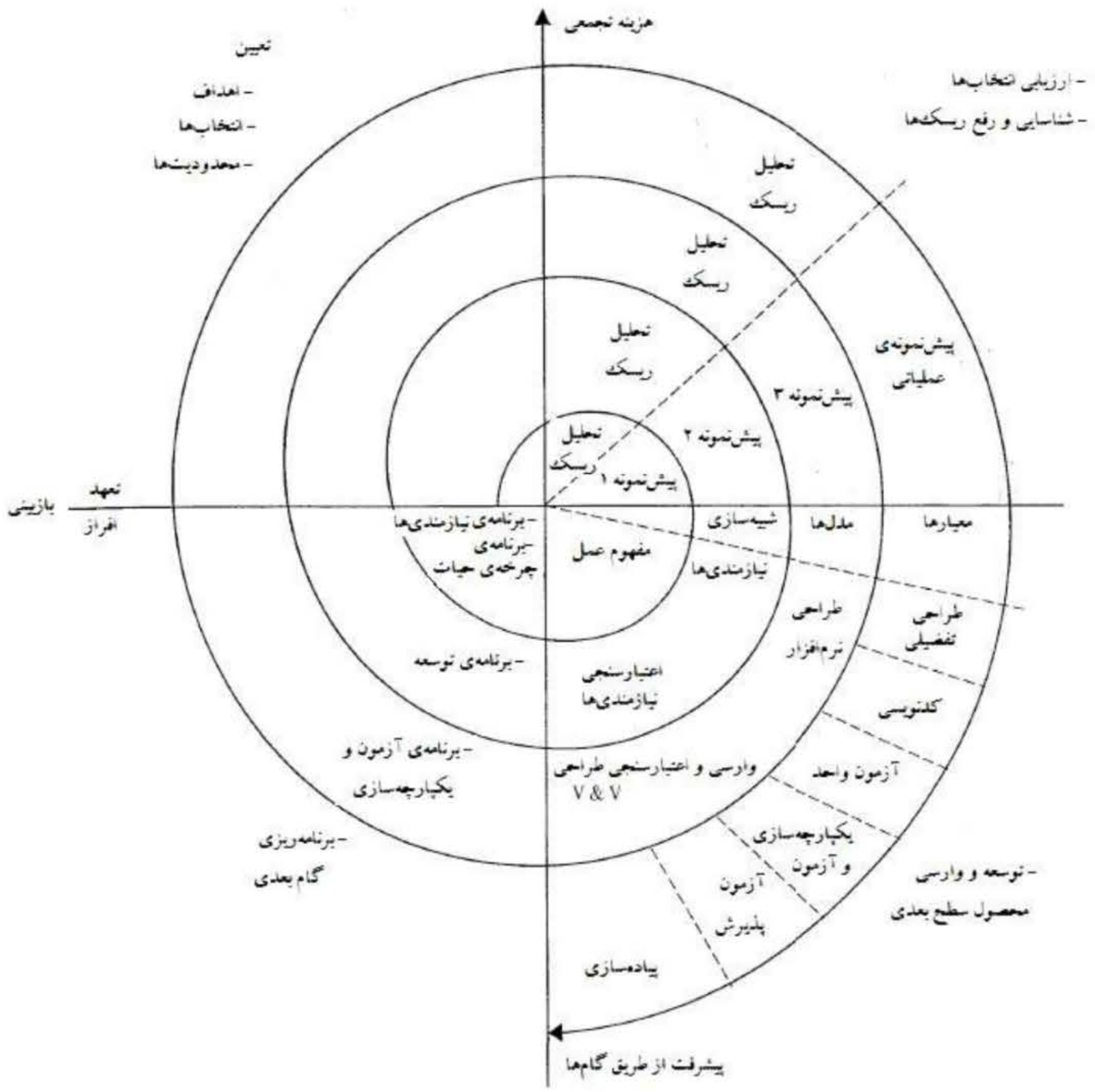
۱. تعیین اهداف، انتخاب‌ها، و قیدهای دور فعلی(گوشه‌ی شمال غربی شکل حلزونی)، در این گام، اهداف، روش‌های جایگزینی برای تحقق اهداف، و محدودیت‌هایی که باید ارضا شوند شناسایی می‌گردند. ریسک‌های پروژه نیز شناسایی و اولویت‌بندی می‌شوند.
۲. ارزیابی انتخاب‌ها، شناسایی و حل‌وفصل ریسک‌ها (گوشه‌ی شمال شرقی شکل حلزونی)، روش‌های جایگزین (انتخاب‌ها) برای رسیدن به اهداف، با توجه به محدودیت‌ها مورد ارزیابی قرار می‌گیرند. ریسک‌های نرسیدن به اهداف، شناسایی و راه‌هایی برای حل‌وفصل آن‌ها تهیه می‌شود. از جمله تکنیک‌های حل‌وفصل ریسک عبارتند از: نمونه‌سازی، شبیه‌سازی، مدل‌سازی، ومحک‌زنی[۲]. بسته به نتیجه‌ی تحلیل و حل‌وفصل ریسک، گام بعدی یکی از موارد زیر می‌باشد:
   - اگر ریسک‌هایی باقی بمانند، دراین صورت گام‌های بعدی آن‌هایی هستند که در قسمت غربی شکل قرار دارند، یعنی برنامه‌ریزی برای سطح بعدی نمونه‌سازی، که پس از آن یک دور جدید نمونه‌سازی نیز تکرار شود.
   - اگر دورهای قبلی اصلی‌ترین ریسک‌های شناخته شده را حل‌وفصل نموده باشند، دراین صورت گام‌های بعدی را می‌توان مانند مدل آبشاری ادامه داد، آن گونه که در گوشه‌ی جنوب شرقی مدل مشاهده می‌شود.
   - اگر نمونه‌ای که در دورهای قبلی تولید شده است، به قدر کافی عملیاتی و مستحکم است که بتواند به‌صورت یک سیستم نهایی تکامل پیدا کند، در این صورت گام‌های بعدی می‌توانند از نمونه استفاده کرده و آن را بر طبق مدل نمونه‌سازی تکاملی، توسعه دهند. بدین معنی که به سمت گوشه‌ی شمال شرقی حرکت می‌کنیم.
۳. توسعه و وارسی سیستم سطح بعدی(گوشه‌ی جنوب شرقی)، همان‌طور که در شکل ۲-۲ نشان داده شده است، این گام مانند مدل آبشاری پیش می‌رود.
۴. برنامه‌ریزی برای مراحل بعدی(گوشه‌ی جنوب غربی)، خواه پروژه جدید باشد، خواه ادامه‌ی یک پروژه‌ی جاری، این گام برنامه‌ی نیازمندی‌ها، فعالیت‌های چرخه‌ی حیات، برنامه‌ی یکپارچه‌سازی و برنامه‌ی آزمون را برای مرحله‌ی بعدی تعریف می‌نماید.

۲-۵-۴- فرایند یکنواخت

فرایند یکنواخت رشنال[۳] یا فرایند یکنواخت، همان‌طور که در شکل ۲-۳ نشان داده شده است، شامل دنباله‌ای از دورهاست. نتیجه‌ی حاصل از هر دور، یک نسخه[۴] از سیستم است. هر دور تعدادی تکرار دارد و تکرارها درون چهار مرحله گروه‌بندی می‌شوند: آغاز[۵]، تشریح[۶]، ساخت[۷]، و انتقال[۸]. هر مرحله با یک نقطه‌ی عطف تمام می‌شود، که در آن نقطه، مدیر تصمیم

---

[۱] Barry Boehm
[۲] Benchmarking
[۳] Rational Unified Process (RUP)
[۴] Release
[۵] Inception
[۶] Elaboration
[۷] Construction
[۸] Transition



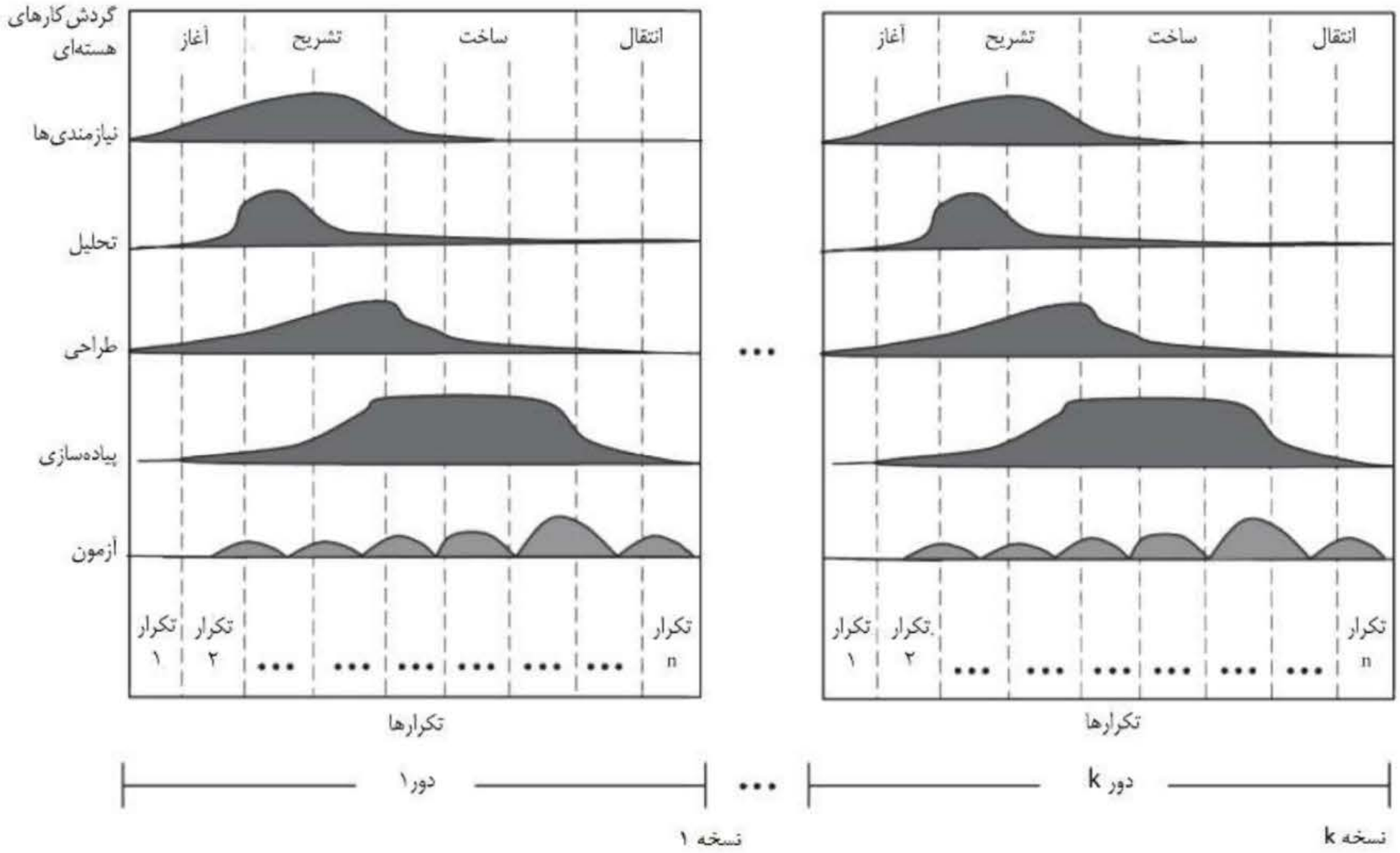
شکل ۲-۲: فرایند حلزونی

مهمی در خصوص ادامه یا توقف پروژه می‌گیرد. در هر تکرار، تعدادی فعالیت گردش کار[۱] مانند تحلیل نیازمندی‌ها، طراحی، پیاده‌سازی و آزمون انجام می‌شود. نواحی خاکستری در شکل ۲-۳ بیان‌گر تخمین‌های غیر دقیقی از میزان فعالیت‌های گردش کار هستند که درحین مراحل انجام می‌شوند. تمرکز فعالیت‌های هر کدام از مراحل چهارگانه در زیر توصیف شده‌اند.

آغاز تکرارهای اول یا دوم، مرحله‌ی آغازین را شکل می‌دهند. در این مرحله یک مدل مورد کاربردی ساده، یک معماری پیشنهادی نرم‌افزار، و یک برنامه‌ی پروژه تهیه می‌شود. به بیان ساده، هر مورد کاربرد یک فرایند کسب‌وکاری از محصولی که سیستم برای آن ساخته می‌شود را مدل می‌کند. برای توصیف مورد کاربردها، عبارت‌های فعلی-اسمی[۲] به‌کار می‌روند. به‌عنوان مثال، واریز وجه، برداشت وجه، و دریافت صورت‌حساب از جمله مورد کاربردهای یک سیستم عابربانک[۳] می‌باشند. مدل مورد کاربرد معمولاً شامل حیاتی‌ترین مورد کاربردهای یک سیستم نرم‌افزاری می‌شود.

تشریح مرحله‌ی تشریح شامل چند تکرار بعدی می‌شود. در طی این مرحله، اغلب مورد کاربردها مشخص شده و نمودارهای UML که بیان‌گر طراحی معماری سیستم هستند تولید می‌شوند. حیاتی‌ترین مورد کاربردهای سیستم نرم‌افزاری نیز

[۱] Workflow
[۲] Verb-Noun Phrase
[۳] Automatic Teller Machine (ATM)



شکل ۲-۳: شرح فرایند یکنواخت

طراحی و پیاده‌سازی می‌شوند.

ساخت در طول مرحله ساخت، مابقی مورد کاربردها به‌صورت تکراری ساخته شده و درون سیستم یکپارچه می‌شوند. سیستم را می‌توان به مرور (به‌طور افزایشی) در محیط مورد نظر نصب نمود.

انتقال در طول مرحله‌ی انتقال، برای استقرار سیستم فعالیت‌هایی صورت می‌گیرد، مانند آموزش کاربران، آزمون بتا [۱]، تصحیح اشکالات، و ارتقای عملیات سیستم.

تاکید UP بر روی شناسایی مورد کاربردها و استفاده از آن‌ها برای برنامه‌ریزی تکرارهاست تا بتوان سیستم را ساخته و مستقر نمود. از این نظر، UP به‌عنوان یک فرایند مورد کاربرد رانده [۲] شناخته می‌شود. UP در همان اوایل چرخه‌ی‌حیات، ساختار کلی یا معماری سیستم را مشخص می‌کند، و از این معماری برای هدایت امر توسعه‌ی سیستم نرم‌افزاری استفاده می‌کند. به همین دلیل، UP را فرایندی معماری محور [۳] می‌گویند. دو ویژگی دیگر UP عبارتند از تکراری و افزایشی، زیرا سیستم به‌صورت تکراری و افزایشی تهیه و تحویل می‌شود.

---

[۱] Beta Testing
[۲] Use Case Driven
[۳] Architecture Centric

۲-۵-۵- فرایند نرم‌افزار شخصی

فرایند نرم‌افزار شخصی[۱] به‌صورت یک چارچوب[۲] جامع و با این هدف طراحی شده که به مهندسین نرم‌افزار بیاموزد چگونه فرایندهای نرم‌افزاری شخصی‌شان را بهبود دهند. PSP از تعدادی اسکریپت، فرم، استاندارد، و خط‌مشی[۳] تشکیل شده است که یک مهندس نرم‌افزار می‌تواند برای انجام تعدادی تمرین برنامه‌سازی از پیش تعریف شده، آن‌ها را به‌کار ببندد. به‌جای تحمیل یک روش توسعه‌ی مشخص، PSP به مهندس نرم‌افزار امکان می‌دهد روش‌های توسعه‌ی خود را انتخاب کند. در حین فعالیت آموزش، PSP به مهندس نرم‌افزار کمک می‌کند تا جاهایی که امکان بهبود در آن‌ها وجود دارد را شناسایی کند. همچنین به مهندس نرم‌افزار کمک می‌کند چنان توانایی‌هایی را در خود ایجاد نماید که در یک محیط کار تیمی به درد می‌خورند، مانند گسترش توانایی انجام تخمین‌های دقیق‌تر و توانایی رعایت تعهدات[۴]. همان‌گونه که بیان شد، هدف از PSP بهبود فرایند نرم‌افزار شخصی یک مهندس نرم‌افزار است، نه این‌که به‌عنوان یک فرایند توسعه‌ی نرم‌افزار انتخاب شود. یعنی، بعد از یادگیری، انتظار می‌رود که مهندس نرم‌افزار فرایندهای نرم‌افزاری خودش را به‌کار ببندد تا نرم‌افزار با کیفیتی تولید نماید. اعتقاد بر این است که افزایش کیفیت منجر به افزایش بهره‌وری می‌شود زیرا تلاش و زمان صرف شده برای آزمون و اشکال‌زدایی[۵] کاهش می‌یابد.

تکامل فرایند PSP

برای اینکه یادگیری PSP آسان بشود، این فرایند از یک روش تکاملی استفاده می‌کند. بدین مفهوم که این چارچوب به‌صورت دنباله‌ای از فرایندهای از پیش تعریف شده ارائه می‌شود که با PSP0، PSP0.1، PSP1، PSP1.1، PSP2، PSP2.1، PSP3.0 نام‌گذاری شده‌اند.

PSP0 ، PSP0.1. این دو فرایند، اندازه‌گیری و انضباط را معرفی می‌کنند. PSP0 فرایند خط‌پایه[۶]، ثبت زمان، ثبت خرابی‌ها، و استاندارد نوع خرابی را معرفی می‌کند. PSP0.1 استاندارد کدگذاری، سنجش اندازه ، و پیشنهادیه بهبود فرایند را معرفی می‌کند.

PSP1 ، PSP1.1. این دو فرایند تخمین و برنامه‌ریزی را معرفی می‌کنند. به‌طور خاص، PSP1 تخمین اندازه و گزارش آزمون و PSP1.1 برنامه‌ریزی و زمان‌بندی را در بر می‌گیرد.

PSP2 ، PSP2.1. این دو فرایند طراحی و مدیریت کیفیت را معرفی می‌کنند. به‌طور مشخص، PSP2 بازبینی کد و بازبینی طراحی را ارائه می‌کند و PSP2.1 نیز یک قالب طراحی در اختیار می‌گذارد.

PSP3.0. این فرایند برای هدایت توسعه‌ی برنامه‌های در سطح قطعه طراحی شده است.

اسکریپت PSP

در PSP تمام فرایندها با استفاده از اسکریپت‌های فرایند یا (اسکریپت برای اختصار) توصیف می‌شوند. هر اسکریپت هدف، معیارهای ورود، گام‌ها یا فعالیت‌های فرایند، و معیارهای خروج را مشخص می‌نماید. به‌عنوان مثال فرایند خط‌پایه‌ی PSP0 شامل شش مرحله می‌شود: برنامه‌ریزی، طراحی، کدگذاری، ترجمه، آزمون، و کالبدشکافی[۷]. این فرایند را می‌توان توسط اسکریپتی که در شکل ۲-۴ نشان داده شده است، توصیف کرد.

این فرایند شامل سه گام می‌شود: (۱) برنامه‌ریزی، (۲) توسعه، که شامل طراحی، کدگذاری، ترجمه، و آزمون می‌شود، و (۳) کالبدشکافی. معیارهای ورود برای یک اسکریپت، دست‌ساخته‌های نرم‌افزاری که باید قبل از شروع فرایند در دسترس

---

[۱] Personal Software Process (PSP)
[۲] Framework
[۳] Guideline
[۴] Commitments
[۵] Debugging
[۶] Baseline
[۷] Postmortem

مقصود	برای هدایت توسعه برنامه‌ای در سطح قطعه
معیارهای ورود	• توصیف مسئله • فرم خلاصه‌ی برنامه پروژه (PSPO) • یادداشت‌های ثبت زمان و خرابی • استاندارد نوع خرابی • زمان‌سنج (اختیاری)

گام	فعالیت‌ها	توصیف
۱	برنامه‌ریزی	• بیان نیازمندی‌ها را را تولید یا بدست آورید. • زمان توسعه را تخمین بزنید. • فرم خلاصه‌ی برنامه پروژه را پر کنید. • یادداشت ثبت زمان را تکمیل کنید.
۲	توسعه	• برنامه را طراحی کنید. • طراحی را پیاده‌سازی کنید. • برنامه را ترجمه کرده، تمام خطاهای یافته‌شده را برطرف و یادداشت کنید. • یادداشت ثبت زمان را تکمیل کنید.
۳	کالبدشکافی	فرم خلاصه‌ی طرح پروژه را با درج داده‌های واقعی از زمان، خرابی و اندازه تکمیل کنید.
معیارهای خروج		• یک برنامه کاملاً آزمایش شده • فرم خلاصه برنامه پروژه با داده‌های تخمینی و واقعی تکمیل شود. • یادداشت‌های کامل‌شده ثبت زمان و خرابی‌ها

شکل ۲-۴: فعالیت‌های PSP به‌وسیله‌ی اسکریپت‌ها توصیف می‌شوند.

باشند را مشخص می‌نماید. گام‌ها هم فهرستی از فعالیت‌ها و توصیفی از آن‌ها را ارائه می‌نمایند. معیارهای خروج نیز می‌گویند چه دست‌ساخته‌های نرم‌افزاری باید پس از خاتمه‌ی فرایند تولید شده باشند.

گام کالبدشکافی در شکل ۲-۴ یکی از ویژگی‌های منحصر به فرد PSP است. این گام از مهندس نرم‌افزار می‌خواهد که فرم خلاصه‌ی برنامه پروژه را با اطلاعات واقعی مربوط به زمان، خرابی‌ها و اندازه پر کند. این فرم در بخش «فرم‌های PSP» که در ادامه می‌آید نشان داده شده است. مهندس نرم‌افزار این فرم را در پایان هر کار برنامه‌نویسی انجام می‌دهد. فرم به مهندس نرم‌افزار امکان می‌دهد که کارهای نرم‌افزاری شخصی‌اش را مشاهده کند، جاهایی را که امکان بهبود دارند شناسایی نماید، و داده‌هایی را که در تخمین‌ها می‌توانند استفاده شوند جمع‌آوری نماید.

در PSP یک نگاه بازگشتی به فرایند توسعه می‌شود. یعنی، هر فرایند شامل دنباله‌ای از فعالیت‌هاست و هر فعالیت توسط یک فرایند سطح پایین توصیف می‌شود. به‌عنوان مثال، گام برنامه‌ریزی در شکل ۲-۴، یک فرایند است، که می‌تواند توسط اسکریپت دیگری توصیف شود. اسکریپت می‌تواند شامل یک گام نیازمندی‌ها، یک گام تخمین منابع، و یک گام زمان‌بندی باشد. به طرز مشابه، گام توسعه در شکل ۲-۴ می‌تواند توسط یک اسکریپت که شامل گام‌های طراحی، کدگذاری، ترجمه، و آزمون است توصیف شود.

فرم‌های PSP

برای مستندسازی در PSP از فرم استفاده می‌شود. هر فرم اطلاعات متداولی هم‌چون نام دانشجو، تاریخ، نام برنامه، شماره برنامه، مدرس، و زبان برنامه‌سازی را تعیین می‌کند. برای حفظ سازگاری در فرم‌ها، در ادامه اصطلاحات دانشجو و مهندس نرم‌افزار به‌جای یکدیگر استفاده شده‌اند. برخی فرم‌ها در زیر تشریح شده‌اند.

یاداشت ثبت زمان [۱] همان‌گونه که در شکل ۲-۵ نشان داده شده است، این فرم هفت ستون دارد که در آن اختلاف زمان [۲] عبارت است از تاریخ و زمان خاتمه منهای تاریخ و زمان شروع منهای زمان وقفه [۳] . هر دانشجو اقلام موردنظر را برای هر گام یا مرحله از فرایند در یک خط از فرم درج می‌نماید. به‌عنوان مثال، بعد از کامل کردن فرایند خط پایه PSP0 ، دانشجو برای هر یک از گام‌های برنامه‌ریزی، طراحی، کدگذاری، ترجمه، و آزمون یک خط از فرم را تکمیل می‌کند.

یاداشت ثبت خرابی[۴] شکل ۲-۶ فرم یادداشت ثبت خرابی را نشان می‌دهد. در بالای فرم ۱۰ نوع خرابی نوشته شده‌اند، که در شکل ۲-۷ توضیح داده شده‌اند. هر دانشجو خرابی‌هایی که در طول فرایند یافت و رفع شده‌اند را مشخص می‌کند. برای هر خرابی، دانشجو شماره برنامه، تاریخ یافتن خرابی، شماره شناسایی خرابی، نوع خرابی، مرحله‌ای که خرابی در آن یافته و مرحله‌ای که خرابی در آن رفع شده است، زمان صرف شده برای یافتن و برطرف کردن خرابی، شماره‌ی خرابی‌ای که در حین رفعش این خرابی فعلی اتفاق افتاده است، و شرح مختصری از خرابی شامل خطای مشاهده شده و دلیل معرفی آن را وارد فرم می‌کند.

PSP Time Recording Log

Student _____ Date _____
Program _____ Program# _____
Instructor _____ Language _____

Project	Phase	Start Date and Time	Interrupt Time	Stop Date and Time	Delta Time	Comments

شکل ۲-۵: یاداشت ثبت زمان PSP

فرم خلاصه برنامه پروژه [۵] فرم خلاصه برنامه پروژه که در بخش قبلی ذکر شد، در شکل ۲-۸ نشان داده شده است. این فرم شامل چهار بخش می‌شود. در بالای فرم اطلاعات توصیفی ذکر می‌شود که نام دانشجو، تاریخ، نام برنامه، شماره برنامه، مدرس، و زبان برنامه‌سازی استفاده شده را مشخص می‌کند. بخش «زمان در مرحله» (بر حسب دقیقه) زمان برنامه‌ریزی برای فرایند، و زمان واقعی، زمان تا امروز [۶] ، و درصد زمان تا امروز [۷] را برای هر مرحله مشخص می‌کند.

برای مثال، اگر یک دانشجو ۳۰ دقیقه روی مرحله‌ی طراحی برنامه‌ی اول، و ۲۵ دقیقه برای مرحله‌ی طراحی برنامه‌ی دوم وقت گذاشته باشد، آن‌گاه زمان‌های واقعی[۸] و تا امروز برای اولین برنامه ۳۰ دقیقه می‌باشند. ولی برای برنامه‌ی دوم زمان‌های واقعی و تا امروز به ترتیب ۲۵ و ۵۵ دقیقه می‌باشند. اگر کل زمان تا امروز برای برنامه‌ی اول ۱۱۷ دقیقه باشد، آن‌گاه درصد زمان تا امروز برای برنامه‌ی اول ۲۵/۶% می‌باشد (یعنی ۳۰ تقسیم بر ۱۱۷). بخش خرابی‌های تزریق شده نیز ستون‌ها و

---

[۱] The Time Recording Log
[۲] Delta Time
[۳] Interrupt Time
[۴] The Defect Recording Log
[۵] The Project Plan Summary Form
[۶] To Date
[۷] To Date%
[۸] Actual

Defect Types	
10 Documentation	60 Checking
20 Syntax	70 Data
30 Build, Package	80 Function
40 Assignment	90 System
50 Interface	100 Environment

PSP Defect Recording Log

Student Student 3 Date 1/19
Program Standard Deviation Program# 1
Instructor Brown Language C++

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
1	1/19	1	20	Code	Comp.	1	

Description Missing semicolon.

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
		2	20	Code	Comp.	1	

Description Missing semicolon.

شکل ۲-۶: یاداشت ثبت خرابی PSP

شناسه	نوع خرابی	شرح
۱۰	مستندسازی	توضیحات، پیام‌ها، دفترچه‌های راهنما
۲۰	نحو	املا، نشانه‌گذاری، غلط‌های تایپی، قالب دستورالعمل‌ها
۳۰	ساخت، بسته‌بندی	مدیریت تغییرات، کتابخانه، کنترل نسخه
۴۰	انتساب	اعلان، نام‌های تکراری، قلمرو، حدود
۵۰	واسط	فراخوانی روال‌ها و مرجع‌ها، ورودی/خروجی ،(I/O) قالب کاربر
۶۰	بررسی	پیام‌های خطا، بررسی‌های ناکافی
۷۰	داده	ساختار، محتوا
۸۰	تابع	منطق، اشاره‌گرها، حلقه‌ها، بازگشت، محاسبات، خرابی توابع
۹۰	سیستم	پیکربندی، زمان‌بندی، حافظه
۱۰۰	محیط	طراحی، ترجمه، آزمون، یا سایر مشکلات پشتیبانی سیستم

شکل ۲-۷: خلاصه انواع خرابی در PSP

ردیف‌هایی مشابه بخش زمان در مرحله [۱] دارد و تعداد خرابی‌های هر مرحله را یادداشت می‌کند. بخش خرابی‌های حذف شده[۲] نیز مشابه بخش خرابی‌های تزریق شده است و تعداد خرابی‌های رفع شده را یاداشت می‌کند.

PSP همچنین شامل روش‌هایی است که در تخمین و برنامه‌ریزی به مهندس نرم‌افزار کمک می‌کند. به‌علاوه، PSP روال‌های تضمین کیفیت دارد که به مهندس نرم‌افزار کمک می‌کند کیفیت نرم‌افزار را بهبود بخشد. این‌ها در پیسوت آ معرفی شده‌اند.

[۱] Time in Phase

[۲] Defects Removed

PSP0 Project Plan Summary Form

Student	Student 3	Date	1/19
Program	Standard Deviation	Program #	1
Instructor	Brown	Language	C++

Time in Phase (min.)	Plan	Actual	To Date	To Date %
Planning		5	5	4.3
Design		30	30	25.6
Code		32	32	27.4
Compile		15	15	12.8
Test		5	5	4.3
Postmortem		30	30	25.9
Total	- 180	117	117	100.0

Defects Injected	Actual	To Date	To Date %
Planning	0	0	0.0
Design 2 2 28.6	2	2	28.6
Code 5 5 71.4	5	5	71.4
Compile 0 0 0.0	0	0	0.0
Test 0 0 0.0	0	0	0.0
Total Development	7	7	100.0

Defects Removed	Actual	To Date	To Date %
Planning	0	0	0.0
Design	0	0	0.0
Code	0	0	0.0
Compile	6	6	85.7
Test 1 1 14.3	1	1	14.3
Total Development	7	7	100.0
After Development	0	0	

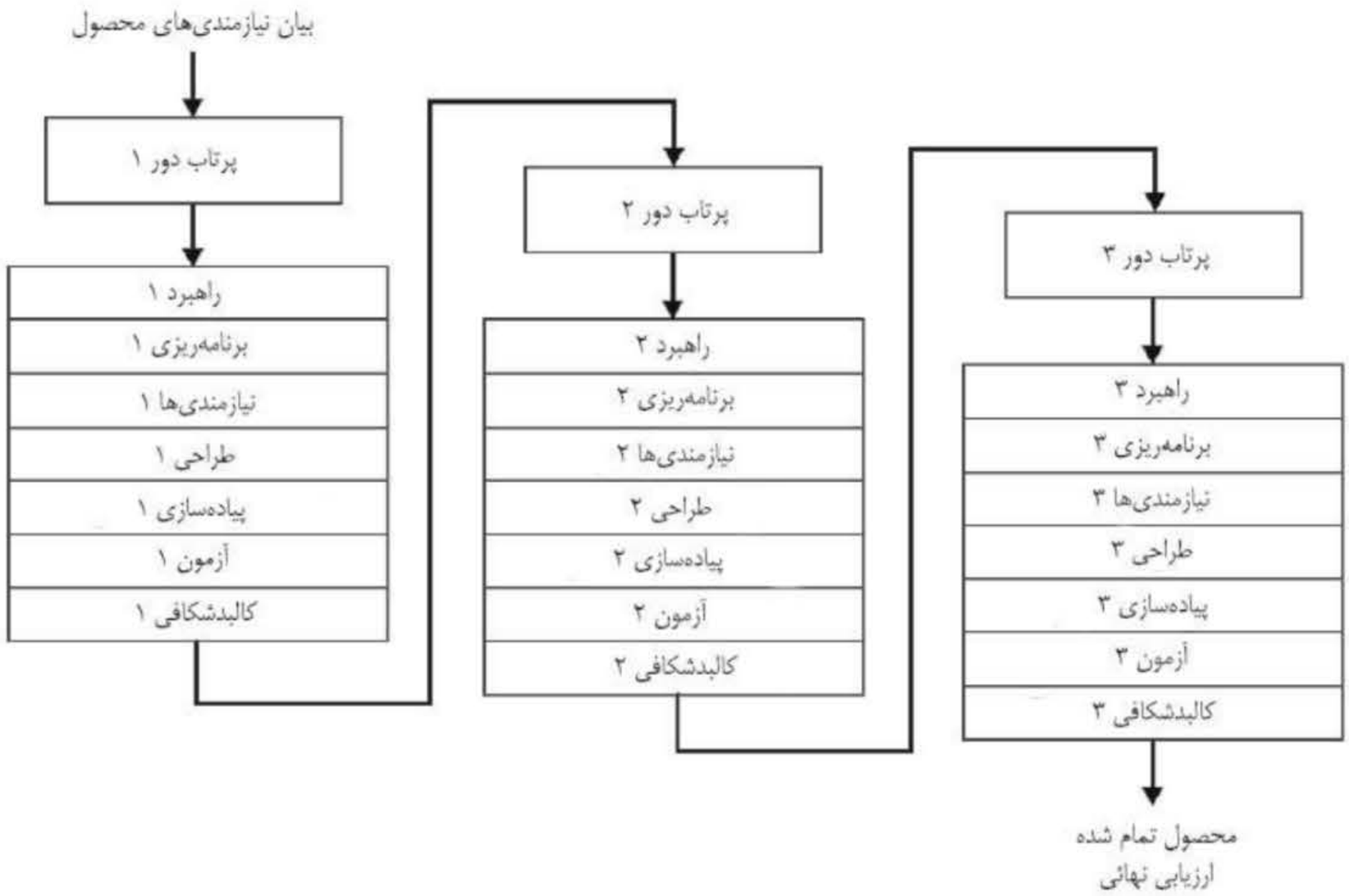
شکل ۲-۸: فرم خلاصه برنامه‌ی پروژه PSP0

۲-۵-۶- فرایند نرم‌افزار تیمی[۱]

فرایند نرم‌افزار تیمی توسط موسسه‌ی مهندسی نرم‌افزار[۲] طراحی شده است با این هدف که اعضای تیم که با PSP آشنا شده‌اند بتوانند با یکدیگر کار کنند تا یک پروژه‌ی تیمی را به سرانجام برسانند. همان‌گونه که در شکل ۲-۹ نشان داده شده است، TSP شامل تعدادی دور است. هر دور نیز تعدادی فعالیت انجام می‌دهد. هر پروژه‌ی TSP با یک فرایند پرتاب[۳] TSPشروع می‌شود تا تیم تشکیل و برنامه‌ی پروژه تهیه شود. فرایند پرتاب توسط یک مربی[۴] مجرب و آموزش‌دیده‌ی TSP هدایت می‌شود. این فرایند نیازهای مشتری را شناسایی می‌کند، نقش‌ها را به اعضای تیم منسوب می‌نماید، و یک مفهوم اولیه از سیستم، یک راهبرد[۵] توسعه، و یک برنامه برای توسعه‌ی سیستم تولید می‌کند. به‌علاوه تیم TSP یک برنامه‌ی کیفیت و یک برنامه‌ی مدیریت ریسک نیز تهیه می‌کند. برنامه‌ها به مدیریت ارائه می‌شود و او ممکن است آن را تصویب نموده یا تغییراتی را خواستار شود. آخرین گام از هر دور، کالبدشکافی است. در جلسه کالبدشکافی، تیم، فرایندهای پرتاب را بازبینی می‌نماید، پیشنهادها برای

[۱] Team Software Process (TSP)
[۲] Software Engineering Institute (SEI)
[۳] Launch
[۴] Coach
[۵] Strategy

بهبود را شناسایی و ثبت می‌کند، و کارهای بعدی را به اعضای تیم واگذار می‌نماید.



شکل ۲-۹: فرایند نرم‌افزار تیمی

فعالیت‌های TSP برای هر دور در قالب یک اسکریپت تبیین می‌شوند. شکل ۲-۱۰ یک اسکریپت را نشان می‌دهد که برای استفاده در متدولوژی ارائه شده در این کتاب مختص شده است. بدین معنی که، متدولوژی فرایند TSP را پیاده‌سازی می‌کند.

اسکریپتی که در شکل ۲-۱۰ نشان داده شده است برای یک کار تیمی یک ترمی طراحی شده است که شامل آموزش نیز می‌شود. این پیشنهاد، چندین بار آزمایش شده است، ولی با این حال می‌تواند اصلاح شود، یا برای موقعیت‌های مختلف تنظیم شود. به‌عنوان مثال، می‌تواند در زمان کوتاه‌تری اجرا شود. در این حالت، تنها یک یا دو دور خواهد داشت. همچنین می‌توان برخی مباحث را حذف کرد، یا به عنوان مثال اعمال الگوهای خاص موقعیت را به درس دیگری منتقل کرد. گزینه‌ی دیگر این است که از این اسکریپت فقط برای تولید طراحی (و نه پیاده‌سازی) استفاده نمود.

۲-۵-۷- فرایندهای چابک

فرایند آبشاری برای مسائل خوش‌رفتار به خوبی جواب می‌دهد، چون این مسائل چند خصوصیت جالب دارند. ولی توسعه‌ی نرم‌افزارهای کاربردی یکی از مسائل بدرفتار است. به همین دلیل، نیاز به فرایندی دارد که برای حل مسائل بدرفتار طراحی شده باشد. فرایندهای چابک از این قبیل فرایندها می‌باشند. فرایندهای چابک بر کار تیمی، توسعه‌ی برنامه همراه با کاربران، طراحی برای تغییر[۱]، و توسعه‌ی سریع و تحویل مداوم افزایش‌های کوچک در تکرارهای کوتاه تاکید دارند. توسعه‌ی چابک بر ارزش‌ها، اصول، و بهترین روش‌های[۲] چابک استوار است. تمام این موارد ویژگی‌های مسائل بدرفتار را در نظر گرفته‌اند.

[۱] Design for Change

[۲] Best Practice

یک اسکریپت برای فرایند نرم‌افزار تیمی

مقصود	برای راهنمایی یک تیم از طریق توسعه‌ی یک محصول نرم‌افزاری	
معیارهای ورود	• مدرس یک یا چند تیم پنج نفره را از دانشجویان را راهنمایی و هدایت می‌کند. • دانشجویان همگی PSP را فراگرفته‌اند. • مدرس مواد، تسهیلات و منابع موردنیاز برای حمایت از تیم‌ها را دراختیار دارد. • مدرس اهداف فعلی محصول را تشریح کرده است.	
عمومی	فرآیند PSP برای پشتیبانی از سه شکل تیم طراحی شده است. آن اسکریپتی که به کار می‌آید را دنبال کنید: ۱) یک محصول نرم‌افزاری با اندازه‌ی کوچک تا متوسط در دو یا سه دور توسعه می‌یابد. ۲) یک محصول کوچکتر در یک دور ساخته شود. ۳) یک جز از یک محصول، مانند نیازمندی‌ها، طراحی، یا برنامه‌ی آزمون در بخشی از یک دور تولید شود.	
هفته	**گام**	**فعالیت ها**
۱	مرور	• معرفی درس و مرور PSP • پیشگفتار، مقدمه، و این فصل را مطالعه کنید، برروی بخش PSP تمرکز کنید.
۲	LAU1	• اهداف درس را مرور کنید و تیم‌های دانشجویان و نقش‌ها را مشخص کنید. • بخش‌های TSP و نیز مروری به متدلوژی یکنواخت چابک در این فصل را بخوانید.
	STRAT1	• طراحی مفهومی را بسازید، راهبرد توسعه را مشخص کنید، تخمین اندازه بزنید، و ریسک‌ها را ارزیابی کنید. • یک سبک طراحی معماری نرم‌افزاری را به کار ببرید(در اکثر موارد معماری n-تایی انتخاب می‌شود). • فصول مربوط به طراحی معماری و مدیریت پروژه را بخوانید، با تیکه برروی بخش‌های تخمین و مدیریت ریسک
۳	PLAN1, REQ1	• نیازمندی‌ها را تعریف و بررسی کنید، با تکیه برروی نیازمندی‌های بااهمیت‌تر. • از روی نیازمندی‌ها، مورد کاربردها را بدست آورید، نمودارهای مورد کاربرد و ماتریس ردیابی را تولید کنید، مورد کاربردهای سطح بالا را مشخص کنید. • مورد کاربردها را به دورها تخصیص دهید و ماتریس تخصیص را بسازید. • مورد کاربردها، نمودارهای مورد کاربرد، تعیین مورد کاربردهای سطح بالا و ماتریس‌ها را بازنگری کنید. • فصول مهندسی سیستم، جمع‌آوری نیازمندی‌های نرم‌افزار، و تضمین کیفیت را بخوانید، با تمرکز برروی بخش‌های مرتبط با نیازمندی‌ها، فصل استخراج مورد کاربرد را نیز بخوانید.
۴	PLAN1, DES1	• دور ۱ مدل‌سازی دامنه را انجام دهید (طوفان فکری، طبقه‌بندی مفهومی دامنه، و مصورسازی مدل دامنه) • مورد کاربردهای گسترش‌یافته‌ی دور ۱ را مشخص کنید، مورد آزمون‌های مبتنی بر مورد کاربرد را تولید کنید. • مدل دامنه، مورد کاربردهای گسترده، و مورد آزمون‌های مبتنی بر مورد کاربرد را بازبینی کنید. • فصول مربوط به مدل‌سازی دامنه، مدل‌سازی تعامل کنشگر سیستم، و آزمون نرم‌افزار (آزمون مبتنی بر مورد کاربرد) را بخوانید.
۵	DES1	• سناریوها، جداول سناریو، و نمودارهای توالی مربوط به دور ۱ را تولید و مرور کنید. • نمودار کلاس طراحی (DCD)، و طراحی واسط کاربر مربوط به دور ۱ را بدست آورده و بازرسی کنید. • فصول مدل‌سازی تعامل شئ، استخراج نمودار کلاس طراحی، و طراحی واسط کاربر را بخوانید.
۶	IMP1	• توسعه‌ی آزمون‌رانده را برای دور ۱ شروع کنید (احتمالاً آن را با برنامه‌نویسی دوتایی ترکیب کنید) بدین منظور که ۱۰۰٪ پوشش شاخه را برآورده سازد. • موارد آزمون واحد را بازبینی کنید. • فصول پیاده‌سازی، تضمین کیفیت، و آزمون نرم‌افزار را بخوانید.
۷	TEST1	• ساخت و یکپارچه‌سازی دور ۱ را انجام دهید، مورد آزمون‌های مبتنی بر مورد کاربرد را اجرا کنید. • نرم‌افزار دور ۱ را برای مشتری و کاربران نمایش دهید، بازخوردها را جمع‌آوری و ثبت کنید. • مستندات کاربری مربوط به دور ۱ را تولید کنید.
۸	PM1	• کالبدشکافی انجام دهید و گزارش نهایی دور را بنویسید. • ارزیابی نقش‌ها و تیم را برای دور ۱ انجام دهید.
	LAU2	• تیم‌ها و نقش‌ها را برای دور ۲ بازسازی کنید.
	STRAT2,PLAN2,REQ2	• برای دور ۲، راهبرد و برنامه را تولید کنید، ریسک‌ها را ارزیابی کنید. • نیازمندی‌ها، مدل دامنه، مورد کاربرد، ماتریس ردیابی، و ماتریس تخصیص را اصلاح و بازنگری کنید.
۹	DES2	• الگوهای GRASP را اعمال کنید، و نمودارهای توالی دور ۱ را مرور و اصلاح کنید. • مورد کاربردهای گسترش‌یافته و موارد آزمون مبتنی بر مورد کاربرد دور ۲ را تولید و بازرسی کنید. • الگوهای GRASP را به کار برده و سناریوها، جداول سناریو، و نمودارهای توالی را برای دور ۲ تولید و بازبینی کنید. • فصل اعمال الگوهای انتساب مسئولیت را بخوانید.

شکل ۲-۱۰: اسکریپت توسعه‌ی TSP

۱۰	IMP2	• توسعه‌ی آزمون‌رانده و بازرسی دور ۲ را انجام و به پوشش کد ۱۰۰٪ برسید. • موارد آزمون واحد و کد را بازبینی کنید.
	TEST2	• ساخت، یکپارچه‌سازی، و آزمون دور ۲ انجام شود، نرم‌افزار دور ۲ را به مشتری کاربران نمایش دهید، بازخورد بگیرید و یادداشت کنید. • مستندات کاربران را برای دور ۲ تهیه کنید.
۱۱	PM2	• کالبدشکافی انجام دهید و گزارش نهایی دور ۲ را بنویسید. • ارزیابی نقش‌ها و تیم را برای دور ۲ تهیه کنید.
	LAU3	• تیم‌ها و نقش را برای دور ۳ از نو سازماندهی کنید.
	STRAT3, PLAN3, REQ3	• راهبرد و برنامه را برای دور ۳ تهیه کنید، ریسک‌ها را ارزیابی کنید. • نیازمندی‌ها، مدل دامنه، مورد کاربردها، ماتریس ردیابی، و ماتریس تخصیص را اصلاح و بازبینی کنید.
۱۲	DES3	• الگوهای GoF یا الگوهای خاص موقعیت را اعمال کنید، نمودارهای طراحی دورهای ۱ و ۲ را بروز کنید. • مورد کاربردهای گسترش‌یافته و نیز موارد آزمون مبتنی بر مورد کاربرد مربوط به دور ۳ را تولید و بازرسی کنید. • نمودارهای توالی دور ۳ را تولید و بازبینی کنید (الگوهای خاص موقعیت اعمال شده‌اند). • فصل اعمال الگوهای خاص موقعیت را بخوانید.
۱۳	IMP3	• توسعه آزمون‌رانده و بازرسی دور ۳ را انجام و به پوشش کد ۱۰۰٪ برسید. • موارد آزمون واحد و کد را بازبینی کنید.
	TEST3	• ساخت، یکپارچه‌سازی و آزمون دور ۳ را انجام دهید، محصول نهایی را به مشتری و کاربران نمایش دهید، بازخوردها را جمع‌آوری و یادداشت کنید. • راهنمای کاربران را در مورد محصول نهایی تولید و بازبینی کنید.
۱۴	PM3	• کالبدشکافی انجام دهید و گزارش نهایی دور ۳ را بنویسید. • ارزیابی نقش‌ها و تیم را برای دور ۳ تهیه کنید. • محصول تولید شده فرآیندهای استفاده شده را بازبینی کنید، درس‌های گرفته شده را بشناسید و پیشنهادهای خود را برای بهبود فرآیند ارائه دهید.
معیارهای خروج		• محصول یا جزئی از محصول و نیز مستندات کاربران تکمیل شده است. • دفتر یادداشت پروژه بروزرسانی و تکمیل شده است. • ارزیابی تیم و گزارش دورها مستند شده‌اند.

شکل ۲-۱۰: ادامه

بیانیه‌ی چابکی

بر طبق بیانیه‌ی چابکی[۱]، توسعه‌ی چابک برای چهار جنبه از کارهای توسعه‌ی نرم‌افزار ارزش قائل می‌شود که با آنچه در روش‌های رقیب سنتی و برنامه-رانده مرسوم است فرق دارد. این ارزش‌ها در ادامه فهرست شده و توضیح داده شده‌اند.

- توسعه‌ی چابک برای افراد و تعاملات بین آن‌ها بیش از فرایندها و ابزارها ارزش قائل است.
- توسعه‌ی چابک برای نرم‌افزار در حال کار بیش از مستندات جامع ارزش قائل است.
- توسعه‌ی چابک برای همکاری با مشتری بیش از مذاکره پیرامون قرارداد ارزش قائل است.
- توسعه‌ی چابک برای واکنش نشان دادن به تغییر بیش از پیروی از یک برنامه ارزش قائل است.

۱. **چابکی برای افراد و تعاملات بین آن‌ها بیش از فرایندها و ابزارها ارزش قائل است.**

روش‌های سنتی و برنامه-رانده براین باورند که داشتن یک فرایند نرم‌افزاری خوب برای موفقیت یک پروژه‌ی نرم‌افزاری ضروری است. یکی از پندهای متداول این است که «کیفیت نرم‌افزار همان قدر خوب است که فرایند نرم‌افزار.» باوجودی که این پند متداول هنوز خوبی‌های خودش را دارد، ولی تجربه نشان داده است که توانایی‌های اعضای تیم و نیز کار گروهی مهم‌ترند. در هر حال، این اعضای تیم هستند که فرایند نرم‌افزاری را اجرا می‌کنند. اگر اعضای تیم ندانند چگونه طراحی بکنند یا به‌طور موثری با یکدیگر ارتباط برقرار نکنند، نتیجه‌ی خوبی حاصل نخواهد شد. روش‌های سنتی وزن زیادی برای استفاده از ابزارها قائل هستند. به همین دلیل بسیاری از شرکت‌ها سرمایه‌گذاری سنگینی برای

[۱] Agile Manifesto (www.agilemanitesto.org)

توسعه‌ی ابزارها و محیط‌ها می‌کنند. البته برخی از ابزارها خوبند و مسائل مربوط را حل می‌کنند، ولی این ابزارها فقط توسط افراد مناسب، که چگونگی استفاده از ابزار را می‌دانند، می‌توانند به‌کار برده شوند. اگر مهندس نرم‌افزار نداند چگونه طراحی OO را انجام دهد، یک ویرایش‌گر نمودارهای UML نمی‌تواند کمکی بکند. گرچه ویرایش‌گر، نمودارهای به ظاهر زیبایی را تولید می‌کند، ولی این‌ها لزوماً به معنای طراحی‌های خوب نیستند.

برخلاف روش‌های سنتی، روش‌های چابک به افراد و کار تیمی بهای بیش‌تری می‌دهند. دلیل این امر این است که نرم‌افزار یک محصول مفهومی است و فعالیت‌های توسعه به شدت فکری[۱] هستند. اگر اعضای تیم مجبور باشند با همدیگر کار کنند تا مشترکاً محصول نرم‌افزاری را بسازند، در این صورت توانایی‌های اعضای تیم در تعامل و سهیم بودن در این تلاش جمعی، برای موفقیت پروژه یک ضرورت به حساب می‌آید. قطعاً فرایندهای نرم‌افزاری و ابزارها نیز مطرح هستند، ولی افراد و تعاملاتشان اصل محسوب می‌شود.

۲. **چابکی برای نرم‌افزاری که کار بکند بیش از مستندات جامع ارزش قائل است.**

برای سال‌ها و حتی دهه‌ها، شرکت‌ها تلاش بسیار زیادی روی تهیه‌ی مستندات تحلیل و طراحی صرف می‌کردند. دلیل این امر تا حدودی به بازرسی استانداردها برمی‌گشت و تا حدودی نیز به خاطر این اعتقاد بود که «نرم‌افزار خوب از مستندات طراحی خوب، و مستندات طراحی خوب نیز از مدل‌های تحلیل خوب به دست می‌آیند.» این باورها فقط تا حدودی صحیح هستند. بسیاری از مهندسان نرم‌افزار این را تجربه کرده‌اند که در برخی موارد تعیین نیازهای واقعی، یا این‌که آیا طراحی واقعاً کار می‌کند، تا زمانی که کد نوشته و آزمایش نشود غیرممکن است. دراین‌گونه موارد، مستندات جامع کمکی نمی‌کند و حتی ممکن است مضر باشد زیرا این‌طور برداشت می‌شود که یک راه‌حل کاری پیدا شده است. همچنین، تهیه مستندات جامع بدین معنی است که وقت کمتری برای کدگذاری و آزمون، که در این‌گونه موارد تنها روش‌هایی هستند که می‌توان نیازمندی‌های واقعی و طراحی مناسب را شناسایی نمود، وجود دارد.

به‌عنوان مثال، نرم‌افزاری که هدفش بهینه‌سازی فهرست انبار[۲] یک شرکت بزرگ است را در نظر بگیرید. فهرست انبار شامل توصیف متنی میلیون‌ها قلم کالاست که در طی چند دهه‌ی گذشته توسط کارمندان مختلفی نوشته شده است. فعالیت‌های مختلفی که برای جمع‌آوری و نیز ادغام صورت گرفته، تعداد اقلام، طبقه‌ی اقلام و قالب و سبک توصیف آن‌ها را افزایش داده است. برای پردازش توصیف‌های مندرج در فهرست انبار، نیاز به یک نرم‌افزار است. هدف، ساده‌کردن کار انبار و کاهش هزینه‌های آن است. واضح است که نیازمندی‌های نرم‌افزار آن چیزهایی است که نرم‌افزار می‌تواند برای رسیدن به این هدف انجام دهد. ولی بدون پیاده‌سازی نرم‌افزار، هیچ‌کس نمی‌داند نرم‌افزار چه‌کار خواهد کرد. این مثالی از یک مسئله‌ی بدرفتار است. مشخصات و پیاده‌سازی را نمی‌توان از هم جدا کرد. فرض کنید که نیازمندی‌ها به نحوی بدون نیاز به پیاده‌سازی نرم‌افزار شناسایی شده‌اند. در این صورت، طراحی داده ساختارها و الگوریتم‌ها خود یک چالش جدی است. زیرا بسیار مشکل خواهد بود که بدانیم آیا الگوریتم‌ها اصلاً کار می‌کنند و اگر بلی، تا چه حدی. دلیل این مشکلات، تنوع در توصیف فهرست کالاها، ناسازگاری‌ها، اطلاعات ناقص، غلط‌های نگارشی و مختصرنویسی‌های متفاوت می‌باشد. به نظر می‌رسد، روش آزمون و خطا[۳] برای این پروژه مناسب‌تر باشد.

روش‌های چابک به نرم‌افزار در حال کار بها می‌دهند زیرا نرم‌افزار در حال کار انتهای ماجراست[۴]. به هر حال، تیم توسعه باید نرم‌افزاری که کار می‌کند (نرم‌افزار در حال کار) را تحویل مشتری بدهد. فقط نرم‌افزار در حال کار را می‌توان آزمایش کرد و مطمئن شد که سیستم نرم‌افزاری قابلیت‌های خواسته‌شده را خواهد داشت. از این رو، نرم‌افزار در حال کار همان نیازمندی‌هاست و برعکس. پروژه‌ی طبقه‌بندی توصیفات فهرست کالاها که در بالا بدان اشاره شد این موضوع را نشان می‌دهد. البته از این بحث نباید نتیجه گرفت که روش‌های چابک تمایلی به تحلیل و طراحی ندارند. برعکس،

---

[۱] Intellectual
[۲] Inventory
[۳] Trial-and-Error
[۴] Bottomline

روش‌های چابک نیز مدل‌های تحلیل و طراحی رامی‌سازند. به هر شکل، اصول چابکی طرفدار ایده‌ی مدل‌سازی فقط به اندازه‌ی کافی هستند. این ایده، مدل‌سازی را فقط تا حدی که به فهم مسئله و تبادل ایده‌های طراحی بین اعضای تیم کمک کند، صلاح می‌داند.

۳. **چابکی برای همکاری با مشتری بیش از مذاکره پیرامون قرارداد ارزش قائل است.**

در فرایندهای سنتی یک مرحله مذاکره در مورد قرارداد وجود دارد تا آنچه مشتری می‌خواهد، شناسایی شود. سپس یک مسئله تبیین نیازمندی‌ها تولید و جزیی از قرارداد می‌شود. در حین فرایند توسعه، مشتری فقط در تعدادی جلسه بازبینی طراحی و ازمون پذیرش شرکت می‌کند. بسیاری از تصمیمات مهم طراحی که باید به همراه مشتری اتخاذ شوند، توسط تیم توسعه گرفته می‌شوند. گرچه تیم توسعه در اتخاذ تصمیم‌های فنی وارد هستند، ولی ممکن است دانش و پیش‌زمینه‌ی لازم را برای این‌که برای مشتری تصمیم بگیرند، نداشته باشند. به‌عنوان مثال، این خواسته که باید بیش از یک DBMS پشتیبانی شود ممکن است مشخص نکند که منظور کدام DBMSها است. از نظر فنی، تیم ممکن است بداند کدام DBMSها بهترین هستند و باید پشتیبانی شوند. ولی ممکن است مشتری عوامل دیگر را مهم‌تر بداند. ازجمله این عوامل می‌توان به موارد زیر اشاره نمود: توانایی کارکنان بخش فناوری اطلاعات در نگهداری از انواع DBMSها، هزینه معرفی چنین سیستم‌هایی، قابلیت سازگاری با سیستم‌های موجود. اگر تیم توسعه برای مشتری چنین تصمیم‌هایی را بگیرد، ممکن است سیستم به دست آمده نیازهای کسب‌وکاری مشتری را برآورده نسازد.

همکاری مشتری برای موفقیت یک پروژه ضروری است. این همکاری ارتباط و درک متقابل تیم و مشتری را بهبود می‌بخشد. بهبود در ارتباط به شناسایی نیازمندی‌های واقعی و کاهش احتمال سوءتفاهم در نیازمندی‌ها کمک می‌کند. درک متقابل منجر به اشتراک ریسک می‌شود، و بنابراین احتمال شکست پروژه را کم می‌کند. در بسیاری از پروژه‌ها، پیش‌بینی نتیجه دقیق سیستم، یک تصمیم طراحی، یا یک الگوریتم بسیار مشکل و یا حتی غیرممکن است. در این موارد، همکاری با مشتری بسیار مهم است. درک متقابل بدین معنی است که تیم توسعه درک خوبی از دامنه کسب‌وکاری، عملیات، چالش‌ها، و اولویت‌های مشتری دارد. بدین صورت، تیم قادر است سیستم را طوری طراحی و پیاده‌سازی کند که نیازهای کسب‌وکار مشتری را برآورده نماید.

درک متقابل همچنین بدین معنی است که مشتری محدودیت‌های فناوری، که روش‌هایی برای پیاده‌سازی راه‌حل‌های کسب‌وکار فراهم می‌کند را می‌فهمد، فناوری به تنهایی مسائل کسب‌وکار را حل نمی‌کند. مشتری باید محدودیت‌های تیم توسعه‌دهنده را بفهمد، همان‌گونه که تجربه مولف که در ادامه می‌آید نشان می‌دهد. مشتری اصرار داشت که یک محصول نرم‌افزاری با اندازه متوسط می‌بایست در یک ماه تولید شود، بدون توجه به این واقعیت که مولف گفته بود که این غیرممکن است. علاوه بر کمبود زمان، کمبود برنامه‌نویسان باتجربه، نیز چالش دیگری بود. بعد از شش ماه، تیم هنوز نتوانسته بود محصول را توسعه دهد و لذا پروژه شکست خورد. در این داستان، مشتری سیستم را در یک ماه می‌خواست، ولی هیچ تیمی نمی‌توانست این تقاضا را پاسخ دهد چون قرار بود مجموعه کاملاً جدیدی از ایده‌های کسب‌وکاری نوآورانه در سیستم پیاده‌سازی شود. همکاری مشتری می‌توانست این پروژه را نجات دهد. برای مثال، دو طرف می‌توانند سعی کنند محدودیت‌ها و اولویت‌های یکدیگر را بفهمند و یک برنامه چابک واقع‌بینانه تهیه کنند تا به‌طور افزایش ویژگی‌های نوآورانه را تکمیل نماید.

۴. **چابکی برای پاسخگویی به تغییر، بیشتر از پیروی از برنامه، ارزش قائل است.**

روش‌های سنتی بر روی «کنترل تغییرات» تاکید دارند زیرا تغییر هزینه‌بر است. وقتی یک دست‌ساخته، مانند تبیین نیازمندی‌ها، نهایی شد، تغییرات بعدی باید از یک فرایند کنترل تغییر جدی و خشک عبور کنند. این فرایند تیم را از واکنش به درخواست تغییر بازمی‌دارد. روش‌های چابک به پاسخ به تغییر بیش از پیروی از برنامه بها می‌دهند زیرا تغییر جزیی از زندگی است. در دنیای امروز که سریعاً در حال تغییر است، هر کسب‌وکاری مجبور است به تغییرات در شرایط

کسب‌وکار سریعاً واکنش نشان دهد تا دوام آورده و بتواند رشد کند. بنابراین، تغییر در نرم‌افزار امری اجتناب‌ناپذیر است. پیشرفت در فناوری‌های اینترنت به کسب‌وکارها امکان می‌دهد و آن‌ها را مجبور می‌کند تا وب‌کاربردهای خود را به سرعت و به‌طور مداوم به‌روزرسانی کنند. انعطاف‌ناپذیری روش‌های سنتی برنامه-رانده نمی‌تواند نیازهای چنین برنامه‌هایی را برآورده نماید. بنابراین روش‌های چابک پدیدار می‌شوند.

اصول چابکی

ارزش‌های چابکی موارد تاکید فرایندهای چابک را بیان می‌کنند. برای هدایت توسعه‌ی چابک، جامعه‌ی چابک مجموعه‌ای از اصول راهنما را نیز تهیه کرده است که به آن‌ها «اصول توسعه‌ی چابک» یا به اختصار اصول چابکی می‌گویند. این اصول به شرح زیر می‌باشند.

۱. درگیر شدن فعالانه‌ی کاربر ضروری است.

بیشتر روش‌های چابک درگیر شدن فعالانه‌ی کاربر را واجب می‌دانند. دلیل این امر این است که در اکثر پروژه‌های توسعه نرم‌افزار شناسایی نیازهای واقعی دشوارترین بخش کار است. روش‌های سنتی ۱۵% الی ۲۵% از کل تلاش توسعه را برای تحلیل نیازها صرف می‌کنند، و سپس کنترل تغییرات سخت و محکمی را پیاده می‌کنند تا نیازها را منجمد (بی‌تغییر) سازند. به نظر نمی‌رسد این کارها مشکل را حل کند. در واقع، مشکل کمبود زمان یا تلاش نیست، بلکه این ناتوانی نوع بشر است که نمی‌تواند نیازهای واقعی را در مراحل اولیه فرایند توسعه بشناسد. به‌علاوه، دنیا در حال تغییر است، لذا نیازها هم باید تکامل یابند.

درگیر شدن فعالانه کاربر بدین معنی است که نمایندگانی از جامعه کاربران به‌طور تنگاتنگ و در مواقع مورد نیاز با تیم توسعه تعامل خواهند داشت. به‌عنوان‌مثال، تعدادی از نمایندگان آگاه از طرف کاربر به پروژه تخصیص داده می‌شوند. این افراد یا همراه تیم بوده و با آن‌ها کار می‌کنند و یا به‌طور منظم چند بار در هفته با تیم ملاقات می‌کنند. این برنامه‌ریزی تا حد زیادی ارتباط و تفاهم بین تیم و کاربران را بهبود می‌بخشد. علاوه بر این، اطمینان خواهیم داشت که سوءبرداشت از نیازمندی‌ها در همان مراحل اولیه اصلاح‌شده، بازخوردهای گرفته‌شده از کاربران به درستی و در زمان مناسب مورد رسیدگی قرارگرفته، و تصمیم‌ها در خصوص سیستم با مشارکت کاربران اتخاذشده‌اند. نتیجه‌ی تمام موارد فوق این خواهد بودکه نیازهای واقعی شناسایی و اولویت‌بندی شده،و سیستم به‌گونه‌ای ساخته می‌شود که انتظارات کاربران را برآورده نماید.

۲. تیم باید قدرت لازم را برای تصمیم‌گیری‌ها داشته باشد.

توسعه چابک برای افراد و تعاملات ایشان بیش از فرایندها و ابزارها ارزش قائل است. این اصل نیز همین ارزش را واقعیت می‌بخشد. بدین معنی که، اعضای تیم هم لازم است و هم تشویق می‌شوند که تصمیم‌سازی کنند و مسئولیت و مالکیت را بپذیرند. برای انجام این کار، اعضای تیم باید در طول پروژه به‌عنوان یک تیم کار کنند و با یکدیگر و با کاربران تعامل داشته باشند.

۳. نیازها تکامل پیدا می‌کنند ولی مقیاس زمان ثابت است.

برخلاف روش‌های سنتی که نیازها را منجمد و بی‌تغییرفرض می‌کنند، فرایندهای چابک به‌گونه‌ای طراحی‌شده‌اند که از تغییرات استقبال می‌کنند. منظور این است که برای غلبه بر تغییر نیازها، قلمرو[۱] کار می‌تواند تکامل یابد، ولی محدوده زمان و بودجه توافق شده ثابت خواهد ماند. بدین معنی که نیازهای جدید یا تغییریافته در ازای هزینه سایر نیازها،

---

[۱] Scope

پذیرفته می‌شوند. در این صورت، تلاش اضافی که صرف می‌شود با کنار گذاشتن دیگر نیازها که بحرانی نیستند، جبران خواهد شد.

۴. نیازها را درسطح بالا جمع‌آوری کنید، سبک‌وزن و دیداری.

توسعه چابک برای نرم‌افزار در حال کار بیش از مستندات جامع ارزش قائل می‌شود. به هر حال، هدف اصلی این است که نرم‌افزار در حال کار تحویل داده شود، نه مستندات تحلیل و طراحی. برای دست‌یابی به این هدف، روش‌های چابک نیازها را تنها تا حد لازم در قالب داستان‌های کاربر، ویژگی‌ها، یا مورد کاربردها که بر روی کارت‌های داستان با اندازه کوچک نوشته شده‌اند، کسب می‌کنند. برای به تصویر کشیدن این نیازها، از تابلوهای داستان یا دنباله‌ای از عکس‌های صفحه، نقشه‌ها، یا روش‌های دیگر دیداری استفاده می‌شود تا نحوه تعامل کاربر با سیستم نشان داده شود. به این دلیل که به اشتراک‌گذاری و کار با کارت‌های داستان یا تابلوهای داستان ساده است، این روش‌ها از مستندسازی سنگین جلوگیری نموده و تغییر و کم و زیاد کردن نیازها را تسهیل می‌کنند.

۵. نسخه‌های کوچک و افزایشی بسازید و این کار را تکرار کنید.

پروژه‌های چابک سیستم را در افزایش‌های کوچک ساخته و مستقر می‌کنند تا امکان تحویل مورد کاربردها، داستان‌های کاربر،و یا خصوصیات به‌صورت تکراری فراهم آید. بدین ترتیب مزایای متعددی برای فرایندهای چابک می‌توان نام برد: پیشرفت پروژه مشهود است، در هر زمان کاربر فقط باید تعداد کمی ویژگی جدید را یاد بگیرد، ارائه بازخورد برای کاربران آسان‌تر می‌شود، و در نهایت این‌که افزایش‌های کوتاه ریسک شکست پروژه را کاهش می‌دهد.

۶. بر تحویل مداوم محصولات نرم‌افزاری تاکید داشته باشید.

قبل از معرفی توسعه‌ی چابک نیز روش‌های تکراری مانند فرایند حلزونی و فرایند یکنواخت وجود داشته‌اند. تفاوت فرایندهای چابک با پیشینیانش در تحویل مداوم سیستم نرم‌افزاری در افزایش‌های کوچک است. فرایندهای چابک پیشنهادهای مختلفی برای طول مدت هر تکرار ارائه می‌کنند، که از یک روز تا سه ماه متغیر است. برای مثال، روش توسعه سیستم پویا[۱] مدت دو تا شش هفته، و برنامه‌نویسی افراطی[۲] یک تا چهار هفته را پیشنهاد می‌دهند. در روش اسکرام مدت هر کدام از تکرارها که دوی سرعت[۳] نامیده می‌شود، معمولاً ۳۰ روز در نظر گرفته می‌شود. طول هر تکرار از متدولوژی معرفی‌شده در این کتاب می‌تواند بین دو هفته تا سه ماه باشد.

۷. قبل از رفتن به سراغ خصیصه[۴] بعدی، خصیصه‌ی جاری را کامل کنید.

منظور از این اصل این است که قبل از آن‌که به سراغ خصیصه‌ی بعدی برویم، هر خصیصه باید ۱۰۰% پیاده‌سازی شده و کاملاً آزمایش شود. در اینجا چالش این است که چگونه می‌توان فهمید که یک خصیصه کاملاً آزمایش شده است؟ روش توسعه آزمون-رانده[۵] و معیارهای پوشش آزمون برای این مشکل راه‌حلی دارند. در TDD باید آزمون هر خصیصه قبل از پیاده‌سازی آن، نوشته شده باشد. معیارهای پوشش آزمون نیز، نیازهای پوششی را که آزمون باید برآورده سازد تعریف می‌کنند. به‌عنوان‌مثال، بسیاری از شرکت‌ها معیار پوشش شاخه‌ی[۶] ۱۰۰% را استفاده می‌کنند، که می‌گوید هر شاخه از هر دستور شرطی در کد منبع برنامه باید حداقل یکبار آزمایش شود.

۸. قانون ۸۰-۲۰ را به‌کار ببرید.

این قانون همچنین به شکل دیگری نیز بیان می‌شود: «به‌اندازه‌ی کافی خوب، کافی است»[۷]. قانون فوق بر این باور استوار است که ۸۰% از کار یا نتیجه تنها توسط ۲۰% از قابلیت‌های سیستم تولید می‌شود. بنابراین، ما باید اولویت را به آن ۲۰% از نیازها بدهیم که ۸۰% نتیجه را تولید می‌کنند. این اصل به تیم توسعه توصیه می‌کند که مشتری و

---

[۱] Dynamic System Development Method (DSDM)
[۲] Extreme Programming (XP)
[۳] Sprint
[۴] Feature
[۵] Test Driven Development (TDD)
[۶] Branch Coverage
[۷] Good Enough is Enough

کاربران را طوری راهنمایی کنند که بتوانند نیازها را شناسایی و اولویت‌بندی کنند. این قانون همچنین به اعضای تیم موضوع کاهش بازدهی به دلیل کارهای اضافی را گوشزد می‌کند. این موارد شامل خصیصه‌هایی می‌شود که داشتن آن‌ها خوب است، و یا هدفشان بهبود کارایی است که در واقع نیاز هم نبوده است. و سایر موارد این‌چنینی. به‌عنوان مثال، ارزش ندارد که تلاش پیاده‌سازی مضاعفی را برای یک الگوریتم بهینه صرف کنیم، درحالی‌که یک الگوریتم ساده‌تر وجود دارد که برای پردازش داده‌ها به حد کافی سریع هست.

۹. آزمون در تمام چرخه‌ی حیات پروژه یکپارچه شده است، هرچه زودتر و مرتب آزمون را انجام دهید.

این اصل و اصول ۵ الی ۷ یکدیگر را کامل می‌کنند. بدین معنی که آزمون یک جزء جدایی‌ناپذیر از تحویل مداوم افزایش‌های کوچکی از سیستم که کاملاً پیاده‌سازی شده‌اند می‌باشد. ابزارهای آزمون مختلفی برای پشتیبانی این اصل وجود دارند، مانند Junit که یک ابزار آزمون بازگشت[۱] و آزمون واحد برای کلاس‌های جاوا[۲] می‌باشد. برای استفاده از چنین ابزاری، برنامه‌نویس باید مشخص کند که یک خصیصه‌ی تحت آزمون چگونه باید فراخوانی شود و نتایج آزمون چگونه باید ارزیابی شوند. سپس ابزار به‌طور خودکار آزمون‌ها را تولید، اجرا، و آن‌گاه نتایج آزمون را بررسی می‌کند. لذا آزمون‌ها را می‌توان هر چند بار که نیاز باشد اجرا نمود.

۱۰. یک روش همکارانه و همدستانه بین تمام ذی‌نفعان ضروری است.

روش‌های سنتی برای تبادل و ارتباط نیازمندی‌ها بین اعضای تیم بر مستندات جامع تکیه می‌کنند. درپروژه‌های چابک، نیازمندی‌ها به‌صورت سبک‌وزن و در سطح بالا جمع‌آوری می‌شوند. بنابراین، همکوشی و همکاری[۳] بین تیم توسعه و نمایندگان مشتری و کاربران ضروری می‌باشد. طرفین باید یکدیگر را درک کرده و در طول چرخه‌ی حیات با هم همکاری کنند تا نیازها را شناسایی و تکمیل نمایند. به دلیل این‌که سیستم جدید ممکن است عادات کاری و نیز کارایی کاربران را به طرزقابل‌توجهی تغییر داده یا تحت تاثیر قرار دهد همکاری و همدستی بین تیم و کاربران برای موفقیت پروژه ضروری است.

۲-۶ متدولوژی توسعه‌ی نرم‌افزار

توسعه‌ی نرم‌افزار نه تنها به یک فرایند، بلکه به یک متدولوژی یا روش توسعه نیز نیاز دارد. متاسفانه، واژه‌ی «متدولوژی» اغلب بدون تعریف رها می‌شود و این منجر به قدری ابهام می‌شود. به‌عنوان مثال، متدولوژی اغلب با فرایند اشتباه گرفته می‌شود. در مهندسی نرم‌افزار، فرایند و متدولوژی مفاهیم مهمی هستند. هر دو مفهوم به هم مربوط می‌شوند ولی یکی نیستند. در ادامه تعریفی از متدولوژی نرم‌افزار ارائه شده است.

تعریف ۲-۲

متدولوژی نرم‌افزار گام‌ها یا چگونگی انجام فعالیت‌های یک فرایند نرم‌افزار را تعریف می‌کند.

فرایند در حالت کلی فقط فعالیت‌ها و چگونگی ارتباط آن‌ها به یکدیگر را مشخص می‌کند، ولی چگونگی انجام این فعالیت‌ها را تعیین نمی‌کند. در واقع این در اختیار سازمان توسعه نرم‌افزار گذاشته می‌شود تا یک متدولوژی که مناسب سازمان

---

[۱] Regression
[۲] Java
[۳] Cooperation

است را برای خودش انتخاب و یا حتی تدوین نماید. تعریف فوق بدین معنی است که هر متدولوژی پیاده‌سازی یک فرایند است. توسعه‌ی نرم‌افزار نیازمند یک فرایند و یک متدولوژی است.

۲-۶-۱- تفاوت بین فرایند و متدولوژی

شکل ۲-۱۱ تفاوت‌های میان فرایند و متدولوژی را به‌صورت فهرست‌وار نشان می‌دهد. با وجودی که یک فرایند نرم‌افزاری فعالیت‌های مرحله به مرحله یا آن‌چه که باید در هر مرحله انجام شود را تعریف می‌کند، ولی از چگونگی انجام فعالیت‌ها چیزی نمی‌گوید. در عوض، متدولوژی نرم‌افزار گام‌های دقیق یا چگونگی انجام فعالیت‌های یک فرایند را تعریف می‌کند. فرایند نرم‌افزار ورودی و خروجی هر مرحله را مشخص می‌کند،ولی راجع به نحوه‌ی بازنمایش[۱] ورودی و خروجی چیزی را اجبار نمی‌کند. متدولوژی گام‌ها، معیارهای ورود و خروج از هر گام، و روابط بین گام‌ها را تعریف می‌کند. همچنین متدولوژی برای هر گام، رویه‌ها و روش‌ها، اصول و راهنمایی‌ها، ورودی و خروجی گام، و نحوه‌ی نمایش ورودی و خروجی را تعیین می‌کند. نحوه‌ی نمایش دست‌ساخته‌ها توسط متدولوژی بسته به نگرش[۲] یا نوع نگاه متدولوژی به جهان دارد. به‌عنوان‌مثال، نگرش شیءگرا (OO) جهان و سیستم‌ها را به‌صورت اشیا در حال تعامل باهم نگاه می‌کند. بنابراین، برای مدل‌سازی اشیا، نمودارهای تحلیل و طراحی شیءگرا ایجاد می‌شوند. با این تفسیر متدولوژی‌ها وابسته به نگرش هستند. یک متدولوژی نرم‌افزاری را می‌توان به‌عنوان یک پیاده‌سازی عینی[۳] از یک فرایند نرم‌افزاری دانست. پس معلوم می‌شود که می‌توان بیش از یک متدولوژی نرم‌افزاری را به‌عنوان پیاده‌سازی یک فرایند نرم‌افزاری داشت.



شکل ۲-۱۱: تفاوت متدولوژی و فرایند نرم‌افزاری

---

[۱] Representation
[۲] Paradigm
[۳] Concrete

۲-۶-۲- فواید متدولوژی

استفاده از یک متدولوژی خوب توسعه‌ی نرم‌افزار فواید زیادی دربردارد، از جمله:

۱. یک متدولوژی خوب تیم توسعه‌دهنده را قادر می‌سازد بر روی وظایف مهم تمرکز کند، و بداند چگونه این فعالیت‌ها را به درستی انجام دهد تا سیستم نرم‌افزاری مطلوب را تولید کند.
۲. یک متدولوژی خوب ارتباط و همکاری را بهبود می‌بخشد، زیرا:
   - متدولوژی زبان مشترکی برای مدل‌سازی، تحلیل، و طراحی تعریف می‌کند.
   - متدولوژی گام‌هایی را برای انجام موثر یک کار توسعه تعریف می‌کند که هر کس آن گام‌ها را می‌داند و دنبال می‌کند.
۳. یک متدولوژی خوب کیفیت طراحی و بهره‌وری نرم‌افزاری را بهبود می‌بخشد، زیرا:
   - به مهندسان نرم‌افزار قدرت می‌دهد که برای ساخت سیستم، مفاهیم و ابزارهای مدل‌سازی، تحلیل و طراحی را به درستی و به‌طور موثر به‌کار گیرند.
   - راهنماهای بازبینی اشتراکی یا فهرست‌های بررسی[۱] به مهندسان نرم‌افزار امکان بازرسی موثر و بازبینی‌های مشترک را می‌دهند که هدفشان شناسایی کاستی‌ها در تبیین نیازمندی‌ها، طراحی، یا کد منبع است. در عوض، این‌ها نیز باعث کاهش هزینه‌های آزمون، اشکال‌زدایی و نگهداری می‌شوند.
۴. یک متدولوژی خوب مبنایی برای بهبود فرایند به حساب می‌آید زیرا اندازه‌گیری کیفیت، بهره‌وری، هزینه و زمان تحویل نرم‌افزار را می‌توان تعریف و اعمال نمود تا ضعف‌ها و قدرت‌ها شناسایی شوند.
۵. یک متدولوژی خوب مبنایی برای خودکارسازی فرایند می‌سازد زیرا بسیاری از گام‌های متدولوژیکی را می‌توان به‌طور مکانیکی (خودکار) انجام داد، و خودکارسازی نرم‌افزار را بسیار آسان تر نمود.
۶. یک متدولوژی که فراگیری و استفاده از آن آسان باشد به مبتدیان نیز امکان تولید نرم‌افزار کیفی را می‌دهد. وقتی فرد مبتدی با متدولوژی آشنا شد، می‌تواند برخی گام‌ها را در توسعه‌ی چابک صرف‌نظر کند.

متدولوژی‌های توسعه‌ی نرم‌افزار در طی چند دهه‌ی اخیر تکامل‌یافته‌اند. در بخش‌های بعدی برخی از متدولوژی‌های شاخص که در این مدت معرفی‌شده‌اند را مرور می‌کنیم.

۲-۶-۳- متدولوژی‌های ساختمند

متدولوژی‌های تحلیل ساختمند و طراحی ساختمند[۲] دردهه‌ی ۱۹۷۰ میلادی معرفی و در دهه‌ی ۱۹۸۰ میلادی به بلوغ رسیدند و تا امروز نیز استفاده می‌شوند. در تحلیل ساختمند برای مدل‌سازی فرایندهای کسب‌وکار مربوط به کاربردهای دنیای واقعی از نمودار جریان داده[۳] استفاده می‌شود. DFD دراصل یک گراف جهت‌دار است که در آن گره‌ها نمایانگر موجودیت‌های خارجی، فرایندهای کسب‌وکار، و انبارهای داده هستند، درحالی‌که یال‌های جهت‌دار نشان‌دهنده‌ی جریان داده بین گره‌ها می‌باشند. در حین تحلیل ساختمند از روش تقسیم و حل[۴] استفاده می‌شود تا فرایندهای کسب‌وکاری پیچیده تجزیه‌شده و به نمودارهای جریان داده سطح پایین‌تر تبدیل شوند.

---

[۱] Checklist
[۲] Structured Analysis/Structured Design (SA/SD)
[۳] Data Flow Diagram (DFD)
[۴] Divide-and-Conquer

گام‌های تحلیل ساختمند با ساخت یک DFD سطح بالا شروع می‌شود، که به آن نمودار زمینه[۱] می‌گویند. در این نمودار، سیستم به‌عنوان تنها فرایند موجود ترسیم می‌شود که با موجودیت‌های خارجی و انبارداده‌های خارجی تعامل دارد. درگام‌های بعدی، به‌طور تکراری فرایندهای پیچیده به فرایندهای ساده‌تر تجزیه می‌شوند و این کار تا جایی ادامه پیدا می‌کند که فرایندهای در سطح برگ را بتوان به سادگی پیاده‌سازی نمود. ورودی، خروجی وداده‌ساختارها، و نیز الگوریتم این فرایندها مشخص می‌شوند. DFD هم برای توصیف فرایندهای کسب‌وکار موجود مناسب است و هم برای فرایندهای تجاری پیشنهادی، ولی برای نشان دادن روابط فراخوانی بین پیمانه‌های نرم‌افزاری مناسب نیست. دلیل این امر این است که روابط بین فرایندهای یک DFD در واقع روابط جریان داده هستند، درحالی‌که روابط بین پیمانه‌های نرم‌افزاری روابط جریان کنترل‌اند. طراحی ساختمند این شکاف را پر می‌کند.

گام‌های طراحی ساختمند مدل تحلیل نمودار جریان داده را به یک نمودار ساختاری یا نمودار برنامه تبدیل می‌کند که در آن گره‌ها زیربرنامه‌ها[۲] و یال‌ها فراخوانی تابع از زیربرنامه‌های سطح بالا به زیردستان خود را نشان می‌دهند.

۲-۶-۴- متدولوژی‌های کلاسیک شیءگرا

قبل از UML، متدولوژی‌های OO کلاسیک وجود داشتند که سه‌تای آن‌ها بسیار شناخته‌شده بودند: روش Booch، روش مدل‌سازی شیء، و روش مورد کاربرد-رانده. این سه متدولوژی پایه‌ی UML 1.0 را بنا نهادند. متدولوژی‌های OO کلاسیک توسط تعداد زیادی از سازمان‌های توسعه نرم‌افزار استفاده می‌شدند و در شکوفا شدن نگرش OO سهم موثری داشتند. ولی صنعت نرم‌افزار به زودی دریافت که یکپارچه‌سازی و نگهداری سیستم‌هایی که با استفاده از متدولوژی‌های مختلف توسعه‌یافته باشند دردسر بزرگی است. همچنین پشتیبانی از ابزارهایی که متدولوژی‌های مختلف را استفاده می‌کنند نیز بسیار هزینه‌بر بود. این مسائل لزوم تعریف یک روش مدل‌سازی یکنواخت را آشکار نمود و منجر به تعریف زبان مدل‌سازی یکنواخت و فرایند یکنواخت شد.

۲-۷ روش‌های چابک

مانند مدل تکاملی نمونه‌سازی و مدل حلزونی، تمام روش‌های چابک فرایند توسعه تکراری، افزایشی را اتخاذ می‌کنند، البته، تمام روش‌های چابک از بیانیه‌ی چابکی که در بخش ۲-۵-۷ معرفی گردید نیز پیروی می‌کنند. فرایندهای چابک بر تکرارهای کوتاه و تحویل مداوم افزایش‌های کوچک تاکید دارند. گرچه روش‌های چابک از لحاظ نام‌گذاری مراحل و جزییات هر مرحله با یکدیگر تفاوت‌هایی دارند، ولی تمام آن‌ها کم یا بیش فعالیت‌های نیازمندی‌ها، طراحی، پیاده‌سازی، یکپارچه‌سازی، آزمون، و استقرار را در هر تکرار پوشش می‌دهند. اما، تاکید این روش‌ها نسبت به فرایندهای سنتی متفاوت است. به‌عنوان مثال، فرایندهای چابک به نرم‌افزار در حال کار بیش‌تر از مستندات جامع بها می‌دهند. این به معنای انجام مدل‌سازی فقط تا حد لزوم، در مراحل نیازمندی‌ها و طراحی می‌باشد. در این بخش چند تا از روش‌های چابک که بیش‌تر از بقیه استفاده می‌شوند معرفی خواهند شد. شکل ۲-۱۲ خلاصه‌ی مختصری از برخی روش‌های چابک، که در بخش‌های بعدی با جزییات بیش‌تر توصیف شده‌اند، را ارائه می‌دهد. هر کدام از این روش‌ها یک فهرست طولانی از اصول، خصیصه‌ها، ارزش‌ها، و به-روش‌ها دارند. به‌جای نشان دادن همه‌ی این موارد، در شکل ۲-۱۲ فقط سه تا از منحصر به فردترین خصیصه‌های هر روش چابک فهرست شده‌اند.

---

[۱] Context Diagram

[۲] Subroutin

AUM*	DSDM	SCRUM	FDD	XP
● یک کتاب آشپزی برای کار تیمی که از UML استفاده می‌کنند ● برای مبتدیان و برنامه‌سازان فصلی ● مناسب برای روش‌های چابک و برنامه‌رانده و بروژه‌های تیمی کوچک یا بزرگ	● چارچوبی که با RUP و XP کار می‌کند ● مبتنی بر اصل ۲۰-۸۰ ● مناسب برای پروژه‌های چابک یا برنامه‌رانده	● شامل نقش‌هایی چون رئیس اسکرام، مالک محصول، و تیم ● جلسات گزارش وضعیت روزانه‌ی ۱۵ دقیقه‌ای برای بهبود ارتباط ● نگاه به گذشته توسط تیم به منظور بهبود فرآیند.	● خصیصه‌رانده و مدل‌رانده ● مدیریت پیکربندی، بازبینی و بازرسی، و ساخت‌های منظم ● مناسب برای پروژه‌های چابک یا برنامه‌رانده	● هر فرد مجاز است در هر مکان و در هر زمان هر کدی را تغییر دهد. ● یکپارچه‌سازی و ساخت چند بار در روز و هر وقت که یک وظیفه کامل شده انجام شود. ● در هفته بیش از ۴۰ ساعت کار ممنوع است.
پیش‌برنامه‌ریزی ۱) نیازمندی‌ها را جمع‌آوری و اولویت‌بندی کنید ۲) مورد کاربردها را بدست آورید ۳) مورد کاربردها را به افزایش‌ها منسوب کنید ۴) طراحی مفهومی انجام دهید **در هر تکرار:** ۱) پذیرفتن تغییرات ۲) مدل‌سازی دامنه ۳) مدل‌سازی تعامل سیستم و کنشگر ۴) مدل‌سازی رفتاری ۵) نمودار کلاس طراحی ۶) توسعه‌ی آزمون‌رانده / برنامه‌نویسی دوتایی ۷) آزمون یکپارچه‌سازی ۸) استقرار	**مطالعه‌امکان‌سنجی** ۱) سنجش تناسب DSDM برای پروژه ۲) شناسایی ریسک‌ها **مطالعه کسب‌وکار** ۱) تولید نیازمندی‌های اولویت‌بندی شده ۲) طراحی معماری ۳) حل‌وفصل ریسک **مدل عملکردی تکرار** ۱) شناسایی عملکردهای پیش‌نمونه ۲) ساخت، بازبینی، و تایید پیش‌نمونه **طراحی و ساخت تکرار** ۱) ساخت سیستم ۲) انجام آزمون بتا **پیاده‌سازی** ۱) استقرار سیستم ۲) ارزیابی تاثیر برروی کسب‌وکار	**جلسه برنامه‌ریزی هر نسخه** ۱) شناسایی و اولویت‌بندی نیازمندی‌ها (پشته‌ی محصول) ۲) شناسایی نیازمندی‌های با اولویت بالا در هر افزایش که دوسرعت نامیده می‌شود. ۳) شناسایی فعالیت‌های توسعه‌ی sprint **تکرار Sprint** ۱) جلسه‌ی برنامه‌ریزی sprint برای تعیین آنچه باید در مرحله بعد ساخته شود و چگونگی ساخت آن ۲) جلسه اسکرام روزانه برای اعضای تیم تا گزارش وضعیت دهند **جلسه بازبینی sprint** ۱) نمایش افزایش تهیه شده ۲) نگاه به گذشته توسط تیم **استقرار**	**توسعه‌ی مدل کلی** ۱) مرور و کار با سیستم ۲) توسعه‌ی مدل‌های گروهی کوچک ۳) استخراج مدل کلی ۴) تولید توصیف مدل **ساخت فهرست خصیصه‌ها** ۱) شناسایی فعالیت‌های کسب‌وکاری که باید خودکارسازی شوند ۲) شناسایی خصیصه‌های فعالیت‌های کسب‌وکار **برنامه‌ریزی برحسب خصیصه** ۱) زمان‌بندی توسعه‌ی فعالیت‌های کسب‌وکار ۲) واگذاری فعالیت‌های کسب‌وکاری به برنامه‌سازان ارشد ۳) واگذاری کلاس‌ها به اعضای تیم **طراحی برحسب خصیصه** ۱) تولید نمودارهای توالی برای نمایش تعامل اشیا با استفاده از خصیصه‌ها ۲) محفظه‌سازی خصیصه‌ها برای تشکیل کلاس‌هایی که باید پیاده‌سازی شوند **ساخت برحسب خصیصه** ۱) پیاده‌سازی کلاس‌ها **استقرار**	**اکتشاف** ۱) جمع‌آوری اطلاعات درباره‌ی کاربرد ۲) انجام مطالعه امکان‌سنجی **برنامه‌ریزی** ۱) تعیین داستان‌ها برای نسخه‌ی بعدی ۲) برنامه‌ریزی برای نسخه‌ی بعدی **تکرار تا اولین نسخه** ۱) تعریف/تغییر معماری ۲) انتخاب و پیاده‌سازی داستان‌های هر تکرار ۳) اجرای آزمون‌های وظیفه‌مندی توسط مشتری **تولید** ۱) ارزیابی و بهبود کارایی سیستم ۲) تایید و آزمون سیستم برای تولید و استفاده **نگهداشت** ۱) بهبود نسخه‌ی جاری ۲) تکرار فرایند با هر نسخه جدید **مرگ** ۱) تولید مستندات سیستم اگر پروژه انجام شده است، یا ۲) جایگزینی سیستم اگر نگهداشت خیلی هزینه‌بر است

شکل ۲-۱۲: خلاصه‌ای از چند روش چابک

۲-۷-۱- روش توسعه‌ی پویای سیستم‌ها

روش DSDM در اوایل دهه‌ی ۱۹۹۰ به‌عنوان جایگزینی برای روش توسعه کاربرد سریع[۱] در انگلستان ظهور کرد. این روش در واقع یک چارچوب فرایند است که پروژه‌های مختلف می‌توانند آن را انتخاب و برای انجام توسعه‌ی کاربرد سریع تطبیق دهند. برخی مولفین این روش را به عنوان مناسب‌ترین روش برای کاربردهای خدمات مالی معرفی کرده‌اند. DSDM یک فرایند تکراری، افزایشی است که توسط مجموعه‌ای از اصول DSDM، مشابه ۱۰ اصل چابک معرفی‌شده در بخش ۲-۵-۷ هدایت می‌شوند. همان‌گونه که در شکل ۲-۱۳ نشان داده شده است، فرایند DSDM شامل پنج مرحله می‌شود. دو مرحله‌ی اول تنها یک‌بار انجام می‌شوند، درحالی که سه مرحله‌ی دیگر تکراری هستند:

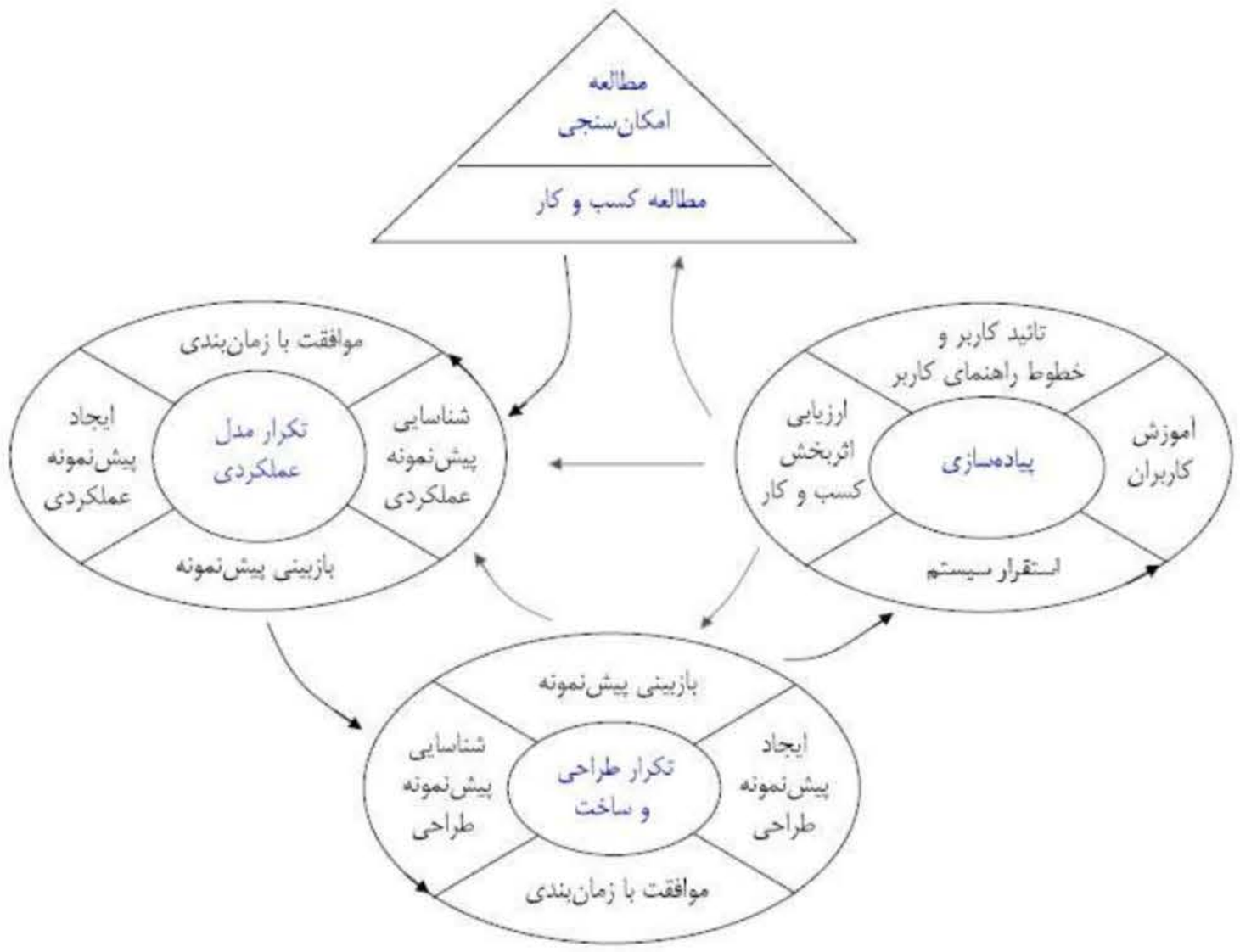
۱. مطالعه‌ی امکان‌سنجی. در این مرحله، قابلیت به‌کارگیری DSDM و امکان‌سنجی فنی پروژه مشخص می‌شود. محصولات نهایی این مرحله شامل یک گزارش امکان‌سنجی، یک برنامه کلی پروژه، و احتمالاً یک پیش‌نمونه که برای ارزیابی امکان‌سنجی پروژه ساخته‌ایم، می‌باشند. احتمال دارد که پیش نمونه تکامل‌یافته و به سیستم نهایی تبدیل شود.
۲. مطالعه‌ی کسب‌وکار. در این مرحله نیازمندی‌ها شناسایی و اولویت‌بندی می‌شوند، و معماری اولیه‌ای از سیستم ترسیم می‌شود. محصولات نهایی این مرحله عبارتند از: تعریف محدوده کسب‌وکار، تعریف معماری سیستم، و برنامه کلی پیش نمونه‌سازی.
۳. تکرار مدل کارکردی. در این مرحله، یک پیش نمونه‌ی کارکردی به‌صورت تکراری و افزایشی ساخته می‌شود. محصول نهایی این مرحله شامل یک مدل کارکردی می‌شود که حاوی کد پیش نمونه و مدل‌های تحلیل، فهرستی از کارکردهای اولویت‌بندی شده، مستندات بازبینی پیش‌نمونه کارکردی، فهرستی از نیازمندی‌های غیرکارکردی، و تحلیل ریسک برای توسعه‌ی بیش‌تر می‌باشد. مستندات بازبینی پیش نمونه بازخوردهای کاربر را که باید در افزایش‌های بعدی مورد توجه قرار گیرند مشخص می‌کند. پیش‌نمونه کارکردی تکامل‌یافته و به سیستم نهایی تبدیل می‌شود.
۴. تکرار طراحی و ساخت. در این مرحله، سیستم طراحی و پیاده‌سازی می‌شود تا نیازمندی‌های کارکردی و غیرکارکردی را برآورده نماید، و سپس توسط کاربران آزمایش می‌شود. بازخوردهای کاربران مستند شده و در توسعه‌های آتی سیستم مدنظر قرار می‌گیرند.
۵. پیاده‌سازی. در این مرحله، سیستم در محیط مورد نظر (محیط مقصد) نصب‌شده و آموزش کاربر انجام می‌شود. محصولات نهایی شامل راهنمای کاربر و گزارش بازبینی پروژه می‌شود، که نتایج پروژه و آنچه که باید در آینده انجام شود را خلاصه می‌کند.

۲-۷-۲- اسکرام[۲]

اسکرام چارچوبی است که به سازمان‌ها اجازه می‌دهد روش‌های توسعه‌ی نرم‌افزار خودشان را به‌کار بسته و بهبود بخشند. اسکرام متشکل از تیم‌های اسکرام، نقش‌های درون تیم، جعبه‌های زمانی، دست‌ساخته‌ها، و قواعد اسکرام می‌شود. اسکرام روشی تکراری، افزایشی است که هدفش بهینه کردن پیش‌بینی‌پذیری و کنترل ریسک است. همان‌گونه که در شکل ۲-۱۴ نشان داده شده است، در این روش یک جلسه‌ی برنامه‌ریزی نسخه وجود دارد که پشته‌ی محصول[۳]، اولویت نیازمندی‌ها، و نیز برنامه‌ریزی برای تکرارها، که به آن‌ها دوی سرعت یا اسپرینت[۴] می‌گویند، را معین می‌کند. در طی مرحله‌ی تکرار اسپرینت، تیم فعالیت‌های توسعه را انجام می‌دهد تا افزایش‌هایی از محصول را توسعه و تحویل نماید. هر اسپرینت با یک جلسه‌ی برنامه‌ریزی اسپرینت شروع می‌شود که در آن جلسه تیم و مالک محصول مشخص می‌کنند که چه اقلامی از پشته‌ی محصول قرار

---

[۱] Rapid Application Development (RAD)
[۲] Scrum
[۳] Product Backlog
[۴] Sprint



شکل ۲-۱۳: فرایند روش توسعه‌ی پویای سیستم

است در قدم بعدی تحویل شوند و چگونه باید توسعه یابند. معمولاً هر اسپرینت ۳۰ روز طول می‌کشد، ولی می‌تواند کوتاه‌تر یا طولانی‌تر هم باشد. یکی از ویژگی‌های متمایز روش اسکرام، جلسات روزانه‌ی ۱۵ دقیقه‌ای است. این جلسات به اعضای تیم امکان تبادل وضعیت پیشرفت و بهبود درک متقابل از پروژه را می‌دهد. ویژگی متمایز دیگر روش اسکرام پس‌نگری اعضای تیم در انتهای هر اسپرینت اسکرام است. این جلسه به تیم امکان می‌دهد روش‌هایش را بهبود بخشد.

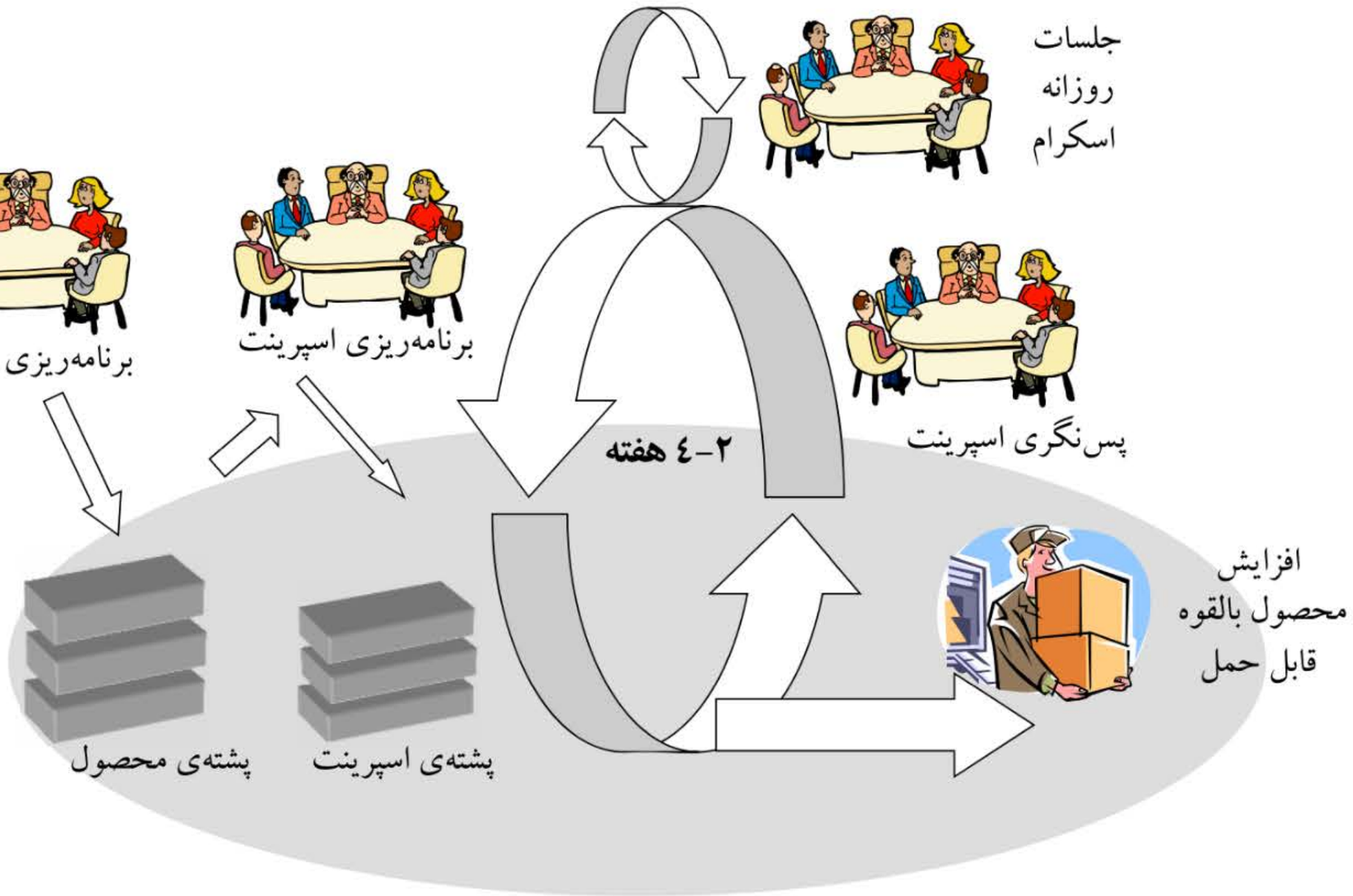
۲-۷-۳- توسعه‌ی خصیصه-رانده

همان‌گونه که در شکل ۲-۱۲ نشان داده شده است، روش توسعه‌ی خصیصه-رانده شامل شش گام یا مرحله می‌شود. سه مرحله‌ی اول فقط یک‌بار و سه مرحله‌ی آخر به‌صورت تکراری انجام می‌گیرند. طرفداران روش FDD می‌گویند این روش برای توسعه‌ی سیستم‌های بحرانی[۱] مناسب تر است. شش مرحله‌ی FDD به اختصار در زیر معرفی شده‌اند.

۱. توسعه‌ی مدل کلی[۲]. در این مرحله، یک فرد خبره‌ی دامنه، گذری بر کل سیستم می‌کند، که ممکن است شامل تجزیه سیستم به اجزاء و زیرسیستم‌ها باشد. مرور مضاعفی بر زیرسیستم‌ها یا اجزاء نیز ممکن است توسط خبرگان همان دامنه‌ها انجام شود. بر اساس این مرورها، گروه‌های کوچکی از توسعه‌دهندگان مدل‌های شئ را برای دامنه‌های مربوطه تهیه می‌کنند. تیم‌های توسعه با یکدیگر همکاری نموده و یک مدل کلی از سیستم تهیه می‌نمایند.
۲. ساخت یک فهرست خصیصه[۳]. در این مرحله، تیم یک فهرست خصیصه تولید می‌کند که بیانگر عملیات کسب‌وکاری است که باید توسط سیستم ارائه شوند. خصیصه‌های درون این فهرست را می‌توان توسط اعمال یا خصیصه‌های سطح پایین‌تر پالایش نمود. این فهرست با همکاری کاربران و سرمایه‌گذاران مورد بازبینی قرار می‌گیرد.

---

[۱] Mission Critical
[۲] Develop Overall Model
[۳] Build by feature



شکل ۲-۱۴: فعالیت‌های توسعه‌ی اسکرام

۳. برنامه‌ریزی بر حسب خصیصه[۱]. در این مرحله، تیم یک برنامه کلان تولید می‌کند تا توسعه و استقرار افزایشی خصیصه‌ها را برحسب اولویت‌ها و وابستگی‌هایشان هدایت نماید. خصیصه‌ها به برنامه‌سازان ارشد واگذار می‌شود. برنامه‌ساز ارشد تصمیم‌گیرنده اصلی در تیم است. این سازمان‌دهی تیمی با عنوان سازمان تیمی برنامه‌ساز ارشد شناخته می‌شود. کلاس‌هایی که در مدل کلی مشخص‌شده‌اند به برنامه‌سازان (توسعه‌دهندگان) واگذار می‌شوند که به ایشان مالکان کلاس[۲] گفته می‌شود. زمان‌بندی پروژه نیز که شامل نقاط عطف می‌باشد تهیه می‌گردد.
۴. طراحی بر حسب خصیصه، ساخت بر حسب خصیصه، و استقرار.[۳]

این سه مرحله، مراحلی تکراری هستند که در طی آن‌ها افزایش‌ها طراحی، پیاده‌سازی، بازبینی، آزمایش و مستقر می‌شوند. ممکن است چند تیم مختلف به‌طور همزمان روی مجموعه‌های متفاوتی از خصیصه‌ها کار کنند. هر افزایش بین چند روز تا چند هفته به طول می‌انجامد.

نقش‌ها و مسئولیت‌ها در یک پروژه FDD مشابه عناوین شغلی متداول می‌باشند. این عناوین شامل مدیر پروژه، معمار ارشد، مدیر توسعه، برنامه‌ساز ارشد، مالک کلاس، خبره‌ی دامنه، مدیر نسخه، کارشناس ابزار[۴] سرپرست سیستم[۵]، آزمون‌گر، و نویسنده‌ی فنی می‌گردند.

---

[۱] Plan by feature
[۲] Class Owner
[۳] Deployment
[۴] Toolsmith
[۵] Admin

۲-۷-۴- برنامه‌نویسی افراطی

برنامه‌نویسی افراطی یا XP روشی چابک و مناسب تیم‌های کوچک است که با نیازمندی‌های مبهم و متغیر روبرو هستند. اصول محرکه‌ی XP در واقع اصول و روش‌های شناخته‌شده را تا حد نهایت پیش می‌برند [۲۱]. برای مثال، اگر ساختن مکرر محصول خوب است، در این صورت تیم‌ها باید هر روز به دفعات زیادی عمل ساخت را انجام دهند. فرایند XP شامل شش مرحله است:

۱. اکتشاف[۱]. در این مرحله، تیم توسعه به‌طور مشترک با مشتری، داستان‌های کاربر را برای سیستم تا حدی توسعه می‌دهند که مشتری متقاعد شود که مواد کافی برای ساختن یک نسخه‌ی خوب وجود دارد. یک داستان کاربر، خصیصه‌ای را که یک کاربر مشخص از سیستم انتظار دارد، مشخص می‌کند. برای مثال، «به‌عنوان یک عضو، من می‌خواهم بتوانم مستندات را از سیستم کتابخانه خارج کنم.» علاوه بر این‌ها، تیم توسعه فناوری‌های موجود را بررسی کرده و یک مطالعه‌ی امکان‌سنجی برای پروژه انجام می‌دهد. این مرحله نباید بیش از دو هفته طول بکشد.
۲. برنامه‌ریزی. در این مرحله، تیم توسعه و مشتری با همکاری یکدیگر داستان‌های مورد نظر برای نسخه بعدی را شناسایی می‌کنند، که شامل کوچک‌ترین و باارزش‌ترین داستان‌ها از نظر مشتری می‌باشد. داستان‌های انتخاب‌شده باید حدود شش ماه تلاش برای پیاده‌سازی نیاز داشته باشند. همچنین، یک برنامه نیز برای نسخه‌ی بعدی تهیه می‌شود. این مرحله نباید بیش از دو روز به‌طول انجامد.
۳. تکرارها تا نسخه‌ی اول[۲]. در این مرحله، معماری کلی سیستم تعریف می‌شود. مشتری داستان‌ها را انتخاب می‌کند. تیم آن داستان‌ها را پیاده‌سازی می‌نماید، و مشتری عملکرد سیستم را آزمایش می‌کند. این فعالیت‌ها به‌طور تکراری انجام می‌شوند تا این‌که نرم‌افزار مناسب برای تولیدو استفاده باشد. هر کدام از تکرارها بین یک تا چهار هفته طول می‌کشد.
۴. محصول سازی[۳]. در این مرحله، مواردی چون کارایی و قابلیت اعتماد که برای تولید و استفاده قابل‌قبول نیستند بررسی و مرتفع می‌گردند. سیستم، مورد آزمون قرار گرفته و برای تولید و استفاده، تاییدیه می‌گیرد. سیستم در محیط تولید نصب می‌گردد.
۵. نگهداری[۴]. آن طور که Beck [۲۱] می‌گوید، این مرحله در واقع وضعیت عادی یک پروژه‌ی XP است. در طی این مرحله، سیستم دست‌خوش تغییرات و بهبودهای مداوم از قبیل بازسازی، پذیرش فناوری‌های جدید، و بهبود در عملکرد با داستان‌های جدیدی از کاربر قرار می‌گیرد. این فرایند برای هر نسخه جدید از سیستم تکرار می‌شود.
۶. مرگ[۵]. سیستم در طول مرحله‌ی نگهداری تکامل می‌یابد تا این‌که به‌طور کامل نیازهای کسب‌وکار مشتری را ارضاء نماید و لذا داستان جدیدی توسط مشتری اضافه نشود. وقتی این اتفاق بیافتد، پروژه تمام شده و وارد مرحله‌ی مرگ می‌شود، که در آن مرحله تیم، مستندات سیستم را جهت آموزش، تعمیرات و مراجعات بعدی تولید می‌نماید. حالت دیگری نیز هست که پروژه وارد مرحله مرگ می‌شود، و آن وقتی است که نتواند برای پاسخ‌گویی به انتظارات مشتری موثر باشد.

۲-۷-۵- چابک یا برنامه-راننده[۶]

گرچه محبوبیت روش‌های چابک در صنعت نرم‌افزار در حال رشد است، ولی این بدان معنی نیست که این روش‌ها هیچ محدودیتی ندارند. در [۳۵]، Boehm و Turner اشاره می‌کنند که روش‌های چابک و راه‌کارهای برنامه-راننده «هر یک در زمینه‌ی خاصی، به‌وضوح بر دیگری برتری دارد.»

---

[۱] Exploration
[۲] Iterations to first release
[۳] Productionizing
[۴] Maintenance
[۵] Death
[۶] Plan-Driven

روش‌های چابک برای پروژه های کوچک تا متوسط که با تغییرات مداوم در نیازمندی‌ها مواجه هستند به خوبی کار می‌کنند. راه‌کارهای برنامه-رانده برای سیستم‌های بزرگ و پیچیده و سازندگان تجهیزات که قابلیت پیش‌بینی در آن‌ها مهم است، همچنان به‌عنوان انتخاب غالب شناخته می‌شوند. بنابراین، هر دو روش مورد نیازند. آن‌طور که Boehm وTurner [۳۵] می‌گویند، روش‌های چابک و برنامه-رانده «هر دو کمبودهایی دارند که اگر مورد توجه قرار نگیرند می‌تواند منجر به شکست پروژه بشود. چالش این است که توازنی بین هر دو روش برقرار شود که از قدرت هر دو بهره‌برداری کرده و ضعف‌هایشان را جبران کند.» به‌عبارت دیگر، ما نیاز به روش‌هایی داریم که آن‌ها را با شرایط و فرهنگ‌های سازمان‌ها و پروژه‌های مختلف توسعه‌ی نرم‌افزار تطبیق دهیم. از جمله این روش‌ها می‌توان به کریستال نارنجی [۵۰]، DSDM [۴۰]، FDD [۱۱۹]، توسعه‌ی لاغر[۱۲۴] ، فرایند یکنواخت سبک‌وزن[۱] [۱۰۴] و نیز متدولوژی ارائه شده در این کتاب اشاره نمود.

۲-۸ مروری بر فرایند و متدولوژی این کتاب

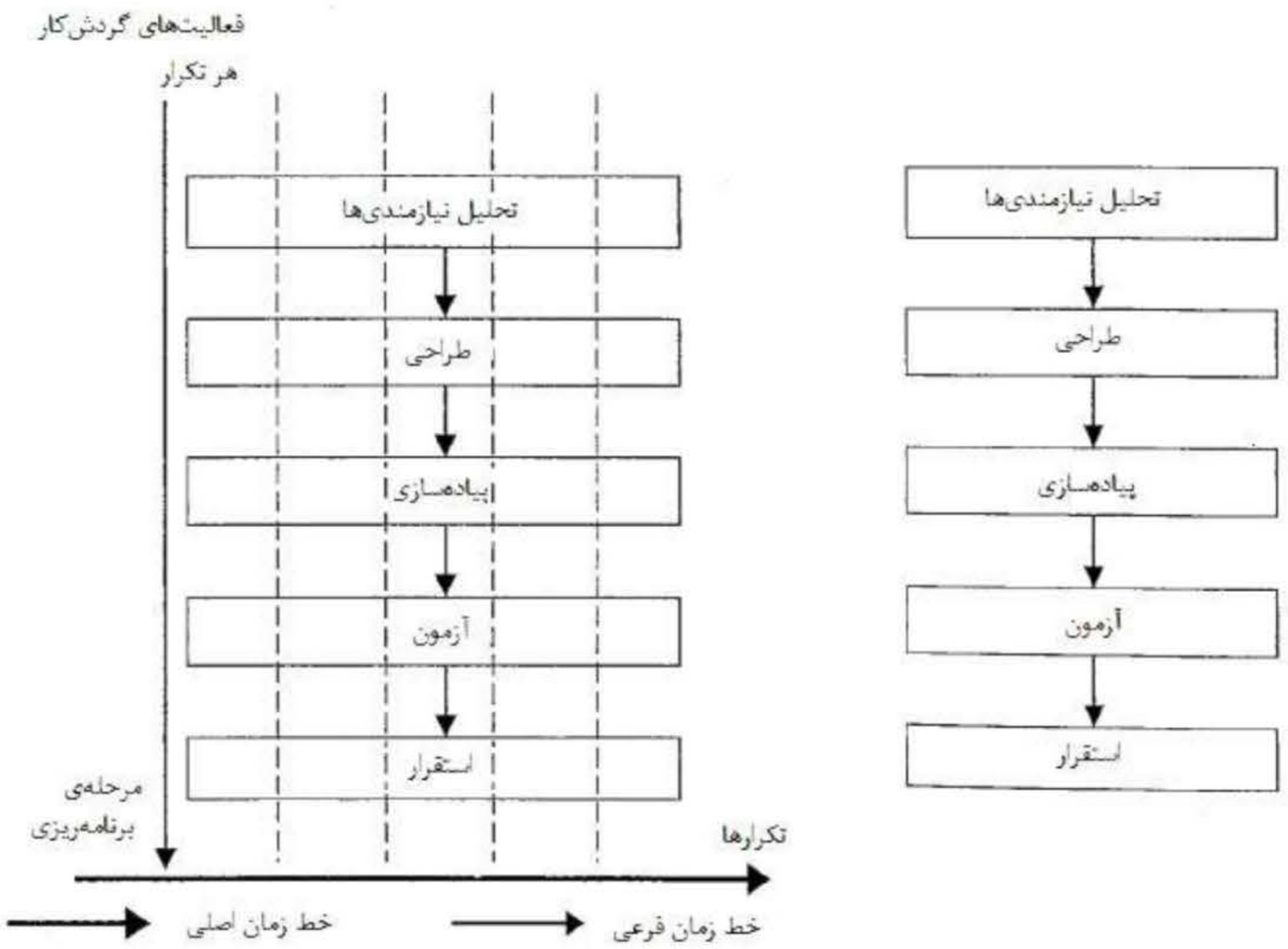
این بخش فرایند یکنواخت چابک و متدولوژی استفاده شده در این کتاب را معرفی می‌نماید. همان‌گونه که در شکل ۲-۱۵(ب) نشان داده شده است، این فرایند را می‌توان به‌صورت یک برش عمودی از فرایند آبشاری دانست. هر برش یک تکرار را مشخص می‌کند که از یک هفته تا سه ماه متغیر است، برای این فرایند دو محور می‌توان تصور کرد. محور افقی تکرارها و محور عمودی فعالیت‌های جریان کار هر یک از تکرارها را نشان می‌دهد. در هر تکرار بیشتر فعالیت‌های گردش کار به‌صورت فرایند آبشاری اجرا می‌شوند با این تفاوت که فعالیت‌ها مربوط به مورد کاربردهایی می‌شوند که به آن تکرار تخصیص یافته‌اند. همان‌گونه که در شکل ۲-۱۵(ب) نشان داده شده است، قبل از تکرارها، یک فاز کوتاه برنامه‌ریزی وجود دارد که حدود دو هفته طول می‌کشد و به‌منظور شناسایی نیازمندی‌ها، به‌دست آوردن مورد کاربردها و انتساب آنها به تکرارها انجام می‌شود. در صورتی که تغییری در نیازمندی‌ها رخ دهد، در فاز نیازمندی‌های هر کدام از تکرارهای بعدی این تغییرات باید لحاظ شوند. تغییر در نیازمندی‌ها ممکن است منجر به تغییر در فهرست مورد کاربردها و تغییر در موعد تحویل مورد کاربردها بشود.

شکل ۲-۱۶ گام‌های متدولوژی یکنواخت چابک[۲] و نیز ورودی، خروجی و روابط بین این گام‌ها را نشان می‌دهد. این متدولوژی شامل دو مرحله‌ی اصلی می‌گردد: الف) مرحله‌ی برنامه‌ریزی و ب)مرحله‌ی تکراری. در مرحله‌ی برنامه‌ریزی، تیم توسعه با نمایندگان مشتری و کاربران ملاقات می‌کند تا نیازمندی‌ها را شناسایی کرده و مورد کاربردها را از نیازمندی‌ها استنتاج[۳] نماید. همچنین تیم توسعه یک طراحی معماری و یک برنامه تهیه می‌کند تا بتواند در حین مرحله‌ی تکرار، مورد کاربردها را توسعه داده و مستقر نماید. در حین مرحله‌ی برنامه‌ریزی، نمودارهای مورد کاربرد تولید می‌شوند تا مورد کاربردها و زیرسیستم‌ها و مولفه‌هایی که حاوی مورد کاربردها می‌باشند را نمایش دهد. برای مثال، یک سیستم اطلاعات کتابخانه [۴] را در نظر بگیرید که احتمالاً ده‌ها نیازمندی دارد. به‌عنوان‌مثال، سه‌تای آن‌ها با شماره‌های R1، R2، و R3 به شرح زیر می‌باشند.

R1) LIS باید به اعضا اجازه‌ی امانت گرفتن مستندات را بدهد.
R2) LIS باید به اعضا اجازه‌ی برگرداندن مستندات را بدهد.
R3) LIS باید به اعضا اجازه‌ی جست‌وجوی مستندات با استفاده از معیارهای مختلف را بدهد.

از این نیازمندی‌ها، سه مورد کاربردی مقابل به ترتیب استنتاج می‌شود: امانت مستندات[۵]، برگرداندن مستندات[۶] و جست‌وجوی مستندات[۷]. علاوه بر این موارد، LIS نیازمندی‌ها و مورد کاربردهای بسیار دیگری نیز دارد. لذا برای تجزیه‌ی

[۱] Lightweight Unified Process (LUP)
[۲] Agile Unified Methodology (AUM)
[۳] Derive
[۴] Library Information System (LIS)
[۵] Document Checkout
[۶] Return Document
[۷] Search for Document



ب) فرایند چابک یکنواخت

الف) فرایند آبشاری

شکل ۲-۱۵: فرایند آبشاری و فرایند چابک

مجموعه‌ی بزرگی از مورد کاربردها نیاز به یک سازوکار[۱] می‌باشد. نمودارهای مورد کاربرد برای سازمان‌دهی مورد کاربردها به زیرسیستم‌ها، به‌گونه‌ای که با هر زیرسیستم بتوان به‌طور مجزا برخورد کرد، مفید می‌باشند. هر نمودار مورد کاربرد، مورد کاربردهای سیستم یا زیرسیستم و رابطه‌ی بین مورد کاربردها و کاربران سیستم که تحت عنوان کنش‌گر[۲] از آن‌ها یاد می‌شود را نمایش می‌دهد. آن گونه که شکل ۲-۱۷ نشان می‌دهد زیرسیستم گردش مستندات[۳] سه مورد کاربرد دارد. به همین صورت، نمودارهای مورد کاربرد دیگری نیز می‌توان برای زیرسیستم فهرست‌برداری[۴] و زیرسیستم خرید[۵] داشت. این زیرسیستم‌ها را می‌توان برای طراحی و پیاده سازی به تیم‌های مختلف توسعه سپرد.

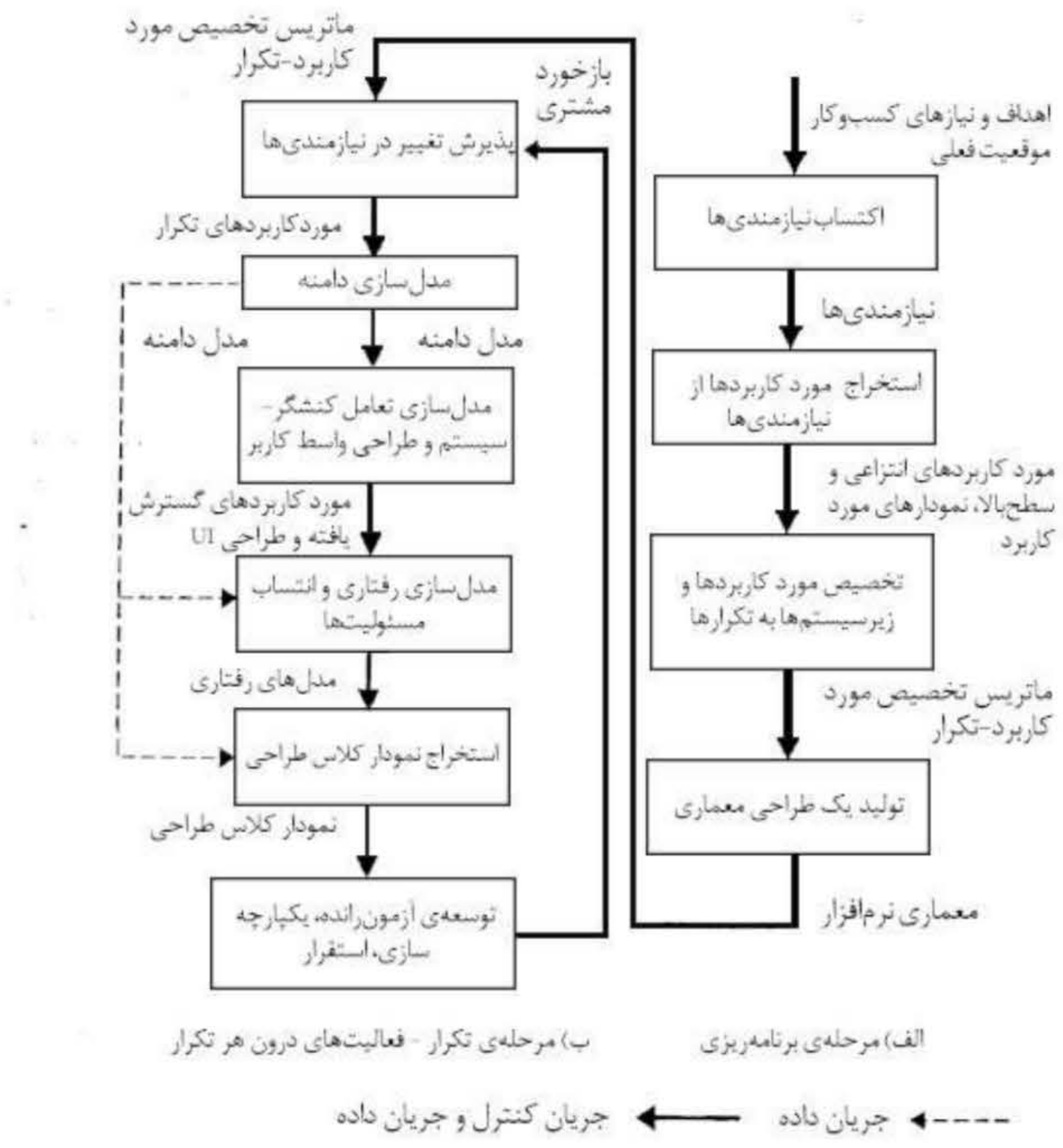
در حین مرحله‌ی تکرار، مورد کاربردها طبق زمان‌بندی تهیه شده در مرحله‌ی برنامه‌ریزی، به‌طور تکراری توسعه و در محیط مقصد استقرار می‌یابند. به خصوص، در هر تکرار برای مورد کاربردهایی که به آن تکرار واگذار شده‌اند گام‌های زیر برداشته می‌شوند.

پذیرش تغییر نیازمندی‌ها

تغییر نیازمندی‌ها در اکثر پروژه‌های نرم‌افزاری امری عادی است. این تغییر، در صورت وجود، باید در شروع هر تکرار پردازش شود. تغییر در نیازمندی‌ها منجر به تغییر در مورد کاربردها و نیز تغییر در برنامه‌ی توسعه و استقرار مورد کاربردها می‌شود. بعد

---

[۱] Mechanism
[۲] Actor
[۳] LIS/Circulation
[۴] Cataloging
[۵] Purchaising



شکل ۲-۱۶: نمای کلی از متدولوژی یکنواخت چابک

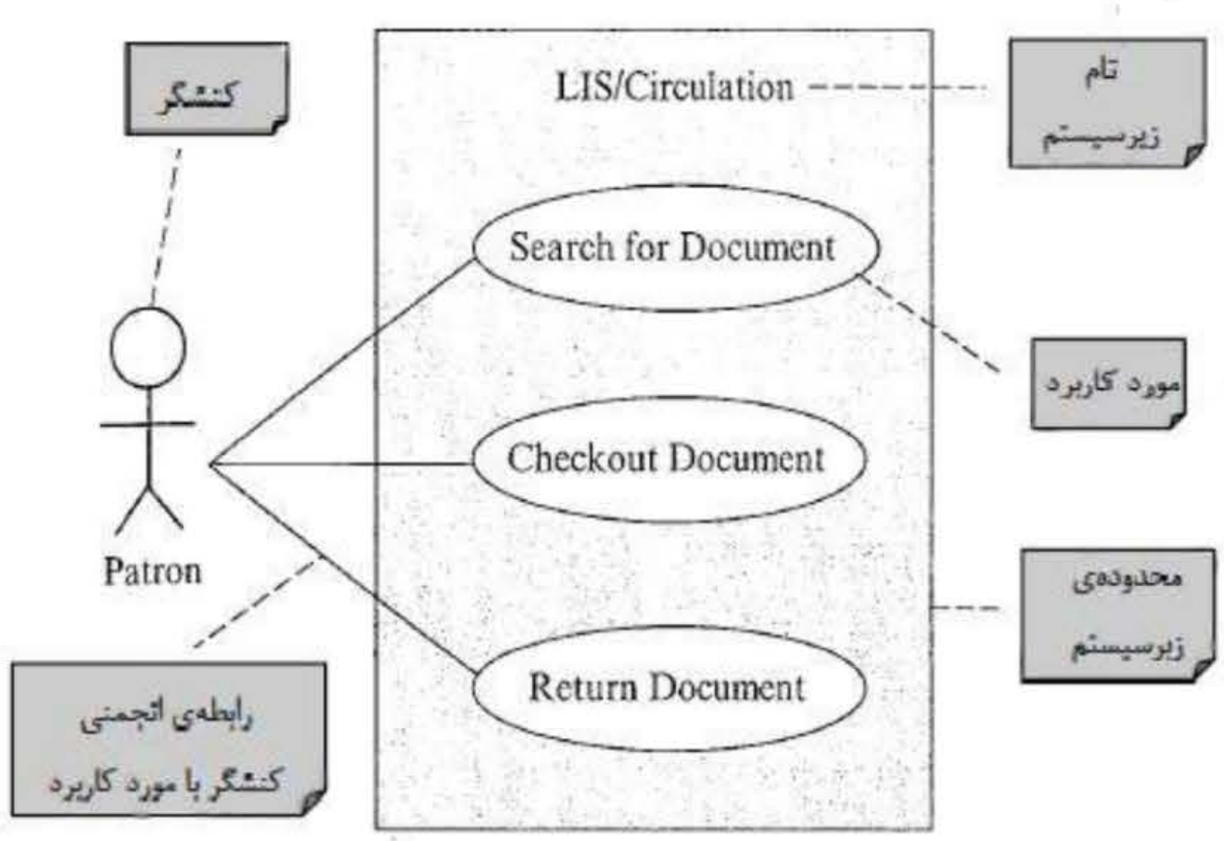
از اعمال این تغییرات، زمان‌بندی تکرار اصلاح شده و توسعه طبق زمان‌بندی اصلاح شده و با انجام گام‌های زیر ادامه می‌یابد.

مدل‌سازی دامنه

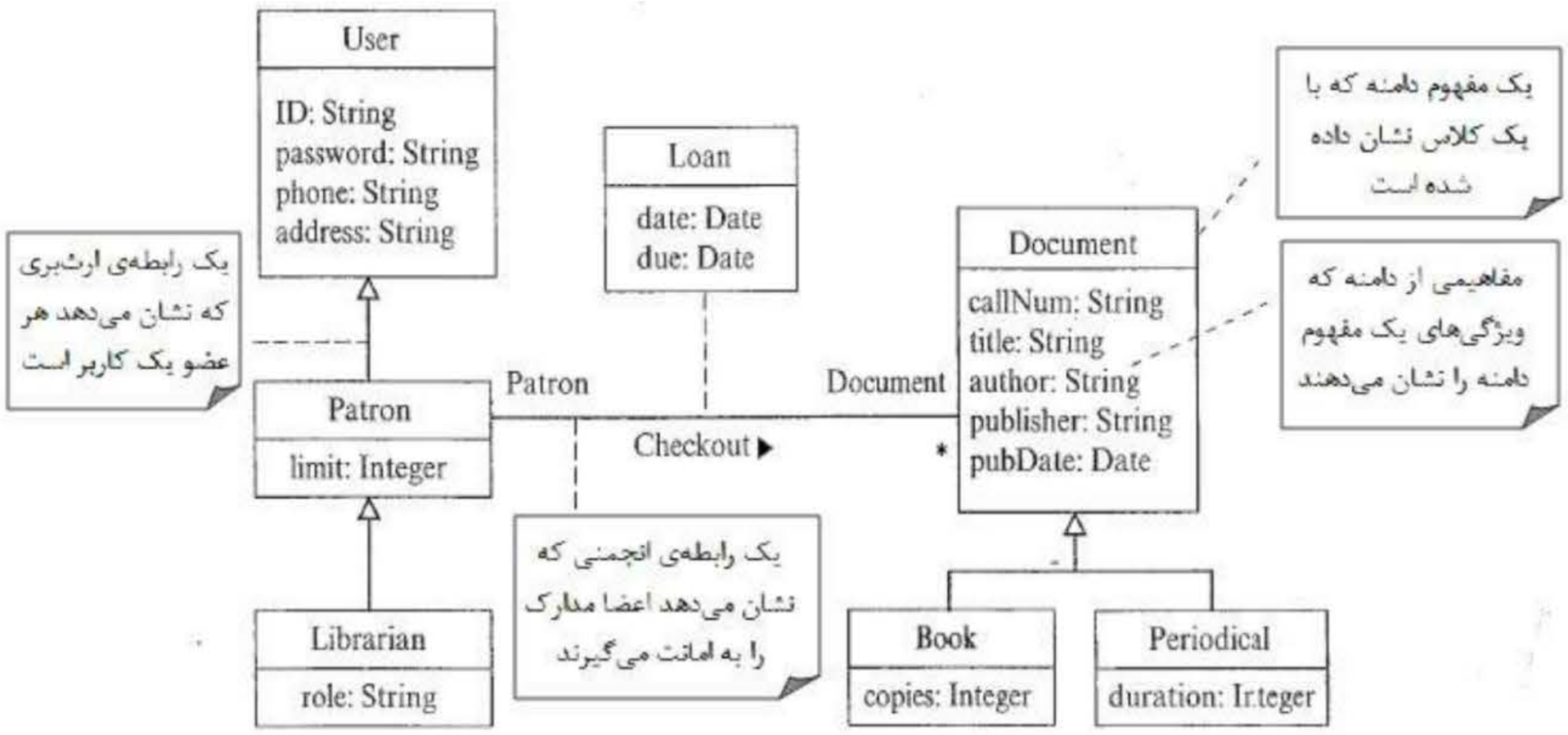
مدل‌سازی دامنه[۱]، فرایندی برای کشف مفاهیم مهم دامنه است. برای مثال، در یک سیستم کتابخانه، مفاهیم مهم دامنه عبارتند از: کاربر، عضو، شماره تلفن، مدرک، کتاب، مجله، تاریخ امانت، مدت مجاز امانت، تاریخ برگشت، امانت، کتابدار، و بسیاری دیگر. تیم توسعه از طریق ارتباط با نمایندگان مشتری، کاربران و خبرگان دامنه این دانش دامنه را به دست می‌آورد. سپس مفاهیم دامنه در قالب کلاس‌ها، صفات[۲] کلاس‌ها و رابطه بین کلاس‌ها طبقه‌بندی می‌شوند. ماحصل کار یک مدل دامنه است، که با استفاده از نمودار کلاس UML به تصویر کشیده می‌شود. یک مدل دامنه‌ی نمونه برای سیستم کتابخانه در شکل ۲-۱۸ نشان داده شده است.

[۱] Domain Modeling

[۲] Attribute



شکل ۲-۱۷: نمودار مورد کاربرد برای سیستم کتابخانه



شکل ۲-۱۸: یک مدل دامنه‌ی نمونه برای سیستم کتابخانه

مدل‌سازی تعامل کنش‌گر-سیستم

کاربران به‌منظور دریافت خدماتی که سیستم ارائه می‌دهد با آن تعامل می‌کنند. منظور از مدل‌سازی تعامل کنش‌گر-سیستم[۱] این است که مشخص شود کاربران چگونه با سیستم تعامل می‌کنند تا مورد کاربردها را اجرا کنند. این مشخصات را مورد کاربردهای گسترش‌یافته[۲] می‌نامیم. برای روشن‌شدن موضوع، شکل ۲-۱۹ مشخصات یک مورد کاربردی گسترش‌یافته را برای مورد کاربردی امانت مدرک نشان می‌دهد. در این نمایش جدولی، ستون چپ جدول ورودی کاربر و اعمال کاربر را نشان

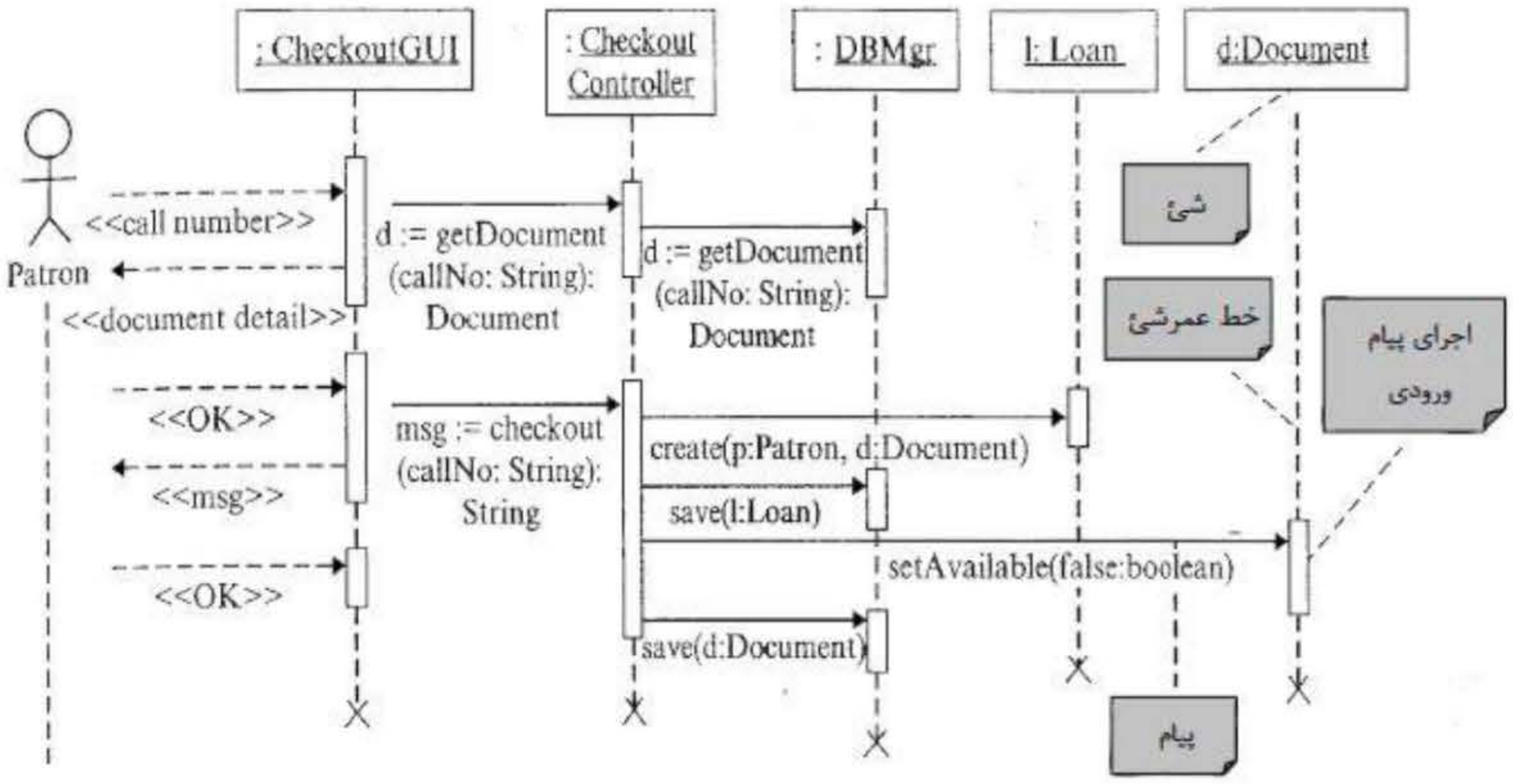
[۱] Actor-system Interaction Modeling

[۲] Expanded Use Cases

موردکاربرد «خروج مدرک»	
سیستم : LIS	کنش‌گر : مراجعه‌کننده
۰. سیستم فهرست اصلی را نشان می‌دهد.	
۲. سیستم فهرست «خروج مدرک » را نشان می‌دهد.	۱. شروع این مورد کاربرد با کلیک روی دکمه‌ی «امانت مدرک» از فهرست اصلی توسط عضو می‌باشد.
۴. سیستم جزییات مدرک را جهت تایید نمایش می‌دهند.	۳. مراجعه‌کننده شماره مدرک موردنظر را وارد کرده و روی دکمه «ارسال» کلیک می‌کند.
۶. سیستم (پایگاه‌داده را بروزرسانی نموده و) یک پیغام جهت تکمیل خروج نمایش می‌دهد.	۵. مراجعه‌کننده روی دکمه «OK» کلیک کرده و خروج مدرک را تایید می‌کند.
	۷. این مورد کاربرد وقتی خاتمه می‌یابد که مراجعه‌کننده روی دکمه‌ی «OK» از جعبه‌ی گفت‌وگوی تکمیل خروج کلیک کند.

شکل ۲-۱۹: نمونه ای از یک مورد کاربرد گسترش‌یافته

می‌دهد، در حالی که ستون راست پاسخ‌های متناظر که سیستم می‌دهد را نمایش می‌دهد.



شکل ۲-۲۰: مدل‌سازی تعامل اشیا برای امانت مدرک

مدل‌سازی رفتاری

مدل‌سازی تعامل کنش‌گر-سیستم، محاسبات یا الگوریتم‌هایی که درخواست‌های کاربر را پردازش می‌کنند را نشان نمی‌دهد. این موضوع بحث مدل‌سازی رفتاری[۱] است. برای مثال، فرض کنید که در گام ۳ از شکل ۲-۱۹ عضو شماره مدرکی که می‌خواهد امانت بگیرد را وارد نماید. برای نمایش نتایج در گام ۴، سیستم باید درخواست عضو را پردازش کند. این پردازش شامل پردازش‌های پس‌زمینه‌ای است که توسط اشیا نرم‌افزاری انجام می‌شوند. مدل‌سازی رفتاری چگونگی تعامل اشیا با یکدیگر جهت پردازش درخواست را توصیف می‌کند. نمودار توالی شکل ۲-۲۰ چگونگی پردازش این درخواست را نشان می‌دهد. جعبه‌های مستطیلی ردیف بالایی نمودار توالی، اشیایی که با هم در تعامل هستند را مشخص می‌کند. خط‌چین‌های عمودی، خط‌عمر[۲] اشیا را نشان می‌دهد. بدین معنی که در سیستم اشیا ایجاد شده و وجود دارند. پیکان‌های ممتد فراخوانی توابع بین اشیا و پیکان‌های خط چین تعامل بین کنش‌گر و سیستم را نشان می‌دهند. نمودار نشان داده شده در شکل ۲-۲۰ را می‌توان به این صورت تفسیر نمود. شروع این کار وقتی است که عضو «شماره‌ی مدرک» را به شئ واسط کاربر امانت مدرک[۳] می‌فرستد، که او هم این درخواست را با فراخوانی تابع ()getDocument از شئ کنترلگر خروج[۴] تکرار می‌کند، که او هم شئ مدرک[۵] را از پایگاه‌داده بازیابی می‌کند و نیز یک شئ امانت[۶] ایجاد کرده و آن را درون پایگاه‌داده ذخیره می‌کند، و سرانجام نیز شئ مدرک را بروزرسانی می‌نماید.

استخراج نمودار کلاس طراحی

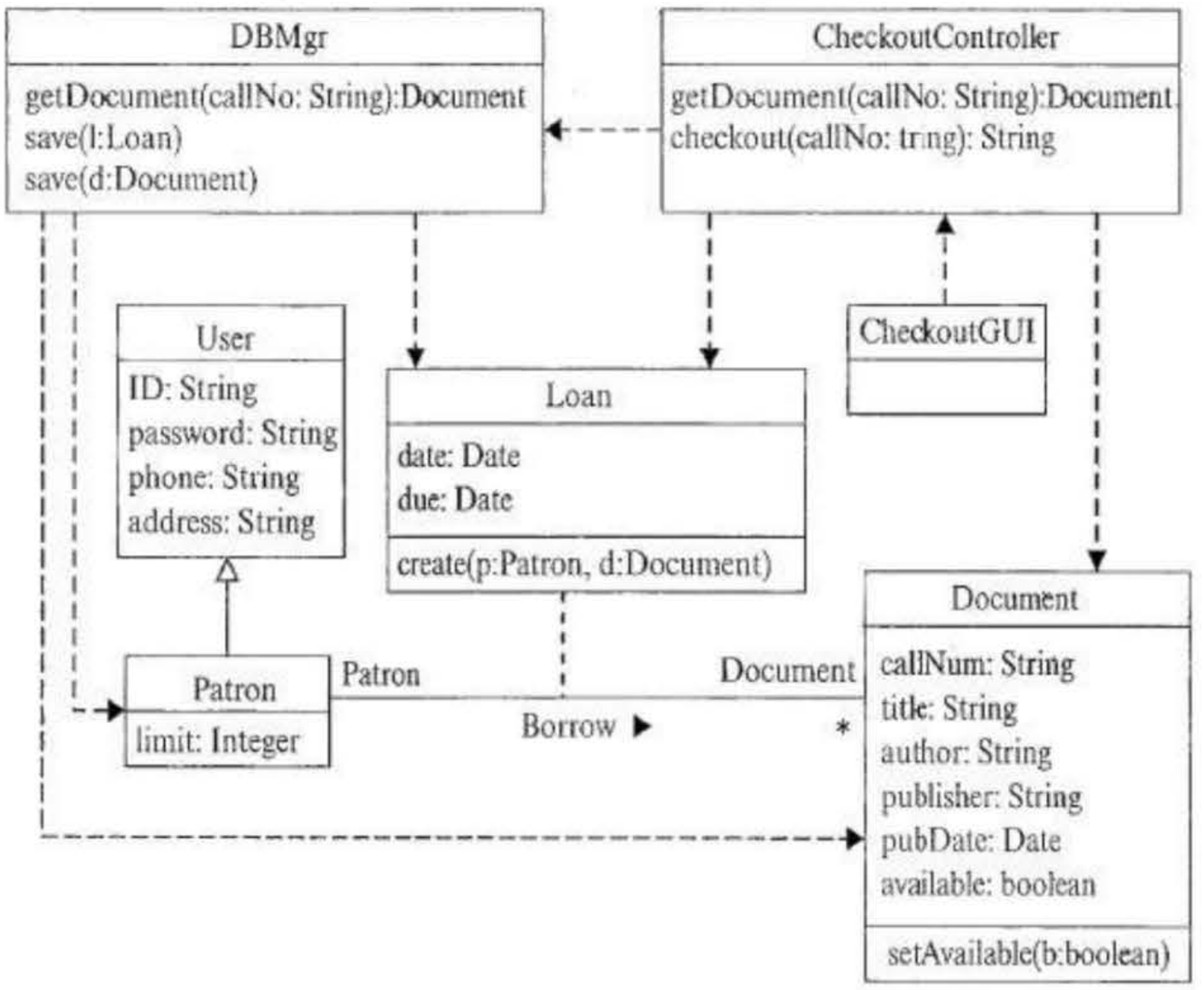
طراحی و مدل‌سازی رفتاری باعث ایجاد نمودارهای توالی، نمودارهای حالت و نمودارهای فعالیت می‌شود. این نمودارها، طراحی الگوریتم‌هایی را مشخص می‌کنند که فرایندهای کسب‌وکاری را پیاده‌سازی می‌کنند. پیاده‌سازی کلاس‌ها (کلاس‌ها) از روی این نمودارها راحت نیست، چون ممکن است یک کلاس در چند نمودار مختلف ظاهر شده باشد. برای مثال، کلاس مدرک[۷] در نمودارهای توالی امانت مدرک[۸] و برگشت مدرک[۹] به چشم می‌خورد. تیم نرم‌افزاری به یک نگرش یک‌پارچه از طراحی نیاز دارد. این نگرش یک‌پارچه با استفاده از یک نمودار کلاس UML[۱۰] که به آن نمودار کلاس طراحی[۱۱] می‌گوییم ساخته می‌شود. شکل ۲-۲۱ یک نمودار کلاس طراحی را نشان می‌دهد که از نمودار توالی شکل ۲-۲۰ استنتاج شده است. پیکان‌های خط‌چین در نمودار کلاس طراحی بیانگر روابط استفاده‌نمودن[۱۲] یا وابستگی[۱۳] بین کلاس‌ها می‌باشد.

پیاده‌سازی[۱۴]، آزمون یکپارچه‌سازی و استقرار

در حین توسعه‌ی آزمون-رانده کلاس‌هایی که مورد کاربردهای تکرار فعلی را پیاده‌سازی می‌کنند، پیاده‌سازی و آزمایش می‌شوند. کلاس‌ها یکپارچه‌سازی‌شده و آزمایش می‌شوند تا اطمینان حاصل شود که می‌توانند با یکدیگر کار کنند. سرانجام، سیستم نرم‌افزاری که مورد کاربردهای تکرار جاری را پیاده‌سازی می‌کنند در محیط مقصد نصب و آزمایش می‌شوند.

---

۱ Behavioral Modeling
۲ Lifeline
۳ Checkout GUI
۴ Checkout Controller
۵ Document
۶ loan
۷ Document Class
۸ Checkout Document
۹ Return Document
۱۰ UML Class Diagram
۱۱ Design Class Diagram
۱۲ Uses
۱۳ Dependency
۱۴ Implementation



شکل ۲-۲۱: نمودار کلاس طراحی با ترتیب پیاده‌سازی

خلاصه

توسعه‌ی نرم‌افزار با دو گونه چالش مواجه است: چالش‌های پروژه و چالش‌های محصول. برای غلبه بر این چالش‌ها، به فرایند نرم‌افزار نیاز است. اما، فرایند سنتی آبشاری با مشکلات زیادی همراه است، چون این فرایند برای حل مسائل خوش‌رفتار مناسب است، در حالی که توسعه‌ی نرم‌افزار کاربردی به‌طور کلی یک مسئله‌ی بدرفتار است. کنکاش برای فرایندهای بهتر منجر به ایجاد مدل‌های فرایند نرم‌افزاری متعددی گردید که فرایندهای چابک از جدیدترین آن‌ها محسوب می‌شوند.

در حالی که یک فرایند نرم‌افزاری فعالیت‌های مرحله به مرحله را مشخص می‌کند، متدولوژی گام‌ها یا چگونگی انجام هر کدام از این فعالیت‌ها را توصیف می‌نماید. در این فصل تفاوت میان فرایند و متدولوژی مورد بحث قرار گرفت. متدولوژی نرم‌افزار از نگرش نرم‌افزاری که برای توسعه‌ی سیستم نرم‌افزاری انتخاب شده است تاثیر می‌پذیرد. زیرا این «نگرش» است که تعیین می‌کند تیم توسعه چگونه به سیستم‌ها و دنیای واقعی نگاه کند، و به همین ترتیب قطعات اصلی سازنده‌ی سیستم نرم‌افزاری و متدولوژی توسعه را مشخص می‌کند. برای مثال، نگرش OO، دنیا و سیستم‌ها را متشکل از اشیا به هم مرتبط و تعامل‌کننده می‌بیند. لذا نتیجه این خواهد شد که قطعات سازنده‌ی یک سیستم OO، اشیا هستند. بنابراین، یک متدولوژی توسعه‌ی نرم‌افزار OO باید چگونگی انجام فعالیت‌های مدل‌سازی، تحلیل، طراحی، پیاده‌سازی و آزمون شیءگرا را بیان کند.

پیدایش فرایندها و روش‌های چابک نشان‌گر این واقعیت است که پیشرفت قابل‌توجهی در پذیرش بدرفتار بودن توسعه‌ی نرم‌افزار حاصل شده است. بیانیه‌ی چابکی، اصول چابکی، فرایندهای چابک و روش‌های چابک، همه و همه برای حمله به توسعه‌ی نرم‌افزار، به‌عنوان یک مسئله‌ی بدرفتار طراحی شده‌اند. گرچه مقبولیت چابکی در صنعت نرم‌افزار به سرعت در حال رشد است، ولی روش‌های توسعه‌ی سنتی برنامه-رانده برای مدت طولانی باقی خواهند ماند، زیرا این روش‌ها نیز در صنعت

نرم‌افزار زمینه‌ی استفاده‌ی خود را خواهند داشت. لذا هم روش‌های معروف به تطبیقی [۱]، که می‌توان آن‌ها را برای شرایط چابک نیز تطبیق داد و هم روش‌های توسعه‌ی برنامه-رانده هر دو جذاب هستند.

برای مطالعه‌ی بیش‌تر

مدل حلزونی، فرایند نرم‌افزار انفرادی، فرایند نرم‌افزار تیمی، و فرایند یکنواخت به ترتیب در [۳۲،۸۶،۸۷،۹۱] توصیف شده‌اند. متدولوژی کلاسیک OO به نام OMT در [۱۳۱] معرفی شده است که بیشتر نمادگان UML را تعریف می‌کند. این متدولوژی توسط Rumbaught، Blaha بازنگری شده تا از UML 2.0 استفاده کند [۳۰]. یک ارائه‌ی عالی از تحلیل و طراحی OO با استفاده از UML و الگوها در [۱۰۴] یافت می‌شود.

روش‌های چابک که در این فصل معرفی شدند به تفصیل در [۲۱،۵۲،۱۱۹،۱۴۰] آمده‌اند. دیگر روش‌های چابک عبارتند از کریستال نارنجی [۵۰] ، توسعه‌ی لاغر [۱۲۴] ، توسعه‌ی نرم‌افزار تطبیقی [۸۰] و مدل‌سازی چابک [۱۱]. آبراهامسون و همکاران [۲] یک خلاصه و مقایسه‌ی خیلی خوب از روش‌های مختلف چابک ارائه کرده‌اند. در [۱۵]، Avison و Fitzgerald تاریخچه‌ی متدولوژی‌های توسعه‌ی نرم‌افزار را مرور کرده و تعدادی مسیر برای آینده پیشنهاد می‌دهند. مجموعه مقالات همایش سالانه‌ی توسعه‌ی چابک[۲]، مقالاتی را در زمینه‌های مختلف توسعه‌ی چابک منتشر می‌کند. برخی مقالات جالب دیگر عبارتند از [۳۵،۶۹،۸۸،۱۲۶،۱۴۴،۱۴۵].

سوال‌های مروری فصل

۱. فرایند نرم‌افزار چیست و مدل‌های فرایند معرفی شده در این فصل کدامند؟
۲. ضعف‌ها و قوت‌های فرایند آبشاری را برشمارید.
۳. متدولوژی توسعه‌ی نرم‌افزار چیست؟ چه تفاوت‌هایی بین یک فرایند و یک متدولوژی وجود دارند؟
۴. فرایندهای چابک و روش‌های چابک کدامند؟ فعالیت‌های چرخه‌ی حیات روش‌های چابک که در این فصل معرفی شدند، کدامند؟
۵. ویژگی‌های مسائل خوش‌رفتار و مسائل بدرفتار به ترتیب کدامند؟
۶. چرا توسعه‌ی نرم‌افزار به‌طور کلی یک مسئله‌ی بدرفتار است؟
۷. فرایند چابک چگونه به توسعه‌ی نرم‌افزار به‌عنوان یک مسئله‌ی بدرفتار حمله می‌کنند؟
۸. آیا توسعه‌ی چابک جایگزین روش‌های سنتی از قبیل فرایند آبشاری خواهد شد؟

---

[۱] Adaptive

[۲] Agile Development Conference

تمرین‌ها

۲-۱ شباهت‌ها و تفاوت‌ها میان مدل سنتی آبشاری و مدل فرایند یکنواخت کدامند؟
سه مورد از محاسن و معایب هر کدام از این دو مدل را شناسایی و دو مورد آن‌ها را توضیح دهید.

۲-۲ در مورد اینکه چگونه یک فرایند خوب و یک متدولوژی خوب کمک می‌کنند تا به چالش‌های پروژه و محصول حمله کنیم یک نوشتار بنویسید. تعداد صفحات را حداکثر پنج صفحه در نظر بگیرید یا بر طبق آنچه که استاد درس تعیین می‌کند عمل کنید.

۲-۳ نوشتار خلاصه‌ای در خصوص تفاوت‌های میان یک فرایند نرم‌افزاری و یک متدولوژی نرم‌افزاری بنویسید.

۲-۴ متنی بنویسید که درباره‌ی موارد زیر بحث کند.

الف) مزایا و معایب توسعه‌ی مدل-رانده و فرایندهای توسعه چابک.

ب) چرا و چگونه توسعه‌ی چابک خواهد یا نخواهد توانست جایگزین روش‌های برنامه-رانده شود.

۲-۵ مقاله‌ی کوتاهی بنویسید که به پرسش‌های زیر پاسخ دهد:

الف) شباهت‌ها و تفاوت‌های فرایند حلزونی، فرایند یکنواخت، و یکی از فرایندهای چابک کدامند؟

ب) مزایا و معایب هر کدام از این فرایندها چیست؟

ج) چه نوع پروژه‌هایی باید هر یک از این فرایندها را به‌کار گیرند؟

۲-۶ در یک نوشتار توضیح دهید که چرا فرایند آبشاری، فرایندی برای حل مسائل خوش‌رفتار است؟

۲-۷ در یک نوشتار توضیح دهید که چگونه توسعه‌ی چابک به توسعه‌ی نرم‌افزار کاربردی به‌عنوان یک مسئله‌ی بدرفتار حمله می‌کند؟

۳

مهندسی سیستم

نکات کلیدی

- مهندسی سیستم، یک روش بین‌رشته‌ای برای توسعه‌ی سیستم‌هایی است که شامل اجزای سخت‌افزاری، نرم‌افزاری، و انسانی می‌شوند.
- مهندسی سیستم نیازمندی‌ها و قیدهای سیستم را تعریف می‌کند، نیازمندی‌ها را به زیرسیستم‌های سخت‌افزاری، نرم‌افزاری و انسانی تخصیص می‌دهد، و سپس این زیرسیستم‌ها را یکپارچه می‌کند تا سیستم را تشکیل دهند.
- مهندسی نرم‌افزار بخشی از مهندسی سیستم است.

خیلی از سیستم‌ها، سیستم جاسازی‌شده هستند. یک سیستم جاسازی‌شده متشکل از اجزای سخت‌افزاری، نرم‌افزاری، و انسانی می‌شود. این اجزای با یکدیگر تعامل می‌کنند تا هدف سیستم محقق شود. به‌عنوان نمونه می‌توان به سیستم جابجایی چمدان در فرودگاه[۱] اشاره نمود. این سیستم مسئول نقل و انتقال چمدان‌ها از یک محل به محل دیگر در فرودگاه است. سخت‌افزار این سیستم شامل نوار نقاله‌ها، و ماشین‌های مخصوص حمل وسایل (DCV) هستند که در مسیرهای با سرعت بالا در حرکت هستند. نرم‌افزار این سیستم وظیفه‌ی مدیریت تحویل بار و نیز کنترل سخت‌افزار را به عهده دارد. کارکنان نیز سخت‌افزار و نرم‌افزار را به‌کار می‌گیرند. مثال‌های دیگری از سیستم‌های جاسازی‌شده عبارتند از سیستم‌های مدیریت نامه‌ها، سیستم‌های کنترل ترافیک هوایی، و سیستم‌های کنترل فرایند تولید. در توسعه‌ی چنین سیستم‌هایی باید کل سیستم مدّنظر قرار گیرد نه فقط سیستم نرم‌افزار آن، لذا یک روش مهندسی سیستم نیاز است.

توسعه‌ی یک سیستم صرفاً نرم‌افزاری ممکن است نیاز به یک روش مهندسی سیستم نیز داشته باشد. برای نمونه، یک سیستم برنامه‌ریزی منابع سازمانی[۲] را در نظر بگیرید. سیستم ERP عبارت است از یک سیستم اطلاعات مدیریت یکپارچه برای کل سازمان، که تمام کارکردهای سازمان اعم از بازاریابی، فروش، ساخت، مدیریت پروژه، مدیریت زنجیره‌ی تأمین، حسابداری، پشتیبانی مشتریان، مدیریت روابط با مشتریان، و کنترل دسترسی را به‌طور یکپارچه پشتیبانی می‌کند. سیستم ERP، تمام اعمال کسب‌وکاری فوق و گردش‌های کاری بین آن‌ها را خودکار می‌کند. به دلایل متعدد می‌توان دید که برای توسعه‌ی چنین سیستمی نیاز به مهندسی سیستم است. سیستم مذکور شامل تعداد زیادی اعمال کسب‌وکاری و سازمانی می‌شود و این اعمال

[۱] Airport Baggage Handling System (ABHS)

[۲] Enterprise Resource planning (ERP)

نیز رشته‌های مختلفی را دربرمی‌گیرند. لذا یک تیم توسعه‌ی چند رشته‌ای مورد نیاز است. این سیستم بزرگ و پیچیده است و ممکن است شامل سیستم‌های موروثی[۱] بشود که در سال‌های دور تولید شده و همچنان در حال استفاده هستند. چگونگی طراحی این سیستم و یکپارچه‌سازی آن با پایگاه‌داده‌ها و فرایندهای کسب‌وکار موجود یک چالش بزرگ است. لذا به یک روش مهندسی سیستم نیاز می‌باشد.

مهندسی سیستم به خودی خود یک رشته است، لذا پوشش دادن آن در یک فصل غیرممکن است. بنابراین، این فصل تنها یک مقدمه است که می‌خواهد به مهندس نرم‌افزار یک درک مقدماتی از فرایندها و فعالیت‌های مهندسی سیستم ارائه نماید. با مطالعه‌ی این فصل شما با موارد زیر آشنا خواهید شد:

- فرایند مهندسی سیستم
- روش‌های مدل‌سازی و طراحی سیستم
- تخصیص نیازمندی‌های سیستم به زیرسیستم‌ها
- مدیریت پیکربندی سیستم

۳-۱ سیستم چیست؟

یک سیستم از اجزایی تشکیل شده است که با یکدیگر تعامل می‌کنند تا به یک هدف برسند. سیستم می‌تواند بزرگ یا کوچک، ساده یا پیچیده باشد و به‌صورت فیزیکی یا فقط مفهومی وجود داشته باشد. به‌عنوان مثال، جهان[۲] یک سیستم خیلی بزرگ است که از میلیون‌ها سال پیش وجود داشته است. در عوض، مورچه یک سیستم خیلی کوچک است. این‌ها سیستم‌های طبیعی هستند. در مقابل سیستم‌های طبیعی، سیستم‌های زیادی نیز وجود دارند که ساخته‌ی دست بشر هستند. سیستم‌های ساخته‌ی بشر ممکن است به‌صورت فیزیکی و یا صرفاً مفهومی وجود داشته باشند. منطق ریاضی، سیستم‌های عددی، سیستم‌های اندازه‌گیری، و بسیاری از سیستم‌های دسته‌بندی همه و همه جزء مثال‌هایی از سیستم‌های مفهومی به حساب می‌آیند. سیستم‌های آبیاری قطره‌ای که باغ‌ها و مزرعه‌ها را آبیاری می‌کنند و سیستم‌های تلفنی که میلیون‌ها خانواده را به یکدیگر مرتبط می‌کنند، همه جزء نمونه‌هایی از سیستم‌های ساخته‌ی بشرند که به‌طور فیزیکی موجودند. سیستم‌های مبتنی بر کامپیوتر نیز سیستم‌های ساخته‌ی بشرند که نرم‌افزار در آن‌ها به‌عنوان یک جزء یا زیرسیستم قرار گرفته است. نمونه‌هایی از سیستم‌های مبتنی بر کامپیوتر عبارتند از: سیستم‌های مخابراتی، سیستم‌های رایانامه، سیستم‌های اطلاعات کتابخانه، و سیستم‌های کنترل فرایند. برخی از این‌ها، مثلاً سیستم رایانامه، سیستم‌های صرفاً نرم‌افزاری‌اند. سایر آن‌ها، مثلاً ABHS، شامل زیرسیستم‌های نرم‌افزاری، سخت‌افزاری و انسان می‌شوند. تمام سیستم‌ها مجموعه‌ای از خصوصیات و ویژگی‌ها را دارند که در زیر معرفی می‌شوند:

۱. هر سیستم شامل مجموعه‌ای از زیرسیستم‌ها، اجزای یا عناصر به هم مرتبط و تعاملی می‌شود. به‌عنوان مثال، ABHS شامل چندین زیرسیستم می‌شود که با یکدیگر در تعامل هستند تا به هدف سیستم برسند.

۲. هر سیستم ممکن است زیرسیستمی از یک سیستم بزرگ‌تر باشد، که آن هم به نوبه‌ی خود ممکن است زیرسیستمی از یک سیستم دیگر باشد. برای مثال سیستم نقاله و سیستم نرم‌افزاری تحویل بار در فرودگاه هردو زیرسیستمی از ABHS هستند. ولی ABHS خود یک زیرسیستم از سیستم فرودگاه تلقی می‌شود. بدین ترتیب، رابطه‌ی بین سیستم‌ها و زیرسیستم‌ها یک رابطه‌ی بازگشتی و کل به جزء است. این رابطه یک سلسله مراتب کل- جزء را تشکیل می‌دهد. هر کدام از سیستم‌ها در این سلسله مراتب یک جایگاه خواهد داشت.

---

[۱] Legacy Systems

[۲] Universe

۳. هر سیستم در یک محیط جای می‌گیرد و با محیط خود تعامل می‌کند. برای مثال، ABHS در محیط فرودگاه قرار دارد و با سیستم اطلاعات پرواز تعامل می‌کند. به‌علاوه، ABHS به‌منظور بارگیری و تخلیه‌ی بارها با پروازهای ورودی/خروجی کار می‌کند.
۴. سیستم‌ها به‌دلیل عوامل داخلی و خارجی همچنان در حال تکامل هستند. به‌عنوان مثال، با رشد اقتصاد تعداد مسافران تجاری و گردشگران نیز رشد خواهد کرد. لذا فرودگاه‌ها هم نیاز به پایانه‌ها و خروجی‌های بیشتری دارند، و در نتیجه ABHS نیز نیاز به گسترش خواهد داشت. پیشرفت فناوری ممکن است مدیریت فرودگاه را ترغیب کند که پویشگرهای بارکد را با دستگاه‌های شناسایی فرکانس رادیویی[۱] جایگزین کنند. عوامل داخلی عبارتند از ایرادهای طراحی کشف شده، نقص مولفه، خطاهای نرم‌افزاری و سایر موارد مشابه. تمام این موارد نیاز به تغییر در سیستم خواهند داشت و باعث می‌شوند سیستم تکامل پیدا کند.

۳-۲ مهندسی سیستم چیست؟

برای توسعه‌ی سیستم ABHS باید کل سیستم را در نظر گرفت نه فقط سیستم نرم‌افزاری را، علاوه براین، ABHS شامل چندین رشته مهندسی از قبیل مهندسی برق و الکترونیک، مهندسی مکانیک، مهندسی عمران و مهندسی نرم‌افزار خواهد بود. مهندسان این رشته‌ها باید با یکدیگر همکاری نموده و سیستم را بسازند. به‌طور کلی، توسعه‌ی سیستم برای سیستم‌های جاسازی‌شده شامل چند رشته‌ی مهندسی و غیرمهندسی می‌شود:

- مهندسی برق و الکترونیک. بسیاری از سیستم‌ها از اجزایی مانند مدارهای مجتمع با تراکم زیاد[۲]، مدارهای یکپارچه خاص-کاربرد[۳] حس‌گر، رله[۴]، سوییچ[۵] و منبع تغذیه[۶] تشکیل می‌شوند. برای مثال، در ABHS از چاپگر بارکد برای تولید برچسب بار و از پویش‌گر بارکد در تقاطع نوار نقاله‌ها برای خواندن بارکدها استفاده می‌شود. مهندسان برق و الکترونیک تحلیل، طراحی، یکپارچه‌سازی و آزمایش این قبیل قطعات و زیرسیستم‌ها را برعهده دارند.
- مهندسی مکانیک. در بسیاری از سیستم‌ها که کار فیزیکی انجام می‌دهند، وسایل مکانیکی به‌کار رفته‌اند. برای مثال، در ABHS از نوار نقاله برای حرکت دادن بسته‌ها و از فشاردهنده[۷] در تقاطع‌ها برای هدایت بسته به طرف مقصد مورد نظر استفاده می‌شود. همچنین در این سیستم از نوارهای پرسرعت استفاده می‌شود تا کالا بین پایانه‌های مختلف حمل شود. بنابراین، تحلیل، طراحی، ساخت، آزمایش و نصب ABHS نیاز به همکاری کاردانان و مهندسان مکانیک دارد.
- مهندسی عمران. برای حفاظت از وسایل و بارها، ABHS نیاز به سازه‌های بزرگ و متصل به هم دارد. نقاله‌ها و نوارهای پرسرعت باید بر روی پایه‌های بتنی نصب شوند. کارهایی از قبیل طراحی، ساخت و آزمایش این سازه‌ها و پایه‌ها وظیفه مهندسی عمران است.
- مهندسی نرم‌افزار. نرم‌افزار ، مسئول انجام بیشتر کارهای بحرانی و چالشی در سیستم جامع است. انجام فعالیت‌های پردازش اطلاعات و نیز کنترل سایر دستگاه‌ها و تجهیزات در کل سیستم بر عهده‌ی نرم‌افزار است. در ABHS، زیرسیستم نرم‌افزاری تحویل بار، امکان دریافت بار مسافران را به مأمور خطوط هوایی می‌دهد. نرم‌افزار پردازش بارکد نیز فشاردهنده‌های مکانیکی مستقر در تقاطع شبکه‌ی نقاله‌ها را طوری تنظیم می‌کند که بار به سمت مقصد مورد نظر هدایت شود. مهندسان نرم‌افزار مسئول تحلیل، طراحی، پیاده‌سازی، آزمون ، و استقرار قطعات و زیرسیستم‌های نرم‌افزاری سیستم جامع هستند.

---

[۱] Radio Frequency Identification (RFID)
[۲] Very Large-Scale Integrated Circuits (VLSIC)
[۳] Application Specific Integrated Circuit (ASIC)
[۴] Relays
[۵] Switches
[۶] Power Supply
[۷] Pusher

- علم کامپیوتر. علوم کامپیوتر و اطلاعات مبانی مهندسی نرم‌افزار بوده و فناوری‌هایی را ارائه می‌دهند که در اکثر سیستم‌ها مورد استفاده قرار می‌گیرند. برای مثال، در ABHS دست کم مباحث زبان‌های برنامه‌سازی، داده‌ساختارها و الگوریتم‌ها، سیستم‌های پایگاه‌داده، تحلیل و طراحی سیستم‌های جاسازی‌شده بلادرنگ و امنیت کامپیوتری مطرح هستند.
- مدیریت بازرگانی و اقتصاد. در اکثر پروژه‌های توسعه‌ی سیستم، جنبه‌های تجاری از ملاحظات مهم محسوب می‌شوند، برای مثال، مهندسی سیستم در ABHS باید تاثیر اقتصاد محلی، ملی و جهانی را بر روی کسب‌وکار فرودگاه و به ویژه ABHS تحلیل نماید. بدین معنی که باید حجم کالاهایی که باید جابجا شوند، رشد آتی، ارزش کالاها، هزینه‌ها و بازگشت سرمایه را تخمین بزند. برای انجام این کار، نیاز به دانش حسابداری، مالی، مدیریت و اقتصاد خواهد بود. همچنین، تحلیل، طراحی و پیاده‌سازی زیرسیستم منابع انسانی در ABHS نیز نیاز به دانش و تجربه در مدیریت منابع انسانی دارد.

فهرست فوق کامل نیست. خیلی از سیستم‌ها نیاز به رشته‌های دیگر مهندسی و غیرمهندسی از قبیل ریاضی، علوم تجربی، علوم اجتماعی، حقوق، و علوم انسانی خواهند داشت. علاوه براین، مهندسی سیستم باید جنبه‌های دیگری مانند ایمنی[۱]، امنیت[۲]، و قابلیت اطمینان[۳] را در نظر گیرد. برای مثال، در طراحی ABHS، باید ایمنی کارکنان که با نوار نقاله و نوارهای پرسرعت کار می‌کنند مدّنظر قرار گیرد. همچنین سیستم باید تجهیزاتی داشته باشد که بتواند بررسی کند قوانین امنیتی توسط کالاها رعایت شده‌اند. قابلیت اطمینان در ABHS بحرانی است، زیرا بر صدها هزار مسافر تاثیر خواهد داشت.

به‌طور خلاصه، توسعه‌ی سیستم یک تلاش بین‌رشته‌ای است و شامل توسعه‌ی سخت‌افزار، نرم‌افزار و منابع انسانی می‌گردد. همین امر باعث چالش‌های متعددی می‌شود. مثلاً این‌که، چگونه می‌توان نیازمندی‌های چنین سیستم‌هایی را شناسایی و تدوین کرد؟ طراحی و ساخت چنین سیستم‌هایی چگونه باید انجام شود؟ چگونه می‌توان فعالیت‌های توسعه را مدیریت نمود؟ چگونه می‌توان کارایی، قابلیت اطمینان، ایمنی و سایر عوامل را در چنین سیستم‌هایی اندازه‌گیری کرد؟ چگونه باید تاثیر سیستم را بر روی جامعه و محیط ارزیابی نمود؟ مهندسی سیستم، رشته‌ای از مهندسی است که به این پرسش‌ها پاسخ می‌دهد. مهندسی سیستم این چالش‌ها را پاسخ می‌دهد. به ویژه، مهندسی سیستم بر روی عناصر زیر تاکید می‌نماید.

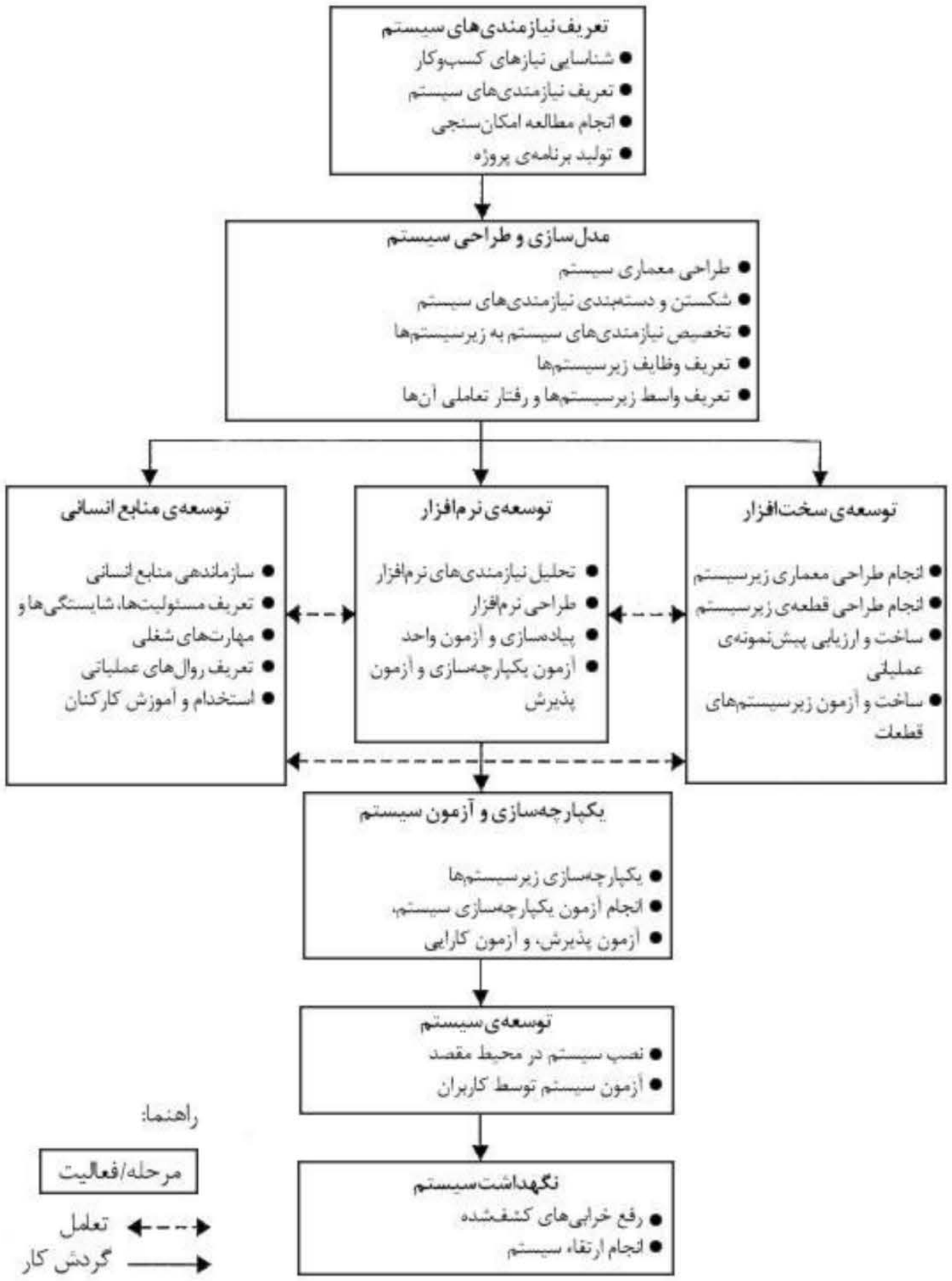
۱. یک فرایند مهندسی سیستم که کل چرخه‌ی حیات سیستم را پوشش دهد. فرایند توسعه‌ی سیستم، مراحل و فعالیت‌هایی که باید در هر مرحله انجام شوند تا سیستم توسعه یابد را تعریف می‌کند. این فرایند تمامی فعالیت‌های چرخه‌ی حیات سیستم را که از مفهوم سیستم اولیه شروع و به نگهداری و بازنشستگی سیستم ختم می‌شود، مدّنظر قرار می‌دهد. این رویکرد سیستم-گرا[۴] تیم‌های مهندسی را تشویق می‌کند که بر روی نیازها و اولویت‌های کسب‌وکاری مشتری تمرکز کرده و سیستم‌هایی را تهیه کنند که این نیازها و اولویت‌ها را برآورده نماید.
۲. یک رویکرد بالابه‌پایین و تقسیم‌وحل. این رویکرد، کل سیستم را به سلسله‌مراتبی از زیرسیستم‌ها و اجزای (قطعات) تجزیه می‌کند، و هر کدام را به‌طور جداگانه توسعه می‌دهد. با این رویکرد، ABHS به زیرسیستم‌هایی شکسته می‌شود. هر زیرسیستم، توسط یک تیم از مهندسان ساخته می‌شود. این راهبرد تقسیم‌وحل، امکان توسعه‌ی سیستم‌های پیچیده و خیلی بزرگ را به تیم‌های مهندسی می‌دهد.
۳. یک رویکرد بین‌رشته‌ای برای توسعه‌ی سیستم. همان‌طور که پیش‌تر گفته شد، توسعه‌ی سیستم یک تلاش بین‌رشته‌ای است. بدین معنی که برای انجام فعالیت‌های توسعه‌ی سیستم، تیم‌های بین رشته‌ای با یکدیگر کار می‌کنند.

شکل ۳-۱ مراحل و گردش‌کارهای یک فرایند مهندسی سیستم را نشان می‌دهد. مستطیل‌ها بیان‌گر مراحل یا فعالیت‌ها هستند. پیکان‌های ممتد گردش‌های کاری بین فعالیت‌ها را نشان می‌دهند. پیکان‌های خط‌چین بیان‌گر تعامل بین فعالیت‌های

---

[۱] Safety
[۲] Security
[۳] Reliability
[۴] System-Oriented

سخت‌افزاری، نرم‌افزاری و منابع انسانی می‌باشند. نوع سیستم بر روی فعالیت‌های مربوط تاثیر می‌گذارد. مثلاً، اگر سیستم یک سیستم صرفاً نرم‌افزاری باشد، در این صورت فعالیت‌های توسعه‌ی سخت‌افزاری اجرا نخواهند شد. در ادامه، مراحل فرایند مهندسی سیستم ابتدا برجسته شده و سپس در بخش‌های بعدی به تفضیل توضیح داده خواهند شد.



شکل ۳-۱: فرایند مهندسی سیستم

۱. تعریف نیازمندی‌های سیستم. سیستم به‌منظور برآورده‌کردن نیازهای کسب‌وکار ساخته می‌شود. برای مثال، سیستم ABHS برای مرتفع ساختن نیازهای یک فرودگاه تهیه می‌شود. هدف از مرحله‌ی تعریف نیازمندی‌های سیستم، شناسایی نیازهای کسب‌وکار و مشخص کردن قابلیت‌هایی برای سیستم است که نیازهای کسب‌وکار را مرتفع نمایند. نتیجه‌ی این مرحله، تبیین نیازمندی‌های سیستم و نیز برنامه‌ی پروژه برای طراحی، پیاده‌سازی، آزمون و استقرار سیستم می‌باشد.
۲. طراحی معماری سیستم[۱]. نیازمندی‌های سیستم، توانایی‌های مورد نیاز سیستم برای حل مسائل کسب‌وکار را مشخص می‌کند، ولی در خصوص چگونگی انجام این توانایی‌ها چیزی نمی‌گوید. در عوض، طراحی معماری سیستم یک طرح اولیه از راه حل و چگونگی ارائه این قابلیت‌ها تهیه می‌کند. به‌طور خاص، این فعالیت، معماری سیستم یا ساختار کلان سیستم را تعریف می‌کند، بدین معنی که زیرسیستم‌ها و روابط بین آن‌ها را مشخص می‌کند. همچنین، این فعالیت نیازمندی‌های سیستم را به زیرسیستم‌ها تخصیص می‌دهد.
۳. تعیین وظایف زیرسیستم‌ها و واسط‌های آن‌ها. وظایف و واسط‌های زیرسیستم‌ها، براساس نیازمندی‌هایی که به آن‌ها تخصیص داده شده، مشخص می‌شوند. همچنین در این فعالیت مشخص می‌شود که زیرسیستم‌ها چگونه با یکدیگر تعامل می‌کنند.
۴. توسعه‌ی زیرسیستم‌ها. پس از مدل‌سازی و طراحی سیستم، زیرسیستم‌ها به‌طور هم‌زمان توسط تیم‌های مختلف مهندسی توسعه می‌یابند. تیم‌های مهندسی، نیازمندی‌های سیستم که به زیرسیستم‌های مجزا تخصیص داده شده‌اند را پالایش نموده و سپس زیرسیستم‌ها را به‌گونه‌ای طراحی و پیاده‌سازی می‌نمایند که نیازمندی‌ها را برآورده نمایند. مثلاً، تیم مهندسی نرم‌افزار نیازمندی‌های نرم‌افزار را پالایش می‌کند و سپس براساس همین نیازمندی‌ها، زیرسیستم‌های نرم‌افزاری را طراحی، پیاده‌سازی و آزمایش می‌نماید.
۵. یکپارچه‌سازی، آزمون، استقرار و نگهداری سیستم. زیرسیستم‌ها باید یکپارچه شوند و سپس آزمون یکپارچه‌سازی انجام شود تا مطمئن شویم که می‌توانند با یکدیگر کار کنند. آزمون پذیرش سیستم انجام می‌شود تا مطمئن شویم که سیستم واقعاً قابلیت‌هایی که در سند تبیین نیازمندی‌های سیستم مشخص شده بود را ارائه می‌نماید. سپس سیستم در محیط مقصد نصب و توسط کاربر آزمایش می‌شود. خطاها، نقص‌ها و بازخوردهای گرفته شده از کاربر مورد رسیدگی قرار می‌گیرند. از این پس، سیستم وارد مرحله نگهداری می‌شود.

۳-۳ تعریف نیازمندی‌های سیستم

تعریف نیازمندی‌های سیستم، نیازهای کسب‌وکار را شناسایی کرده و نیازمندی‌های سیستم را مشخص می‌کند. این مرحله با یک مفهوم اولیه از سیستم شروع می‌شود و سپس این مفهوم گسترش یافته و پالایش می‌شود. در طی این فرایند، مجموعه‌ای از قابلیت‌هایی که سیستم باید تحویل دهد شناسایی می‌شود و سپس این قابلیت‌ها به شکل نیازمندی‌های سیستم مدون می‌شوند. نیازمندی‌های سیستم شامل نیازمندی‌های کارکردی، نیازمندی‌های کیفی، نیازمندی‌های کارایی و دیگر نیازمندی‌های خاص سیستم می‌باشند. در این بخش فعالیت تعریف نیازمندی‌های سیستم تشریح می‌شود.

۳-۳-۱- شناسایی نیازهای کسب‌وکار

شناسایی نیازهای کسب‌وکار[۲] با فعالیت جمع‌آوری اطلاعات شروع می‌شود، که در آن اطلاعاتی در خصوص اهداف و موقعیت فعلی کسب‌وکار جمع‌آوری می‌گردد. تیم باید شکاف بین موقعیت فعلی و اهداف کسب‌وکار را شناسایی کرده و نیازهای کسب‌وکار را استنتاج کند. به‌عنوان مثال، فرودگاه یک شهر کوچک را در نظر بگیرید که می‌خواهد یک ABHS نصب نماید.

---

[۱] System Architectural Design

[۲] Business Needs

جهت شناسایی نیازهای کسب‌وکار، تیم اطلاعاتی پیرامون فرودگاه فعلی و اهداف کسب‌وکاری فرودگاه جدید به دست می‌آورد. اطلاعات مربوط به فرودگاه جدید می‌تواند شامل جمعیت ماهیت کسب‌وکارهای آن شهر کوچک، تعداد پایانه‌ها، خروجی‌ها و محدوده‌های پذیرش مسافر و نیز حجم چمدان‌هایی که باید حمل شوند و سایر عوامل دیگر باشد. فعالیت جمع آوری اطلاعات به پرسش‌های زیر پاسخ می‌دهد.

۱. کسب‌وکاری که سیستم به دنبال خودکارسازی آن است چیست؟
۲. محیط یا زمینه سیستم چیست؟
۳. اهداف کسب‌وکار یا اهداف محصول کدامند؟
۴. موقعیت فعلی کسب‌وکار چیست و چگونه این کسب‌وکار در حال انجام است؟
۵. فرایندهای کسب‌وکار موجود کدامند و این فرایندها چگونه به هم مرتبط می‌شوند؟
۶. مشکلات سیستم فعلی چیست؟
۷. کاربران سیستم فعلی و سیستم آتی چه کسانی هستند؟
۸. مشتری و کاربران از سیستم آتی چه می‌خواهند؟ اولویت‌های کسب‌وکاری آن‌ها کدامند؟
۹. ملاحظات کیفی، کارایی و امنیتی کدامند؟

فعالیت جمع‌آوری اطلاعات از تکنیک‌های مختلف جمع‌آوری اطلاعات استفاده می‌کند.

۱. ارائه‌ی مشتری. ارائه‌ی مشتری یک روش موثر برای جمع‌آوری اطلاعات است که در ابتدای هر پروژه انجام می‌شود. یکی از مدیران یا نمایندگان ارشد از سوی مشتری، مروری بر روی کسب‌وکار جاری، مسائل و مشکلات شناخته شده آن، انتظارات مشتری و کاربران از سیستم واولویت‌های کسب‌وکاری ایشان را ارائه می‌نماید. فهرست پرسش‌هایی که در بالا اشاره شد می‌تواند به‌عنوان محور ارائه انتخاب شود. مدت این ارائه و پرسش و پاسخ‌های مربوطه نباید بیش از دو ساعت طول بکشد.
۲. مطالعه‌ی عملیات کسب‌وکار فعلی. تیم می‌تواند بازدیدی از محیط کسب‌وکاری داشته باشد. همچنین، تیم می‌تواند درخواست کند بازدید همراه با راهنما از اعمال کسب‌وکاری مشتری داشته باشد و در این حین فرم‌های کاغذی، شرح رویه‌های عملیاتی، سیاست‌ها و سایر موارد مرتبط را جمع‌آوری نماید و سپس آن‌ها را به دقت مطالعه نماید. این فعالیت‌ها در فهم موجودیت‌های کسب‌وکار مشتری، فرایندهای کسب‌وکار و گردش کارها به تیم کمک می‌کنند. گردش کارها ممکن است شامل گردش اطلاعات و گردش مواد بشوند.
۳. تحقیق از کاربر. تحقیق از کاربر، برای جمع‌آوری نقطه‌نظرات کاربران در مورد سیستم فعلی و انتظارات ایشان از سیستم جدید مفید است. پرسشنامه‌های تحقیق باید مختصر بوده و بر روی موارد با اهمیت تمرکز کنند. برای گروه‌های مختلف کاربران از پرسشنامه های تحقیق متفاوت استفاده کنید. مواردی که نیاز به شفاف‌سازی دارند بایستی شناسایی شده و در حین مصاحبه‌ها مطرح و روشن شوند.
۴. مصاحبه با کاربر. وقتی جمع‌آوری اطلاعات از طریق تحقیق مشکل باشد، از مصاحبه استفاده می‌شود. مصاحبه با کاربر همچنین برای روشن نمودن مواردی که در تحقیق از کاربر واضح نبوده‌اند نیز مفید می‌باشد. مصاحبه‌ها با افراد منتخب از میان کاربران صورت می‌گیرند. مصاحبه‌ها با افراد منتخب از بین کاربران، به‌صورت فردی یا دسته‌جمعی، انجام می‌گیرد. جلسات مصاحبه باید حدود یک ساعت به طول انجامد. قبل از جلسه مصاحبه لازم است فهرستی از مواردی که باید در حین مصاحبه مورد بحث قرار گیرند، آماده شود. این فهرست را می‌توان قبل از مصاحبه در اختیار کاربران نیز قرار داد.

۵. تحقیق در ادبیات[۱]. جست‌وجو و تحقیق در ادبیات اطلاعات بیشتری را در اختیار تیم توسعه می‌گذارد تا دامنه کاربرد موردنظر را بشناسند. تحقیق در ادبیات باید روی پروژه‌های مشابه، دانش دامنه، فرایندهای کسب‌وکار، مقررات دولتی و استانداردهای صنعتی متمرکز باشد.

۳-۳-۲- تعریف نیازمندی‌های سیستم

گام بعدی این است که از روی نیازهای کسب‌وکار شناسایی شده، نیازمندی‌های سیستم استنتاج گردند. برای مثال، قابلیت‌های ABHS برای ارضای نیازهای ABHS تعریف شده‌اند. ولی به‌دلیل قیدهای بودجه، موعد تحویل، فناوری، سیاسی و نیز ملاحظات هزینه-فایده، نمی‌توان تمام نیازها را برآورده نمود. گاهی یک مطالعه‌ی امکان‌پذیری انجام می‌شود تا مطمئن شویم تیم می‌تواند قابلیت‌های مدّنظر را با توجه به قیدهای بودجه و زمان، تحویل نماید. قابلیت‌هایی که سیستم باید داشته باشد در قالب نیازمندی‌های سیستم مدون می‌شوند. مثلاً، پروژه‌ی ABHS ممکن است برای ارضای نیازها، نیازمندی‌های متعددی را شناسایی کند. برای نمونه، برخی از این نیازمندی‌ها در ادامه ذکر شده‌اند. برای سادگی ارجاع نیازمندی‌ها شماره گذاری شده‌اند.

R1) ABHS باید بسته‌های بار را دریافت و مطابق با مقصد مسافران آن‌ها را به سکوهای خروجی و نیز بخش دریافت چمدان منتقل نماید.

R2) ABHS باید به کارکنان خطوط هوایی امکان بررسی وضعیت بسته و یافتن آن را بدهد.

R3) ABHS باید تمام بسته‌ها را بررسی و اشیا ممنوعه را تشخیص دهد.

R4) ABHS باید آماده خدمت‌رسانی به ۲۰۰۰۰ مسافر در روز باشد.

یکی از محصولات نهایی این مرحله، سند تبیین نیازمندی‌های سیستم است. این سند می‌تواند شامل قیودی باشد که فضای حل[۲] را محدود می‌کنند. برای مثال، ممکن است ABHS شامل قیدهایی در مورد فضای موجود برای ساخت نوار نقاله و نوار پرسرعت باشد. همچنین می‌توان قیودی روی نوع تجهیزات مورد استفاده داشت. محصول دیگر این مرحله، برنامه‌ی پروژه است. برنامه‌ی پروژه، نقاط عطف پروژه و زمان‌بندی فعالیت‌های توسعه را مشخص کرده و منابع را به فعالیت‌ها تخصیص می‌دهد. به‌علاوه، یک برنامه‌ی آزمون سیستم نیز ممکن است تهیه شود. برنامه‌ی آزمون، اهداف آزمون سیستم، رویه‌های آزمون و منابع مورد نیاز را مشخص می‌کند.

۳-۴ طراحی معماری سیستم

بعد از اینکه نیازمندی‌های سیستم شناسایی گردیدند، گام منطقی بعدی این است که سیستم به‌گونه‌ای طراحی شود که نیازمندی‌ها را برآورده نماید. ایده‌آل این است که سیستم توسط مهندسانی طراحی و پیاده‌سازی شود که در تمام زمینه‌های مهندسی مورد نیاز خبره هستند. ولی متاسفانه، چنین مهندسانی به سختی یافت می‌شوند و هزینه‌ی به کارگیری ایشان نیز زیاد است. به همین دلیل، معمولاً سیستم‌ها به‌صورت سلسله‌مراتبی از زیرسیستم‌ها شکسته می‌شوند تا امکان توسعه‌ی آن‌ها توسط مهندسانی از رشته‌های جداگانه وجود داشته باشد. برای مثال، زیرسیستم‌های برقی توسط مهندسان برق ساخته می‌شوند. زیرسیستم‌های مکانیکی توسط مهندسان مکانیک ساخته می‌شوند، و زیرسیستم‌های نرم‌افزاری توسط مهندسان نرم‌افزار توسعه می‌یابند. سپس در حین مرحله‌ی یکپارچه‌سازی و آزمون سیستم، این زیرسیستم‌ها با یکدیگر یکپارچه می‌شوند. با این روش، یافتن مهندسان واجد شرایط آسان‌تر است، همچنین پیچیدگی و هزینه‌های توسعه‌ی سیستم نیز کاهش می‌یابد. طراحی معماری سیستم فعالیت‌های زیر را، که به‌هم مرتبط می‌باشند، انجام می‌دهد.

---

[۱] Literature Survey

[۲] Solution Space

۱. تجزیه‌ی سیستم به سلسله‌مراتبی از زیرسیستم‌ها. در این گام، سیستم به سلسله‌مراتبی از زیرسیستم‌های نسبتاً مستقل تجزیه می‌شود. این کار باعث کاهش پیچیدگی و هزینه‌های توسعه‌ی سیستم می‌شود، چون طراحی و پیاده‌سازی زیرسیستم‌ها ساده‌تر می‌باشد.
۲. تخصیص نیازمندی‌های سیستم به زیرسیستم‌ها. این فعالیت نیازمندی‌های سیستم را بین زیرسیستم‌ها تقسیم می‌کند، با این هدف که تعداد نیازمندی‌های مشترک بین زیرسیستم‌ها کاهش یابد. بدین ترتیب، مسئولیت‌های زیرسیستم‌ها به وضوح تعریف می‌شود. به‌علاوه، نیازمندی‌هایی که به یک زیرسیستم واگذار می‌شود باید از نظر عملکردی به‌هم مرتبط باشند. عمل تخصیص توسط یک ماتریس ردیابی‌پذیری زیرسیستم-نیازمندی نشان داده می‌شود.
۳. به تصویر کشیدن معماری سیستم. این فعالیت طراحی معماری را به‌صورت نموداری متشکل از زیرسیستم‌ها و روابط بین زیرسیستم‌ها نشان می‌دهد.

۳-۴-۱- تجزیه‌ی سیستم[۱]

یک وظیفه‌ی مهم طراحی معماری سیستم، شناسایی زیرسیستم‌های سیستم است. برای این کار اغلب از یک روش بالابه‌پایین و تقسیم‌وحل استفاده می‌شود. به ویژه، در این روش سیستم به شکل سلسله‌مراتبی از زیرسیستم شکسته می‌شود. این روش باعث کاهش پیچیدگی توسعه‌ی سیستم می‌شود. زیرا طراحی و پیاده‌سازی هر یک از زیرسیستم‌ها ساده‌تر خواهد شد. روش‌های مختلفی برای تجزیه‌ی یک سیستم وجود دارند، لذا، نتیجه‌ی کار یکتا نخواهد بود. هدف از تجزیه‌ی سیستم برآورده ساختن اهداف زیر می‌باشد:

۱. نتیجه باید به‌گونه‌ای باشد که تیم‌های مهندسی جدا بتوانند زیرسیستم‌ها را توسعه دهند. این کار باعث کاهش هزینه‌های توسعه‌ی سیستم می‌شود، زیرا نیاز به تعداد مهندس‌هایی که باید در رشته‌های مختلف مهندسی خبره باشند را کم می‌کند. علاوه‌براین، باعث آسان‌تر شدن ارتباط میان تیم‌ها و لذا کاهش سربار ارتباطی می‌شود.
۲. نتیجه باید استفاده از قطعات تجاری آماده فروش[۲] را آسان کند. منظور از COTS محصولات آماده‌ای است که می‌توان آن‌ها را خرید یا تهیه کرد. استفاده از COTS بهره‌وری و کیفیت را افزایش داده و در عین حال هزینه و زمان تا تحویل محصول به بازار را کاهش می‌دهد. برای مثال، در طراحی ABHS باید استفاده از COTS در سیستم‌های نقاله، نوارهای پرسرعت، بارکدخوان‌ها، و نرم‌افزار پذیرش بار مدّنظر قرار گیرد، به‌جای این‌که این محصولات بخواهند از نو ساخته شوند.
۳. نتیجه باید نیازمندی‌های سیستم را (حتی‌الامکان) افراز نماید. بدین معنی که، تعداد اندکی از زیرسیستم‌ها نیازمندی‌ها را با یکدیگر به اشتراک گذارند. این کار باعث می‌شود احتمال برآورده نشدن برخی نیازمندی‌ها، به‌دلیل سوءتفاهم میان تیم‌هایی که نیازمندی‌ها را به اشتراک گذاشته‌اند، کم شود.
۴. وظایف هر زیرسیستم باید به خوبی مشخص شده باشد. این کار باعث می‌شود فهم زیرسیستم آسان‌تر شود. برای مثال، زیرسیستم دریافت بسته‌ها، با دریافت بار سروکار دارد. زیرسیستم نقاله، مسئول حرکت بارها درون پایانه است.
۵. زیرسیستم‌ها باید نسبتاً مستقل باشند. بدین معنی که، تغییر یک زیرسیستم بر روی دیگر زیرسیستم‌ها تاثیری نگذارد. چنین طراحی پیمانه‌ای، نگهداری سیستم را آسان می‌کند. برای مثال، زیرسیستم دریافت بسته و زیرسیستم نوار نقاله، دو زیرسیستم مستقل هستند. هر کدام از آن‌ها را می‌توان بدون تاثیر بر دیگری، جایگزین نمود.
۶. یکپارچه‌سازی زیرسیستم‌ها باید آسان باشد. منظور این است که زیرسیستم‌ها باید واسطه‌ها و رفتارهای تعاملی ساده‌ای داشته باشند، در نتیجه ارتباط بین زیرسیستم‌ها آسان خواهد شد. همچنین یکپارچه‌سازی، آزمون یکپارچگی، و نگه‌داری سیستم تسهیل می‌گردد.

---

[۱] System Decomposition

[۲] Commercial Off-the-Shelf (COTS)

تجزیه‌ی سیستم، چند راهبرد که ممکن است روی یکدیگر تاثیر داشته باشند را به‌کار می‌بندد. بنابراین، این راهبردها را باید تا دستیابی به نتیجه مطلوب به‌طور تکراری اعمال نمود. این راهبردها عبارتنداز:

۱. تجزیه‌ی سیستم بر حسب وظایف سیستم.
۲. تجزیه‌ی سیستم برحسب رشته (شاخه تخصصی).
۳. تجزیه‌ی سیستم بر حسب معماری موجود.
۴. تجزیه‌ی سیستم بر حسب واحدهای کارکردی سازمان.
۵. تجزیه‌ی سیستم بر حسب مدل‌های کاربرد.

تجزیه‌ی سیستم بر حسب وظایف آن یکی از راهبردهای متداول است. در این روش، وظایف سیستم از روی نیازمندی‌های سیستم شناسایی می‌شوند. بدین‌صورت، نیازمندی‌های سیستم از نظر کارکردی به خوشه‌های تقریباً مجزا افراز می‌شوند. سپس بر اساس همین خوشه‌های کارکردی، زیرسیستم‌ها تعیین می‌شوند. مثلاً، اگر سه خوشه‌ی کارکردی پیدا شوند، سیستم نیز به سه زیرسیستم کارکردی شکسته می‌شود. شکل ۳-۲(الف) تقسیم سیستم به سه زیرسیستم به نام‌های Subsys-1 ، Subsys-2 و Subsys-3 را نشان می‌دهد. پس از آن، زیرسیستم‌ها نیز به زیرسیستم‌های خاص دامنه، یعنی زیرسیستم‌های سخت‌افزاری، نرم‌افزاری، و منابع انسانی تجزیه می‌شوند. این روش در مواقعی استفاده می‌شود که تیم‌های توسعه برای ساخت زیرسیستم‌های کارکردی موجود باشند یا این‌که زیرسیستم‌های کارکردی را بتوان از یک شرکت ثالث[1] تهیه نمود. برخی مواقع مهندسان باید نیازمندی‌های سیستم را به نیازمندی‌های سطح پایین بشکنند تا زیرسیستم‌های نسبتاً مستقل بدست آیند. برای نشان دادن این موضوع، نیازمندی‌های زیر را برای ABHS فرض کنید:

R1) سیستم ABHS باید بسته‌های بار را دریافت کند و آن‌ها را مطابق با مقصد مسافران، به سکوهای خروجی و نیز بخش تحویل بار منتقل نماید. شکل ۳-۲ تجزیه‌ی سیستم به‌صورت سلسله‌مراتبی از زیرسیستم‌ها این نیازمندی‌ها شامل اعمالی چون دریافت بسته‌ها و حمل بسته‌ها می‌شود. چنان‌چه فرودگاه بیش از یک پایانه داشته باشد، در این صورت عمل حمل بسته‌ها شامل نوار نقاله‌ها و نوارهای پرسرعت خواهد بود. لذا نیازمندی‌ها باید تجزیه شوند. تجزیه‌ی نیازمندی‌ها باید به‌گونه‌ای باشد که نیازمندی‌های سطح پایین و نیازمندی‌های سطح بالا معادل باشند. بدین معنی که نیازمندی‌های سطح بالا و سطح پایین هر دو قابلیت‌های مشابهی را بیان کنند. فرض کنید که هر کدام از محل‌های دریافت بسته و درگاه‌های خروجی و نیز درگاه‌های پروازهای ورودی و محل‌های تحویل بار در یک پایانه قرار دارند. بنابراین، نیازمندی R1 را می‌توان به‌صورت زیر تجزیه نمود:

R1.1) سیستم ABHS باید به مأموران خطوط هوایی امکان دریافت بسته را بدهد.

R1.2) سیستم ABHS باید بسته‌ها را به مقصدشان در فرودگاه حمل کند.

R1.2.1) سیستم ABHS باید بسته‌ها را از محل‌های دریافت به درگاه‌های خروجی حمل کند.

R1.2.2) سیستم ABHS باید بسته‌ها را از درگاه‌های پروازهای ورودی به محل‌های تحویل بار حمل کند.

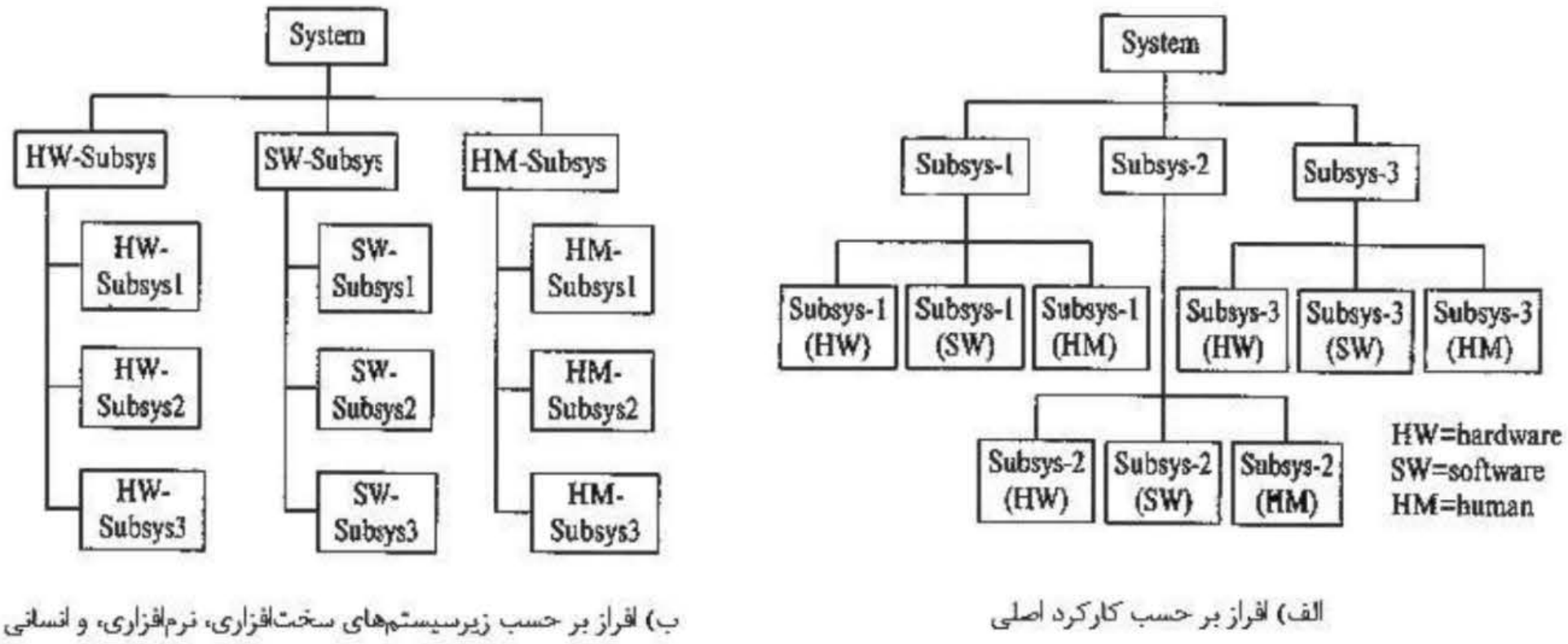
R1.2.3) سیستم ABHS باید بسته‌های مسافران عبوری را از درگاه‌های پروازهای ورودی به درگاه‌های پروازهای خروجی حمل نماید.

R1.2.4) سیستم ABHS باید با استفاده از نوار نقاله‌ها بسته‌ها را درون یک پایانه جابجا کند.

R1.2.5) سیستم ABHS باید با استفاده از DCVهایی که روی نوارهای پرسرعت حرکت می‌کنند، بسته‌ها را بین پایانه‌ها جابجا کند.

---

[1] Third-Party



ب) افراز بر حسب زیرسیستم‌های سخت‌افزاری، نرم‌افزاری، و انسانی

الف) افراز بر حسب کارکرد اصلی

شکل ۳-۲: تجزیه‌ی سیستم به‌صورت سلسله‌مراتبی از زیرسیستم‌ها

R1.3) سیستم ABHS باید حمل و نقل بسته‌ها را درون و بین پایانه‌ها کنترل نماید.

نیازمندی R4 یک نیازمندی کارایی بوده و به چندین زیرسیستم مربوط می‌شود. این نیازمندی نیز باید به‌گونه‌ای تجزیه شود که نیازمندی‌های سطح پایین را بتوان به زیرسیستم‌هایی واگذار نمود. به‌عنوان مثال، در این نیازمندی‌های سطح پایین باید سرعت مورد نیاز و حجم بسته‌ها را برای زیرسیستم‌ها مشخص نمود. مواردی چون تعداد پایانه‌ها، محل‌های دریافت، سیستم‌های نقاله برای هر پایانه، ساعات فعالیت، نیز باید معلوم بشوند. در موقع تجزیه باید درصد معینی از افزونگی را هم برای مواجه با حجم بالای تقاضاها در فصل تعطیلات در نظر بگیرد. در ادامه مثال‌هایی از نیازمندی‌های کارایی سطح پایین آورده شده‌اند.

R4.1) در هر محدوده‌ی دریافت باید بتوان ۱۱۵۰ بسته را در روز دریافت نمود.
R4.2) هر مأمور دریافت باید به‌طور میانگین سه مسافر در دقیقه پذیرش نماید.
R4.3) هر سخت‌افزار نقاله باید در ساعت ۵۰۰ بسته بار را پویش و حمل نماید.
R4.4) نرم‌افزار کنترل ABHS باید امکان پردازش ۲۳۰۰ دریافت بسته در روز و ۱۰۰۰ درخواست پویش بارکد در ساعت را داشته باشد.

پس از آن، نیازمندی‌های سیستم به‌صورت خوشه‌های کارکردی افراز می‌شوند. بدین معنی که نیازمندی‌هایی که کارکرد مشابهی انجام می‌دهند در یک گروه کنار هم قرار می‌گیرند و یک خوشه‌ی کارکردی را تشکیل می‌دهند. خوشه‌های کارکردی خود منشا استنتاج زیرسیستم‌های کارکردی می‌باشند. شکل ۳-۳ خوشه‌ها و زیرسیستم‌های کارکردی که از نیازمندی‌های سیستم استنتاج شده‌اند را نشان می‌دهد.

یک راهبرد دیگر این است که سیستم برحسب رشته‌های مهندسی مورد نیاز تجزیه شود. در نتیجه‌ی این کار، نام زیرسیستم‌ها مطابق با نام رشته‌های مربوطه انتخاب می‌شود، مثل زیرسیستم برقی، زیرسیستم الکترونیکی، زیرسیستم مکانیکی، زیرسیستم نرم‌افزاری. در شکل ۳-۲(ب) کل سیستم به زیرسیستم‌های سخت‌افزاری، نرم‌افزاری، و انسانی تقسیم گردیده است. سپس این‌ها به زیرسیستم‌های کارکردی تجزیه می‌شوند. این راهبرد وقتی به‌کار می‌رود که قرار باشد زیرسیستم‌های سخت‌افزاری، نرم‌افزاری، و انسانی به‌ترتیب توسط تیم‌ها یا بخش‌های مهندسی سخت‌افزار، مهندسی نرم‌افزار، و مهندسی انسانی تهیه شوند.

خوشه کارکردی	توصیف کارکردی	نیازمندی‌های سیستم	زیرسیستم‌های کارکردی
دریافت بسته	این خوشه کارکردی دریافت بسته را انجام می‌دهد.	R1.1, R4.1, R4.2	زیرسیستم دریافت بسته
نقاله	این خوشه کارکردی مسئول حرکت بسته درون پایانه است.	R1.2.1, R1.2.2, R1.2.3, R1.2.4, R4.3	زیرسیستم نقاله
نوار پرسرعت	این خوشه کارکردی بسته را بین پایانه‌ها حمل می‌کند.	R1.2.3, R1.2.5	زیرسیستم نوار پرسرعت
کنترل نرم‌افزاری	این خوشه کارکردی با کنترل سخت‌افزار بسته‌ها را بین پایانه‌ها حمل می‌کند.	R1.3, R4.4	زیرسیستم کنترل نرم‌افزار

شکل ۳-۳: خوشه‌های کارکردی و زیرسیستم‌های کارکردی

تجزیه‌ی سیستم بر طبق یک معماری موجود وقتی استفاده می‌شود که هدف پروژه گسترش یا بهبود یک سیستم موجود باشد. این راهبرد می‌تواند هزینه‌های توسعه را کاهش دهد و جلوی ریسک‌های احتمالی که به‌دلیل تغییر معماری سیستم موجود ممکن است رخ دهند را بگیرد. همچنین می‌توان سیستم را برحسب واحدهای کارکردی سازمان تجزیه نمود. برای مثال، سیستم دستی مدیریت بار مسافران در فرودگاه یک شهر کوچک ممکن است شامل بخش‌های دریافت، حمل، و تحویل بار باشد. وجود این بخش‌ها می‌گوید که سیستم جدید نیز می‌تواند دارای سه زیرسیستم اصلی باشد.

چنان‌چه برای کمک به فهم برنامه یا سیستم موجود، مدل‌هایی از برنامه یا سیستم ساخته شده باشد، می‌توان از این مدل‌ها در امر تجزیه راهنمایی گرفت. برای مثال، ممکن است برای فهم گردش کار سیستم جاری مدیریت بار فرودگاه، مدل‌هایی ساخته شده باشند. در این مدل‌ها ممکن است گفته شده باشد که بسته‌ها به‌طور دستی دریافت و به سکوهای خروجی حمل می‌شوند. به‌علاوه، بسته‌ها از پروازهای ورودی تخلیه و به محل‌های تحویل بار و نیز سکوهای خروجی حمل می‌شوند. از روی این مدل‌ها می‌توان زیرسیستم‌ها را شناسایی کرد و آن را به‌عنوان تجزیه‌ی اولیه‌ای از سیستم به حساب آورد. سپس این تجزیه‌ی اولیه را پالایش کرده و تکمیل نمود تا به شکل سلسله‌مراتبی از زیرسیستم‌ها درآید.

۳-۴-۲- تخصیص نیازمندی‌ها

وقتی سیستم به‌صورت سلسله‌مراتبی از زیرسیستم‌ها تجزیه شد، نیازمندی‌های سیستم به زیرسیستم‌ها واگذار می‌شود. در موقع تخصیص، عوامل متعددی مانند هزینه‌های عملیاتی و توسعه، کارایی، کیفیت خدمات، مقرون به صرفه بودن، و آسانی تغییر را باید مدنظر گرفت. برای مثال، برخی وظایف سیستم می‌تواند توسط سخت‌افزار یا نرم‌افزار پیاده‌سازی گردد. به‌طور کلی، مدارهای مجتمع خاص- کاربرد کارایی بهتری ارائه می‌دهند. اگر کارایی مطرح نباشد، برخی از نیازمندی‌های سیستم را می‌توان به‌جای زیرسیستم سخت‌افزاری به زیرسیستم نرم‌افزاری تخصیص داد. تخصیص نیازمندی‌های سیستم بسته به نتایج گام‌های پیشین دارد. چنان‌چه زیرسیستم‌ها از روی خوشه‌های کارکردی بدست آمده باشند، در این‌صورت تخصیص در واقع همان نگاشت از خوشه‌های کارکردی به زیرسیستم‌ها می‌باشد. این همان چیزی است که در مورد مثال ABHS انجام شد و در شکل ۳-۳ نشان داده است.

تخصیص نیازمندی‌های سیستم به زیرسیستم‌ها توسط یک ماتریس ردیابی‌پذیری به تصویر کشیده می‌شود که در شکل ۳-۴

نشان داده شده است. در این جدول، ردیف‌ها بیان‌گر نیازمندی‌ها بوده و ستون‌ها، زیرسیستم‌ها را نشان می‌دهند. خانه‌هایی از جدول که علامت خورده‌اند تخصیص نیازمندی‌ها را نشان می‌دهند. در یک پروژه‌ی واقعی، جدول ردیابی‌پذیری چون شامل صدها نیازمندی و تعداد زیادی زیرسیستم خواهد بود، جدول بزرگی خواهد شد. پس، شکل ۳-۴ تنها یک نمونه است. استفاده از ماتریس محاسن زیادی دارد:

۱. می‌توان مطمئن شد که هر کدام از نیازمندی‌ها به یک زیرسیستم تخصیص داده شده است. منظور این است که در هر ردیف مربوط به یک نیازمندی سطح برگ (آخرین سطح نیازمندی‌ها)، دست کم یک علامت ضربدر (×) وجود دارد. اگر در یک ردیف مربوط به سطح برگ علامت ضربدر نباشد، معلوم می‌شود که آن نیازمندی به هیچ زیرسیستمی واگذار نشده است.
۲. می‌توان بررسی کرد که آیا تخصیص مناسب است؟ از واگذاری یک نیازمندی سیستم به چند زیرسیستم باید پرهیز نمود زیرا تقسیم عملکرد در بین زیرسیستم‌ها کار دشواری است. با شمردن تعداد ضربدرها در هر ردیف از جدول ردیابی‌پذیری می‌توان چنین مواردی را کشف نمود. چنان‌چه ردیفی شامل چند ضربدر باشد، نشان‌دهنده‌ی این است که نیازمندی به چند زیرسیستم واگذار شده است. در این حالت، باید نیازمندی را تجزیه نمود.
۳. می‌توان دید چه نیازمندی‌هایی به یک زیرسیستم واگذار شده است. ضربدرهای یک ستون از جدول بیان‌گر نیازمندی‌هایی است که به زیرسیستم مندرج در آن ستون واگذار شده است. زیرسیستم‌ها باید نیازمندی‌های واگذار شده را برآورده نمایند.
۴. می‌توان بررسی کرد که آیا به یک زیرسیستم تعداد بسیار زیادی مسئولیت واگذار شده است؟ چنان‌چه در یک ستون از جدول تعداد ضربدرها زیاد باشد، بدین معنی است که به زیرسیستم مربوطه مسئولیت‌های متعددی واگذار شده است. در این صورت، می‌توان زیرسیستم را تجزیه نمود و نیازمندی‌ها را در میان زیرسیستم‌های سطح پایین‌تر توزیع نمود.
۵. می‌توان چسبندگی[۱] کارکردی زیرسیستم‌ها را بررسی نمود. نیازمندی‌هایی که به یک زیرسیستم واگذار می‌شوند باید از نظر کارکردی (عملکردی) به‌هم مرتبط باشند. درصورتی که نیازمندی‌های متناظر با ضرب‌درهای مندرج در یک ستون از جدول به‌هم مرتبط نیستند، در این‌صورت زیرسیستم متناظر با آن ستون چسبندگی کارکردی کمی دارد. لذا باید نیازمندی‌ها را به چند خوشه‌ی کارکردی افراز نمود. زیرسیستم نیز باید به‌صورت تعدادی زیرسیستم کارکردی که متناظر با خوشه‌های کارکردی بدست آمده هستند، تجزیه شود.

۳-۴-۳- نمودارهای طراحی معماری

ساختن مدل‌های برنامه (کاربرد) و مدل‌های سیستم در حین طراحی سیستم امری متداول است. این مدل‌ها به تیم کمک می‌کنند دامنه‌ی کاربرد، فرایندهای کسب‌وکار، و گردش کارها را درک و تحلیل نموده و بتواند مسائل را شناسایی و راه‌حل‌هایی را برای این مسائل توسعه داده و ارزیابی نماید. برای ترسیم جنبه‌های مختلف از کاربرد و سیستم نمودارهای مختلفی استفاده می‌شوند. در مدل‌سازی سیستم و نیز در طراحی از نمودارهای بلوکی، زبان مدل‌سازی یکنواخت گسترش‌های آن، زبان مدل‌سازی سیستم[۲]، و نمودارهای جریان داده به‌طور گسترده‌ای استفاده می‌شود.

در نمودارهای بلوکی از مستطیل برای نمایش زیرسیستم‌ها و اجزای، و از یال‌های جهت‌دار برای نشان‌دادن گردش‌کار بین زیرسیستم‌ها و اجزای استفاده می‌شود. به‌عنوان مثال، شکل ۳-۵ یک نمودار بلوکی سطح بالا را برای ABHS نشان می‌دهد. این نمودار بلوکی نشان می‌دهد که ABHS شامل زیرسیستم‌های پایانه است که به یک زیرسیستم نوار پرسرعت متصل می‌باشند.

---

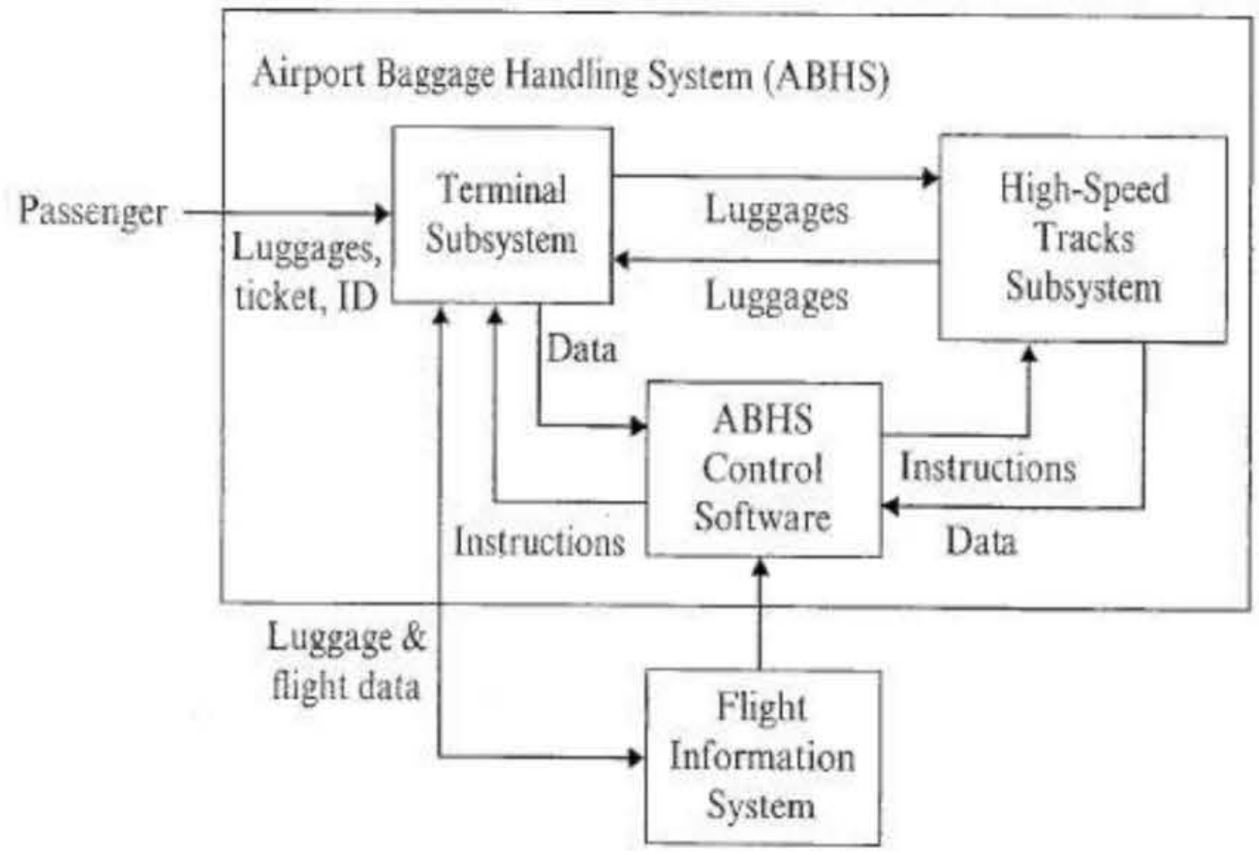
[۱] Cohesion

[۲] System Modeling Language (SysML)

	زیرسیستم بسته‌ها	زیرسیستم نقاله	زیرسیستم نوار پرسرعت	زیرسیستم کنترل نرم‌افزاری
R1				
R1.1	X			
R1.2				
R1.2.1		X		
R1.2.2		X		
R1.2.3		X	X	
R1.2.4		X		
R1.2.5			X	
R1.3				X

شکل ۳-۴: جدول ردیابی‌پذیری نیازمندی‌ها به زیرسیستم‌ها

مسافران بارهایشان را تحویل زیرسیستم پایانه می‌دهند. زیرسیستم نوار پرسرعت بارها را بین پایانه‌ها حمل می‌کنند. هردوی این زیرسیستم‌ها با نرم‌افزار کنترل ABHS در تعامل هستند، و آن هم با سیستم اطلاعاتی فرودگاه تعامل دارد.

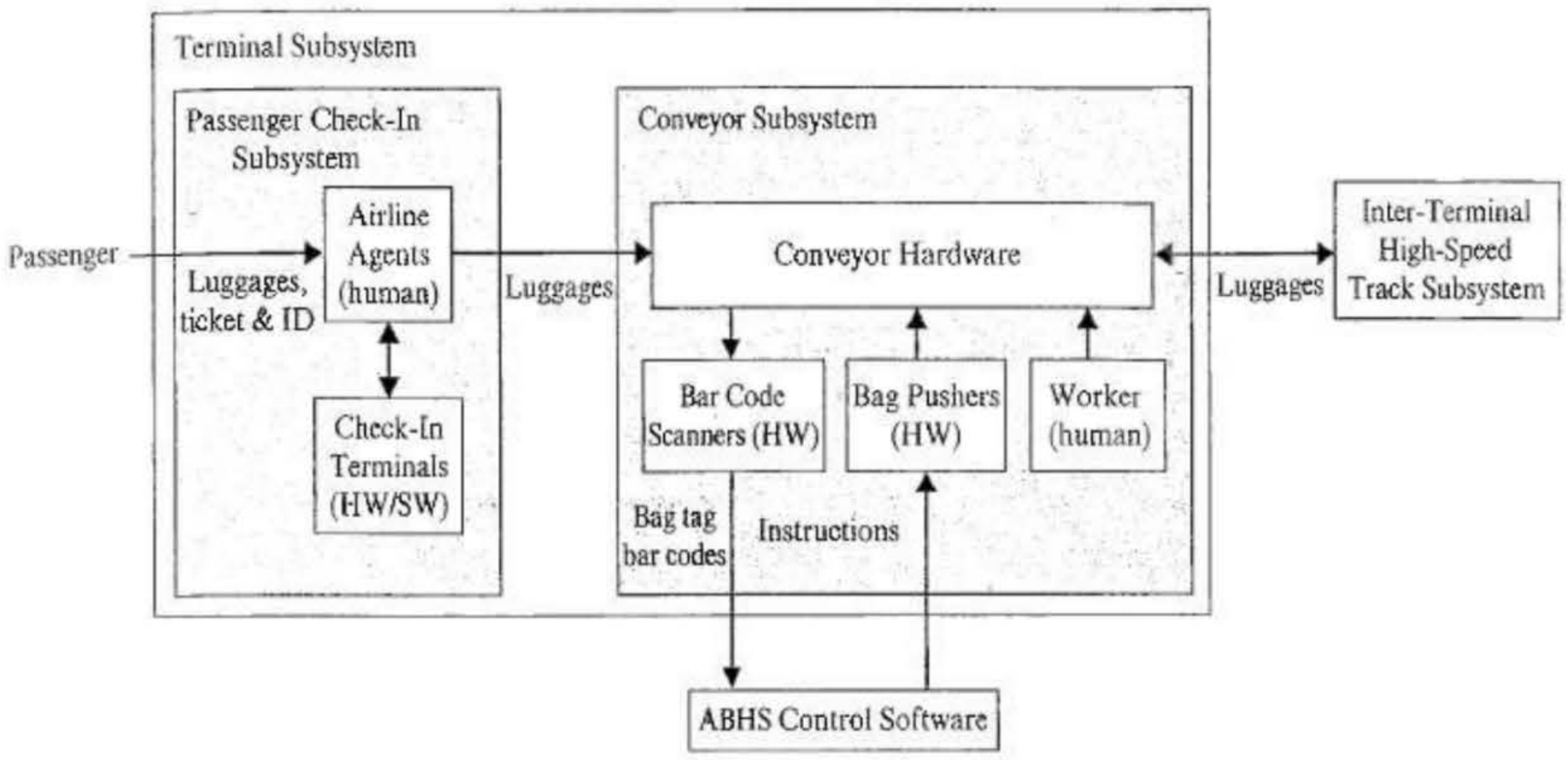


شکل ۳-۵: نمودار بلوکی برای یک سیستم مدیریت بار در فرودگاه

در شکل ۳-۶ پالایشی از زیرسیستم پایانه نشان داده شده است، که در آن زیرسیستم‌های سخت‌افزاری، نرم‌افزاری، و انسانی به چشم می‌خورند.

زبان مدل‌سازی یکنواخت در ابتدا برای مدل‌سازی سیستم‌ها و کاربردهای نرم‌افزاری ایجاد شد. البته عمومیت این زبان باعث شد تا برای مدل‌سازی سیستم نیز ابزاری مناسب باشد. برای این کار از سازوکار استریوتایپ[1] که در UML تدارک

[1] Stereotype



شکل ۳-۶: نمودار بلوکی که پالایش سیستم را نشان می‌دهد

دیده شده است استفاده می‌شود. به‌عنوان مثال، شکل ۳-۷ یک نمودار مولفه را نشان می‌دهد که سخت‌افزار و نرم‌افزار یک سیستم ارتباطات رادیویی[۱] را به تصویر می‌کشد. در این شکل، استریوتایپ «Subsystem» برای گسترش ساختار مدل‌سازی مولفه استفاده شده تا بتوان یک زیرسیستم را مدل‌سازی کرد. نمودار نشان می‌دهد که هر RCS شامل سه زیرسیستم می‌شود: زیرسیستم ایستگاه زمینی، زیرسیستم مدیریت حساب‌ها، و زیرسیستم واحدهای سیّار، کار RCS شبیه به یک شبکه‌ی تلفن همراه است، با این اختلاف که RCS تنها یک ایستگاه زمینی با دریافت‌کننده‌های توان بالا دارد. دریافت‌کننده‌های توان بالا می‌توانند محدوده‌ی بسیار گسترده‌تری از یک ایستگاه زمینی در یک شبکه‌ی تلفن همراه را پوشش دهند. به کمک رله‌کردن دریافت‌کننده‌های توان بالا، واحدهای سیار می‌توانند در محدوده‌ی خدماتی موردنظر با یکدیگر ارتباط داشته باشند. شکل نشان می‌دهد که هر واحد سیار شامل یک زیرسیستم سخت‌افزاری سیار و نیزیک زیرسیستم نرم‌افزاری سیار می‌شود. سخت‌افزار رویدادها را به نرم‌افزار می‌فرستد، و او هم دستوراتی برای عملکرد سخت‌افزار صادر می‌نماید. برای مثال، وقتی کاربری تماس می‌گیرد، رویداد به نرم‌افزار ارسال می‌شود. نرم‌افزار نیز به سخت‌افزار دستور می‌دهد که از طریق یک کانال فرکانس رادیویی به نام اتصال هوایی[۲] با ایستگاه زمینی تماس بگیرد. دریافت‌کننده‌های توان بالا در ایستگاه زمینی درخواست را دریافت نموده و آن را به سخت‌افزار کنترلگر ارسال می‌نمایند، که آن هم درخواست را به نرم‌افزار کنترلگر ارجاع می‌دهد. نرم‌افزار کنترلگر درخواست تماس را اعتبارسنجی نموده و در صورتی که تماس‌گیرنده و مخاطب جزء مشترکین معتبر باشند، به سخت‌افزار و به دریافت‌کننده‌ها دستور می‌دهد که ارتباط را برقرار نمایند.

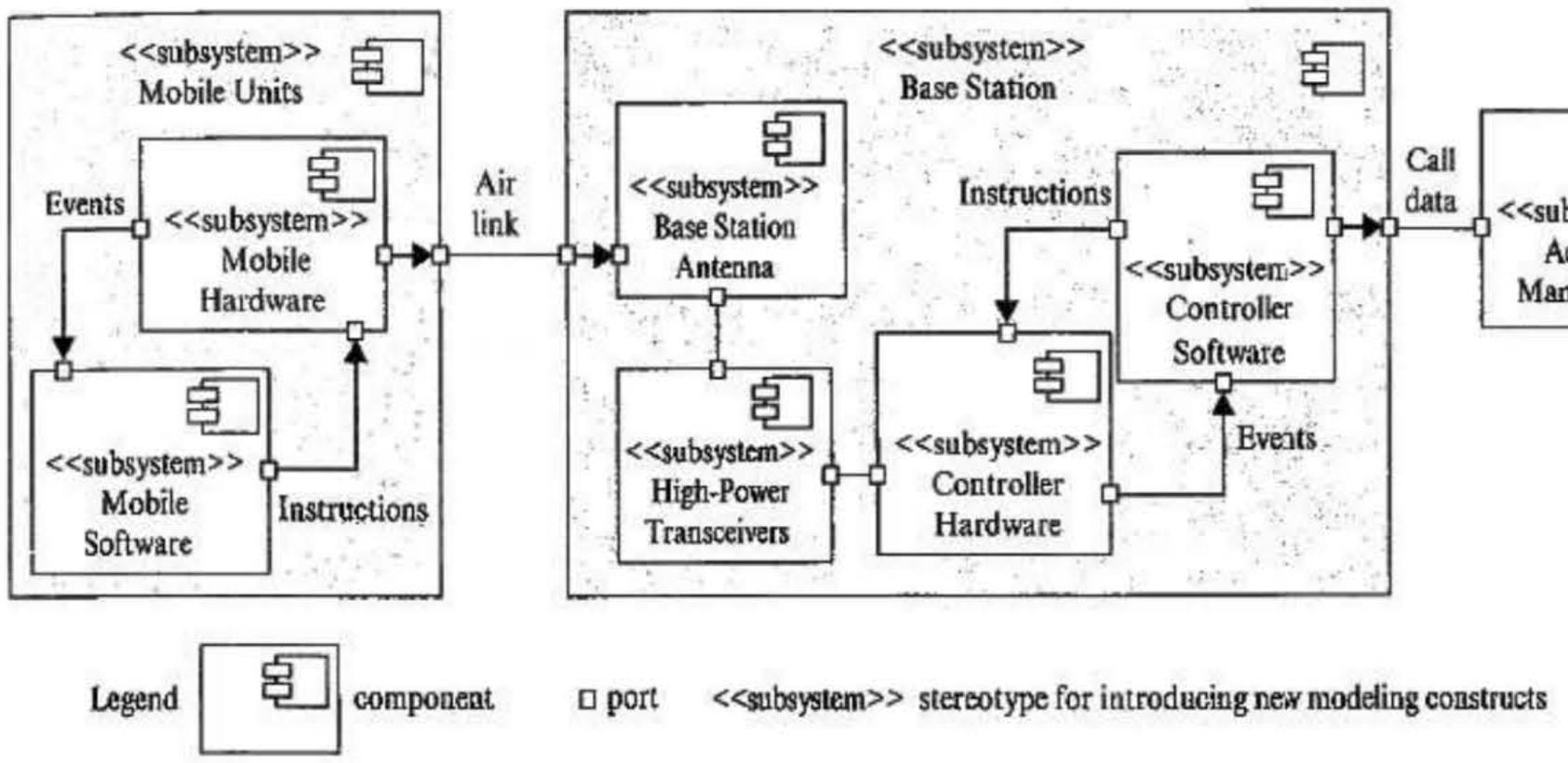
توانایی UML در پشتیبانی از مدل‌سازی سیستم منجر به ایجاد گسترشی از UML به نام زبان مدل‌سازی سیستم گردیده است. نُه نمودار SysML و چگونگی ارتباط آن‌ها با UML در شکل ۳-۸ خلاصه شده‌اند. آخرین ستون جدول، نشان می‌دهد کدام فصل از این کتاب نمودار UML مربوطه را پوشش می‌دهد.

برای نمونه، شکل‌های ۳-۹ و۳-۱۰ به‌ترتیب نمودارهای تعریف بلوک و نمودار داخل بلوک از SysML را که برای ABHS ترسیم شده‌اند نشان می‌دهند. این نمودارها، به‌ترتیب متناظر با نمودار بلوکی که در شکل‌ها ۳-۵ و ۳-۶ نشان داده شده‌اند،

---

[۱] Radio Communication System (RCS)

[۲] AirLink



شکل ۳-۷: مدل‌سازی سیستم با استفاده از یک نمودار مولفه‌ی استریوتایپ‌دار

هستند.

گاهی اوقات، علاوه بر جریان اطلاعات، مایلیم جریان مواد [۱] را نیز در مدل نشان دهیم. برای مثال، ABHS بسته‌ها را از محدوده‌ی دریافت به سکوهای خروجی و از سکوهای ورودی به محدوده‌ی تحویل بار منتقل می‌کند. به این قبیل جریان‌ها، جریان مواد می‌گوییم. در شکل‌های ۳-۵ و ۳-۶ جریان اطلاعات و جریان مواد با نمادهای مشابهی نشان داده شده‌اند. نمودارهای SysML که در شکل‌های ۳-۹ و ۳-۱۰ ترسیم شده‌اند نیز از یک نماد برای نمایش جریان داده و جریان مواد استفاده کرده‌اند، ولی جهت تمایز بهتر است. جریان مواد به شکل دیگری مدل‌سازی شود. به‌عنوان مثال، شکل ۳-۱۱ یک نمودار جریان داده را برای یک سیستم کتابخانه نشان می‌دهد. پیکان‌های بلوکی، جریان مواد را نمایش می‌دهند، بیضی‌ها بیان‌گر وظایف یا اعمال پردازشی هستند، نام موجودیت یا زیرسیستمی که آن عمل را پردازش می‌کند در بخش پایین بیضی نوشته شده است. بخش بالای بیضی نیز شماره‌ی وظیفه را نشان می‌دهد. جریان مواد در موقع تخصیص نیازمندی‌های سیستم به زیرسیستم‌های سخت‌افزاری، نرم‌افزاری، و انسانی مفید خواهد بود. برای مثال، جریان مواد در شکل ۳-۱۱ پیشنهاد می‌دهد که وظایفی که با جریان مواد مانند پیش‌خوان دریافت، اتاق فهرست‌نویسی، و قفسه‌های کتاب، سروکار دارند، باید به زیرسیستم‌های سخت‌افزاری و انسانی مناسب تخصیص داده شوند.

۳-۴-۴- تبیین وظایف و واسط زیرسیستم‌ها

در این گام کارکردی هرکدام از زیرسیستم‌ها و چگونگی تعامل آن‌ها با یکدیگر مشخص می‌شود. کارکردی‌ها با توجه به نیازمندی‌هایی از سیستم که به زیرسیستم‌ها واگذار شده است، مشخص می‌شوند. در این گام نیازمندی‌های واگذارشده به هر زیرسیستم پالایش نیز می‌شوند.

واسط‌های بین زیرسیستم‌ها مشخص می‌کند که زیرسیستم‌ها چگونه به‌هم متصل شده و با یکدیگر ارتباط دارند. رفتار تعاملی نیز توالی پیام‌های تبادل‌شده بین زیرسیستم‌ها را نشان می‌دهد و به تیم‌هایی که زیرسیستم‌ها را می‌سازند این امکان را می‌دهد که بدانند چه رفتار تعاملی و چه واسط‌هایی را باید انتظار داشته و فراهم نمایند. در سیستم‌های جاسازی‌شده، واسط‌های

[۱] Material Flows

SysML	UML	شرح	توضیح	فصل
نمودار فعالیت (Activity Diagram)	Activily Diagram	با استفاده از جریان‌کار، فعالیت‌هایی که به‌هم مرتبط هستند را مدل‌سازی می‌کند. رابطه‌هایی چون توالی، استثناء همگانی، و همروندی را نشان می‌دهد.	نمودار فعالیت SysML گسترش‌یافته‌ی نمودار فعالیت UML است.	۱۴
نمودار تعریف بلوک (Block Definition Diagram)		عناصر ساختاری (که بلوک نامیده می شوند) و تشکیل و طبقه‌بندی آن‌ها را مدل‌سازی می‌کند	نمودار تعریف بلوک گسترش‌یافته‌ی نمودار کلاس UML است.	
نمودار داخل بلوک (Internal Block Diagram)		اتصالات و واسطهای مابین عناصر داخلی یک بلوک را مدل‌سازی می‌کند	نمودار بلوک داخلی گسترش‌یافته‌ی نمودار ساختار مرکب UML است.	
نمودار بسته (Package Diagram)	نمودار بسته	سازمان‌دهی منطقی موضوعات مدل‌سازی و مصنوعات نرم‌افزار را مدل‌سازی می‌کند.	همان نمودار UML است	
نمودار عملی (Pragmatic Diagram)		محدودیت‌هایی را که برای پشتیبانی از تحلیل مهندسی لازم است مشخص می‌کند.		
نمودار نیازمندی (Requirement Diagram)		نیازمندی‌های مبتنی بر متن و رابطه‌ی بین آن‌ها را با دیگر نیازمندی‌ها و مصنوعاتی از قبیل عناصر طراحی، موارد آزمون و غیره مدل‌سازی می‌کند.		
نمودار توالی (Sequence Diagram)	نمودار توالی	رفتار تعامل بین اشیا را به‌ترتیب زمانی مدل‌سازی می‌کند.	همان نمودار UML است.	۹
نمودار حالت (State Diagram)	نمودار حالت	رفتار وابسته به حالت هر شئ را مدل‌سازی می‌کند.	همان نمودار UML است.	۱۳
نمودار مورد کاربرد (Use Case Diagram)	نمودار مورد کاربرد	عملکردها یا فرایندهای کسب و کاری یک کاربرد یا یک سیستم را باضافه‌ی رابطه‌ی آن‌ها با موجودیت‌های خارجی به‌نام کنشگر مدل‌سازی می‌کند.	همان نمودار UML است.	۷

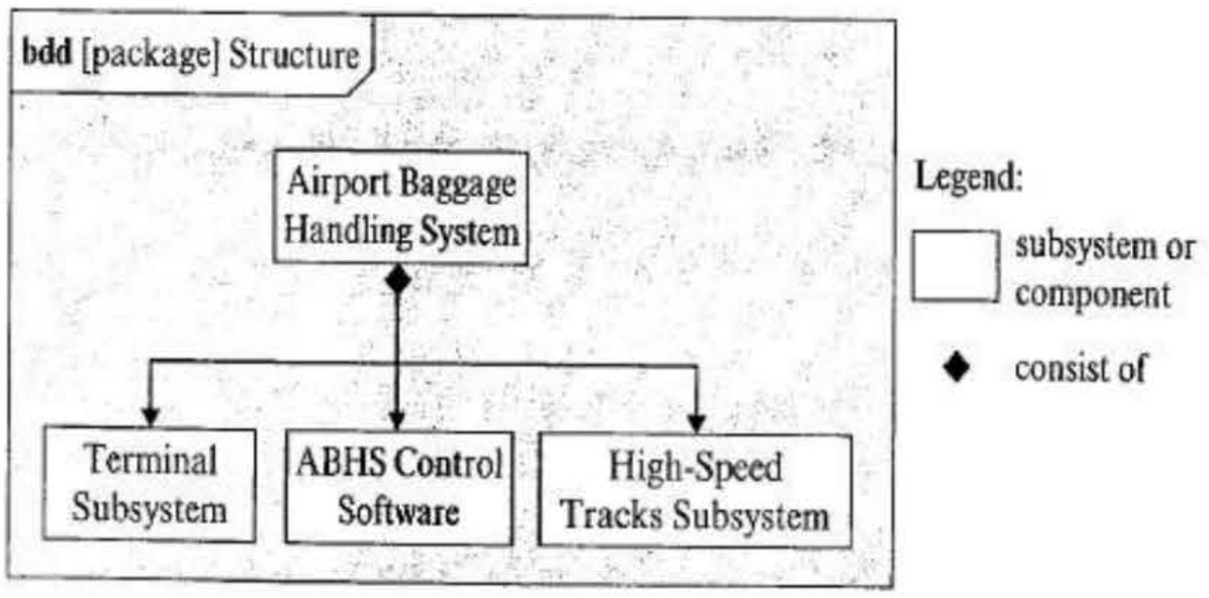
شکل ۳-۸: ارتباط بین نمودارهای SysML و UML

سخت‌افزار-نرم‌افزار متداول‌ترین واسط‌ها هستند. در بیشتر مواقع، یک مدار مجتمع میکروکنترلی برای این کار استفاده می‌شود. شکل ۳-۱۲ این موضوع را برای یک دستگاه تهویه مطبوع[۱] نشان می‌دهد. البته در شکل تنها اقلام مرتبط نشان داده شده‌اند، مثلاً، پین‌های[۲] استفاده نشده باید در واقع به سایر قطعات الکترونیکی متصل شوند.

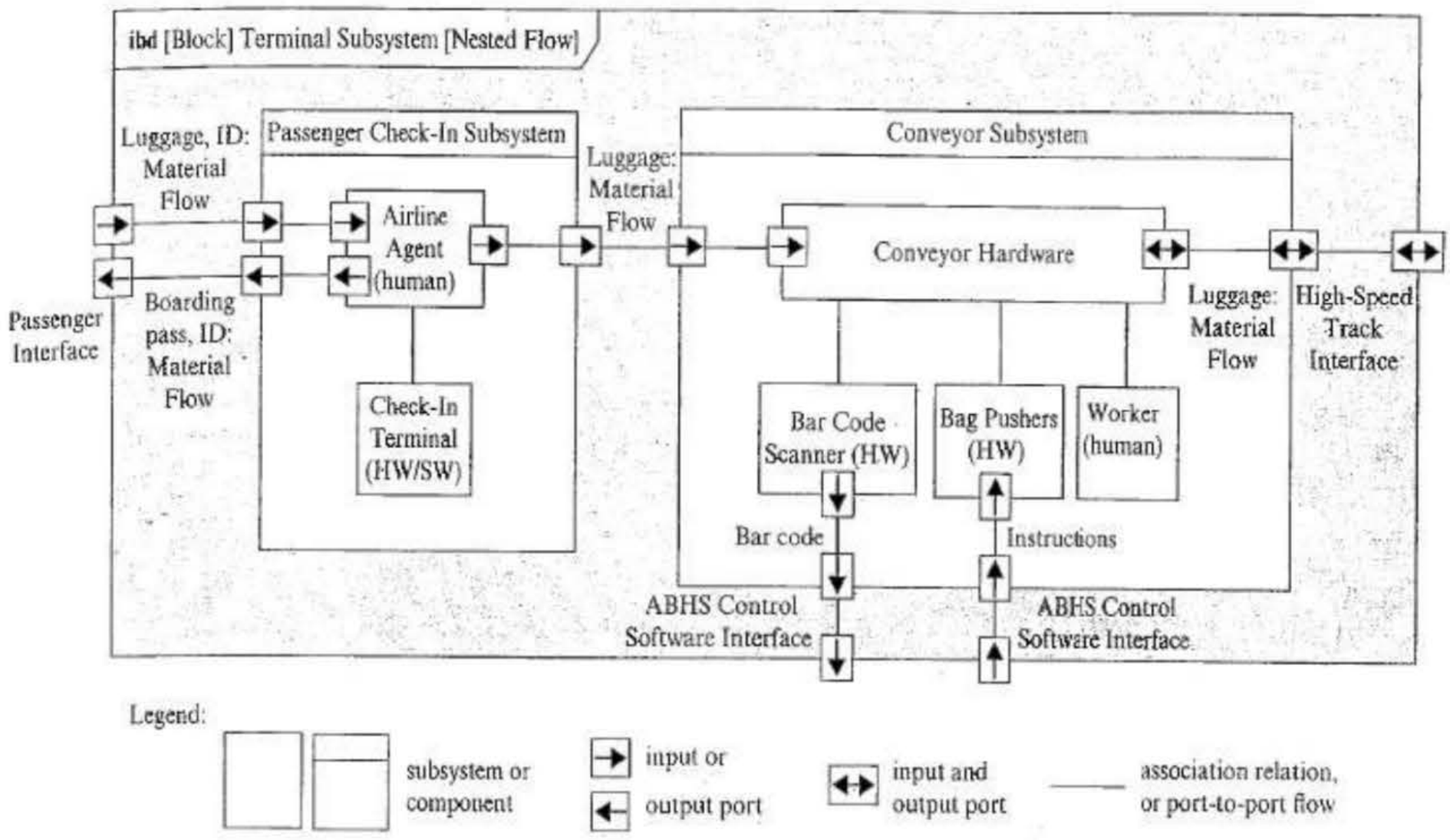
نمودار حالت نشان داده شده در شکل ۳-۱۲ نرم‌افزاری که درون تراشه IC اجرا می‌شود را نشان می‌دهد. وقتی کلید برق روشن می‌شود، نرم‌افزار به حالت قطع رله[۳] وارد می‌شود. این موضوع توسط انتقالی که از حالت Power off به حالت Relay

---

[۱] Air Conditioner
[۲] Pin
[۳] Relay off



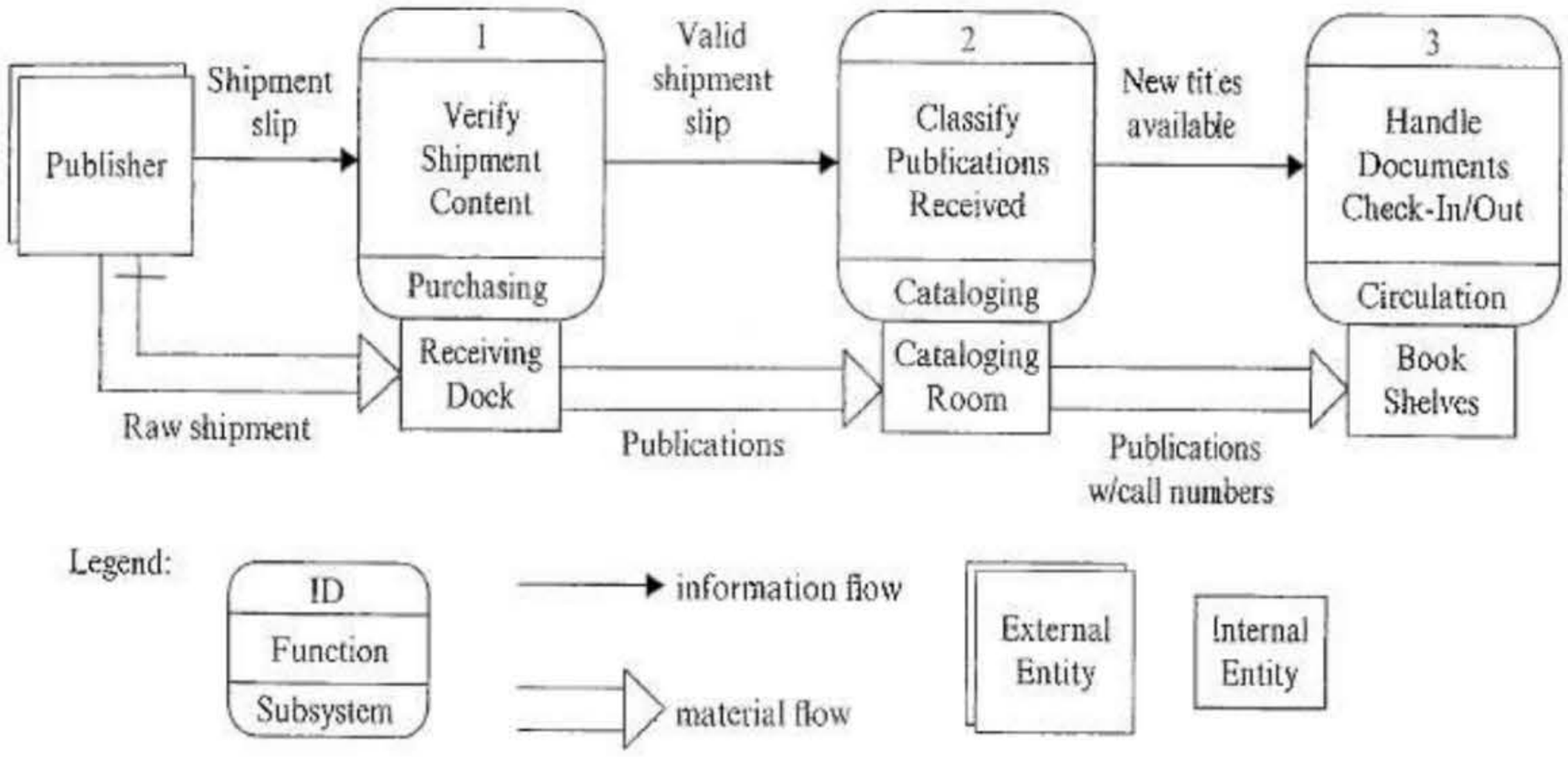
شکل ۳-۹: نمودار تعریف بلوک SysML برای ABHS



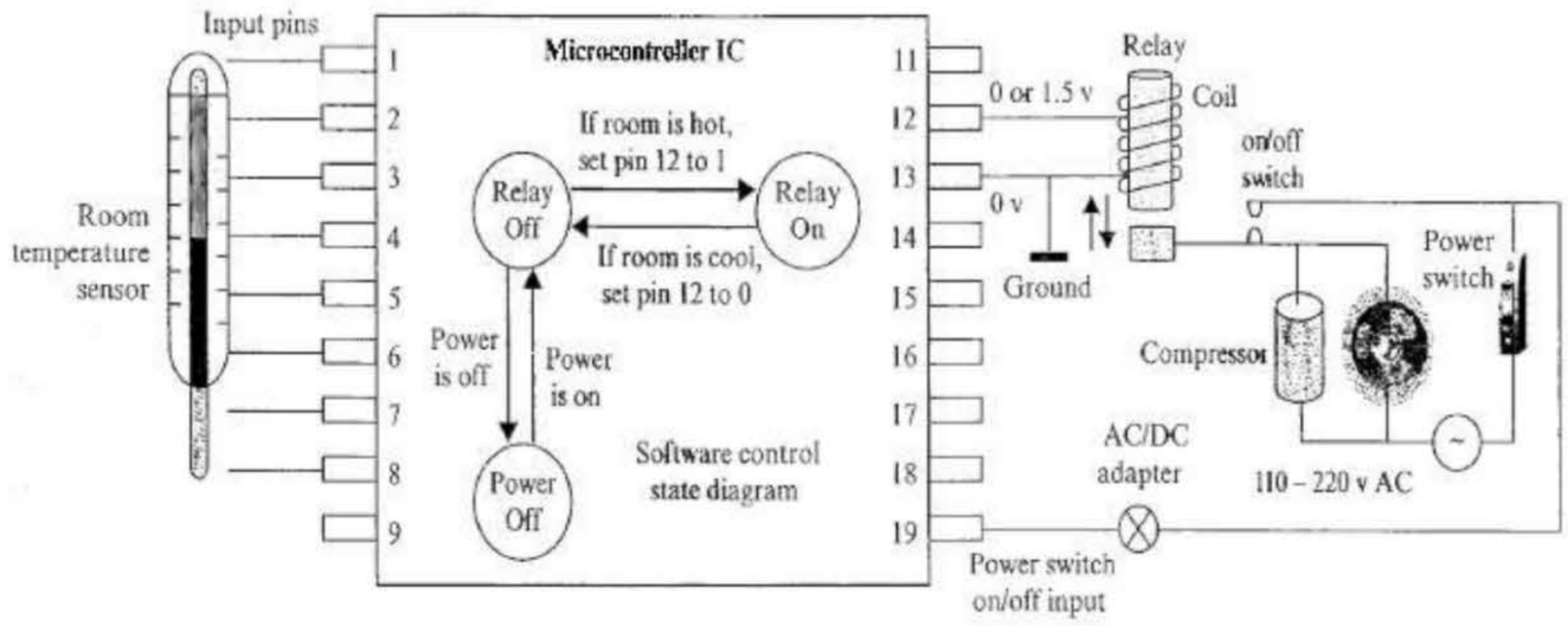
شکل ۳-۱۰: نمودار داخل بلوک SysML برای زیرسیستم پایانه از ABHS

off ترسیم شده نشان داده شده است. نرم‌افزار به‌طور متناوب بایتی که بیان‌گر هشت پایه‌ی متصل به حس‌گر دما است را می‌خواند تا دمای اتاق را به‌دست آورد. اگر اتاق گرم باشد، نرم‌افزار مقدار پایه‌ی ۱۲ را ۱ می‌کند. این کار یک جریان مستقیم ۱/۵ ولتی به سیم‌پیچ رله اعمال می‌کند و سیم‌پیچ هم یک میدان مغناطیسی تولید می‌کند که کلید روشن/خاموش را جذب کرده و لذا مدار برق کمپرسورو پنکه[1] را وصل می‌کند. بنابراین، سیستم خنک‌کننده شروع به‌کار می‌کند. وقتی اتاق خنک شد، نرم‌افزار مقدار پایه ۱۲ را ۰ می‌کند و این‌کار باعث می‌شود مدار برق کمپرسور و پنکه قطع شده و در نتیجه عمل خنک‌کردن متوقف شود. در موردی که در شکل ۳-۱۲ نشان داده شده است، تبیین واسط و رفتار تعاملی، پایه‌هایی که دمای اتاق را نشان می‌دهند و

[1] Fan



شکل ۳-۱۱: مدل‌سازی جریان مواد



شکل ۳-۱۲: نمایشی از واسط سخت‌افزار-نرم‌افزار

معنی بایتی که توسط نرم‌افزار خوانده می‌شود را تعریف می‌کند. هم‌چنین تبیین فوق پایه‌های خروجی و معنی مقادیر خروجی را تعریف می‌کند. در این مورد، پایه‌ی خروجی پایه‌ی شماره ۱۲ است. وقتی این پایه مقدار ۱ را بگیرد، رله درگیر شده و خنک‌کردن شروع می‌شود. علاوه بر واسط نرم‌افزاری-سخت‌افزاری، واسط‌های دیگری نیزاز قبیل واسط‌های انسان-نرم‌افزار، انسان-سخت‌افزار، انسان-انسان، سخت‌افزار-سخت‌افزار وجود دارند. این واسط‌ها و رفتار تعاملی به‌طور مشابه تبیین می‌گردند.

در طراحی سیستم، برنامه‌ی آزمون یکپارچه‌سازی سیستم و برنامه‌ی آزمون پذیرش نیز تولید می‌شوند. این مستندات، رویه‌های آزمون را مشخص می‌کنند که اطمینان می‌دهند زیرسیستم‌ها همان‌طور که انتظار می‌رود با یکدیگر کار می‌کنند، و سیستم آن توانایی‌هایی را که در سند تبیین نیازمندی‌های سیستم مشخص‌شده، ارائه می‌دهد. به‌علاوه، منابع لازم برای انجام آزمون یکپارچه‌سازی و آزمون پذیرش سیستم نیز باید مشخص شده باشند. این برنامه‌ی آزمون، درحین مراحل آزمون یکپارچه‌سازی و آزمون پذیرش سیستم مورد استفاده قرار می‌گیرند.

۳-۵ توسعه‌ی زیرسیستم‌ها

بعد از اینکه طراحی سیستم و تخصیص انجام شد، زیرسیستم‌ها به تیم‌های مهندسی مختلف واگذار می‌شوند. تیم‌های مهندسی نیز زیرسیستم‌ها را جداگانه ساخته و آزمایش می‌کنند. به‌طور خاص، تیم مهندسی نرم‌افزار (SE) فرایندها و متدولوژی‌های مهندسی نرم‌افزار را به‌کار برده و زیرسیستم نرم‌افزاری را توسعه می‌دهد. این تیم همچنین فعالیت‌های تضمین کیفیت و مدیریتی را انجام می‌دهد. تیم‌های مهندسی با یکدیگر تماس نزدیکی داشته، وضعیت پیشرفت‌شان را با هم در میان گذاشته و برای حل مشکلات طراحی و پیاده‌سازی بین رشته‌ای با یکدیگر همکاری می‌نمایند. این بخش فعالیت‌هایی که مرتبط با تیم مهندسی نرم‌افزاری است را معرفی می‌کند.

۳-۵-۱- نمودار زمینه‌ی شیءگرا

فعالیت توسعه‌ی نرم‌افزار باید زمینه‌ی زیرسیستم نرم‌افزاری را در نظر بگیرد. زمینه متشکل است از اشیای دنیای واقعی و سایر زیرسیستم‌هایی که با زیرسیستم نرم‌افزاری تعامل دارند. مهندسی نرم‌افزار شیءگرا، دنیا و سیستم نرم‌افزاری را به‌صورت اشیایی که با یکدیگر در تعامل هستند مدل‌سازی می‌کند. برای مثال، زمینه‌ی نرم‌افزار تهویه مطبوع در شکل ۳-۱۲ شامل حسگر دمای اتاق، تطبیق‌دهنده‌ی AC/DC، و رله می‌شود. زمینه‌ی نرم‌افزار کنترل ABHS که در شکل ۳-۵ نشان داده شد، شامل زیرسیستم پایانه، زیرسیستم نوار سرعت بالا، و زیر سیستم اطلاعات پرواز می‌گردد. شکل ۳-۶ نشان می‌دهد که نرم‌افزار با پویش‌گرهای بارکد و فشاردهنده‌های بسته، که اجزایی از زیرسیستم پایانه هستند، در تعامل است. توسعه‌ی نرم‌افزار شیءگرا این موارد را به‌عنوان اشیایی که با نرم‌افزار در حال ساخت درتعامل هستند، مدل‌سازی می‌کند، نتیجه به‌صورت یک نمودار کلاس UML که به آن مدل دامنه‌ی زمینه[۱] یا نمودار زمینه[۲] می‌گویند نشان داده می‌شود. مدل دامنه‌ی زمینه، اشیای زمینه، ویژگی‌های[۳] اشیای زمینه، و روابط بین اشیای زمینه و نرم‌افزار را نشان می‌دهد. برای نمونه، شکل ۳-۱۳ یک نمودار زمینه را برای کنترل ABHS به تصویر می‌کشد. این نمودار، نوع اشیایی که با نرم‌افزار کنترل ABHS در تعامل هستند و هم‌چنین واسط‌ها و پروتکل‌های تعامل بین این اشیا را مشخص می‌کند. اطلاعاتی که در نمودار زمینه نهفته است برای طراحی زیرسیستم نرم‌افزاری مفید است.

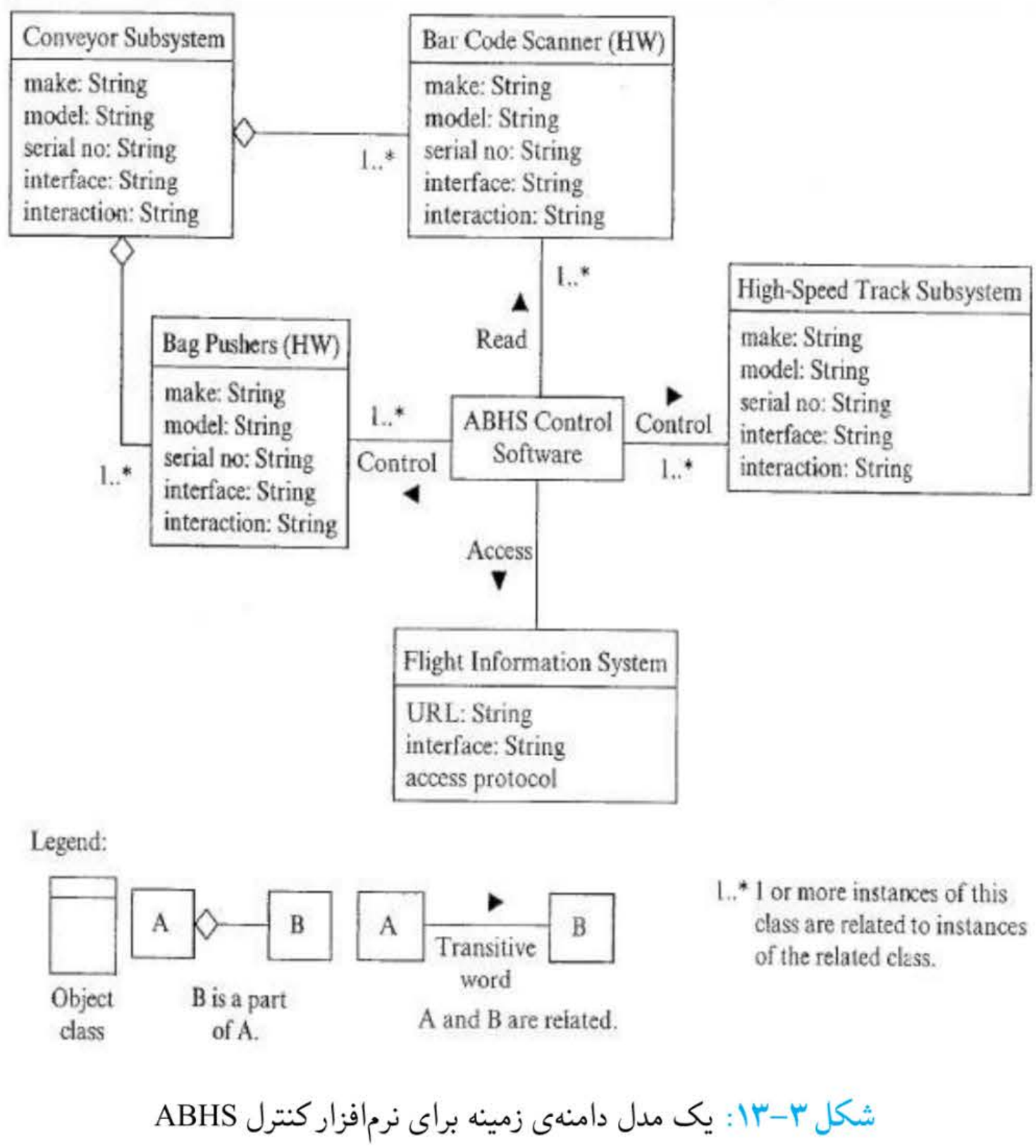
۳-۵-۲- فواید نمودار زمینه‌ی شیءگرا

مزایای نمودار زمینه‌ی شیءگرا به شرح زیر می‌باشند.

۱. یک دید یکنواخت از اشیای نرم‌افزاری و اشیای زمینه فراهم می‌آورد. این موضوع به تیم کمک می‌کند تا اشیای زمینه و ارتباط این اشیا با زیرسیستم نرم‌افزاری را درک کند.
۲. واسط‌ها و تعامل با اشیای زمینه را برجسته می‌کند. مدل دامنه‌ی زمینه با نرم‌افزار به‌عنوان یک جعبه‌ی سیاه برخورد می‌کند و فقط اشیای زمینه، ارتباطات آن‌ها، و صفت‌هایی که برای طراحی زیرسیستم نرم‌افزاری مفید هستند رانشان می‌دهد.
۳. به توسعه‌ی زیرسیستم نرم‌افزاری کمک می‌کند. برای مثال، صفت‌های پویش‌گرهای بارکد برای طراحی و پیاده‌سازی راه‌هایی که با پویش‌گرهای بارکد تعامل دارند مفید است. تعدد[۴] مانند *..1 در شکل ۳-۱۳، مشخص‌کننده‌ی تعداد پویش‌گرهایی است که نرم‌افزار کنترل ABHS باید پشتیبانی نماید. اطلاعات نشان داده شده در نمودار زمینه هم‌چنین برای آزمون و نگهداری هم مفید است. برای مثال، وقتی یک نوع جدید از پویش‌گر بارکد اضافه شود، تنها لازم است کلاس‌هایی که به پویش‌گر جدید ربط دارند طراحی و پیاده‌سازی شوند.

---

[۱] Context Domain Model
[۲] Context Diagram
[۳] Properties
[۴] Multiplicity



شکل ۳-۱۳: یک مدل دامنه‌ی زمینه برای نرم‌افزار کنترل ABHS

۴. ارتباط و همکاری با سایر تیم‌های مهندسی را تسهیل می‌کند. نمودار زمینه‌ی شئ‌گرا واسط‌ها و تعامل با سایر زیرسیستم‌های مهندسی را برجسته می‌کند، این کار به تیم مهندسی نرم‌افزار کمک می‌کند که وقتی با سایر تیم‌های مهندسی کار می‌کنند، بر روی مسائل واسط و تعامل تمرکز کنند.
۵. برای آموزش مفید است. زیرسیستم نرم‌افزاری از یک سیستم پیچیده‌ی بزرگ، لازم است که با بسیاری سیستم‌های پیچیده‌ی دیگر تعامل داشته باشد. لذا، درک زمینه ساده نیست. نمودار زمینه ابزاری مفید است برای اعضای جدید تا زمینه‌ی یک زیرسیستم نرم‌افزاری را فراگیرند.

۳-۵-۳- همکاری تیم‌های مهندسی

در حین فرایند توسعه‌ی زیرسیستم‌ها، تیم‌های مهندسی، تماس نزدیکی با یکدیگر دارند تا وضعیت را با هم در میان گذاشته و مشکلات بین رشته‌ای را حل نمایند. برای مثال، طراحی و پیش‌نمونه‌سازی یک زیرسیستم دریافت‌کننده‌ی رادیویی را در نظر بگیرید. تیم مهندسی برق[1] ممکن است کشف کند که یک خصیصه‌ی معین باید به‌جای سخت‌افزار، با نرم‌افزار پیاده سازی شود. تیم EE ممکن است با تیم SE ملاقات کند تا امکان‌پذیری این کار را بررسی کند. اگر دو تیم با تغییر موافق باشند، یک

[1] Electrical Engineering (EE)

پیشنهاد تغییر ارسال خواهد شد. یک هیئت کنترل تغییر پیشنهاد را بازبینی کرده و تغییر را قبول یا رد می‌نماید. اگر پیشنهاد پذیرفته شود، تغییر پیاده‌سازی می‌شود.

۳-۶ یکپارچه‌سازی، آزمون و استقرار سیستم

زیرسیستم‌هایی که توسط تیم‌های مهندسی مختلف ساخته شده‌اند، سپس یکپارچه شده و آزمون‌های یکپارچه‌سازی و سیستم اجرا می‌شوند. آزمون یکپارچه‌سازی سیستم مطابق با برنامه‌ی آزمون یکپارچه‌سازی سیستم که در مرحله‌ی طراحی سیستم و تخصیص تولید شده است، انجام می‌شود. به‌دلیل تغییراتی که درحین مرحله‌ی توسعه‌ی زیرسیستم‌ها ممکن است رخ داده باشد، ممکن است تغییراتی نیز در برنامه‌ی آزمون ضروری باشد. آزمون یکپارچه‌سازی سیستم، رویه‌های آزمون مشخص شده در برنامه‌ی آزمون را پالایش کرده و این آزمون‌ها را اجرا می‌کند تا اطمینان حاصل شود که زیرسیستم‌ها از طریق واسط‌هایشان به‌درستی با یکدیگر ارتباط برقرار می‌کنند.

آزمون سیستم بر طبق برنامه‌ی آزمون پذیرش انجام می‌گردد. این آزمون اطمینان می‌دهد که سیستم در واقع می‌تواند تمام قابلیت‌هایی که در نیازمندی‌های سیستم بیان شده است را ارائه نماید. بسته به سیستم، آزمون‌های سیستم و یکپارچه‌سازی ممکن است به تجهیزات خاصی نیاز داشته باشد. برای مثال، برای سیستم‌های مخابراتی، تجهیزات آزمون باید میلیون‌ها تماس فرضی تولید کند تا سیستم در حال ساخت را آزمایش کرده و بازدهی [۱]، کارایی، و زمان پاسخ سیستم را به‌دقت اندازه‌گیری نماید. پس از آزمون سیستم، سیستم حمل شده و در محیط مقصد نصب می‌شود و در آن‌جا توسط کاربران آزمایش می‌شود که به آن آزمون بتا[۲] می‌گویند. نقص‌هایی که در حین آزمون بتا پیدا می‌شوند به تیم توسعه گزارش می‌شوند. نقص‌ها برطرف شده و سیستم دوباره آزمایش می‌شود. بعد از آزمون بتا، سیستم وارد مرحله‌ی نگهداری می‌شود. در حین نگهداری، به‌سازی‌ها انجام و چنانچه نقص‌های بیشتری یافت شوند برطرف می‌گردند.

۳-۷ مدیریت پیکربندی سیستم

مهندسی سیستم شامل تیم‌های مهندسی مختلفی می‌شود که باید به شکل هماهنگ و کنترل شده با یکدیگر کار کنند. فرض کنید که طراحی یک جزء نرم‌افزاری به طراحی یک قطعه‌ی الکترونیکی وابسته باشد. در این صورت، تا طراحی قطعه‌ی الکترونیکی نهایی نشود طراحی جزء نرم‌افزاری نمی‌تواند نهایی شود. پرسش این است که: چگونه تیم مهندسی نرم‌افزار با خبر می‌شود که طراحی قطعه‌ی الکترونیکی نهایی شده است؟ یک روش، ارسال رایانامه[۳] است، ولی برای یک پروژه‌ی بزرگ توسعه‌ی سیستم این روش غیررسمی مشکل‌ساز خواهد بود. چه کسی مسئول ارسال رایانامه خواهد بود؟ چه کسی باید رایانامه را دریافت کند؟ اگر رایانامه ارسال نشود یا به گیرنده نرسد چه می‌شود؟ در متن رایانامه چه باید نوشته شود؟ لذا یک روش رسمی[۴] برای خبر دادن چنین وقایعی به تیم مهندسی نرم‌افزار نیاز است.

مسئله‌ی دیگری که باید هماهنگ شود، کنترل تغییر است. برای مثال، چنان‌چه طراحی قطعه‌ی الکترونیکی تغییر کند، در این صورت طراحی جزء نرم‌افزاری نیز باید تغییر یابد. این بدان معنی است که تیم مهندسی نرم‌افزار باید باخبر شود، لذا یک سازوکار اطلاع‌رسانی مورد نیاز است. چنان‌چه طراحی قطعه‌ی الکترونیکی بتواند به تناوب و به دلخواه تغییر یابد، در این‌صورت تیم مهندسی نرم‌افزار برای غلبه بر تغییرات احساس مشکل خواهد کرد. از سوی دیگر، اگر نتوان طراحی را تغییر داد، تیم الکترونیک چگونه می‌تواند ایرادات یا خطاهای طراحی را مرتفع نماید؟ مدیریت پیکربندی سیستم این مشکل را حل

---

[۱] Throughput
[۲] Beta Testing
[۳] Email
[۴] Formal

می‌کند.

مدیریت پیکربندی سیستم بر مفهوم خط مبنا استوار است. خط مبنا یک مرحله‌ی مهم از پروژه است. برای مثال، یک پروژه‌ی توسعه‌ی سیستم شامل چندین خط مبنا می‌باشد، از قبیل: خط مبنای نیازمندی‌های سیستم، خط مبنای طراحی سیستم، خط‌مبنای تخصیص، خط‌مبنای طراحی زیرسیستم. خط‌مبنای نیازمندی‌های سیستم، اعلان می‌کند که مرحله‌ی تحلیل نیازمندی‌های سیستم با موفقیت خاتمه یافته است. هر خط مبنا با مجموعه‌ای از دست‌ساخته‌ها به‌نام اقلام پیکربندی[۱] همراه است. برای مثال، خط مبنای نیازمندی‌های سیستم می‌تواند شامل تبیین نیازمندی‌های سیستم و برنامه‌ی پروژه باشد. وقتی این‌ها تولید و بازبینی‌شده و نقص‌هایی که توسط بازبین‌ها پیدا شدند رفع گردیدند، خط مبنای نیازمندی‌های سیستم برقرار می‌شود. تا قبل از برقرارشدن خط مبنای نیازمندی‌های سیستم، می‌توان این دست‌ساخته‌ها را آزادانه تغییر داد. وقتی خط مبنا برقرار شد، برای انجام تغییر دراین دست‌ساخته‌ها باید یک رویه‌ی کنترل تغییر طی شود. بدین صورت که، تغییرات پیشنهادی باید توسط یک هیات کنترل تغییرات، که نمایندگانی از ذی‌نفعان نیز در آن عضو هستند، ارزشیابی شود. اگر تغییرات پیشنهادی پذیرفته شدند، آن‌گاه تغییرات پیاده‌سازی شده و وضعیت پیشرفت کنترل می‌گردد. در صورتی که تغییرات پیشنهادی رد شوند، بایگانی می‌شوند. به‌طورخلاصه، مدیریت پیکربندی سیستم شامل چهار عمل می‌شود.

۱. شناسایی قلم پیکربندی. این عمل خط مبنای پروژه و اقلام پیکربندی مربوطه را تعریف می‌کند. نمونه‌هایی از خط مبنا عبارتند از: خط مبنای نیازمندی‌های سیستم، خط مبنای طراحی سیستم، خط مبنای تخصیص، و خط مبنای طراحی زیرسیستم. مثال‌های اقلام پیکربندی نیز عبارتند از: تبیین نیازمندی‌های سیستم، برنامه‌ی آزمون پذیرش سیستم، تبیین طراحی سیستم، برنامه‌ی تخصیص، برنامه‌ی آزمون یکپارچه‌سازی سیستم. هر قلم پیکربندی را می‌توان قبل از برقراری خط مبنا، آزادانه تغییر داد. خط مبنا وقتی برقرار می‌شود که تمام اقلام پیکربندی مربوطه درون سیستم مدیریت پیکربندی وارد شده باشند. برای مثال، چنان‌چه خط‌مبنای نیازمندی‌های سیستم شامل تبیین نیازمندی‌های سیستم و برنامه‌ی پروژه باشد، آن‌گاه خط مبنا فقط وقتی می‌تواند برقرار شود که هر دوی این مستندات درون سیستم مدیریت پیکربندی وارد شده باشند. وقتی خط مبنا برقرار شد، تغییر در قلم پیکربندی مستلزم طی یک رویه‌ی کنترل تغییر است.

۲. کنترل تغییر پیکربندی. این عمل رویه‌ی کنترل تغییر را تعریف و اجرا می‌کند. شکل ۳-۱۴ رویه‌ی کنترل تغییر را نشان می‌دهد. ابتدا، تغییر درخواستی شناسایی و تحلیل می‌شود. سپس یک پیشنهادیه‌ی تغییر مهندسی[۲] تهیه می‌شود، که تعیین‌کننده‌ی تغییر، تاثیر، مدت، و هزینه‌ی آن تغییر است. این ECP توسط یک هیات کنترل تغییر[۳] ارزیابی می‌شود. این هیات شامل نمایندگانی از طرف‌هایی است که تحت تاثیر تغییر پیشنهادشده قرار می‌گیرند. نتیجه‌ی ارزیابی می‌تواند پذیرش یا رد ECP باشد، که در نتیجه، تغییر پیشنهادی یا پیاده‌سازی شده یا بایگانی می‌گردد.

۳. بازرسی پیکربندی. این عمل به‌طوررسمی خط مبناهای پروژه را برقرار می‌کند و نیز اطمینان می‌دهد که تغییرات مهندسی پیشنهادی به درستی انجام شده‌اند. برای رسیدن به این اهداف، وارسی و اعتبارسنجی در مورد قلم پیکربندی انجام می‌شود. وارسی قلم پیکربندی اطمینان می‌دهد که با برقراری خط‌مبنا، قلم‌های پیکربندی درخواستی نیز تولید شده‌اند. مثلاً تضمین می‌کند که قبل از برقراری خط مبنای نیازمندی‌های سیستم، تبیین نیازمندی‌های سیستم و برنامه‌ی پروژه هر دو تولید شده‌اند. اعتبارسنجی قلم پیکربندی تضمین می‌کند که اقلام پیکربندی درست هستند، برای مثال، تبیین نیازمندی‌های سیستم توسط تیم‌های مهندسی، خبرگان دامنه، مشتری، و کاربران مورد بازبینی قرارگرفته و نقص‌های یافت‌شده برطرف و نگرانی‌هانیز رفع شده‌اند.

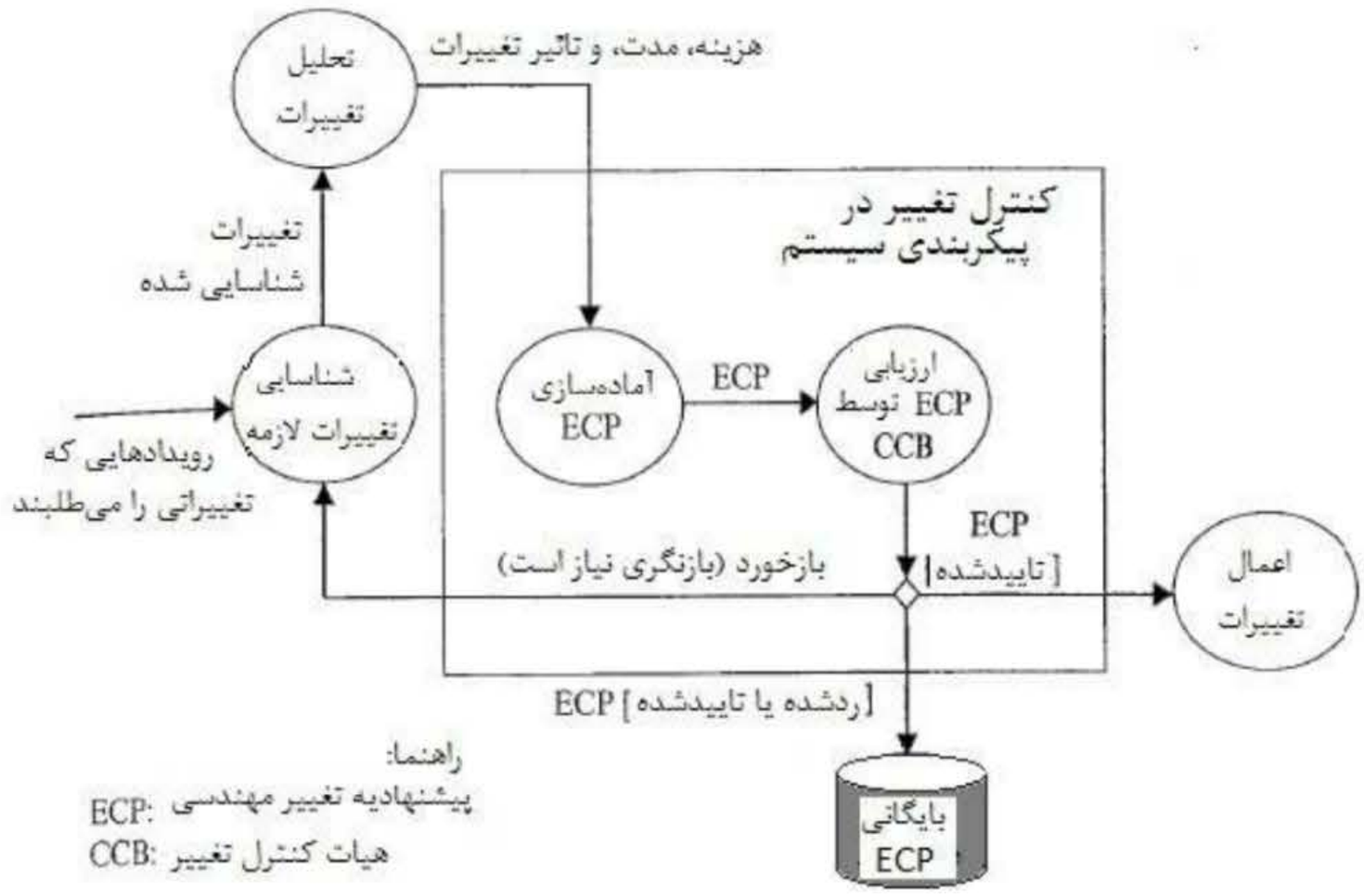
۴. گزارش وضعیت پیکربندی. این عمل پایگاه‌داده‌ای برای از سه عمل دیگر تهیه می‌کند. از این پایگاه‌داده می‌توان اطلاعاتی

---

[۱] Configuration Item

[۲] Engineering Change Proposal (ECP)

[۳] Change Control Board (CCB)



شکل ۳-۱۴: کنترل تغییر پیکربندی

درباره‌ی اقلام پیکربندی پرس‌وجو نمود. این عمل هم‌چنین رویدادهایی درباره‌ی پیکربندی سیستم منتشر می‌نماید. از جمله این رویدادها می‌توان به برقراری خط مبنا و تغییراتی در اقلام پیکربندی اشاره نمود.

خلاصه

این فصل فعالیت‌های توسعه‌ی سیستم که شامل اجزای سخت‌افزاری، نرم‌افزاری، و انسانی می‌شوند را معرفی نمود. هم‌چنین، در این فصل فرایند مهندسی سیستم و تکنیک‌هایی برای مدل‌سازی، طراحی، و تخصیص نیازمندی‌های سیستم تشریح گردید. پس از تخصیص، زیر سیستم‌ها توسط تیم‌های جداگانه‌ای از رشته‌های مختلف مهندسی ساخته می‌شوند. سپس زیرسیستم‌ها یکپارچه و آزمایش می‌شوند تا مطمئن شویم که سیستم نیازمندی‌ها و قیدهای سیستم را برآورده می‌سازد. در طول فرایند تولید سیستم، مستندات مختلف تحلیل، طراحی، پیاده‌سازی و آزمون، تولید و بروزرسانی می‌شوند. بروزرسانی یک مستند ممکن است بر روی سندهای دیگری نیز تاثیر بگذارد. لذا بروزرسانی‌ها باید هماهنگ شوند که این کار توسط مدیریت پیکربندی سیستم انجام می‌شود.

برای مطالعه‌ی بیش‌تر

ارائه‌های بسیار خوبی در مورد جنبه‌های مختلف مهندسی سیستم توسط Blanchard [۳۱] و Kossiakoff et al [۹۹] تهیه شده‌اند. SysML در [۶۵] و UML در [۳۶] معرفی شده‌اند. در [۱۵۲]، Thayler می‌گوید که چگونه اصول مهندسی سیستم برای توسعه‌ی سیستم‌های نرم‌افزاری بزرگ و پیچیده باید اعمال شوند. این مقاله، فرایندها و استانداردهای مهندسی نرم‌افزار IEEE را با فرایند مهندسی سیستم نرم‌افزاری ادغام می‌کند.

سوال‌های مروری فصل

۱. مهندسی سیستم چیست؟
۲. چرا ما به مهندسی سیستم نیاز داریم؟
۳. چرا مهندسی سیستم به‌عنوان یک روش بین‌رشته‌ای درنظرگرفته می‌شود؟
۴. مراحل و فعالیت‌های فرایند مهندسی سیستم، که در این فصل تشریح شدند، کدامند؟
۵. هدف از توسعه‌ی سخت‌افزار، توسعه‌ی نرم‌افزار، و توسعه‌ی منابع انسانی چیست و چرا این‌ها جداگانه انجام می‌شوند؟
۶. فرایند مهندسی سیستم چگونه با چالش‌های مهندسی سیستم مقابله می‌کند؟
۷. رابطه‌ی بین مهندسی سیستم و مهندسی نرم‌افزار چیست؟
۸. در طی مرحله‌ی طراحی، تیم مهندسی سیستم باید چه عواملی را در نظر بگیرد؟
۹. راهبردهای قابل استفاده برای تجزیه‌ی یک سیستم یا زیرسیستم در حین طراحی سیستم کدامند؟
۱۰. مدل‌سازی سیستم چیست؟
۱۱. ماتریس ردیابی‌پذیری نیازمندی-زیرسیستم چیست؟
۱۲. چالش‌های بالقوه برای یکپارچه‌سازی و آزمون سیستم کدامند؟

تمرین‌ها

۳-۱ شرح مختصری از اعمال یک ماشین فروشنده[1] ارائه نمایید. تمام نیازمندی‌های کارکردی و کارایی را شناسایی و تنظیم کنید.

۳-۲ دو سیستم جاسازی‌شده به جز آن‌هایی که در این فصل معرفی شدند شناسایی کنید و برای هر کدام اعمال زیر را انجام دهید:

الف) اعمال سیستم را به اختصار تشریح کنید به‌گونه‌ای که حدود نیم تا یک صفحه بوده و در صورت لزوم دارای نمودار هم باشد.

ب) پنج نیازمندی کارکردی و دو نیازمندی غیرکارکردی را شناسایی و تنظیم کنید. یکی از نیازمندی‌های غیرکارکردی باید در مورد کارایی باشد.

ج) سیستم را تجزیه نموده و نیازمندی‌های سیستم را به زیرسیستم‌ها تخصیص دهید.

۳-۳ یک سیستم خودروشویی[2] سکه‌ای که یک سیستم سلف-سرویس است را درنظربگیرید. مشتری تعدادی سکه‌ی بیست و پنج سنتی (ربع دلاری) داخل سیستم می‌اندازد تا زمان از پیش تنظیم‌شده برای شست‌وشو را خریداری نماید. در حین زمان شست‌وشو، هر زمان که مشتری بخواهد می‌تواند با چرخاندن یک شاخص صابون، کف، آب‌کشی، و واکس را انتخاب نماید. وقتی یک دقیقه به اتمام کار باقی مانده است، سیستم بوق می‌زند و مشتری می‌تواند با انداختن سکه‌های بیشتر زمان شست‌وشو طولانی‌تر نماید. برای این سیستم خودروشویی، کارهای زیر را انجام دهید:

---

[1] Vendor Machine

[2] Carwash

الف) نیازمندی‌های سیستم، اعم از کارکردی و غیرکارکردی را شناسایی و تبیین نمایید.

ب) سیستم را به زیرسیستم‌های کارکردی تجزیه نمایید. در صورت لزوم، نیازمندی‌های سیستم را نیز تجزیه کنید.

ج) نیازمندی‌های سیستم را به زیرسیستم‌ها تخصیص دهید.

د) با استفاده از یک تکنیک نمودارکشی به انتخاب خودتان یا استاد درس، یک نمودار طراحی معماری سیستم تهیه کنید.

۳-۴ متنی بنویسید که درباره‌ی موارد زیر بحث کند. یک سیستم تقاطع راه‌آهن با به‌کارگیری تعدادی حس‌گر، چراغ چشمک‌زن، زنگ‌خطر، و دروازه[1] رفت‌وآمد در یک تقاطع راه‌آهن را کنترل می‌کند. وقتی که قطاری از یک سو به تقاطع نزدیک می‌شود، حس‌گر قطار را حس کرده و با نرم‌افزار ارتباط برقرار می‌کند. نرم‌افزار برای مدت معینی چراغ زرد چشمک‌زن را روشن کرده و زنگ خطر را به صدا در می‌آورد. سپس چراغ را به قرمز چشمک‌زن تغییر داده و دروازه‌ها را می‌بندد. بعد از این‌که قطار از تقاطع عبور کرد، حس‌گر دیگری این واقعه را حس کرده و با نرم‌افزار ارتباط برقرار می‌سازد. نرم‌افزار نیز چراغ‌ها را خاموش، زنگ‌خطر را قطع، و دروازه‌ها را باز می‌کند. کارهای زیر را در مورد سیستم تقاطع راه‌آهن انجام دهید.

الف) زیرسیستم‌های نرم‌افزاری و سخت‌افزاری را شناسایی و عملکرد هر کدام از این سیستم‌ها را تعیین کنید.

ب) بگویید چگونه این سیستم‌ها به‌هم مرتبط بوده و با هم تعامل می‌کنند. به عبارت دیگر، رفتار تعاملی و واسط‌های زیرسیستم‌ها را مشخص کنید.

ج) یک نمودار طراحی معماری سیستم تهیه کنید که روابط بین زیرسیستم‌ها را نشان دهد.

د) نیازمندی‌های ایمنی را شناسایی و مدون کرده و آن‌ها را به زیرسیستم‌ها تخصیص دهید. توضیح دهید چگونه زیرسیستم‌ها نیازمندی‌های ایمنی را برآورده می‌سازند.

۳-۵ مقاله‌ی کوتاهی بنویسید که به پرسش‌های زیر پاسخ دهد: مدیران یک فروشگاه بزرگ می‌خواهند خرده‌فروشی برخط[2] را نیز راه بیاندازند. بدین معنی‌که شرکت می‌خواهد یک سیستم برخط تهیه کند که سفارش‌های مشتریان را به‌صورت برخط دریافت نموده و اقلام سفارش داده‌شده را به‌وسیله‌ی یک سرویس حمل‌ونقل ملی، حمل نماید. برای کاهش هزینه‌ی کارگر، شرکت یک سیستم کاملاً خودکار می‌خواهد. کارهای زیر را انجام دهید:

الف) پنج نیازمندی کارکردی و دو نیازمندی کارایی برای این سیستم شناسایی و تنظیم نمایید.

ب) سیستم را به‌صورت سلسله‌مراتبی از زیرسیستم‌ها تجزیه نموده و نیازمندی‌های سیستم را به زیرسیستم‌ها اختصاص دهید.

ج) یک نمودار طراحی معماری سیستم تولید کنید.

۳-۶ چنان‌چه رسم نمودارهای UML را یاد گرفته‌اید، یک نمودار زمینه‌ی شئ‌گرا برای زیرسیستم نرم‌افزاری سیستم خرده‌فروشی برخط که در تمرین ۳-۵ تولید کردید، ایجاد کنید. در مورد فواید نمودار زمینه برای طراحی، پیاده‌سازی، آزمون، و نگهداری زیرسیستم نرم‌افزاری بحث کنید.

---

[1] Gate

[2] Online

دوم

تحلیل و طراحی معماری

فصل چهارم استخراج نیازمندی‌های نرم‌افزار
فصل پنجم مدل‌سازی دامنه
فصل ششم طراحی معماری

۴

استخراج نیازمندی‌های نرم‌افزار

نکات کلیدی

- نیازمندی‌ها، توانمندی‌هایی هستند که سیستم باید از خود نشان بدهد.
- دشوارترین بخش از ساخت یک سیستم نرم‌افزاری، تصمیم‌گیری دقیق در مورد چیزی است که باید ساخته شود؛ یعنی نیازمندی‌ها (Frederick P. Brooks, Jr.).
- کار استخراج[۱] نیازمندی‌های نرم‌افزار، با هدف شناسایی نیازهای واقعی سیستم انجام می‌گیرد.

مرحله‌ی برنامه‌ریزی متدولوژی چابک یکنواخت در این کتاب سه هدف را دنبال می‌کند. (۱) استخراج نیازمندی‌های نرم‌افزار برای سیستم آینده، (۲) به‌دست آوردن یک مجموعه‌ی مورد کاربرد از نیازمندی‌ها، و (۳) تعریف یک برنامه‌ی تکراری توسعه و استقرار برای تحویل موارد کاربرد. این فصل، گام‌های لازم برای رسیدن به هدف اول را نشان می‌دهد. استخراج نیازمندی‌ها یک کار مهم و دشوار در توسعه‌ی نرم‌افزار است. مهم است، چون توانمندی‌های سیستم را در کنار محدودیت‌های موجود در فضای پاسخ مسئله تعریف می‌کند. این توانمندی‌ها و محدودیت‌ها، پایه‌ی فعالیت‌های آینده‌ی توسعه را تشکیل می‌دهند. اگر نیازمندی‌ها و محدودیت‌ها نادرست، ناقص، ناسازگار، و یا ناکافی باشند، سیستمی که بر اساس آنها ساخته و نصب می‌گردد نیازهای مشتری را برآورده نخواهد کرد. به همین دلیل، یک فصل کامل به موضوع استخراج نیازمندی‌ها، فعالیتی برای شناسایی نیازهای واقعی سیستم، اختصاص یافته است. استخراج نیازمندی‌ها با عنوان تحلیل نیازمندی‌ها نیز شناخته می‌شود. این دو اصطلاح ممکن است در این کتاب با یک معنا مورد استفاده قرار گیرد. با مطالعه‌ی این فصل موارد زیر را خواهید آموخت:

- اهمیت و چالش‌های استخراج نیازمندی‌ها.
- انواع نیازمندی.
- فعالیت‌های استخراج نیازمندی‌ها.
- اولویت‌بندی نیازمندی‌ها.
- تکنیک‌های مرور نیازمندی‌ها.
- به‌کارگیری قوانین چابکی در حین استخراج نیازمندی‌ها.

---

[۱] Elicitation

۴-۱ استخراج نیازمندی‌ها چیست؟

سیستم‌های نرم‌افزاری به دلایل گوناگونی ساخته می‌شوند. بیش از هرچیز، یک سیستم نرم‌افزاری برای برآورده کردن نیازهای کاری یک مشتری، که برای کار توسعه‌ی نرم‌افزار هزینه پرداخت می‌کند، ساخته می‌شود. بیشتر پروژه‌های واقعی، بودجه‌ای محدود دارند. بنابراین، تنها زیرمجموعه‌ای از نیازهای کاری آنها می‌تواند برآورده شود. این نیازها با عنوان نیازمندی‌های نرم‌افزار شناخته می‌شوند.

تعریف ۴-۱

نیازمندی‌های نرم‌افزار توانمندی‌هایی هستند که سیستم باید از خود نشان دهد.

یک پروژه‌ی نرم‌افزاری در صورتی موفق است که سیستم، نیازمندی‌های نرم‌افزاری خود را برآورده سازد و طبق زمان‌بندی تحویل داده شود. بنابراین برای موفقیت پروژه‌ی نرم‌افزاری، شناسایی نیازمندی‌های واقعی ضروری است. شناسایی نیازمندی‌های واقعی، به‌خصوص درشرایطی که باید محدودیت‌های بودجه و زمان توسعه رعایت گردد، کار آسانی نیست. این بدان علت است که ذی‌نفعان مختلف، درک متفاوتی از اولویت‌های کاری و توانمندی‌های مورد نیاز سیستم دارند. این تفاوت‌ها از عوامل مختلفی چون منافع مالی، زمینه‌های تحصیلی، تجربه، جایگاه، و باورها نشأت می‌گیرد. برای مثال، یک ذی‌نفع با سوابق تکنیکی قوی ممکن است بر توان عملیاتی و زمان پاسخ سیستم تأکید داشته باشد، در حالیکه یک ذی‌نفع با سوابق کاری قوی ممکن است اولویت را به هزینه‌ها و رضایت مشتری بدهد.

یک سیستم اطلاعاتی خدمات درمانی[۱] را در نظر بگیرید که به شرکتهای بیمه، پزشکان، پرستاران، بیمارستان‌ها، درمانگاه‌ها، داروسازان، و داروخانه‌ها سرویس می‌دهد. علایق این ذی‌نفعان مختلف است: به همین دلیل درک آنها از نیازمندی‌های واقعی و اهمیت آن‌ها متفاوت است. شرکت‌های بیمه‌خواهان هزینه‌های کمتر برای درمان بیمه‌شدگان هستند. آن‌ها از سیستم می‌خواهند که قوانین سخت‌گیرانه‌ای برای بررسی صحت ادعاهای مشتریان وضع کند و بر اجرای آنها نظارت نماید. این درخواست، نیازمند کار بیشتر پزشکان، نهادهای خدمات درمانی، و داروخانه‌ها برای ثبت مشخصات بیمه‌شوندگان می‌باشد و گاهی ممکن است باعث تأخیر در پرداخت‌ها شود. از سوی دیگر پزشکان، مراکز خدمات درمانی، و داروخانه‌ها انتظار دارند که سیستم، فرایند ثبت مشخصات بیمار و نیز پرداخت حق بیمه‌ها را آسان‌تر و سریع‌تر نماید. اگر این سیستم در سازمان‌های دولتی به‌کارگرفته شود، باید آیین‌نامه‌های دولتی را نیز در نظر بگیرد و نیازمندی‌های دیگری را هم برآورده سازد. حتی اگر زمان و منابع کافی برای توسعه‌ی چنین سیستمی وجود داشته باشد، تصمیم‌گیری در مورد نیازمندی‌های واقعی که بتواند همه‌ی این ذی‌نفعان را راضی کند، کار ساده‌ای نیست.

مشتریان یا نهادهای دولتی معمولاً محدودیت‌هایی را بر سیستم نرم‌افزاری اعمال می‌کنند. برای مثال ممکن است مشتری نیاز داشته باشد که تمام سیستم یا درصدی از آن حتماً به یک زبان خاص برنامه‌نویسی نوشته شود. مشتری ممکن است استفاده از یک محصول خاص و یا کلیه‌ی محصولات یک تولیدکننده‌ی خاص را ممنوع کند. این محدودیت‌ها معمولاً به دلیل ملاحظات سیاسی یا شغلی از سوی مشتری اعمال می‌گردد. محدودیت‌هایی از این دست، معمولاً با عنوان قیود[۲] به‌صورت زیر تعریف می‌شوند:

---

[۱] Health Care System

[۲] Constraints

تعریف ۴-۲

قیود، محدودیت‌هایی بر فضای پاسخ یک سیستم نرم‌افزاری هستند.

تفاوت اصلی بین نیازمندی‌ها و قیود این است که قیود تعداد راه‌های مختلف طراحی و پیاده‌سازی را کاهش می‌دهند. به‌عنوان مثال، حالت ویژه‌ای را در نظر بگیرید که در آن هیچ محدودیتی اعمال نشود. در این حالت، تیم توسعه‌دهنده می‌تواند برای پیاده‌سازی سیستم نرم‌افزاری از طیف وسیعی از محصولات، زبان‌های برنامه‌نویسی، و سکوها انتخاب نماید. از سوی دیگر، اگر زبان برنامه‌نویسی به زبان X محدود شود، همه‌ی زبان‌های برنامه‌نویسی دیگر از دامنه‌ی انتخاب‌ها خارج می‌شوند که نتیجه‌ی این کار، کوچک‌تر شدن فضای پاسخ می‌باشد.

بسیار پیش می‌آید که نیازمندی‌ها به‌اشتباه به‌عنوان قیود درنظر گرفته شوند. برای مثال، فرض کنید که یک شرکت حمل و نقل مانند سرویس پست ایالات متحده[1] عبارت زیر را اظهار کرده باشد: «برای حمل سریع بسته‌های پستی، اندازه‌ی آنها به طول ۱۰۸ اینچ، یا ۱۶۵ اینچ برای مجموع طول و سطح وجه دیگر و یا وزن ۱۵۰ پوند محدود می‌شود.»

آیا این عبارت نشان‌دهنده‌ی یک قید است یا یک نیازمندی؟ این جمله شامل عبارت «محدود می‌شود» است، یعنی می‌تواند یک قید باشد. ولی با کمی دقت می‌توان دریافت که این جمله تنها محدودیتی در مورد بسته‌ای که باید حمل شود بیان می‌کند، و نه یک محدودیت برای فضای پاسخ. بنابراین میتواند یک نیازمندی باشد و نه یک قید. به عبارت دیگر، سیستم باید بسته‌های بزرگ‌تر از ابعاد تعیین‌شده را مرجوع نماید؛ این یک توانمندی است که سیستم باید از خود نشان بدهد.

در عمل، نیازمندی‌ها و قیود در یک مستند بیان می‌شوند که بیانیه نیازمندی‌های نرم‌افزار (SRS) نامیده می‌شود. اولویت نیازمندی‌های مختلف سیستم نیز در این سند مشخص می‌شود. این اولویت‌ها برای برنامه‌ریزی و زمان‌بندی پروژه به کار می‌آیند؛ نیازمندی‌هایی که اولویت بالاتری دارند، باید زودتر توسعه‌یافته و مستقر شوند، به طوری‌که تیم توسعه‌دهنده، زمان و منابع کافی برای تغییر احتمالی سیستم در پاسخ به بازخوردهای کاربران را در اختیار داشته باشد.

برای بیشتر پروژه‌های واقعی، SRS بخشی از قرارداد توسعه‌ی نرم‌افزار است که با خود تعهدات حقوقی به همراه می‌آورد. به این ترتیب در زمان تحلیل نیازمندی‌های نرم‌افزار باید از توانایی سیستم برای برآورده کردن نیازمندی‌ها و محدودیت‌های خواسته شده، اطمینان حاصل گردد. در بیشتر موارد، این تضمین کار ساده‌ای نیست. به‌خصوص در مواردی که سیستم از فناوری‌های پیشرفته استفاده می‌کند، باید محدویت‌های بلادرنگ بودن را برآورده سازد و یا ویژگی‌های خلاقانه‌ای از خود نشان دهد. برای به‌حداقل‌رساندن این تردیدها، مطالعه‌ی امکان‌پذیری معمولاً در مرحله‌ی تحلیل نیازمندی‌ها انجام می‌گیرد.

با بررسی امکان‌پذیری سیستم، پیش از طراحی و پیاده‌سازی سیستم اطمینان حاصل می‌شود که سیستم، نیازمندی‌ها و قیود خواسته‌شده را برآورده می‌سازد. پس از پیاده‌سازی، با انجام آزمون پذیرش می‌توان بررسی کرد که آیا در عمل نیز این اتفاق رخ داده است یا خیر! برای بسیاری از پروژه‌های واقعی، برنامه‌ریزی برای آزمون پذیرش، در مرحله‌ی استخراج نیازمندی‌ها انجام می‌شود. برنامه‌ی آزمون پذیرش، شامل روش‌های آزمون، شرایط پذیرش، منابع و مسئولیت‌های انسانی، تجهیزات موردنیاز، زمان‌بندی آزمون، و موارد مرتبط دیگر است. برنامه‌ی آزمون، بخشی از برنامه‌ی کلی پروژه است که در مرحله‌ی برنامه‌ریزی ایجاد می‌شود تا راهنمای کار توسعه‌ی نرم‌افزار باشد.

---

[1] United States Postal Service (USPS)

۴-۲ اهمیت استخراج نیازمندی‌ها

برای درک بهتر اهمیت استخراج نیازمندی‌ها، به دو ماجرای واقعی توجه نمایید. بیش از ۴۰ سال پیش در آغاز دهه‌ی ۱۹۷۰ میلادی، این فرصت را داشتم که بر روی یک پروژه برای صنعت برق[1] کار کنم. به مدت دو سال، شب و روز کار کردیم؛ با نمایندگان مشتری ملاقات می‌کردیم، و طراحی و پیاده‌سازی و آزمون انجام می‌دادیم. در نهایت، ما سیستم را به مشتری تحویل دادیم و این موفقیت را جشن گرفتیم. شادی ما به‌زودی تبدیل به غم شد، زیرا مشتری با این بهانه که سیستم کاری که انتظار داشته است را انجام نمی‌دهد، حتی یک سنت به ما پرداخت نکرد. پروژه، به تمام معنا شکست خورد. سال‌ها بعد، این تجربه‌ی ناراحت‌کننده مرا برآن داشت که مهندسی نرم‌افزار را به‌عنوان یک حرفه دنبال کنم.

ماجرای دیگر، چند سال قبل بین یک سازمان خدمات درمانی دولتی و یک شرکت نرم‌افزاری خصوصی اتفاق افتاد. شرکت نرم‌افزاری، یک سیستم مدیریت درمان برای سازمان مربوطه ساخته بود. سیستم نصب شده و در حال کار بود، ولی شرکت، تنها بخشی از مبلغ قرارداد را دریافت کرد و مشتری از پرداخت هزینه‌ی توسعه‌ی نرم‌افزار خودداری نمود. شرکت نرم‌افزاری به دلیل عدم پرداخت بقیه‌ی قرارداد، از سازمان دولتی شکایت کرد. در مقابل، سازمان دولتی نیز با مطرح نمودن برآورده نشدن نیازهای خود توسط سیستم، علیه شرکت نرم‌افزاری اقامه‌ی دعوی کرد. در مدت زمانی که شکایت‌ها در جریان بود، هر دو طرف به نوبه‌ی خود متضرر شدند. نتیجه آن‌که شناخت نیازمندی‌های واقعی، هم برای توسعه‌دهنده و هم برای مشتری از اهمیت بالایی برخوردار است.

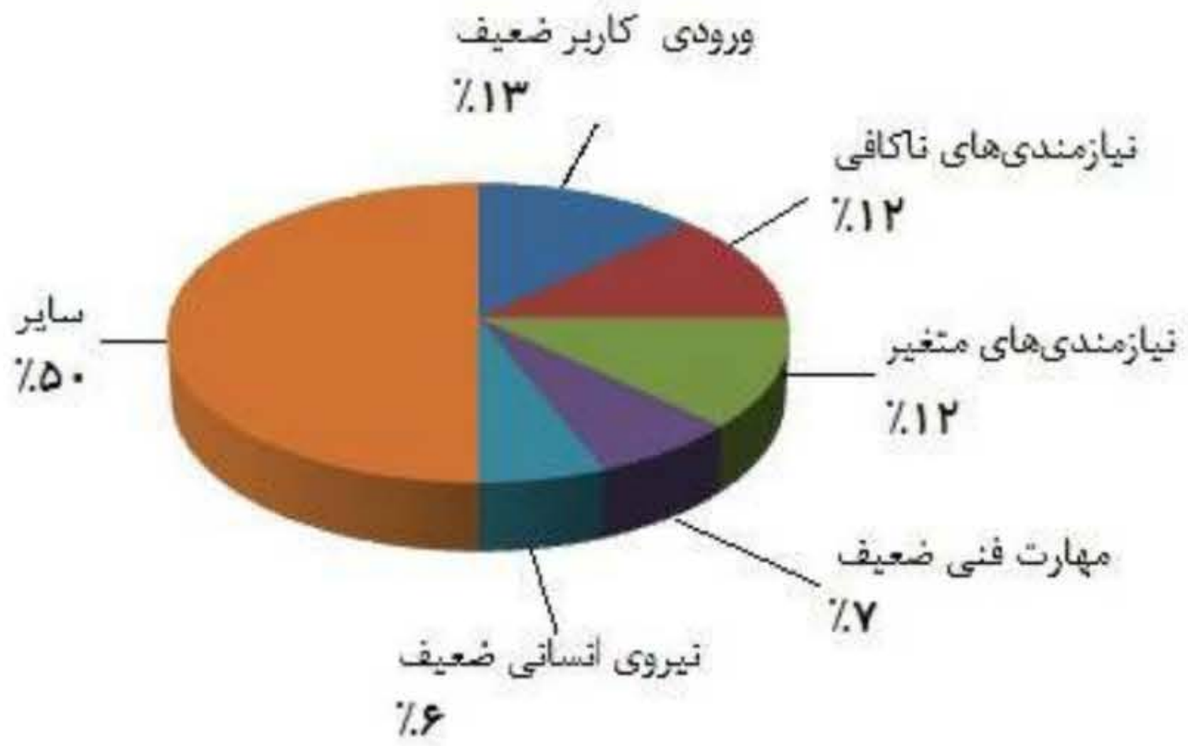
دو ماجرایی که ذکر شد، غیرمعمول و نادر نیستند. بسیاری از پروژه‌ها به دلیل تشخیص نادرست و یا شرح ناکافی نیازمندی‌ها به شکست می‌انجامند. همان‌طور که در شکل ۴-۱ نشان داده شده است، ۳۷% از کل مشکلات توسعه‌ی نرم‌افزار به نیازمندی‌های آن مربوط می‌شود [۹۲]. فردریک بروکز، برنده جایزه‌ی تورینگ در سال ۱۹۹۹، از تجربیات متعدد خود در زمینه‌ی توسعه‌ی نرم‌افزار صحبت می‌کند و دلیل این امر را چنین می‌گوید: «دشوارترین بخش از ساخت یک سیستم نرم‌افزاری، تصمیم‌گیری دقیق در مورد چیزی است که باید ساخته شود. هیچ بخش دیگری از کار به سختی این نیست که نیازمندی‌های تفصیلی تکنیکی، شامل همه‌ی واسط‌ها با افراد، ماشین‌ها، و سایر سیستم‌های نرم‌افزاری را مشخص کنیم. درست انجام ندادن هیچ بخش دیگری از کار، این چنین سیستم را از کار نمی‌اندازد. اصلاح بعدی هیچ بخش دیگری از کار، به این اندازه دشوار نیست.»

۴-۳ چالش‌های استخراج نیازمندی‌ها

شناسایی نیازمندی‌های واقعی دشوارترین بخش از توسعه‌ی نرم‌افزار است، زیرا باید در خلال فرایند استخراج نیازمندی‌ها بر مشکلات مهمی غلبه گردد. برخی از این مشکلات از این واقعیت مهم نشأت می‌گیرند که توسعه‌ی نرم‌افزار، آن‌گونه که در فصل ۲ شرح داده شد، معمولاً یک مسئله‌ی بدرفتار است. نمونه‌هایی از این مشکلات عبارتند از:

۱. تیم توسعه، اطلاعات کافی در مورد کاربرد و دامنه‌ی آن ندارد. برای بسیاری از پروژه‌های واقعی، وجود دانش عمیق در مورد کاربرد و دامنه‌ی آن برای موفقیت پروژه حیاتی است. این بدان علت است که در بسیاری از موارد، سیستم نرم‌افزاری هم وظایف همتای دستی خود را انجام می‌دهد و هم از طریق دستگاه‌های سخت‌افزاری با کاربرد در تعامل است. در این شرایط، باید درک خوبی از کاربرد و دامنه‌ی آن وجود داشته باشد. این در حالی است که بخش بزرگی از دانش کاربرد و دامنه‌ی آن فقط می‌تواند پس از سال‌ها تجربه‌ی کاری به‌دست بیاید. این مسئله، دلیل تأکید روش‌های چابک بر همکاری مشتری و مشارکت فعال کاربر برای کسب دانش دامنه می‌باشد.

[1] Electric Utility Industry



شکل ۴-۱: مشکلات مربوط به نیازمندی‌ها در توسعه‌ی نرم‌افزار

۲. مشتری‌ها و کاربران نمی‌دانند که نرم‌افزار می‌تواند چه کاری انجام دهد و چگونه می‌توانند نیازهایشان را بیان کنند. برای بیشتر کاربران، نرم‌افزار یک معما و یک پازل است. آنها نمی‌دانند که نرم‌افزار چیست، چگونه کار می‌کند، برای آنها چه کاری می‌تواند انجام بدهد، چگونه با آنها کار خواهد کرد، و چگونه با نیازهای آنان ارتباط برقرار خواهد کرد. رسم بر آن است که در این‌گونه موارد، توسعه‌دهندگان نرم‌افزار، بدون دریافت نظر مشتری و کاربران برای آنها تصمیم می‌گیرند. گاهی توسعه‌دهنده‌ی نرم‌افزار این‌گونه می‌گوید: «شما نمی‌دانید، پس ما برایتان تصمیم می‌گیریم»، یا «این‌ها ویژگی‌های خوبی است که شما باید دوست داشته باشید.» یک خاطره، احساس کاربران را از این نوع برخورد نشان می‌دهد: نویسنده و دانشجویانش برای گرفتن بازخورد کاربران از واسط کاربری طراحی‌شده برای یک سیستم خدمات درمانی با این افراد ملاقات کردند. یک پرستار ارشد، این بازدید را چیز جدیدی می‌دانست: «در گذشته همیشه به ما یک نرم‌افزار داده می‌شد که از آن استفاده کنیم، ولی این‌بار شما به نظرات ما گوش می‌دهید.» واقعیت این است که تیم توسعه‌دهنده باید در زمان استخراج نیازمندی‌ها از نزدیک با مشتری و کاربران کار کند، و آنها را برای بیان آنچه می‌خواهند و آنچه دوست ندارند تشویق نماید.

۳. نبودن زمینه‌ی مشترک بین تیم، مشتری، و کاربران، بین آنها یک سد ارتباطی ایجاد می‌نماید. فقدان درک متقابل از زمینه‌ی کاری تیم، مشتری، و کاربران یک سد ارتباطی بین آنها ایجاد می‌نماید. بسیار پیش می‌آید که یک کلمه، اصطلاح، یا عبارت در دامنه‌های مختلف معانی متفاوت داشته باشد. برای مثال، کلمه‌ی «امنیت» می‌تواند به معنای حفاظت و امنیت فیزیکی کارمندان باشد، و یا حفاظت از اطلاعات، منابع محاسباتی، و سرویس‌ها را نشان دهد. در یک دامنه‌ی مالی، امنیت در مورد محصولات سرمایه‌ای مانند سرمایه، قرارداد، پشتوانه‌ی مالی، و مانند آن مطرح می‌گردد. این، یکی از مثال‌های بی‌شماری است که معانی مختلف یک عبارت در دامنه‌های گوناگون و یا زمینه‌های مختلف یک دامنه را نشان می‌دهد.

۴. نیازمندی‌های نرم‌افزار نمی‌توانند به صراحت توصیف شوند؛ توصیف و پیاده‌سازی قابل جدا کردن نیستند. این امر، از ویژگی‌های مسائل بدرفتار است. متأسفانه، توسعه‌ی نرم‌افزار ذاتاً یک مسئله‌ی بدرفتار است (فصل ۲ را ببینید). برای مثال، نیازمندی‌های ساخت یک «واسط کاربرپسند» برای یک سیستم جدید را در نظر بگیرید. توصیف دقیق و کامل این نیازمندی‌ها، عملاً کاری ناممکن است. این بدان علت است که «کاربرپسند بودن» یک ویژگی کیفی است و هیچ راهی برای قضاوت علمی در مورد آن وجود ندارد. معمولاً برای کمک به توصیف این نیازمندی، واسط کاربر نمونه‌سازی

می‌گردد ولی این کار نیز به معنای پیاده‌سازی است و نه فقط توصیف.

۵. مدیران، مشتریان، کاربران، و نیز توسعه‌دهندگان معمولاً اهمیت و دشواری استخراج نیازمندی‌ها را دست‌کم می‌گیرند. نتیجه‌ی این‌کار، تخصیص زمان و منابع ناکافی برای استخراج نیازمندی‌ها می‌باشد.

۶. نیازمندی‌های غیرکارکردی، شناسایی نمی‌شوند و یا دست کم گرفته می‌شوند. نیازمندی‌های غیرکارکردی معمولاً به قابلیت‌هایی اطلاق می‌شود که به جنبه‌هایی چون کیفیت، کارایی، امنیت، و قابلیت اطمینان سیستم مربوط می‌شوند. به‌علاوه، نیازمندی‌های غیرتکنیکی، مانند نیاز به رعایت آیین‌نامه‌ها و استانداردهای صنعتی، نیز وجود دارند که معمولاً از آنها صرف‌نظر می‌شود. در بسیاری از موارد، نیازمندی‌های غیرکارکردی شناسایی و یا دست کم گرفته می‌شوند و این مشکل معمولاً تا مراحل بعدی پروژه مخفی می‌ماند. این مسئله می‌تواند زمان و هزینه‌های توسعه را بسیار افزایش دهد.

۷. نیازمندی‌ها در طول چرخه‌ی عمر نرم‌افزار، و حتی بعد از شروع به کار آن در محیط نهایی، تغییر می‌کنند. این چالش، نتیجه‌ی مستقیم چالش‌های قبل است. نیازمندی‌ها از روز اول تا آخر چرخه‌ی عمر نرم‌افزار تغییر می‌کنند. تغییر، محدود به نیازمندی‌های مربوط به کارکرد و کارایی نرم‌افزار نمی‌شود. تغییر در واسط کاربری، خصوصاً در ابتدای یک پروژه، بسیار معمول است.

۴-۴ انواع نیازمندی

نیازمندی‌های نرم‌افزار معمولاً به دو دسته‌ی کارکردی[۱] و غیرکارکردی[۲] تقسیم می‌شوند. نیازمندی‌های کارکردی، قابلیت‌های سیستم در پردازش اطلاعات را نشان می‌دهند. این نیازمندی‌ها به‌صورت جملات خبری نشان داده می‌شوند. در این جملات، سیستم نرم‌افزاری نقش نهاد را بازی می‌کند و پس از آن کلماتی چون «باید»، «می‌تواند»، و «انتظار می‌رود»، آمده و سپس توانمندی‌هایی که سیستم باید داشته باشد به‌عنوان گزاره‌ی جمله قرار می‌گیرند. برای مثال، یک نیازمندی کارکردی برای یک سیستم اجاره‌ی خودرو[۳] می‌تواند به صورت زیر باشد: «CRS باید به مشتری امکان بدهد که با جستجوی شرایطی مانند کارخانه‌ی سازنده، مدل، تاریخ اجاره، تاریخ پایان اجاره، حدود قیمت، و کلاس خودرو (اندازه‌ی کوچک، اندازه‌ی بزرگ، لوکس)، در مورد دردسترس‌بودن خودرو مورد نظرش برای اجاره تحقیق کند.»

مهم است که استفاده از عبارت‌هایی مانند «باید»، «می‌تواند»، و «انتظار می‌رود» در سراسر توصیف به‌صورت یکنواخت و سازگار صورت پذیرد. به بیان دیگر، اگر «باید» مورد استفاده قرار گرفته است، برای حرفه‌ای به‌نظر رسیدن شرح نیازمندی‌ها، باید این کلمه در توصیف همه‌ی نیازمندی‌های دیگر نیز به‌کار رود. راه دیگر آن است که نیازمندی‌ها به شکل تواناییهای کاربران بیان شود. به‌عنوان مثال، نیازمندی بالا می‌تواند به شکل زیر نوشته شود: «مشتری باید بتواند با جستجوی شرایطی مانند کارخانه‌ی سازنده، مدل، تاریخ اجاره، تاریخ پایان اجاره، حدود قیمت، و کلاس خودرو (سایز کوچک، سایز بزرگ، لوکس)، در مورد در دسترس بودن خودرو مورد نظرش برای اجاره تحقیق کند.»

نیازمندی‌های غیرکارکردی به دسته‌های متعددی از نیازمندی‌ها تقسیم می‌شوند. در ادامه، برخی از معمول‌ترین این دسته‌ها را می‌بینید:

نیازمندی‌های کارایی: مؤثربودن[۴] و بازدهی[۵] سیستم را مشخص می‌کنند. این نیازمندی‌ها جملاتی در مورد توان عملیاتی[۶]، زمان پاسخ[۷]، پردازش بلادرنگ[۸]، و به‌کارگیری منابع[۹] هستند. برای مثال، «سیستم اجاره‌ی خودرو باید در هر روز

---

[۱] Functional
[۲] Non Functional
[۳] Car Rental System (CRS)
[۴] Effectiveness
[۵] Efficiency
[۶] Throughput
[۷] Response Time
[۸] Real-Time Processing
[۹] Resource Utilization

۱۰۰۰۰۰ تراکنش را پردازش کند»، یک نیازمندی کارایی است. یک سیستم آمارگیری جاده‌ای «باید ۱۰۰۰۰۰۰ کیلوبایت داده ورودی را در یک ثانیه پردازش کند»، مثال دیگری است.

نیازمندی‌های کیفیت: جملاتی درمورد کیفیت مورد نیاز نرم‌افزار هستند. این نوع، به محدوده‌ی وسیعی از ویژگی‌های مطلوب نرم‌افزار شامل قابلیت اطمینان، در دسترس بودن، نرخ خطای کشف شده، و مانند آن اطلاق می‌شود. یک نیازمندی قابلیت اطمینان می‌تواند به این شکل بیان شود: «سیستم باید در ۹۹% زمان‌ها در دسترس باشد.»

نیازمندی‌های ایمنی: قابلیت سیستم را در جلوگیری از ورود به یک حالت ناخواسته به دلیل انجام عملیات پیش‌بینی‌نشده توصیف می‌کنند. چنین رخدادی می‌تواند باعث آسیب‌های مالی و جانی، و یا پیامدهای غیرقابل‌انتظار دیگر شود. برای مثال، «وقتی یک تراکنش شکست می‌خورد، سیستم باید سابقه‌ی تراکنش‌های انجام شده را ضبط و از آن نگهداری کند». گاهی نیازمندی‌های ایمنی از استانداردی که برای اطمینان از ایمنی محصولات یا سرویس‌ها تعریف شده است گرفته می‌شود .

نیازمندی‌های امنیت: قابلیت سیستم در حفاظت از منابعی چون داده‌ها و برنامه‌ها در برابر حملات بدخواهانه را شرح می‌دهند. چنین حملاتی می‌توانند از داخل یا خارج سازمان ترتیب داده شوند. به‌عنوان مثال، «سیستم باید همه‌ی فایل‌هایی که از طریق اینترنت انتقال می‌یابد را رمزگذاری کند». نیازمندی‌های امنیتی در فصل ۲۴ شرح داده شده‌اند.

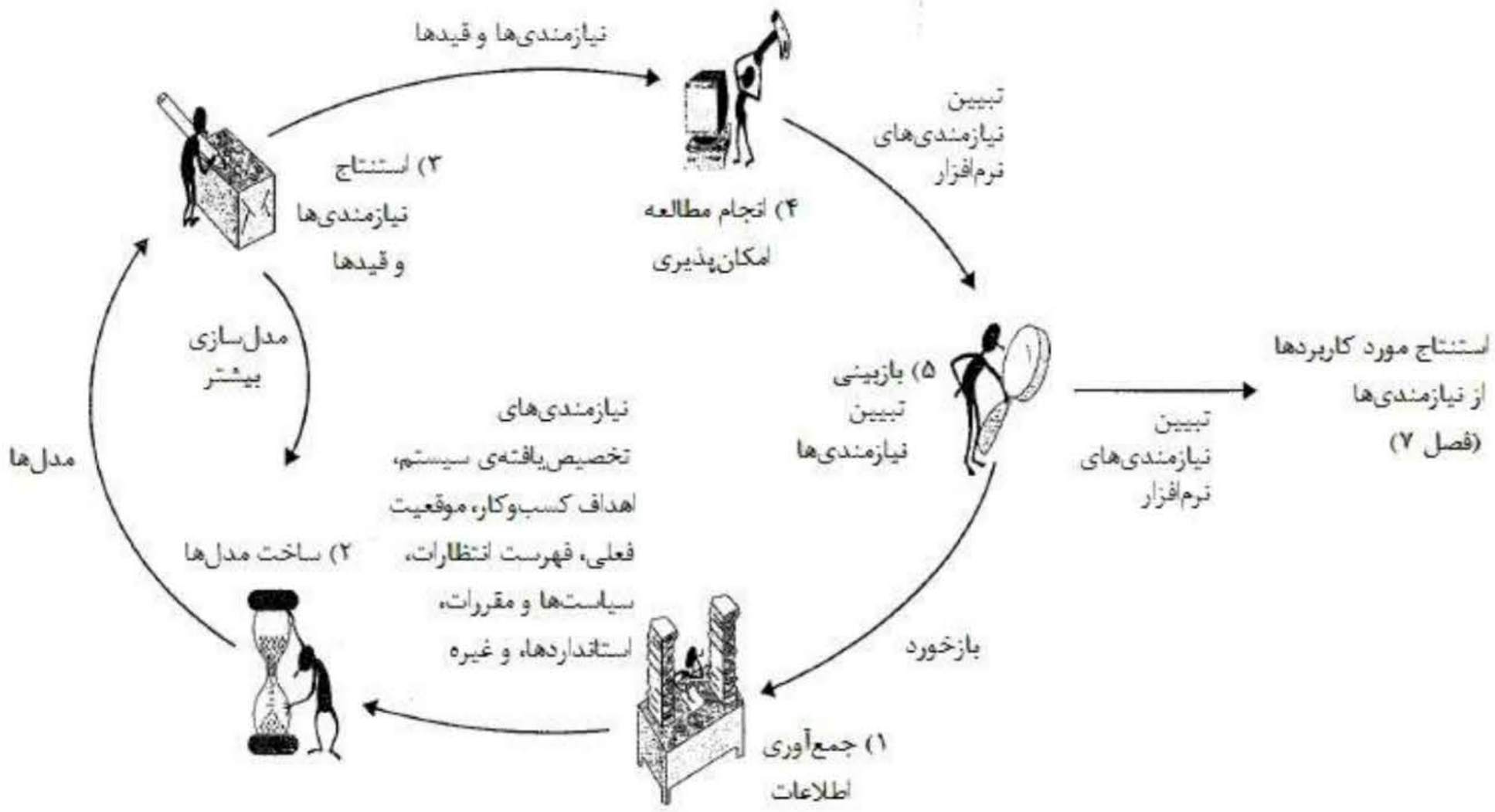
نیازمندی‌های واسط: ظاهر و حس، و رفتار تعاملی و واسطه‌ای که سیستم باید برای ارتباط با موجودیت‌های خارجی از خود نشان دهد را توصیف می‌کند. این نیازمندی‌ها شامل نیازمندی‌های واسط کاربری و نیازمندی‌های واسط سخت‌افزاری/نرم‌افزاری است. اولی به طرح، ظاهر، حس، رفتار تعاملی با کاربر، و سایر نیازمندی‌های مرتبط با واسط کاربری اشاره می‌کند. نیازمندی‌های واسط سخت‌افزاری/نرم‌افزاری، نشان‌دهنده‌ی توانمندی‌های موردنیاز سیستم نرم‌افزاری برای کار با سیستم‌های نرم‌افزاری و یا سخت‌افزاری می‌باشند. این نیازمندی‌ها باید حداقل سیستم‌های سخت‌افزاری/نرم‌افزاری که باید با آن تعامل صورت گیرد، نحوه‌ی ارتباط با هر یک از این سیستم‌ها، و زمان برقراری این ارتباط را مشخص کنند.

۴-۵ گام‌های استخراج نیازمندی‌ها

هدف کلی از استخراج نیازمندی‌های نرم‌افزار، شناسایی توانمندی‌های سیستم آتی است. همان‌طور که در شکل ۴-۲ می‌بینید، استخراج نیازمندی‌ها شامل گام‌های زیر است که ممکن است چندین مرتبه تکرار شوند:

گام ۱: جمع‌آوری اطلاعات در مورد کاربرد.
گام ۲: ایجاد مدل‌های تحلیل در صورت لزوم.
گام ۳: استخراج نیازمندی‌ها و محدودیت‌ها.
گام ۴: انجام مطالعه‌ی امکان‌سنجی.
گام ۵: مرور شرح نیازمندی‌ها.

استخراج نیازمندی‌ها با جمع‌آوری اطلاعات در باره‌ی کاربرد و دامنه‌ی آن آغاز می‌شود. تکه‌های بسیاری از این اطلاعات برای درک کاربرد و استخراج نیازمندی‌ها و محدودیت‌ها مفید هستند. این اطلاعات شامل اهداف کسب‌وکاری، وضعیت فعلی کسب‌وکار، سیاست‌ها، مقررات، و استانداردها می‌باشند. سپس اطلاعات جمع‌آوری‌شده تحلیل می‌گردد تا مشکلات شناخته شود و نیازها از این مشکلات استخراج گردد. گاهی مدل‌هایی برای کمک به فهم کاربرد و فرایندهای کسب‌وکاری ساخته می‌شوند. البته با توجه به محدودیت‌های بودجه و زمان توسعه، لزوماً همه‌ی نیازها نمی‌توانند برآورده گردند. بنابراین، تیم باید



شکل ۴-۲: استخراج نیازمندی‌های نرم‌افزار

با مشتری و کاربران کار کند تا نیازهای با اولویت بالای آنها را شناسایی کرده، نیازمندی‌هایی که این احتیاجات را برآورده می‌سازند استخراج نماید. گاهی برای حذف یا کاهش موارد ریسک، باید در زمان استخراج نیازمندی‌ها، مطالعه‌ی امکان‌پذیری انجام گردد. سپس نیازمندی‌ها و محدودیت‌ها به‌ترتیب توسط همکاران، متخصصین دامنه، و مشتری و کاربران مرور می‌شود. نتایج این کار برای بهبود توصیف نیازمندی‌ها مورد استفاده قرار می‌گیرد، تا موارد کاربرد از آن استخراج شود.

۴-۵-۱- جمع‌آوری اطلاعات

استخراج نیازمندی‌ها یک فرایند تصمیم‌گیری است که طی آن در مورد توانمندی‌های سیستم آینده تصمیم‌گیری می‌شود. برای تصمیم‌گیری، باید اطلاعات تحلیل شوند. بنابراین نخستین گام، جمع‌آوری اطلاعات درخصوص کاربرد است. این بخش به پرسش‌های زیر در مورد جمع‌آوری اطلاعات پاسخ می‌دهد:

۱. چه اطلاعاتی باید جمع‌آوری شود؟
۲. روش‌ها و تکنیک‌های جمع‌آوری اطلاعات کدامند؟
۳. رهنمون‌های جمع‌آوری اطلاعات کدامند؟

نقاط تمرکز فعالیت‌های جمع‌آوری اطلاعات

فعالیت‌های جمع‌آوری اطلاعات باید بر کسب اطلاعات در خصوص کاربرد، فرایندهای مربوط به کسب‌وکار، و دامنه‌ی کاربرد متمرکز شوند. فعالیت‌های جمع‌آوری اطلاعات باید در پاسخ‌گویی به پرسش‌های زیر کمک کنند:

۱. کسب‌وکاری که سیستم کامپیوتری برای آن ساخته می‌شود چیست؟

۲. وضعیت فعلی این کسب‌وکار چیست و چگونه انجام می‌شود؟
۳. محیط یا زمینه‌ی کاری سیستم چیست؟
۴. فرایندهای فعلی این کسب‌وکار، و ورودی و خروجی آنها کدامند و چگونه باهم ارتباط برقرار می‌کنند؟
۵. در سیستم فعلی چه مشکلاتی وجود دارد؟
۶. اهداف کسب‌وکار یا محصول مربوطه چه هستند؟
۷. کاربران سیستم فعلی و سیستم آینده چه کسانی هستند؟
۸. مشتری و کاربران از سیستم آینده چه می‌خواهند و اولویت‌های کسب‌وکار آنها کدامند؟
۹. در زمینه‌ی کیفیت، کارایی، و امنیت چه نکاتی باید مورد توجه قرار گیرد؟

تکنیک‌های جمع‌آوری اطلاعات

روش‌ها و تکنیک‌های جمع‌آوری اطلاعات، برای یافتن پاسخ به سؤالاتی که دیدیم به‌کار می‌روند. این تکنیک‌ها شامل موارد زیر می‌باشند:

۱. معرفی مشتری[۱]
۲. بررسی سوابق موجود[۲]
۳. مطالعه‌ی رویه‌ها و فرم‌های موجود برای کسب‌وکار موردنظر
۴. تحقیق در مورد ذی‌نفعان[۳]
۵. گفت‌وگو با کاربران
۶. نوشتن داستان‌های کاربران[۴]

معرفی مشتری، نقطه‌ی آغاز خوبی برای جمع‌آوری اطلاعات است. با این کار، تیم توسعه به‌سرعت به مشتری معرفی می‌شود و به اعضای تیم اجازه داده می‌شود که با پرسیدن سؤالاتی، موارد ابهام را برطرف نمایند. همچنین این معرفی، مفهوم اولیه‌ای از سیستم در حال توسعه ارائه می‌نماید. برای این‌که این فعالیت، کارآمد باشد توجه به نکات زیر ضروری است:

- مطمئن شوید که کسی که معرفی را انجام می‌دهد، کسب‌وکار موردنظر، و آن‌چه مشتری و کاربران می‌خواهند را می‌شناسد، گرچه اطلاعات او ممکن است کامل یا ۱۰۰% صحیح نباشد. یک مدیر یا رئیس که مسئول کارهای روزانه است، بهترین انتخاب برای این کار است. یک فرد تازه‌کار ممکن است گزینه‌ی خوبی برای این کار نباشد.
- اجازه دهید که مشتری یا نماینده‌ی او، آن‌چه باید پوشش داده شود را بدانند. به‌علاوه، برای به‌دست آوردن اطلاعات اضافی تلاش کنید.
- در صورت امکان، یکی از اعضای تیم باید اسلایدهای معرفی را پیش از جلسه مرور کند تا اطمینان یابد که اطلاعات لازم در آن پیش‌بینی شده است.
- در طول معرفی فقط سؤالاتی را بپرسید که نقاط ابهام را برطرف کند و از توضیحات مربوط به پیاده‌سازی بپرهیزید. هدف از جلسه‌ی معرفی، شناختن کسب‌وکار و آن‌چه کاربر انتظار دارد سیستم برایش انجام دهد است.
- یک نسخه از ارائه را بگیرید و بین افراد تیم توزیع کنید.
- از جلسه‌های معرفی طولانی که بیش از دو ساعت به طول می‌انجامد، پرهیز نمایید.

---

[۱] Customer Presentation
[۲] Literature Survey
[۳] Stakeholder Survey
[۴] User Stories

با مطالعه‌ی پروژه‌های مشابه، فرم‌ها و مستندات مربوط به آن کسب‌وکار، استانداردهای صنعتی، سیاست‌ها و آیین‌نامه‌های دولتی مربوطه، و سایر مطالب مرتبط می‌توانید دید بسیارخوبی نسبت به اطلاعات سیستمی و حرفه‌ای مشتری به دست آورید. علاوه‌براین، تیم توسعه می‌تواند اطلاعاتی را از طریق شرکت در کلاس‌های برگزارشده توسط متخصصین آن دامنه به‌دست بیاورد. اگر اطلاعاتی در مورد پروژه‌های مشابه دردست باشد، می‌تواند دید بسیار ارزشمندی نسبت به سیستم جدید در حال توسعه به‌دست بدهد. برای مثال، نیازمندی‌های نرم‌افزاری سیستم جدید می‌تواند از توانمندی‌های سیستم مشابه نتیجه گرفته شود. همچنین طراحی و پیاده‌سازی سیستم جدید نیز می‌تواند از طراحی و پیاده‌سازی سیستم‌های مشابه الهام گرفته شود.

بسیاری از مستندات کسب‌وکار، رویه‌ها، کتابچه‌های راهنما، و فرم‌ها اطلاعات ارزشمندی هستند. با مطالعه‌ی این منابع، تیم توسعه می‌توانند درک بهتری از فرایندهای کسب‌وکار و ورودی و خروجی این فرایندها به‌دست آورند. روشن است که سیستم آینده باید همه یا برخی از این فرایندهای کسب‌وکار را خودکار نماید. بنابراین تیم توسعه باید از مشتری بخواهد که همه‌ی مستندات و فرم‌های موجود را برایش فراهم کند و سپس آنها را به دقت مطالعه نماید. هر شرکت و واحد صنعتی، استانداردهای خاص خود را دارد. سیستم‌های کامپیوتری باید از این استانداردها پشتیبانی کنند. این استانداردها می‌توانند برای استخراج نیازمندی‌ها یا محدودیت‌ها به کار گرفته شوند. آیین‌نامه‌ها و سیاست‌های خاص دولتی، صنعتی، و درون سازمانی معمولاً محدودیت‌هایی را بر فضای پاسخ اعمال می‌کنند. این محدودیت‌ها معمولاً با عنوان قیود توصیف نیازمندی‌های نرم‌افزار تنظیم می‌شوند.

یک پرسش‌نامه‌ی مطالعاتی، ابزار خوبی برای جمع‌آوری اطلاعات از ذی‌نفعان، شامل پرسنل مدیریتی و کاربران نهایی است. این تکنیک مزایای زیادی دارد. اول اینکه برای حل مسائل بدرفتار، درگیرکردن ذی‌نفعان در فرایند تصمیم‌گیری دارای اهمیت است. نظرخواهی از ذی‌نفعان، آنها را تشویق می‌کند که دیدگاه‌ها، اطلاعات داخلی و توصیه‌های خود برای بهبود کار را بیان کنند. به‌دست آوردن چنین اطلاعاتی از راه‌های دیگر، دشوار است. این اطلاعات می‌تواند برای استخراج نیازمندی‌های سیستم آینده مورد استفاده قرار بگیرد. نکات زیر باید در زمان استفاده از پرسش‌نامه مورد توجه قرار گیرد:

- پرسش‌نامه باید کوتاه و مختصر باشد و تنها بر نکات مهم تمرکز نماید.
- در صورت امکان، پرسش‌نامه‌های مختلف برای گروه‌های متفاوت از ذی‌نفعان طراحی گردد.
- سوالات پرسش‌نامه با همکاری نماینده‌ی مشتری مرور شود تا از مناسب بودن و وضوح آنها اطمینان حاصل شود.

مصاحبه با نمایندگان مشتری، کاربران، و متخصصین دامنه، راه کارامدی برای جمع‌آوری اطلاعات است. این مصاحبه‌ها می‌تواند رودررو، و یا از طریق تلفن انجام شود. طبیعت تعاملی مصاحبه به دو طرف اجازه می‌دهد که در زمینه‌ی موضوعات مربوط به سیستم مورد نظر وارد بحث‌های عمقی شوند.

رهنمون‌های زیر در زمان برگزاری مصاحبه مفید خواهند بود:

- زمان مصاحبه را بعد از آن که تیم توسعه پرسش‌نامه‌های تکمیل‌شده توسط مشتری و کاربران را مطالعه کرد، قرار دهید. ممکن است پرسش‌نامه‌ها حاوی اطلاعاتی باشد که تیم توسعه می‌خواهند بدانند. بنابراین، با موکول کردن مصاحبه به بعد از مطالعه‌ی پرسش‌نامه‌ها در کار و زمان صرفه‌جویی می‌شود. پاسخ پرسش‌نامه‌ها می‌تواند تیم را در مورد زمینه‌ها و نکاتی که باید در طول مصاحبه موردتوجه قرار گیرد، راهنمایی کند.
- مصاحبه باید با تدارک قبلی و تمرکز کامل انجام شود. یعنی تیم توسعه باید قبلاً فهرستی از سوالات را برای پرسیدن در مصاحبه تهیه کند. این لیست می‌تواند از سوالات پرسش‌نامه تهیه شود. سوالات باید بر نکات مهم و یا بحرانی که باید توسط سیستم کامپیوتری حل شود تمرکز نماید.
- طول هر جلسه‌ی مصاحبه نباید بیشتر از یک ساعت باشد.

- به نظرات مشتری و کاربران احترام بگذارید. آنها در حرفه‌ی خود متخصص هستند. از توصیه به آنها در زمان مصاحبه خودداری نمایید.

توجه به قوم‌نگاری[۱] در مرحله‌ی جمع‌آوری نیازمندی‌ها از اهمیت زیادی برخوردار است. دانش مردم‌شناسی به تیم توسعه توصیه می‌کند که به‌جای نگاه از خارج، از داخل به فرهنگ سازمانی کسب‌وکار مورد نظر، که سیستم برای آن ساخته می‌شود، نگاه کنند. یعنی استخراج نیازمندی‌ها به‌جای روش «خارج به داخل» به شیوه‌ی «داخل به خارج» انجام شود. برای این کار، تیم باید کاربران را درک کند و مانند آنها فکر کند. یک تحلیل‌گر بی‌تجربه‌ی سیستم، معمولاً نقش پزشکی را ایفا می‌کند که برای مشکلات حرفه‌ای مشتری دارو تجویز می‌کند. قوم‌نگاری توصیه می‌کند که افراد تیم به‌دقت مشاهده کنند که کاربران چگونه کار می‌کنند و چه‌طور مشکلات کسب‌وکار خود را حل می‌کنند. روش‌های چابک نیز از قانون مشابهی طرفداری می‌کنند و آن، لزوم مشارکت فعال کاربر در فرایند تحلیل است. برای مثال، DSDM این مشارکت فعال را با تعریف نقش‌های «کاربر مشاور[۲]» و «کاربر نماینده[۳]» در تیم پیاده‌سازی می‌کند.

روش‌های چابک به‌طور گسترده‌ای از داستان‌های کاربران به‌عنوان روش جمع‌آوری نیازمندی‌ها استفاده می‌کنند. داستان‌های کاربران، توسط کاربران و یا با مشارکت تیم توسعه تهیه می‌شوند. هر داستان کاربر، به‌اختصار یک توانمندی را شرح می‌دهد که یک کاربر علاقمند است داشته باشد. هر کاربر ممکن است چندین داستان داشته باشد. این داستان‌ها، کارآمد هستند زیرا به روش «داخل به خارج» تهیه می‌شوند و نه «خارج به داخل». موارد زیر، نمونه‌هایی از داستان‌های متعدد کاربران است که در یک پروژه‌ی واقعی ایجاد می‌شوند:

۱. به‌عنوان یک دستیار پزشکی، من باید رکوردهای جدیدی برای بیماران جدید ایجاد کنم.
۲. به‌عنوان یک دستیار پزشکی، من باید رکوردهای بیماران را با استفاده از تاریخ تولد، و یا نام و نام‌خانوادگی بیمار جستجو و بازیابی کنم.
۳. به‌عنوان یک دستیار پزشکی، من باید رکوردهای آزمایش‌های پزشکی بیمار را دنبال کنم.
۴. به‌عنوان یک پزشک، من باید همه‌ی گفت‌وگوها با یک بیمار را ثبت و مستندسازی کنم.
۵. به‌عنوان یک پزشک، من باید برای یافتن اطلاعات خاص، به دفعات، گفت‌وگوهای انجام‌شده با یک بیمار را جستجو کنم.
۶. به‌عنوان یک پزشک، من باید رکوردهای آزمایش‌های پزشکی یک بیمار را جستجو کنم.
۷. به‌عنوان یک پزشک، من می‌خواهم از منزل یا هر جای دیگر به سیستم دسترسی داشته باشم.
۸. به‌عنوان یک مسئول امنیت اطلاعات، من می‌خواهم سیستم را از دسترسی غیرمجاز حفظ کنم.

این گام، فهرستی شامل اهداف کسب‌وکار، وضعیت فعلی کسب‌وکار، داستان‌های کاربران یا انتظارات کاربران و مشتری، و آیین‌نامه‌ها، سیاست‌ها و استانداردهای دولتی و صنعتی ایجاد می‌کند. اگر سیستم نرم‌افزاری یک جزء داخلی از یک سیستم نرم‌افزاری/سخت‌افزاری باشد، نتایج شامل نیازمندی‌های سیستم نرم‌افزاری نیز خواهند بود. به‌عنوان یک مثال، پروژه‌های تعریف‌شده توسط دفتر آموزش بین‌المللی[۴] یک دانشگاه را درنظر بگیرید. سال‌ها پیش، OIE بحث فراگیرکردن تحصیلات را مطرح کرد. در آن زمان، بسیاری از دانشگاه‌های امریکا رشد قابل توجهی در تعداد دانشجویانی که برای تحصیل در برنامه‌های تحصیلی دانشگاه‌های خارج از امریکا ثبت‌نام می‌کردند، مشاهده نمودند. این کار به دانشجویان اجازه می‌داد که برای یک یا دو ترم در دانشگاه‌های خارج از امریکا تحصیل کنند و سپس نمرات درسی خود را به امریکا بیاورند. برای برآورده ساختن این نیاز جدید، OIE پروژه‌ی جدیدی را برای ارتقای برنامه‌ی تحصیلی در خارج از کشور آغاز کرد. در ادامه، خلاصه‌ای از نتایج گام

---

[۱] Ethnography
[۲] Advisory User
[۳] Ambassador User
[۴] Office of International Education (OIE)

جمع‌آوری اطلاعات این پروژه را مشاهده می‌کنید:

وضعیت فعلی کسب‌وکار

OIE در ساختمانی دورتر از فضای اصلی دانشگاه قرار گرفته است. رسیدن به OIE برای دانشجویان دشوار است. برنامه‌ی تحصیل در خارج از کشور OIE کاملاً یک عملیات دستی است. پردازش و رسیدگی به درخواست‌های دانشجویان برای تحصیل در خارج، کاری وقت‌گیر است.

اهداف کسب‌وکار

۱. دسترسی دانشجویان به برنامه‌ی تحصیل در خارج OIE به‌شدت تسهیل شود.
۲. سرویس‌های ارائه‌شده توسط برنامه‌ی تحصیل در خارج، به شدت مؤثرتر و کارآمدتر شود.

فهرست انتظارات

یک وب‌گاه برای برنامه‌ی تحصیل در خارج، با عنوان سیستم مدیریت تحصیل در خارج[۱] مورد انتظار است. فهرستی از سایت‌های مشابه توسط OIE پیشنهاد داده شده بود.

۴-۵-۲- ساخت مدل‌های تحلیل

کاربردهای واقعی، پیچیده‌اند. این کاربردها با انواع مختلفی از اشیای خاص کسب‌وکار، ارتباطات پیچیده‌ی بین این اشیا، و فرایندهای کسب‌وکار درگیر هستند. تیم توسعه نیازمند این است که برای شناسایی و تنظیم نیازمندی‌ها، اطلاعاتی از این دست را از دامنه بداند. دانش دامنه برای طراحی، پیاده‌سازی، آزمون، و نگهداری سیستم نرم‌افزاری نیز مفید است. مدل‌سازی، ابزار کارآمدی برای کسب دانش دامنه است که مکرراً در زمان استخراج اطلاعات مورد استفاده قرار می‌گیرد. مدل‌سازی یک فرایند ادراک و تجسم است که به تیم توسعه در درک و تحلیل کاربرد فعلی کمک می‌کند. به ویژه، مدل‌سازی در مهندسی نرم‌افزار شیءگرا، جهان را به شکل مجموعه‌ای از اشیا می‌بیند که با هم در ارتباط و تعامل هستند. حاصل این تفکر، یک مجموعه نمودار UML است که جنبه‌های مختلف کاربرد موجود را به تصویر می‌کشد. شکل ۴-۳ برخی از نمودارهای UML و نحوه‌ی مدل‌سازی آنها از یک کاربرد موجود را نشان می‌دهد.

به‌عنوان یک مثال دیگر، شکل ۴-۴ نمودار فعالیتی است که نشان می‌دهد چگونه برنامه‌های مبادله‌ی دانشجو به‌طور دستی کنترل می‌شود. مستطیل‌های گوشه‌گرد، فعالیت‌ها را نشان می‌دهند و خطوط جهت‌دار، جریان اشیا و جریان کنترل بین فعالیت‌ها را مشخص می‌کنند. نماد لوزی به معنای یک نقطه‌ی تصمیم است و خط‌چین‌ها نشان‌دهنده‌ی سازمان‌های صاحب فعالیت می‌باشند. همان‌طورکه در نمودار نشان داده شده است، درخواست شرکت در برنامه‌ی مبادله‌ی دانشجو توسط دانشجویان ایجاد می‌شود. این درخواست‌ها توسط مشاورین OIE بررسی می‌شوند و قبول یا رد می‌گردند. اگر درخواست قبول شود، با موسسه مربوطه تماس گرفته می‌شود، به گروه دانشگاهی و دانشجو اطلاع داده می‌شود، و درخواست بایگانی می‌گردد. اگر درخواست قبول نشود، به دانشجو اطلاع داده می‌شود و درخواست بایگانی می‌گردد.

مدل‌سازی، وقت و کار زیادی را به خود اختصاص می‌دهد. به همین دلیل، تنها مدل‌هایی را بسازید که کمک قابل‌توجهی در درک و تحلیل مسئله می‌کنند. از این‌که فقط به خاطر مستندسازی مدل بسازید، بپرهیزید. توسعه‌ی چابک «مدل‌سازی به حد کفایت» را توصیه می‌کند. به‌عبارت‌دیگر کمترین حد مدل‌سازی را انجام دهید، تا آنجا که فقط نیاز شما را برآورده کند و نه چیزی بیشتر از آن.

---

[۱] Study Abroad Management System (SAMS)

نمودار UML	توضیح	کاربرد در مدل‌سازی شرایط فعلی
نمودار کلاس	یک گراف جهت‌دار است که گره‌های آن نشان‌دهنده‌ی کلاس‌ها، و یال‌های جهت‌دار آن نشان‌دهنده‌ی انواع روابط بین کلاس‌ها می‌باشد. به‌علاوه، هر گره حاوی اطلاعاتی است که ویژگی‌های کلاس را توصیف می‌کند.	نمودارهای کلاس برای به تصویرکشیدن مدل‌های دامنه مورد استفاده قرار می‌گیرند. این مدل‌ها به تیم توسعه در درک و برقراری ارتباط با مفاهیم دامنه، و آشنایی با روابط بین آنها کمک می‌کنند.
نمودار موردکاربرد	یک گراف است که گره‌های آن نشان‌دهنده‌ی فرایندهای کسب‌وکار و کنشگرهای آن می‌باشند. یال‌های این گراف نشان می‌دهند که کدام کنشگرها با کدام فرایندهای کسب‌وکار تعامل می‌نمایند.	نمودارهای موردکاربرد برای ارائه‌ی یک دید کلی از فرایندهای کسب‌وکار یک کاربرد موجود و یا یک زیرسیستم، به همراه کاربرانی که سرویس‌هایی را از این فرایندها درخواست می‌کنند، به‌کار می‌روند. به‌علاوه، این نمودارها مرزهای کاربرد، سیستم، یا زیرسیستم‌های موجود را نشان می‌دهد.
نمودار توالی	یک گراف جهت‌دار است که در آن گره‌ها نشان‌دهنده‌ی اشیا، و یال‌های جهت‌دار نشان‌دهنده‌ی پیام‌ها و درخواست‌هایی است که با ترتیب زمانی بین اشیا ردوبدل می‌شوند.	نمودارهای توالی به تیم توسعه کمک می‌کنند که فرایندهای موجود کسب‌وکار را درک و تحلیل نمایند. یعنی نشان می‌دهند که چگونه اشیای خاص آن کسب‌وکار، در محیط حرفه‌ای فعلی درخواست کاربر را پردازش می‌کنند.
نمودار حالت	یک گراف جهت‌دار است که گره‌های آن حالات سیستم را نشان می‌دهند و یال‌های آن، نشان‌دهنده‌ی انتقال از حالتی به حالت دیگر بر اثر رویدادهای داخلی یا خارجی می‌باشند.	سیستم موجود و یا فعالیت‌های آن، بسته به حالتی که در آن قرار دارند، نسبت به رویدادهای جدید واکنش نشان می‌دهند. نمودارهای حالت، این رفتارهای واکنشی را مدل‌سازی می‌نمایند.
نمودار فعالیت	یک گراف جهت‌دار است که گره‌های آن نشان‌دهنده‌ی فعالیت‌های پردازش اطلاعات هستند و یال‌های جهت‌دار آن، جریان‌های داده و کنترل بین فعالیت‌ها را نشان می‌دهند. جریان‌های کنترل شرح می‌دهند که فعالیت‌ها به‌طور متوالی، به‌طور هم‌روند، و یا به شکل هم‌گام انجام شده‌اند.	نمودارهای فعالیت برای مدل‌سازی فعالیت‌های پردازش اطلاعات به‌کار می‌روند. فعالیت‌ها از طریق روابط پیچیده‌ی جریان داده و جریان کنترل با هم مرتبط می‌شوند.

شکل ۴-۳: مدل‌سازی کاربردهای موجود با نمودارهای UML

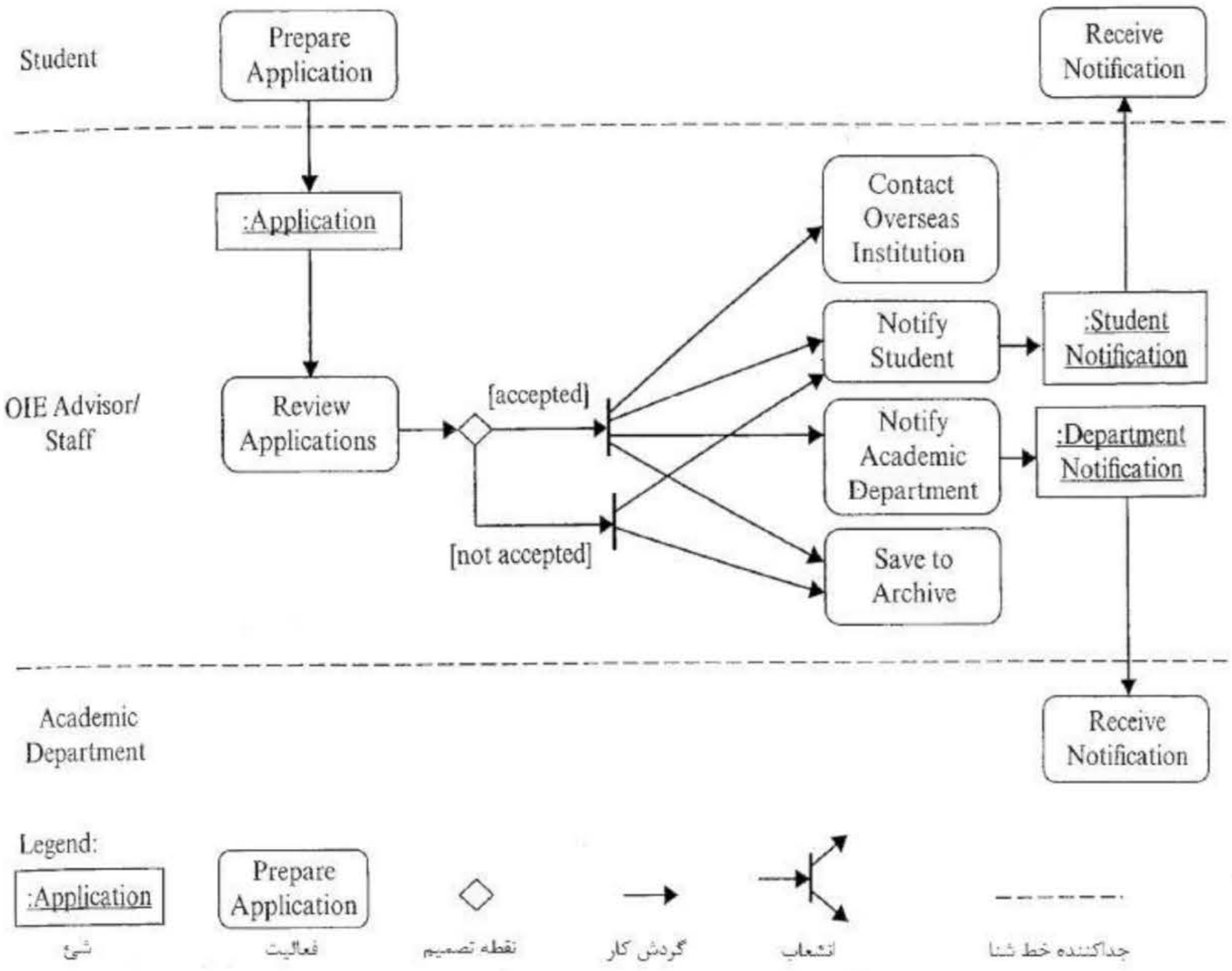
۴-۵-۳- استنتاج نیازمندی‌ها و قیدها

هدف اصلی از این گام، استنتاج نیازمندی‌ها و قیدهای نرم‌افزار از اطلاعات جمع‌آوری شده است. برای رسیدن به این هدف، اطلاعات تحلیل می‌شوند تا نیازها شناسایی گردند و قابلیت‌های مورد نیاز سیستم برای برآورده ساختن این نیازها به‌دست بیایند. در مرحله‌ی بعدی، این قابلیت‌ها به شکل نیازمندی‌ها تنظیم می‌گردند. نیازمندی‌ها با توجه به اولویت‌های کسب‌وکار اولویت‌بندی می‌شوند. تیم توسعه با مشتری، کاربران، و متخصصین دامنه کار می‌کند تا این وظایف را انجام دهد.

شناسایی نیازها از فهرست انتظارات

اگر فهرستی از انتظارات[1] و یا داستان‌های کاربران در دست باشد، تشخیص نیازها از آن کاری نسبتاً ساده است. پروژه‌ی OIE، که در بخش ۴-۵-۱ شرح داده شد، را درنظر بگیرید. پاسخ مسئله قبلاً در لیست انتظارات ذکر شده است: یک وب‌گاه برای مدیریت برنامه‌ی تحصیل در خارج از کشور توسط OIE. در این حالت کار تیم توسعه، شناسایی چیزی است که سایت باید برای تحقق اهداف حرفه‌ای فراهم نماید. برخی از نیازهای شناسایی‌شده را در ادامه می‌بینید:

[1] Wish List



شکل ۴-۴: نمودار فعالیت برای یک فرایند کسب‌وکار موجود

- دانشجویان باید بتوانند با شرایط متنوع جستجو، برنامه‌های تبادل دانشجو را جستجو نمایند.
- کارمندان OIE باید بتوانند برنامه‌های تبادل دانشجو را وارد، ویرایش، فعال، یا غیرفعال نمایند.
- دانشجویان باید بتوانند درخواست‌های خود برای برنامه‌های تبادل دانشجو را ایجاد، ویرایش، و ارسال نمایند.
- مشاوران OIE باید بتوانند درخواست‌های برخط را پردازش نمایند.

شناسایی نیازها برای برآوردن اهداف کسب‌وکار

از وضعیت فعلی کسب‌وکار و اهداف کاری آن نیز می‌توان نیازها را تشخیص داد. ابتدا فاصله‌ی بین وضعیت فعلی و اهداف کاری آینده تعیین می‌گردد. این فاصله برای شناسایی نیازها، یعنی کارهایی که باید انجام شود و یا قابلیت‌هایی که باید برای پرکردن این فاصله فراهم گردد، مورد استفاده قرار می‌گیرد. برای مثال اگر هدف کاری دو برابر کردن فروش در سه سال آینده باشد، یعنی کسب‌وکار باید قابلیت فروش خود را افزایش دهد. این کار می‌تواند به راه‌های مختلفی چون توسعه‌ی نیروهای شاغل در بخش فروش، در نظر گرفتن تخفیف‌های ویژه، و یا عرضه‌ی برخط کالاها انجام گردد. معمولاً یک راه‌حل کامپیوتری یکی از گزینه‌های ممکن است که احتمالاً مقرون‌به‌صرفه‌ترین راه‌حل برای مسئله می‌باشد. برای مثال، عرضه‌ی برخط می‌تواند برای برخی از انواع محصولات فروش را افزایش دهد، ولی این امر برای همه‌ی انواع محصول صدق نمی‌کند. تیم توسعه باید با مشتری، کاربران، و متخصصین دامنه کار کند تا مناسب‌ترین راه را برای حل مشکل آن کسب‌وکار تشخیص بدهد.

استخراج نیازمندی‌ها از نیازها

تیم می‌تواند برای استخراج قابلیت‌های سیستم جدید برای برآورده ساختن نیازها، از خود این نیازها استفاده نماید. گاهی مانند آن‌چه در مورد وب‌گاه برنامه‌ی تحصیل در خارج از کشور OIE دیدیم، نیازها، قابلیت‌ها را نیز بیان می‌کنند. در حالت‌های دیگر، تیم توسعه باید این قابلیت‌ها را از راه استنتاج به‌دست آورد. در هر صورت، قابلیت‌های شناسایی‌شده با عنوان نیازمندی‌ها معرفی می‌گردند. باید توجه داشت که لازم نیست سیستم تحت‌توسعه، همه‌ی نیازهای شناسایی‌شده را برآورده سازد. دلایل متعددی برای این امر وجود دارد. به‌عنوان مثال، ممکن است مشتری و کاربران، در حال‌حاضر نیازی به همه‌ی موارد شناسایی‌شده توسط تیم توسعه نداشته باشند. مشتری ممکن است به دلیل محدودیت بودجه نخواهد که برای برخی از قابلیت‌ها هزینه کند. ممکن است او به دلیل برخی ملاحظات کاری بخواهد که برخی از این ویژگی‌ها در نسخه‌های آینده پیاده‌سازی شوند. به این ترتیب، هر کاربرد یک حالت خاص است که باید در جای خود به آن پرداخته شود.

مثال ۴-۱

بعد از ارائه‌ی یک سخنرانی توسط مدیر عملیاتی OIE و چند جلسه دیگر، تیم پروژه‌ی SAMS در حدود سی نیازمندی کارکردی برای سیستم شناسایی می‌کند. سه تا از این نیازمندی‌ها را در ادامه می‌بینید:

R1) SAMS باید امکان جستجو با تعداد زیادی معیار جستجو برای برنامه‌های تبادل دانشجو فراهم کند.

R2) SAMS باید برای تسهیل جستجو در نتایج، آن‌ها را به شکل سلسله‌مراتبی، از یک خلاصه‌ی سطح بالا تا جزئیات هر برنامه‌ی تبادل دانشجو مرتب کند.

R3) SAMS باید به دانشجویان اجازه دهدکه درخواست‌های خود برای برنامه‌های تبادل دانشجو را به صورت برخط ارسال نمایند.

استخراج نیازمندی‌ها از مدل‌های تحلیل

همانطور که در بخش ۴-۵-۲ نشان داده شد، مدل‌ها برای درک کاربردهای فعلی و فرایندهای کسب‌وکار آن‌ها مفید هستند. همچنین، مدل‌ها برای استخراج نیازمندی‌ها نیز به کار می‌روند. مثال زیر این مطلب را روشن می‌سازد.

مثال ۴-۲

شکل ۴-۴ یک نمودار فعالیت برای یکی از فرایندهای دستی متعلق به دفتر تحصیلات بین‌المللی OIE را نشان می‌دهد. نیازمندی‌های کارکردی را از این نمودار فعالیت شناسایی کنید.

جواب: نمودار فعالیت شکل ۴-۴ حاوی فعالیت‌های زیر است:

۱. آماده‌سازی درخواست. این فعالیت توسط دانشجو انجام می‌شود. درحال حاضر، دانشجو یک فرم را به‌طور دستی پر می‌کند و آن را به OIE ارسال می‌نماید.

۲. بررسی درخواست‌ها. این فعالیت توسط مشاورین OIE انجام می‌شود. مشاورین OIE هر یک از درخواست‌ها را بررسی می‌کنند و تعیین می‌نمایند که آیا درخواست پذیرفته شود یا خیر.

۳. تماس با مؤسسه‌ی تبادل دانشجو. این فعالیت توسط مشاورین OIE انجام می‌شود که بنا به درخواست دانشجویانی که برای برنامه‌های تبادل دانشجو پذیرفته شده‌اند، با مؤسسه‌های مربوطه تماس می‌گیرند.

۴. اطلاع‌رسانی به دانشجو. این کار توسط یکی از کارمندان OIE انجام می‌شود که با ارسال نامه‌هایی به دانشجویان، آنها را از نتایج بررسی درخواست‌هایشان باخبر می‌کند.
۵. اطلاع‌رسانی به گروه دانشگاهی. این کار توسط یکی از کارمندان OIE انجام می‌شود که با ارسال نامه‌ای به هریک از گروه‌های دانشگاهی، آنها را از اسامی دانشجویان پذیرفته‌شده و پذیرفته‌نشده در برنامه‌ی تبادل دانشجو مطلع می‌سازد.
۶. بایگانی کردن. این کار توسط یکی از کارمندان OIE انجام می‌شود که درخواستهای بررسی شده را بایگانی می‌نماید.
۷. دریافت اطلاعیه توسط دانشجو. این‌کار توسط دانشجویان انجام می‌شود که نامه‌هایی که توسط کارمند OIE فرستاده‌شده است را دریافت می‌کنند.
۸. دریافت اطلاعیه توسط گروه‌های دانشگاهی. این‌کار توسط منشی‌های گروه‌های دانشگاهی مربوطه انجام می‌شود که نامه‌های ارسالی توسط کارمند OIE را دریافت و سپس بایگانی می‌کنند.

اگرچه این فعالیت‌ها مربوط به سیستم دستی موجود است، با این وجود به دلیل آن‌که فعالیت‌ها جزیی از عملکرد حرفه‌ای نرم‌افزار کاربردی OIE هستند، سیستم جدید باید راه‌های خودکاری برای پشتیبانی از آن‌ها فراهم نماید. به این ترتیب نیازمندی‌های زیر برای سیستم جدید پیشنهاد می‌شود:

R4. SAMS باید برای دانشجویان، امکان آماده‌سازی برخط درخواست‌ها برای برنامه‌های تبادل دانشجو را فراهم نماید.
R5. SAMS باید بررسی درخواست‌ها توسط مشاوران OIE را تسهیل نماید.
R6. SAMS باید وقتی یک درخواست توسط OIE پذیرفته شد، با مؤسسه‌ی تبادل دانشجو تماس بگیرد.
R7. SAMS باید وقتی یک درخواست پذیرفته یا رد شد، طرف‌های مربوطه را مطلع سازد.

R7.1. SAMS باید در صورت پذیرفته یا رد شدن یک درخواست، به دانشجو اطلاع دهد.
R7.2. SAMS باید در صورت پذیرفته شدن یک درخواست، به گروه دانشگاهی مربوطه اطلاع دهد.

R8. SAMS باید همه‌ی درخواست‌ها، اعم از پذیرفته‌شده یا ردشده را بایگانی کند.
R9. SAMS باید به دانشجویان اجازه دهد که نامه‌ای که آن‌ها را از تصمیم اخذشده در مورد درخواست‌شان مطلع می‌سازد را مشاهده و چاپ نمایند.
R10. SAMS باید به منشی‌های گروه‌های دانشگاهی اجازه دهد که نامه‌های اطلاع‌رسانی را به صورت برخط مشاهده و چاپ نمایند.

علاوه بر موارد ذکرشده، نیازمندی‌ها می‌توانند از نمودارهای مورد کاربرد و نمودارهای کلاس که یک کاربرد موجود را مدل می‌کنند، به‌دست بیایند. نمودارهای مورد کاربرد برای نمایش فرایندهای کسب‌وکار در یک کاربرد موجود به‌کار می‌روند. یعنی هر مورد کاربرد، یک فرایند کسب‌وکار موجود را مدل می‌نماید. به‌این‌ترتیب، نیازمندی‌های خودکارسازی فرایندهای کسب‌وکار می‌توانند از موردهای کاربرد به‌دست بیایند. به‌علاوه، نیازمندی‌ها میتوانند از آنچه کامپیوترها قادرند برای انسان انجام بدهند، به‌دست آیند. به‌عنوان مثال، یک دانشجو تنها می‌تواند برنامه‌های مبادله‌ی دانشجوی ثبت‌شده در سیستم دستی فعلی را مشاهده نماید. حال آن‌که یک سیستم کامپیوتری می‌تواند امکان جستجوی برخط با شرایط متنوع را برای او فراهم سازد. این نیازمندی نشان می‌دهد که اطلاعات برنامه‌های مختلف تبادل دانشجو باید به سیستم وارد شود. به‌این‌ترتیب، سیستم باید به کارمند OIE اجازه دهد که برنامه‌های تبادل دانشجو را به سیستم وارد نموده آنها را ویرایش، فعال، و غیرفعال نماید. همه‌ی اینها، نیازمندی‌های عملیاتی سیستم جدید هستند.

گاهی برای کمک به تیم در درک دامنه‌ی کاربرد، در مرحله‌ی استخراج نیازمندی‌ها مدل‌های دامنه ساخته می‌شوند. رابطه‌ی وابستگی که با یک فعل متعدی نام‌گذاری می‌شود، معمولاً نشان‌دهنده‌ی قابلیت‌هایی است که می‌تواند به‌عنوان نیازمندی شناخته شود. برای مثال، یک مدل دامنه برای کاربرد تحصیل در راه دور OIE ممکن است رابطه‌ی انجمنی بین کلاس Faculty و کلاس Student را به‌صورت زیر توصیف نماید: Faculty recommend Student . این رابطه نشان می‌دهد که سیستم برخط باید به اعضای هیئت علمی اجازه دهد که برای دانشجویان خود توصیه‌نامه ارسال نمایند. این یک نیازمندی کارکردی است.

در برخی از پروژه‌ها علاوه بر نیازمندی‌ها، محدودیت‌هایی نیز برای طراحی و پیاده‌سازی اعمال می‌گردد. این محدودیت‌ها در مرحله‌ی توصیف نیازمندی‌ها با عنوان قیود شناخته می‌شوند. برای نمونه، ممکن است OIE بخواهد که SAMS بر روی سرور مرکز محاسبات دانشگاه اجرا شود و از سیستم مدیریت پایگاه‌داده‌ی طراحی‌شده توسط این مرکز محاسبات استفاده کند. OIE می‌تواند زبان پیاده‌سازی را نیز بنابر توصیه‌ی کارشناسان فناوری اطلاعات خود انتخاب نماید. این‌ها نمونه‌هایی از قیود احتمالی سیستم می‌باشند.

شماره‌گذاری نیازمندی‌ها و قیود

برای برچسب زدن و نام‌گذاری نیازمندی‌ها و قیود از شماره‌هایی مانند R1، R1.1، R1.2.1، R1.2.2، R2، R2.1، R2.2، و C1، C2، C3 استفاده می‌شود. این کار باعث تسهیل ارتباط نیازمندی‌ها و قیود با سایر دست‌ساخته‌های نرم‌افزار، مانند موردهای کاربرد و موردهای آزمون می‌گردد. ارتباط نیازمندی‌ها با موردهای کاربرد به مدیر پروژه کمک می‌کند که توسعه و استقرار موردهای کاربرد را زمان‌بندی نماید. برای مثال، به‌این‌ترتیب اطمینان حاصل می‌شود که همه‌ی نیازمندی‌ها برآورده شده‌اند و نیازمندی‌های پراولویت‌تر، زودتر تحقق یافته‌اند. برای تسهیل شناسایی این روابط، یک ماتریس ردیابی‌پذیری[1] در صنعت مورد استفاده قرار می‌گیرد. ردیف‌های این ماتریس، نیازمندی‌ها و ستون‌های آن، موردهای کاربرد را نشان می‌دهند. برچسب‌گذاری نیازمندی‌ها و قیود سیستم با شماره‌های ذکر شده، ساختن و استفاده از ماتریس ردیابی را آسان می‌نماید. همچنین اعداد این سیستم، نشان‌دهنده‌ی یک سلسله مراتب پالایشی یا تجزیه‌ای بین نیازمندی‌ها و همین‌طور قیود هستند. در مثال ۴-۲ نیازمندی R7 به دو نیازمندی R7.1 و R7.2 پالایش شده است. این بدان معناست که R7 معادل R7.1 و R7.2 است. یعنی R7 برآورده می‌شود، اگر و تنها اگر R7.1 و R7.2 برآورده شوند. فایده‌ی استفاده از این رابطه‌ی برابری این است که برآورده ساختن نیازمندی‌ها در سطوح پایین‌تر، آسان‌تر می‌باشد.

اولویت‌بندی نیازمندی‌ها

پس از استخراج نیازمندی‌ها، یک مقدار صحیح بین ۱ تا ۵، یا ۱ تا ۱۰، به‌عنوان وزن اولویت به هریک از آنها اختصاص می‌یابد. به‌این‌ترتیب، موردهای کاربرد مربوط به نیازمندی‌های پر اولویت‌تر، زودتر توسعه داده شده و تحویل می‌گردند. به‌دلیل محدودیت زمان و منابع، اولویت‌بندی نیازمندی‌ها دارای اهمیت است. این اولویت‌ها به تیم کمک می‌کنند که بر نیازمندی‌های پر اولویت‌تر تمرکز نماید و به‌این‌ترتیب، اولویت‌های کسب‌وکار برآورده می‌گردند. اعضای تیم توسعه باید برای تعیین اولویت نیازمندی‌ها، با مشتری و کاربران کار کنند. همان‌گونه که پیش‌تر گفته شد، برای این کار می‌توان از پرسش‌نامه‌های تحقیقاتی و مصاحبه استفاده کرد.

---

[1] Traceability Matrix

۴-۵-۴- استانداردهای توصیف نیازمندی‌ها

برای پیش‌گیری از اختلافات بعدی، نیازمندی‌ها ومحدودیت‌هایی که در این گام شناسایی می‌شوند باید ثبت گردند. پس از گذشت سال‌ها، صنعت نرم‌افزار استانداردهایی برای بیانیه نیازمندی‌های نرم‌افزار وضع کرده است. شکل ۴-۵ استاندارد IEEE STD 830-1998 SRS را نشان می‌دهد.

۴-۵-۵- انجام مطالعه‌ی امکان‌پذیری

توصیف نیازمندی‌ها بخشی از قرارداد است و با خود پیامدهای قانونی به‌همراه می‌آورد. برای مثال، دو ماجرا در ابتدای این فصل آورده شد. یکی از آنها منجر به آن شد که توسعه‌دهنده و مشتری از یکدیگر به‌دلیل آنکه سیستم نیازمندی‌های موجود برآورده می‌سازد، شکایت کنند. سیستم موردنظر این روایت، می‌بایست در هر زمان و از هرجا در محدوده‌ی سرویس سیستم، به پزشکان امکان دسترسی بی‌سیم و برخط به اطلاعات بیماران را به پزشکان بدهد. این سیستم باید امنیت و محرمانگی اطلاعات بیماران را تضمین نماید. حتی اگر تکنولوژی بتواند این قابلیت‌ها را فراهم کند، محدودیت‌های بودجه و زمان‌بندی می‌تواند توانایی تیم توسعه برای برآورده‌ساختن این نیازمندی‌ها را زیر سؤال ببرد. بنابراین بسیار مهم است که در مرحله‌ی استخراج نیازمندی‌ها، امکان‌پذیری پروژه با درنظرگرفتن محدودیت‌هایی چون توانایی تیم، بودجه، زمان‌بندی، و سایر عوامل مرتبط ارزیابی گردد.

روش‌ها و تکنیک‌های زمان‌بندی و تخمین کار مورد نیاز، که در فصل ۲۳ شرح داده شده است، برای تعیین امکان‌پذیری با توجه به توانایی تیم، بودجه و زمان‌بندی توسعه‌ی سیستم مفید است. یکی از روش‌های موثر برای ارزیابی امکان‌پذیری ازدید تکنولوژی، پیش‌نمونه‌سازی است. برای مثال برای اطمینان از آنکه محدودیت‌های زمانی برآورده می‌شوند، یک نمونه از زیرسیستم مربوطه ساخته می‌شود و مورد استفاده قرار می‌گیرد. اگرچه مطالعه‌ی امکان‌پذیری سیستم، احتمال برآورده‌نشدن نیازمندی‌ها و قیود را کاهش می‌دهد، ولی ریسک‌ها همچنان ممکن است وجود داشته باشند. دراین حالت، برای شناسایی و تحلیل ریسک‌ها مدیریت ریسک انجام می‌شود و معیارهای حل و فصل ریسک تعریف می‌گردد. ریسک‌ها در طول فرایند توسعه تحت‌نظر قرار می‌گیرند و درصورتی که شرایط ریسک رخ بدهد، معیارهای مناسبی برای حل‌وفصل ریسک‌ها اعمال می‌شود. در فصل ۲۳، در مورد مدیریت ریسک صحبت خواهد شد.

۴-۵-۶- مرور شرح نیازمندی‌ها

شرح نیازمندی‌های نرم‌افزار باید توسط همکاران، متخصصین دامنه، مشتری، و کاربران بازنگری گردد. به این بازنگری‌ها، مرورهای تکنیکی، بازبینی متخصصین، و مرور مشتری گفته می‌شود که در ادامه شرح داده می‌شوند:

مرور تکنیکی

مرور تکنیکی یک بازنگری داخلی است که توسط تیم توسعه انجام می‌شود. روش‌های مرور تکنیکی، شامل مرور توسط همکاران[۱]، بازرسی[۲] و گذار[۳] می‌باشد. مرور تکنیکی با هدف کشف مشکلات زیر انجام می‌گیرد:

- ناقص بودن[۴] مانند : (۱) نقص در تعریف (مثلاً برخی از مفاهیم خاص کاربرد تعریف نشده باشد)، (۲) نقص داخلی (مثلاً برخی از نیازمندی‌ها یک بخش «اگر» داشته باشند ولی بخش «در غیر این صورت» نداشته باشند، و (۳) نقص خارجی (مثلاً حالت‌هایی در کاربرد واقعی وجود داشته باشد که در شرح نیازمندی‌ها ذکر نشده باشند).

---

[۱] Peer Review
[۲] Inspection
[۳] Walkthrough
[۴] Incompleteness

بخش	شرح
۱- مقدمه	یک دید کلی از توصیف نیازمندی‌های نرم‌افزار (SRS) ارائه دهید.
۱-۱- هدف	منظور از SRS ومخاطبان موردانتظار آن‌را شرح دهید.
۱-۲- قلمرو	الف – نام محصول را ذکر کنید. ب – آن‌چه محصول انجام می‌دهد و آن‌چه انجام نمی‌دهد را شرح دهید. ج – کاربردهای محصول شامل اهداف، مقاصد، و مزایای آن‌را توضیح دهید.
۱-۳- تعاریف، سرنام‌ها، و کوته‌نوشت‌ها	برای شرح بهتر SRS، همه‌ی اجزا، کوته‌نوشت‌ها، و سرنام‌ها را تعریف کنید. ممکن است در این بخش، ارجاعاتی به ضمیمه‌ها و یا مدارک دیگر انجام شود.
۱-۴- مراجع	مراجعی که در SRS به آنها ارجاع شده است را با نام، شماره‌ی گزارش، تاریخ، ناشر، و محل یا نحوه‌ی دسترسی به مرجع فهرست کنید.
۱-۵- طرح کلی	رئوس کل مطالب SRS و نحوه‌ی سازمان‌دهی آن را طرح‌ریزی کنید.
۲- شرح کلی	یک دید کلی از محصول ارائه دهید.
۲-۱- چشم انداز محصول ۲-۱-۱- واسط‌های سیستم ۲-۱-۲- واسط‌های کاربر ۲-۱-۳- واسط‌های سخت‌افزاری ۲-۱-۴- واسط‌های نرم‌افزاری ۲-۱-۵- واسط‌های ارتباطی ۲-۱-۶- واسط‌های حافظه ۲-۱-۷- واسط‌های عملیات ۲-۱-۸- نیازمندی‌های سازگاری با محیط نصب	زمینه‌ی محصول و روابط و واسط‌های آن با سایر اجزاء کل سیستم را شرح دهید. برای نشان دادن زمینه و روابط می‌توان از نمودارهای بلوکی استفاده کرد. همچنین، ویژگی‌ها و محدودیت‌های حافظه‌ی اصلی و ثانویه، مودهای عملیاتی، پشتیبان‌گیری و بازیافت، و نیازمندی‌های خاص محل نصب را شرح دهید.
۲-۲- کارکرد محصول	خلاصه‌ای از کارهایی که محصول انجام می‌دهد را تهیه کنید.
۲-۳- مشخصات کاربر	مشخصات عمومی کاربران مورد انتظار سیستم، شامل سطح تحصیلات، تجربه، تخصص، و مهارت‌های تکنیکی را شرح دهید.
۲-۴- قیود	محدودیت‌های موجود بر فضای راه‌حل و یا گزینه‌های توسعه دهنده‌ی نرم‌افزار را بیان کنید.
۲-۵- مفروضات و وابستگی‌ها	عوامل مؤثر بر نیازمندی‌ها را فهرست نمائید.
۳- نیازمندی‌های خاص	نیازمندی‌های نرم‌افزار را شرح دهید.
۳-۱- نیازمندی‌های واسط خارجی	برای هر یک از واسط‌های سیستم، واسط‌های کاربر، واسط‌های سخت‌افزاری، واسط‌های نرم‌افزاری، و واسط‌های ارتباطاتی یک شرح تفصیلی بنویسید. شرح هر واسط برای هر ورودی و خروجی شامل نام، فرمت، محدوده‌ی اعتبار، زمان‌بندی، و سایر اطلاعات مرتبط است.
۳-۲- نیازمندی‌های کارکردی	برای کارکرد هریک از نیازمندی‌های کارکردی، شرحی بنویسید که با «سیستم باید.... انجام دهد» شروع شده باشد. این شرح می‌تواند شامل بررسی اعتبار ورودی، توالی عملیات، پاسخ به وضعیت‌های غیرطبیعی، و ارتباطات ورودی و خروجی باشد.
۳-۳- نیازمندی‌های کارایی	همه‌ی توانمندی‌های محصول که با کارایی آن در ارتباطند را برشمارید.
۳-۴- قیود طراحی	همه‌ی محدودیت‌های زمان طراحی، مانند قیودی که استانداردها و محدودیت‌های سخت‌افزاری اعمال می‌کنند، را شرح دهید.
۳-۵- صفت‌های سیستم نرم‌افزاری	همه‌ی نیازمندی‌های مربوط به کیفیت مانند قابلیت اطمینان، امنیت، و در دسترس بودن را شرح دهید.
۳-۶- سایر نیازمندی‌ها	
ضمیمه‌ها	
فهرست	

شکل ۴-۵: استاندارد IEEE Std 830-1998 SRS

- ناسازگاری[۱] مانند : (۱) ناسازگاری در نوع (مثلاً شرح ناسازگار یک نوع داده در شرح نیازمندی‌ها)، و (۲) ناسازگاری منطقی (مثلاً نیازمندی‌ها نشان بدهند که تحت شرایط یکسان ممکن است دو نتیجه‌ی مختلف به‌دست بیاید).
- ابهام[۲] مانند : (۱) ابهام در تعریف مفاهیم خاص کاربرد، و (۲) ابهام در نگارش نیازمندی‌ها.
- افزونگی[۳] مانند : (۱) تعریف دوباره‌ی یک مفهوم، و (۲) شرح دوباره‌ی یک نیازمندی یا قید.
- سرکش‌بودن[۴] (مثلاً نیازمندی‌های سطوح پایین‌تر، به نیازمندی‌های سطوح بالاتر ارتباطی نداشته باشند).
- امکان‌پذیر نبودن[۵] (مثلاً برخی از نیازمندی‌ها یا قیود قابل برآورده ساختن نباشند).
- جزییات غیرضروری پیاده‌سازی

مرور توسط متخصص:

این بازبینی توسط متخصصین دامنه و با هدف پاسخ‌گویی به پرسش‌های زیر انجام می‌شود:

۱. آیا حقوق، قوانین، رفتارها، سیاست‌ها، استانداردها، و آیین‌نامه‌های خاص دامنه به‌طور صحیح و دقیق در SRS بیان شده‌اند؟
۲. آیا اصطلاحات خاص کسب‌وکار موردنظر، به‌طور نادرست، نامناسب، نادقیق، و یا ناسازگار مورد استفاده قرار گرفته است؟
۳. آیا درک درست و دقیقی از دامنه‌ی کاربرد وجود دارد؟
۴. آیا مسائل یا جنبه‌های احتمالی دیگری در خصوص دامنه وجود دارد؟

مرور توسط مشتری:

این بازبینی توسط مشتری و کاربران انجام می‌شود و پاسخ پرسش‌های زیر را می‌یابد:

۱. آیا شرح نیازمندی‌ها، به‌درستی نیازمندی‌های عملیاتی کاربردی که قرار است برایش سیستم ساخته شود یا گسترش یابد، را شرح می‌دهد؟
۲. آیا نیازمندی‌های مربوط به واسط کاربری به‌درستی توصیف شده‌اند؟
۳. آیا نیازمندی‌های غیرکارکردی شامل کارایی، زمان پاسخ، و نیازمندی‌های امنیتی به‌درستی توصیف شده‌اند؟
۴. آیا محدودیت‌های خاص این کاربرد، که مرتبط با محیط عملیات، و سیاست‌ها و آیین‌نامه‌های صنعتی و دولتی می‌باشند، به‌درستی توصیف شده‌اند؟

۴-۶ اعمال قوانین چابکی

توسعه‌ی صنعتی برنامه‌محور، زمان و کار قابل‌توجهی را برای استخراج نیازمندی‌ها اختصاص می‌دهد. برای مثال، فاز تحلیل نیازمندی‌ها برای یک فرایند ابشاری، در حدود ۱۵% و یا بیشتر از کل زمان یا کار لازم برای توسعه را صرف می‌نماید. این بدان علت است که وقتی مرحله‌ی تشخیص نیازمندی‌ها به‌پایان می‌رسد، تغییرات در نیازمندی‌ها سخت و پرهزینه خواهد بود.

توسعه‌ی چابک این واقعیت را می‌پذیرد که تغییر، راه زندگی است و به لطف فواصل کوتاه تکرار و افزایش‌های کوچک نرم‌افزاری، با سرعت و توجه به تغییرات پاسخ می‌دهد. به این ترتیب، استخراج نیازمندی‌ها برای توسعه‌ی چابک باید مقدار

---

[۱] Inconsistency
[۲] Ambiguity
[۳] Redundancy
[۴] Intractability
[۵] Infeasibility

کوچکی زمان و کار صرف کند؛ چیزی در حدود دو هفته برای یک پروژه‌ی شش‌ماهه یا یک‌ساله. هدف، ایجاد فهرستی از نیازمندی‌های پراولویت از نرم‌افزار است که انتظار می‌رود پس از چند دور تکرار برآورده شود. اصول زیر باید در زمان استخراج نیازمندی‌های یک پروژه‌ی چابک اعمال گردند:

۱. همکاری مشتری و درگیر کردن فعال مشتری، ضروری است. همان‌طور که در بخش چالش‌های استخراج نیازمندی‌ها گفته‌شد، برخی موانع ارتباطی میان تیم توسعه و مشتری و کاربران وجود دارد. این موانع به‌دلیل عدم درک کافی این افراد از حرفه‌ی یکدیگر به‌وجود می‌آیند. برای رفع این موانع، تیم توسعه باید در مشارکت‌دادن مشتری و کاربران در فرایند توسعه پیش‌قدم گردد. این‌کار به‌دلیل آن‌که زمان و کار قابل توجهی را مصرف می‌کند، قبل از هر چیز همکاری مشتری را می‌طلبد. معنای همکاری مشتری این است که او اهمیت مشارکت کاربران را درک کند و اجازه بدهد که زمان و کار ارزشمند کارمندانش به پروژه اختصاص یابد. به‌طور مشخص‌تر، لازم است که مشتری کارمندانی که به‌خوبی کاربرد مورد نظر را می‌شناسند را شناسایی کند و آن‌ها را برای کار با تیم توسعه تخصیص دهد. فرکانس تعامل با تیم توسعه، بسته به نوع شیوه‌ی چابک و بسته به نوع پروژه متفاوت است. حداقل دو یا سه جلسه در هفته با تیم توسعه مورد نیاز است.

۲. نیازمندی‌ها را در بالاترین سطح خود ببینید و جزییات را به دورهای تکرار بعدی موکول کنید. روش‌های چابک نیازمندی‌ها را به‌طور تفصیلی شرح نمی‌دهند. این بدان علت است که در بسیاری از پروژه‌های واقعی، توصیف دقیق، کافی، و تفصیلی نیازمندی‌ها امکان‌پذیر نیست. هرچه موارد کاربرد بیشتری در محیط واقعی نصب و عملیاتی می‌شود، مشتری ایده‌های حرفه‌ای بیشتری ارائه می‌کند و بنابراین تغییرات بیشتری برروی نیازمندی‌ها لازم می‌شود. بنابراین، شرح و توصیف تفصیلی نیازمندی‌ها، حتی اگر امکان‌پذیر باشد، چیزی جز اتلاف کار و زمان نیست.

۳. به اندازه‌ی کافی خوب. به‌سرعت فهرستی از نیازمندی‌های نرم‌افزار تهیه کنید و به گام بعدی بروید. نیازی نیست که همه‌ی نیازمندی‌های ممکن را در فهرست خود لحاظ کنید. حتی لازم نیست که نیازمندی‌های انتخاب‌شده را به‌طور ۱۰۰% صحیح و دقیق شرح دهید. بنا به ویژگی‌های مسائل بدرفتار که در فصل ۲ شرح داده شد، نیازمندی‌ها نمی‌توانند به‌طور کاملاً صریح، دقیق، و جامع شناسایی و توصیف گردند. بنابراین روش‌های چابک بر شناسایی، اعتبارسنجی و شرح نیازمندی‌هایی تمرکز می‌کنند که در کسب‌وکار مورد نظر، ماموریتی اصلی، اولویتی زیاد، و ارزشی بالا داشته باشند. این قانون معمولاً با قانون ۸۰/۲۰ مقایسه می‌شود، که بر شناسایی پرکاربردترین قابلیت‌های سیستم با انجام فقط کسری از کل کار تاکید می‌کند. در عمل این قوانین بسیار مفید هستند زیرا در آنِ واحد نمی‌تواند همه‌چیز را باهم داشت، شما چیزی را از دست می‌دهید تا چیز دیگری را به‌دست آورید. این بدان معنی است که تمرکز بر نیازمندی‌هایی که برای مشتری باارزش‌ترند، بهتر از تقسیم وقت و انرژی بین کارهای کوچک و کم‌ارزش است.

۴. فقط کافی است در حد نیاز مدل‌سازی انجام دهید. اگر مدل‌ها برای کمک به استخراج نیازمندی‌ها ساخته می‌شود، پس به‌سرعت کم‌ترین مقدار مدلی که برای رساندن مقصود لازم دارید بسازید و نه چیزی بیشتر از آن. در روش‌های چابک به نرم‌افزاری که کار می‌کند، بیشتر از یک مجموعه‌ی کامل از مستندات بها داده می‌شود. با این حال، مدل‌سازی اگر درست مورد استفاده قرار گیرد، درک و ارتباط بین اعضای تیم را بهبود می‌بخشد. مدل‌سازی در حد نیاز به‌این معناست که اگر مدل‌سازی لازم تشخیص داده شد، شما باید سعی کنید که دامنه‌ی آن را در حدی محدود کنید که فقط نیاز شما، و نه بیشتر از آن را برآورده سازد.

۴-۷ مدیریت نیازمندی‌ها و ابزارهای آن

برای بسیاری از پروژه‌های نرم‌افزاری، تغییر نیازمندی‌ها امری رایج است. این تغییرات شامل اضافه کردن نیازمندی‌های جدید، تغییر، و حذف نیازمندی‌های موجود است. تغییر یک نیازمندی ممکن است بر بسیاری از محصولات میانی و سایر اقلام نیازمندی‌ها تاثیر داشته‌باشد. اثر تغییر، هزینه‌ی پیاده‌سازی آن، و هزینه‌ی اعمال آن باید مورد بررسی قرار گیرد تا معلوم شود که آیا فایده‌ی این تغییر، هزینه‌هایش را جبران می‌کند یا خیر. تغییر نیازمندی‌ها می‌تواند در فاز نگهداری، و برای اصلاح خطاها و بهبود کارکرد سیستم، لازم گردد. مدیریت نیازمندی‌ها، شیوه‌ای برای اداره کردن همه‌ی موضوعات مرتبط با تغییر نیازمندی‌هاست. شکل ۴-۶ فعالیت‌های اصلی مدیریت نیازمندی‌ها را شرح می‌دهد.

مدیریت نیازمندی‌ها باید حجم عظیمی از داده‌ها و وابستگی‌های پیچیده را پردازش کند. ابزارهای بسیاری برای پشتیبانی از فعالیت‌های مدیریت نیازمندی‌ها وجود دارد. IBM Rational RequisitePro، و IBM Rational Doors دو نمونه از این ابزارها هستند. RequisitePro به اعضای تیم اجازه می‌دهد که نیازمندی‌ها را تعریف کنند و ویژگی‌های اقلام نیارمندی‌ها را با استفاده از یک واسط مبتنی بر وب توصیف نمایند. این ابزار از قابلیت ردیابی‌پذیری نیازمندی‌ها، شامل ردگیری بین نیازمندی‌های کلاس بالا و کلاس پایین پشتیبانی می‌نماید. وقتی یک نیازمندی تغییر می‌کند، این ابزار امکان تحلیل تاثیر این تغییر، و اطلاع‌رسانی خودکار از طریق ایمیل را فراهم می‌کند. به‌علاوه، سازگاری با آیین‌نامه‌های دولتی و صنعتی و استانداردها نیز بررسی می‌گردد. این ابزار می‌تواند با مجموعه ابزارهای دیگر Rational نیز یکپارچه شود.

IBM Rational Doors یک ابزار مدیریت نیازمندی‌ها می‌باشد. این ابزار، تعریف نیازمندی‌ها و پیونددادن هر نیازمندی با نیازمندی‌های دیگر، نمودارها، جدول‌ها، و مستندات مرتبط را پشتیبانی می‌نماید. ذی‌نفعان مستقر در مکان‌های مختلف می‌توانند با استفاده از امکان دسترسی به وب این ابزار، نیازمندی‌ها و اطلاعات ردگیری آن‌ها را توصیف و ویرایش نمایند. بررسی براورده‌شدن نیازمندی‌ها در این ابزار به دو راه مختلف انجام می‌پذیرد. توابع ردیابی‌پذیری ابزار می‌توانند نشان دهند که چگونه نیازمندی‌ها، طراحی، و آزمون با استانداردها و آیین‌نامه‌ها مطابقت دارند. همچنین این ابزار می‌تواند این‌که چه‌کسی چه‌چیزی را درچه زمانی تغییر داده‌است، ثبت نماید و بنابراین، همواره دنباله‌ای از تغییرات برای بررسی موجود است. این ابزار می‌تواند با سایر ابزارهای Rational و شرکت‌های دیگر یکپارچه شود تا از عملیات مدیریت تغییر، مدیریت کیفیت، و کارهای مشابه پشتیبانی نماید.

خلاصه

این فصل مفاهیم پایه، گام‌ها، و روش‌های استخراج نیازمندی‌ها را، به‌همراه اهمیت و اهداف آن نشان می‌دهد. گام‌های استخراج نیازمندی‌ها عبارتند از: (۱) جمع‌آوری اطلاعات در مورد کاربرد و دامنه‌ی آن، (۲) ساخت مدل‌های تحلیل، (۳) استخراج نیازمندی‌ها و قیود (۴) انجام مطالعه‌ی امکان‌پذیری، و (۵) مرور نیازمندی‌ها و قیود. سه نوع از روش‌های مرور نیازمندی‌ها شرح داده شده است: مرور تکنیکی، مرور توسط فرد متخصص، و مرور توسط مشتری. هریک از این مرورها بر جنبه‌های مختلفی از توصیف نیازمندی‌ها تاکید می‌نماید. در پایان فصل، قوانین چابکی برای استخراج نیازمندی‌ها شرح داده شده است. SRS، قابلیت‌ها یا آن‌چه سیستم باید انجام دهد را نشان می‌دهد. چگونگی فراهم‌کردن این قابلیت‌ها در SRS مورد توجه قرار نمی‌گیرد و به دورهای تکراری آتی موکول می‌شود. این مسئله می‌تواند سبب خطا در تفسیر نیازمندی‌ها گردد. یعنی تیم توسعه و مشتری و کاربران ممکن است نیازمندی‌ها را به شکل‌های متفاوت تفسیر نمایند. در این حالت، موردهای کاربرد مفید واقع می‌شوند زیرا موردهای کاربرد، بهبودیافته‌ی نیازمندی‌ها هستند. نیازمندی‌ها نشان می‌دهند که سیستم باید چه چیزی را تحویل بدهد، و موارد کاربرد چگونگی تحویل این قابلیت‌ها را بیان می‌کند. به‌عبارت دیگر موردهای کاربرد، نیازمندی‌ها را

تعریف و اولویت‌بندی	• شناسایی نیازمندی‌های کسب‌وکار و اولویت‌های آن‌ها • تنظیم‌کردن نیازمندی‌های کارکردی، نیازمندی‌های غیرکارکردی، و قیود
مدیریت کیفیت	• اطمینان از این‌که نیازمندی‌ها به‌وضوح توصیف شده‌اند، صحیح و سازگارند، و نیازمندی‌های بسیار حیاتی لحاظ شده‌اند. • اطمینان از این‌که نیازمندی‌ها به موارد آزمون پیوند داده شده‌اند. • جمع‌آوری داده و محاسبه‌ی معیارهای کیفیت
مدیریت ریسک	• شناسایی و اولویت‌بندی موارد ریسک در نیازمندی‌ها • شرح معیارهای حل‌وفصل ریسک برای موارد پراولویت ریسک • رصد ریسک‌ها و اجرای اندازه‌گیری‌های حل‌وفصل ریسک • به‌روزرسانی موارد ریسک و معیارهای حل‌وفصل آن‌ها
مدیریت ردیابی‌پذیری	• دنبال‌کردن روابط بین نیازمندی‌ها و نیازهای کسب‌وکار، مستندات مشتری، و همه‌ی محصولات میانی مربوط به توسعه مانند مستندات تحلیل و طراحی، کد، و موارد آزمون
مدیریت تغییر	• تشخیص و تحلیل نیاز به تغییر نیازمندی‌ها • تحلیل‌کردن اثر تغییر و میزان مفیدبودن آن برای هزینه‌ها • اطمینان از این‌که تغییرات تصویب‌شده، به‌درستی پیاده‌سازی شده‌اند.



شکل ۴-۶: فعالیت‌های مدیریت نیازمندی‌ها

پیاده‌سازی می‌کنند. این اثر، امکان تفسیر نادرست را کاهش می‌دهد.

برای مطالعه‌ی بیش‌تر

دیوید های [۷۸] استخراج نیازمندی‌ها را از نقطه‌نظر کسب‌وکار نشان می‌دهد. یک راهنمای عملی برای استخراج نیازمندی‌ها در [۱۶۱] دیده می‌شود. در [۳۰]، یک روش شیءگرا برای تحلیل نیازمندی‌ها با کمک UML شرح داده شده است. کتاب نوشته‌شده توسط آلن دیویس [۵۶] منبع بسیار خوبی برای روش‌های تحلیل و توصیف نیازمندی‌ها می‌باشد. در این کتاب روش‌های مختلف استخراج نیازمندی‌ها معرفی و مقایسه شده است. همچنین حجم وسیعی از مقالات منتشر شده در حوزه‌ی تحلیل نیازمندی‌ها در آن فهرست شده‌اند. کتاب تالیف‌شده توسط برینز و لوچی [۲۵] مرجع ارزشمندی برای روش‌های تحلیل

و مرور نیازمندی‌ها می‌باشد.

سوال‌های مروری فصل

۱. نیازمندی و محدودیت را تعریف کنید. چه تفاوت‌هایی بین این‌دو وجود دارد؟
۲. اهداف استخراج نیازمندی‌ها چیست؟
۳. چرا استخراج نیازمندی‌ها مهم است؟
۴. چالش‌های استخراج نیازمندی‌ها کدامند؟
۵. گام‌ها یا فعالیت‌های استخراج نیازمندی‌ها کدامند؟
۶. در مورد نیازمندی‌ها چه اشتباهاتی می‌تواند پیش بیاید؟
۷. انواع سه‌گانه‌ی مرور نیازمندی‌ها کدامند؟
۸. «مدیریت نیازمندی‌ها» و «ردیابی‌پذیری نیازمندی‌ها» را تعریف کنید.
۹. کدام‌یک از قوانین چابک می‌توانند در استخراج نیازمندی‌ها به‌کار گرفته شوند؟ مفهوم هریک از این قوانین را بنویسید.

تمرین‌ها

۴-۱ برای یک سیستم اطلاعاتی کتابخانه، یک شرح نیازمندی‌های نرم‌افزار بنویسید که مشابه سیستم مورد استفاده در دانشگاه شما باشد. این سیستم، باید حداقل دارای عملیاتی برای جستجو، دریافت، و پس‌دادن کتاب باشد.

۴-۲ برای هریک از سیستم‌های زیر، نیازمندی‌های کارکردی و غیرکارکردی را به‌اختصار بنویسید. برای هرسیستم، شرح نیازمندی‌ها را به حداکثر سه صفحه محدود نمایید.

الف) یک ماشین‌حساب مجازی رومیزی که به کاربر اجازه می‌دهد از ماوس و صفحه‌کلید برای واردکردن اطلاعات استفاده کند.

ب) یک ماشین پاسخگویی تلفن

ج) یک سیستم ایمیل مبتنی بر وب

۴-۳ نیازمندی‌های کارکردی و غیرکارکردی یک سیستم مدیریت درس مبتنی بر وب را شناسایی و توصیف کنید. این سیستم باید به دانشجویان اجازه دهد که درس‌ها را به‌ترتیب جستجو، ثبت‌نام و یا تمام کنند. همچنین به کارمندان اداره‌ی آموزش دانشگاه امکان زمان‌بندی درس‌ها را می‌دهد.

۴-۴ برای سیستم مدیریت تقویم زیر، نیازمندی‌های کارکردی و غیرکارکردی را مشخص نمایید. طول SRS را به دو صفحه محدود کنید:

این نرم‌افزار مدیریت تقویم به کاربر اجازه می‌دهد که فعالیت‌های مشخص خود از قبیل جلسات و کارهایی که باید انجام شود را زمان‌بندی کند. یک فعالیت می‌تواند در یک بازه‌ی زمانی مشخص در آینده رخ دهد. یک فعالیت می‌تواند برای چندین روز متوالی انجام شود. هر فعالیت، دارای یک شرح مختصر خطی است. یک فعالیت می‌تواند از نوع بازگشتی باشد که به‌طور تکراری هر ساعت، هر هفته، یا هر ماه اتفاق می‌افتد. یک کاربر می‌تواند یک فعالیت را با استفاده از یک تقویم ماه‌به‌ماه طوری برنامه‌ریزی کند که ابتدا تاریخ یا تاریخ‌ها را انتخاب کند و سپس زمان دقیق شروع و خاتمه را در یک تاریخ مشخص نماید. سیستم تقویم می‌تواند از طریق ایمیل، پیام متنی، و یا تماس تلفنی در روز قبل از فعالیت و خود آن روز، به کاربر یادآوری کند. کاربر می‌تواند فعالیت‌های قبلی خود را بازبینی کند و فعالیت‌های برنامه‌ریزی شده را به روز کرده و یا حذف نماید.

۴-۵ برای سیستم رزرو اتاق که در ادامه شرح داده شده است، یک SRS تهیه کنید. طول SRS را به حداکثر دو صفحه محدود نمایید. این سیستم تحت وب رزرو تک‌اتاقه، یک سیستم مدیریت منابع است که به کاربران ثبت‌نام شده‌ی یک سازمان اجازه می‌دهد که برای یک دوره‌ی زمانی از یک تاریخ داده شده، اتاق رزرو کنند. به‌علاوه سیستم به کاربر اجازه می‌دهد که انتخاب کند آیا پیامی برای یادآوری به‌طور خودکار برای گروهی از کاربران فرستاده شود یا خیر، و در صورت تمایل، آن پیام در زمان تعیین‌شده ارسال می‌گردد. چون تنها یک اتاق موجود است. نیازی به امکانات جست‌وجو نیست. کاربر می‌تواند با استفاده از یک تقویم ماه‌به‌ماه، به روز مورد نظر خود برسد. وقتی این تقویم به کاربر نشان داده می‌شود، روزهایی که اتاق در آن‌ها از پیش رزرو شده است با رنگ قرمز، و سایر روزها با رنگ دیگر و یا بدون رنگ مشخص می‌شوند. این سیستم برای تعیین چند محدودیت یا ویژگی پیکربندی می‌شود:

الف) برای فعال یا غیرفعال کردن یک صفحه‌ی نمایش که نشان می‌دهد چه کاربرانی در چه بازه‌ی زمانی اتاق را رزرو کرده‌اند .

ب) برای فعال یا غیرفعال کردن قابلیت رزرو اتاق به‌طور بازگشتی در یک بازه‌ی زمانی مشخص توسط یک کاربر. به‌عنوان مثال، ساعت ۱۰ صبح تا ۱۲ بعدازظهر هر چهارشنبه در کل طول ترم.

ج) برای تعیین یک محدودیت برای بیشترین بازه‌ی زمانی که می‌توان در آن به‌طور بازگشتی رزرو انجام داد.

د) رای تعیین یک محدودیت برای بیشترین تعداد رزروی که یک کاربر می‌تواند انجام دهد.

ه) برای فعال یا غیرفعال کردن امکان اطلاع‌رسانی خودکار سیستم.

۴-۶ در پیسوت ت، حرفه‌ی معرفی سرویس‌های مشاغل ملی[1] معرفی شده است. فرض کنید که در حال حاضر این شغل به‌طور دستی انجام می‌شود. شرکت می‌خواهد یک سیستم برخط تحت‌وب برای اتوماسیون این کسب‌وکار ایجاد نماید. این تمرین می‌تواند حرفه‌ای یا بخشی از یک پروژه‌ی سیستمی باشد. تیم دانشجویی باید کارهای زیر را انجام دهد:

الف) با استفاده از شرح این کسب‌وکار، مهم‌ترین فعالیت‌های حرفه‌ای آن را شناسایی و فهرست کند. این فعالیت‌ها، قابلیت‌هایی هستند که سیستم آینده برای کاربر فراهم می‌نماید.
راهنمایی: تعداد این فعالیت‌ها از یک دانشجو به دانشجویی دیگر متفاوت است، ولی در مجموع

[1] National Trade Show Services (NTSS)

قابلیت‌های شناسایی‌شده باید حداقل ۹۰% فعالیت‌های خطی،که شامل مهم‌ترین آن‌هاست، را شامل شود.

ب) سایر قابلیت‌های مورد نیاز برای سیستم برخط را شناسایی کند، به‌طوری‌که این کسب‌وکار بتواند به‌صورت برخط انجام گردد. برای مثال، سیستم باید بتواند کاربران را احراز هویت نماید.

ج) قابلیت‌های شناسایی‌شده را با عنوان نیازمندی‌های نرم‌افزار بازنویسی نماید.

۷-۴ به اختصار شرح دهید که چگونه برای SRS ایجاد شده در تمرین ۴-۶، سه نوع مرور لازم را انجام می‌دهید. پاسخ خود را به حداکثر یک صفحه محدود نمایید.

۸-۴ برای سیستم اجاره‌ی برخط اتومبیل که در پیوست ت شرح داده شده است، نیازمندی‌ها و محدودیت‌ها را شناسایی کنید. با توجه به طبیعت حرفه‌ی اجاره‌ی اتومبیل، به هر نیازمندی وزن اولویت مناسب را اختصاص دهید.

۹-۴ مقاله‌ای بنویسید که توضیح دهد کار استخراج نیازمندی‌ها می‌تواند به ترتیب با استفاده از فرایند آبشاری، و فرایند چابک انجام گردد. تفاوت بین دو روش را بنویسید و به‌طورمختصر شرح دهید.

۵

مدل‌سازی دامنه

نکات کلیدی

- مدل‌سازی دامنه، یک فرایند مفهوم‌سازی برای کمک به تیم توسعه جهت فهم دامنه‌ی کاربرد است.
- پنج گام ساده: جمع‌آوری اطلاعات در مورد دامنه‌ی کاربرد؛ طوفان فکری؛ دسته‌بندی[۱] نتایج طوفان فکری؛ به تصویر کشیدن مدل دامنه با استفاده از یک نمودار کلاس UML؛ و انجام بازرسی و مرور.

مری ساندرس در حال کار بر روی یک پروژه برای یک شرکت بیمه بود، که به‌ناگاه به کار بر روی یک پروژه‌ی دیگر برای یک شرکت خدمات درمانی گمارده شد. مدیرش می‌دانست که او می‌تواند این دامنه‌ی کاربردی جدید را به سرعت فراگیرد، و به‌راستی او توانست پروژه را در زمان مقرر و بدون مشکل خاصی تحویل دهد. راز مری چیست؟ چگونه او توانست چنان کارآمد و موثر از یک دامنه‌ی کاربرد به دامنه‌ی دیگر حرکت کند؟

یک مهندس نرم‌افزار مانند مری که به‌عنوان مشاور فناوری اطلاعات مشغول است، معمولاً روی پروژه‌هایی در دامنه‌های مختلف کاربردی کار می‌کند. حتی اگر یک مهندس نرم‌افزار مشاور هم نباشد، ممکن است لازم باشد بر روی پروژه‌های جدید یا نسخه‌های جدیدی از نرم‌افزارهای موجود کار کند. به‌دلیل آن‌که مهندسان گاه‌به‌گاه باید روی کاربردهای جدید کار کنند، مهندسی نرم‌افزار چالش‌آفرین و پر از هیجان است. همین مزیت است که یک مهندس نرم‌افزار متوسط را از یک مجری ارشد متمایز می‌نماید. یکی از ابزارهایی که مهندسان نرم‌افزار را قادر به این‌کار می‌کند، مدل‌سازی است. در این فصل مطالب زیر را خواهید آموخت:

- مدل‌سازی دامنه
- اهمیت مدل‌سازی دامنه
- مفاهیم شیءگرایی
- گام‌های مدل‌سازی دامنه
- نمودار کلاس UML

---

[۱] Classification

۵-۱ مدل‌سازی دامنه چیست؟

سال‌ها پیش، من درسی را تدریس می‌کردم که در آن تیم‌های دانشجویی می‌بایست یک شبیه‌ساز سیستم ارتباطات موبایل را طراحی و پیاده‌سازی می‌کردند. ما در کلاس مدل‌سازی دامنه را یاد داده‌بودیم و دانشجویان موظف بودند یک مدل‌دامنه برای این پروژه‌ی شبیه‌سازی بی‌سیم بسازند. یکی از دانشجویان فعال کلاس، تقریباً هر هفته به دفتر من سرمی‌زد و یادداشت‌های زیادی که حاصل مطالعاتش در زمینه‌ی ارتباطات بی‌سیم بود را نیز به‌همراه خود می‌آورد. من به او اکیداً توصیه کردم که به‌جای یادداشت‌برداری مدل دامنه را بسازد. متاسفانه این دانشجو نتوانست پروژه‌ی خود را در موعد زمانی مقرر به‌پایان برساند. پس از ارسال مدل دامنه، این دانشجو نزد من آمد و گفت: «من متاسفم که مدل‌سازی دامنه را زودتر شروع نکردم. این‌کار می‌توانست به من در فهم سریع و سازمان‌یافته‌ی ارتباطات بی‌سیم کمک کند.»

حال، مدل‌سازی دامنه چیست؟

تعریف ۵-۱

مدل‌سازی دامنه، یک فرایند مفهوم‌سازی است. این فرایند به شناسایی مفاهیم مهم دامنه، ویژگی‌های آن‌ها، و ارتباط بین این مفاهیم کمک می‌کند. حاصل این فرایند در یک نمودار به نام مدل‌دامنه به‌تصویر کشیده می‌شود.

فرایند مفهوم‌سازی شامل مشاهده، دسته‌بندی، مجردسازی، و تعمیم[1] می‌باشد. به دلیل آن‌که نرم‌افزار از نگاه عام، یک محصول مفهومی است، این فرایند دارای اهمیت است. همه‌ی توسعه‌ی نرم‌افزار در مورد «مفهوم‌سازی» است. برای نمونه، توسعه‌ی یک سیستم بانکی را درنظر بگیرید که در آن، تیم توسعه باید در اولین گام، حرفه‌ی بانک‌داری را درک کند. برای این کار تیم توسعه باید موجودیت‌ها یا اشیای موجود در کاربرد بانک‌داری و چگونگی ارتباط آن‌ها با یکدیگر، ویژگی‌ها و حالت‌های مختلف این اشیا، و مواردی از این قبیل را بشناسد. تیم باید برای رسیدن به این درجه از شناخت، اطلاعات مربوط به این کاربرد را جمع‌آوری کند، آن‌ها را تحلیل نماید، و یک مدل برای نمایش برداشت‌های اولیه‌ی خود بسازد. فرایند انجام این فعالیت‌ها، مدل‌سازی دامنه نامیده می‌شود.

۵-۲ چرا مدل‌سازی دامنه؟

مهندسان نرم‌افزار باید روی پروژه‌هایی از دامنه‌های مختلف کار کنند. این مهندسان، اطلاعات پایه و تجربیات کاری متفاوتی دارند که در سطح دریافت آن‌ها از دامنه‌های کاربردی مختلف موثر است. برای مثال، عبارت امنیت در دامنه‌های مختلف و یا حتی یک دامنه به معانی مختلفی به‌کار می‌رود. در دامنه‌ی علوم کامپیوتر، این عبارت به معنای امنیت کامپیوتر است و در یک دامنه‌ی مالی، در مورد سرمایه‌هایی چون پشتوانه‌ی مالی و یا تنخواه‌های دوجانبه به‌کار می‌رود. به این ترتیب، دو مهندس نرم‌افزار ممکن است دو برداشت مختلف از این عبارت داشته باشند. حتی مهندسانی که سال‌ها در کنار هم در یک شرکت و روی پروژه‌های مشابه کار کرده‌اند، ممکن است برداشت‌های متفاوتی از دامنه‌های کاربردهای موجود داشته باشند. اگر این تفاوت برداشت‌ها جزیی باشد، اثر مهمی در طراحی، پیاده‌سازی، یکپارچه‌سازی، آزمون، و نگهداری محصول نرم‌افزاری نخواهد

[1] Generalization

داشت. با این وجود اگر تفاوت‌ها اساسی باشند، حتماً در یکی از مراحل توسعه‌ی نرم‌افزار خود را نشان خواهند داد. این امر ممکن است موجب دوباره‌کاری و تاخیر قابل‌توجهی در تاریخ تحویل پروژه شود.

ساخت یک مدل‌دامنه به شناسایی و رفع برداشت‌های متفاوت کمک می‌کند. به‌ویژه:

- مدل‌سازی دامنه به تیم توسعه یا تحلیل کمک می‌کند که کاربرد و دامنه‌ی آن را درک کنند.
- مدل‌سازی دامنه به اعضای تیم امکان می‌دهد که با ارتباط با هم حس عمومی خود از کاربرد و دامنه‌ی آن را ارتقا دهند.
- مدل‌سازی دامنه به تیم توسعه امکان می‌دهد که برداشت‌های خود را به مشتری و یا کاربران منتقل کنند و بازخورد آن‌ها را بگیرند.
- مدل‌سازی دامنه یک پایه‌ی مفهومی مشترک برای طراحی، پیاده‌سازی، آزمون، و نگهداری در آینده ایجاد می‌کند.
- یک مدل‌دامنه می‌تواند به اعضای تیم در شناسایی کاربرد مربوطه و دامنه‌ی آن کمک کند.

۵-۳ شیءگرایی و نمودار کلاس

مدل‌سازی دامنه کمک می‌کند که مفاهیم مهم دامنه، ویژگی‌های آن‌ها، و ارتباط بین این مفاهیم شناخته شوند. این مفاهیم به‌صورت کلاس‌ها و روابط بین کلاس‌ها نمایش داده می‌شوند و می‌توانند با نمودارهای کلاس UML به تصویر کشیده شوند. این بخش، برخی از مفاهیم پایه‌ی شیءگرایی، و نحوه‌ی نمایش آن‌ها در نمودارهای کلاس UML را مرور می‌نماید.

۵-۳-۱- تعاریف توسعه‌ای و ذهنی

مدل‌سازی دامنه یک فرایند مفهوم‌سازی است. به‌عنوان بخشی از این فرایند، توسعه‌دهنده نمونه‌های خاصی از یک کاربرد را مشاهده می‌کند و نمونه‌های مشاهده‌شده را دسته‌بندی، مجردسازی، و عمومی می‌کند تا یک مدل مفهومی به نام مدل دامنه ایجاد نماید. یک گام مهم، فرایند دسته‌بندی است. انگاره‌ی «کلاس»، محصول نهایی فرایند دسته‌بندی است. برای روشن شدن موضوع، یک کودک دوساله را در نظر بگیرید که با مادرش در حال قدم زدن است. کودک یک سگ را می‌بیند و مادرش به او می‌گوید که این یک سگ است، سپس او مفهوم مورد برداشت خود از یک سگ را شکل می‌دهد، بعداً او سگ‌ها و گربه‌های دیگری از رنگ‌های گوناگون را می‌بیند. در این لحظه، فرایند مفهوم‌سازی کودک یک فرایند قیاسی است. رهیافتی که او برای یادگیری مفاهیم جدید استفاده می‌کند، بر مبنای تعریف توسعه‌ای[1] است که به‌صورت زیر تعریف می‌شود:

تعریف ۵-۲

یک تعریف توسعه‌ای، مفهومی را با برشمردن نمونه‌هایی از آن مفهوم تعریف می‌نماید.

یک تعریف توسعه‌ای از عددهای زوج به صورت «مجموعه‌ای از اعداد شامل ... ،۴،۲،۰،-۲،-۴، ...» می‌باشد. برای کودک، تعریف توسعه‌ای «سگ» از سگ همسایه، سگ کنار خیابان، سگی که نزدیک پارک زندگی می‌کند، و مانند آن تشکیل می‌شود. هرچه کودک دنیا را بیشتر می‌بیند، یاد می‌گیرد که سگ‌ها و گربه‌ها رفتار متفاوتی دارند ولی هر دو حیوان‌اند، پس او یاد می‌گیرد چیزهای مختلف را مجردسازی، دسته‌بندی، و عمومی‌سازی نماید تا بسته به ویژگی‌هایشان از آن‌ها کلاس‌هایی تشکیل دهد. او همچنین می‌داند که چگونه این کلاس‌ها را به‌هم مربوط سازد.

---

[1] Extensional

یک روز آن پسر بسیار هیجان‌زده است، زیرا به‌زودی به ملاقات عمه‌اش می‌رود که به‌تازگی به یک خانه‌باغ نقل‌مکان کرده است. پسر به‌دلیل آن‌که خانه‌های شهرش باغ ندارند بسیار کنجکاو است. او براساس دانشی که از خانه‌ها و باغ‌ها دارد، یک خانه‌باغ را نقاشی می‌کند. سپس او این خانه‌باغ را با لگوهایش (Lego) می‌سازد. در این حالات، کودک مفهوم شخصی خود را از یک تعریف ذهنی[1] می‌سازد.

انگاره	معنا	نمادگان
کلاس صفت عملیات	هر کلاس یک گونه است صفت‌ها و عملیات یک کلاس اشیای آن کلاس را مشخص می‌کنند.	نمایش فشرده: نام کلاس نمایش گسترده: نام کلاس / فهرست صفت‌ها / فهرست عملیات
وراثت	رابطه‌ی تعمیم/تخصیص بین دو کلاس	ابَرکلاس ← زیرکلاس
تجمیع	رابطه‌ی بخشی-از بین دو کلاس رابطه‌ی جزء/کل	کل ◇— جزء کل ◆— جزء
انجمن، جهت، تعدد، نقش	رابطه‌ای دوطرفه بین دو کلاس	کلاس1 [m] [برچسب] ▶ [n] کلاس2 [نقش1] [نقش2]
کلاس انجمن	کلاسی که یک انجمن را توصیف می‌کند	کلاس انجمن
		[x] بدین معنی است که x اختیاری است

شکل ۵-۱: چند انگاره و نماد پراستفاده در نمودار کلاس

تعریف ۵-۳

یک تعریف ذهنی، یک مفهوم را از طریق توصیف ویژگی‌ها و رفتارهایی که نمونه‌هایی از آن مفهوم دارد، تعریف می‌کند.

مدل‌سازی دامنه، مانند درک کودک از جهان اطراف خود است و از هر دو نوع تعریف توسعه‌ای و ذهنی استفاده می‌کند. تیم به یک دامنه‌ی کاربردی جدید وارد می‌شود. تیم، مانند یک کودک، چیزهایی که می‌بیند را برمی‌شمرد و آن‌ها را به کلاس‌ها، ویژگی‌ها، و روابطی دسته‌بندی می‌نماید. نتیجه در قالب نمودارهای کلاس UML که به شکل زیر تعریف می‌شود به تصویر کشیده می‌شود.

[1] Intentional

تعریف ۵-۴

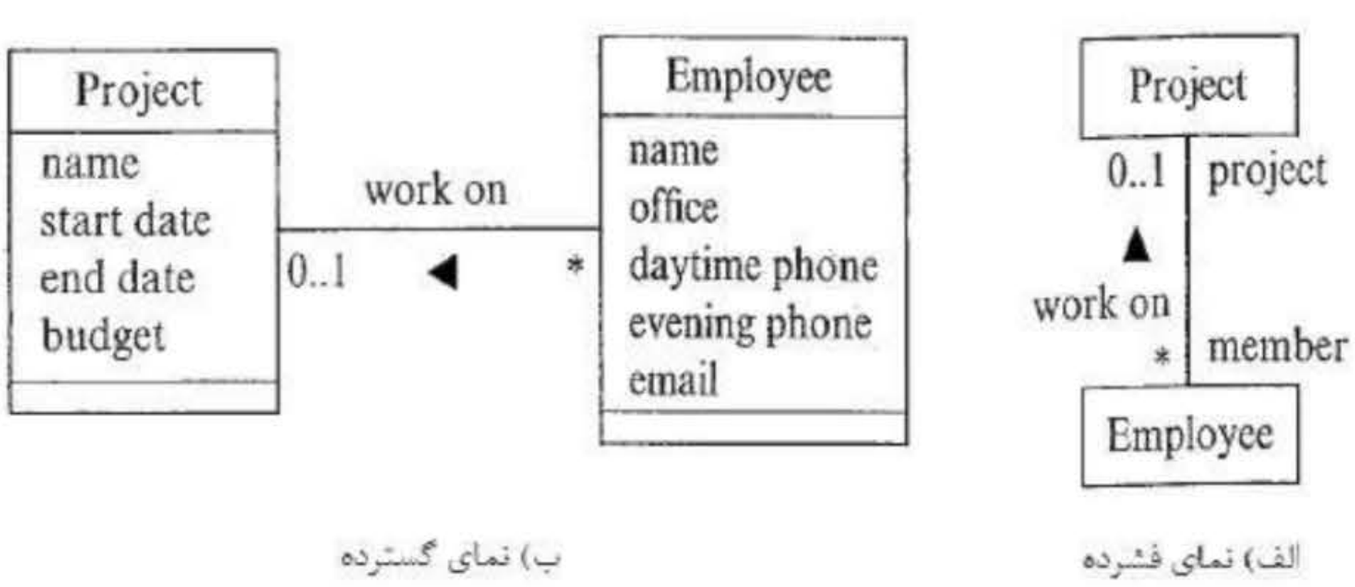
یک نمودار کلاس UML، یک نمودار ساختاری است که کلاس‌ها و صفت‌ها و عملیات آن‌ها را به همراه روابط بین کلاس‌ها به تصویر می‌کشد.

شکل ۵-۱ انگاره‌ها و نمادهای نمودار کلاس را، که در مدل‌سازی دامنه مورد استفاده قرار می‌گیرند، نشان می‌دهد. این مفاهیم در بخش‌های آینده به تفصیل شرح داده خواهند شد.

۵-۳-۲- کلاس و شئ

تعریف ۵-۵

یک کلاس، یک نوع است؛ یک تعریف ذهنی از یک مفهوم. یک کلاس، صفت‌ها و عملیاتی را در خود کپسوله (Encapsulate) می‌کند که نمونه‌های آن کلاس را مشخص می‌نماید. یک شئ، نمونه‌ای از یک کلاس است.



شکل ۵-۲: نمایش کلاس‌ها از دو نمای فشرده و گسترده

همان‌طور که در شکل‌های ۵-۲-الف و ۵-۲-ب دیده می‌شود، کلاس‌ها می‌توانند هم از نمای فشرده و هم از نمای گسترده نشان داده شوند. اگر نیازی به نمایش جزییات کلاس نباشد، از نمای فشرده استفاده می‌شود. نمای گسترده برای نمایش صفت‌ها و عملیات یک کلاس به‌کار می‌رود. هر دو نمای گسترده و فشرده می‌توانند در یک نمودار کلاس مورد استفاده قرار گیرند. یعنی ممکن است برخی از کلاس‌ها به شکل فشرده و سایر کلاس‌ها به شکل گسترده نشان داده شوند.

ویژگی‌ها و عملیات یک کلاس، خصیصه‌های کلاس نیز نامیده می‌شوند. واژه‌ی «خصیصه» به‌خوبی منظور از نمایان‌کردن و برجسته‌نمودن اساسی‌ترین و لازم‌ترین ویژگی‌ها و رفتارهای کلاس را انتقال می‌دهد. یک کلاس یک نوع است، خواه یک نوع داخلی باشد و یا یک نوع تعریف‌شده توسط کاربر. این یک تعریف ذهنی است، زیرا یک کلاس می‌تواند با یک گزاره تعریف شود. به‌عنوان مثال، وقتی کودک مفاهیم خود از سگ، گربه، حیوان، انسان، و خانه را شکل می‌دهد، در ذهن خود برای هر یک از این‌ها یک گزاره دارد. وقتی او خانه‌باغ را نقاشی می‌کند هم یک گزاره دارد. در تفکر شئ‌گرا، گزاره‌ای که یک کلاس را تعریف می‌کند مجموعه‌ای از صفت‌ها و عملیات کلاس است. برای سادگی بحث، قراردادهای زیر در این فصل مورد استفاده

قرار می‌گیرد:

- نام کلاس با یک حرف بزرگ شروع می‌شود و به شکل مفرد مورد استفاده قرار می‌گیرد: Account، Bank، Customer می‌توانند به‌عنوان نام کلاس مورد استفاده قرار گیرند.
- یک متغیر x از نوع X به شکل x:X نشان داده می‌شود. این قانون برای هر دو نوع عدد و کلاس مورد استفاده قرار می‌گیرد. برای مثال، متغیر c از نوع کلاس Customer به شکل c:Customer نشان داده می‌شود. وقتی امکان هیچ‌گونه اشتباه و سردرگمی وجود نداشته باشد و نوع x از زمینه‌ی مورد بحث قابل تشخیص باشد، نوع و علامت «:» می‌تواند حذف شود.
- یک نمونه c از کلاس C به شکل c:C نشان داده می‌شود. برای مثال، a1:Account، a2:Account، a3:Account سه شئ a1، a2 و a3 از نوع Account را نشان می‌دهد. وقتی امکان هیچ‌گونه اشتباه و سردرگمی وجود نداشته باشد و نوع کلاس از متن قابل تشخیص باشد، این اشیا با نام a1، a2،و a3 مورد ارجاع قرار می‌گیرند. توجه به نکات زیر مفید خواهد بود: (۱) Account: یک شئ ناشناخته از Account را نشان می‌دهد. چنین نوع نمایشی در برخی از زمینه‌ها به‌کار می‌آید، برای مثال اشیایی که در یک حلقه مورد ارجاع قرار می‌گیرند. (۲) x: شئ یتیم را نشان می‌دهد، یعنی یک شئ از یک کلاس نامعلوم. این شکل نمایش در یک زمینه‌ی چندریختی (Polymorphic Context) و یا جایی که ذکر نوع کلاس اهمیتی ندارد، به‌کار می‌رود.
- مجموعه‌ی همه‌ی نمونه‌ها از کلاس C در یک نقطه‌ی مشخص زمان با U(C) نشان داده می‌شود، که در آن U(C)={c|c:C}، یا برای مختصرنویسی به‌صورت U(C)={c1,c2,c3,…..,cn} وقتی اشیا بتوانند شمارش شوند. به این ترتیب، یک نمونه c از کلاس C می‌تواند به‌صورت $c \in U(C)$ نشان داده شود. در این‌جا U مجموعه‌ی جهانی مورد بحث[۱] و U(C)، تجسم[۲] U روی C نامیده می‌شود.
- اگر c یک نمونه از کلاس C باشد، آن‌گاه c.a و c.f(…) به ترتیب برای ارجاع یا دسترسی به یک صفت a و یک تابع f(…) از c به‌کار می‌رود.

۵-۳-۳- شئ و صفت

در عمل گاهی دشوار است که تصمیم بگیریم آیا چیزی باید یک صفت باشد و یا خود یک شئ (یا یک کلاس) باشد. برای مثال، «رنگ» می‌تواند یک صفت از «خودرو» باشد ولی «رنگ» می‌تواند در یک کاربرد فروشگاه رنگ، خود یک «شئ» باشد. مثال دیگر «ساختمان» است که می‌تواند یک شئ و یا یک صفت (مثلاً مکان یک گروه دانشگاهی) باشد. رهنمون‌های زیر برای تشخیص اشیا از صفت‌ها مورد استفاده قرار می‌گیرند:

رهنمون ۵-۱

یک شئ (از یک دامنه‌ی کاربردی)، وجود مستقلی در کاربرد یا دامنه‌ی آن دارد. یک صفت این وجود مستقل را ندارد.

در این‌جا، وجود مستقل به این معنا است که کاربرد مایل است اطلاعات یا حالت این شئ را نگهداری نماید. برای مثال، «تعداد صندلی‌ها» وجود مستقلی ندارد زیرا می‌تواند تعداد صندلی‌های یک خودرو، یک کلاس، و یا یک هواپیما باشد که همه از اشیا هستند. «وجود مستقل» محدود به وجود فیزیکی نمی‌باشد، بلکه وجود مفهومی یا ذهنی را نیز در برمی‌گیرد. برای مثال، یک کتاب وجود فیزیکی دارد ولی رشته‌ی تحصیلی مفهومی ذهنی است.

---

[۱] The Universe of Discourse

[۲] Projection

رهنمون ۵-۲

صفت‌ها، اشیا را توصیف و مشخص می‌نمایند.

رهنمون ۵-۳

صفت‌ها می‌توانند از طریق ابزار ورودی (مانند صفحه‌کلید) وارد شوند، ولی اشیا نمی‌توانند. اشیا با فراخوانی یک سازنده ساخته می‌شوند.

کسی نمی‌تواند یک شیء «دانش‌آموز» را از طریق صفحه کلید وارد کند، ولی می‌تواند نام دانش‌آموز، شماره‌ی دانش‌آموز، آدرس، شماره تلفن، و سایر اطلاعات او را وارد کند. این موارد صفت هستند، نه شیء. شیء دانش‌آموز با فراخوانی سازنده‌ی[۱] کلاس Student ساخته می‌شود.

۵-۳-۴- انجمن[۲]

مدل دامنه، علاوه بر کلاس‌ها و صفت‌ها، ارتباطات بین کلاس‌های اشیا را نیز ثبت می‌نماید. به‌طور معمول سه نوع رابطه در مدل‌سازی دامنه مورداستفاده قرار می‌گیرد که در این بخش و بخش‌های آتی معرفی می‌گردد:

تعریف ۵-۶

انجمن، رابطه‌ای بین یک یا چند کلاس است. این رابطه بیان می‌کند که اشیای یک کلاس می‌توانند با اشیای کلاس‌های دیگر مرتبط باشند.

رابطه‌ی انجمنی، یک ارتباط عمومی بین اشیای یک یا چند کلاس را تعریف می‌نماید. یعنی هر رابطه‌ای که در کاربرد موردعلاقه باشد می‌تواند رابطه‌ای انجمنی باشد. برای مثال، دریک سیستم بانکی مشتریان صاحب حساب‌ها هستند. به‌این ترتیب، یک رابطه‌ی انجمنی بین کلاس Customer و کلاس Account وجود دارد. پول می‌تواند بین حساب‌ها جابه‌جا شود. پس یک رابطه‌ی انجمنی بین کلاس Account وخودش وجود دارد. در یک دانشگاه، اعضای هیأت علمی درس‌ها را تدریس می‌کنند و دانشجویان در درس‌ها ثبت‌نام می‌کنند. پس روابط انجمنی بین Faculty و Course، و Student و Course وجود دارد. یک رابطه‌ی انجمنی بین دو کلاس، با یک خط توپر که دو کلاس را متصل می‌کند نشان داده می‌شود.

در شکل ۵-۲ یک رابطه‌ی انجمنی بین Employee و Project نشان داده شده است که به‌معنی آن است که کارمند روی پروژه‌ها کار می‌کند. مثلث کوچک جهت انجمن را نشان می‌دهد، بدین معنی که کارمندان روی پروژه‌ها کار می‌کنند، ولی عکس این گزاره صحیح نیست.

نمایش جدولی به درک ارتباطات انجمنی کمک می‌نماید. رابطه‌ی انجمنی «مشتری صاحب حساب است» رادر نظر بگیرید که با Own(Customer, Account) نشان داده می‌شود. Own از نوع دوتایی (Customer, Account) است. به‌عبارت دیگر می‌توان نوشت Own:(Customer, Account) که برای سادگی علامت : آن حذف شده است. با استفاده از قراردادهای ذکرشده

---

[۱] Constructor

[۲] Association

مالکیت

Customer	Account
C1	a1
C1	a2
C2	a2
C2	a3
C3	a4

الف) نمونه‌هایی از یک انجمن دودویی

کار نظارت‌شده توسط

Student	Project	Professor
Chen	OOM	Baker
Chen	SOA	Liu
Gupta	SOA	Liu
Rosa	Security	Brown
Smith	Security	Shah

ب) نمونه‌هایی از یک انجمن سه‌سه‌ای

شکل ۵-۳: نمونه‌هایی از یک انجمن

در بخش ۵-۳-۲، فرض کنید U(Account)={a1,…,a5} و U(Customer)={c1,…,c4}. c1 صاحب a1 و a2، c2 صاحب a2 و a3، c3 صاحب a4 است و c4 حسابی ندارد، و هیچ مشتری صاحب a5 نیست. همان‌طور که در شکل ۵-۳-الف نشان داده شده است، رابطه‌ی انجمنی Own(Customer, Account) از پنج دوتایی تشکیل‌شده‌است: (c1,a1)، (c1,a2)، (c2,a2)، (c2,a3)، (c3,a4). جدول به همراه U(Customer) و U(Account) نشان می‌دهد که یک مشتری می‌تواند صاحب صفر یا چند حساب باشد و یک حساب می‌تواند متعلق به صفر یا چند مشتری باشد. درواقع، هر یک از زیرمجموعه‌های مجموعه‌ی ۲۰ زوجی، که با جفت‌کردن هر یک از چهار مشتری با هر یک از پنج حساب به‌دست می‌آید، قابل قبول است. یعنی 2^{20} زیر مجموعه از این‌دست وجود دارد و هر یک از آن‌ها یک نمونه‌ی معتبر از رابطه‌ی انجمنی Own می‌باشد.

بیشترین رابطه‌های انجمنی که تاکنون مشاهده‌شده، روابط انجمنی دودویی[۱] می‌باشند. انجمن‌های دودویی رابطه‌هایی بین اشیای یک یا دو کلاس هستند. برای مثال، این‌که یک حساب می‌تواند به حساب دیگر پولی واریز کند، نشان‌دهنده‌ی یک رابطه‌ی دودویی بین اشیای یک کلاس می‌باشد. به‌عنوان یک مثال دیگر، یک کارمند می‌تواند یک کارمند دیگر را سرپرستی کند، که این یک رابطه‌ی انجمنی بین اشیای کلاس Employee خواهد بود. انجمن بین اشیای سه کلاس و یا یک رابطه‌ی انجمنی سه‌سه‌یی[۲] نیز می‌تواند وجود داشته باشد. برای مثال رابطه‌ی بین استاد (Professor)، دانشجو (Student) و پروژه (Project) که در آن استادان، دانشجویانی که روی یک پروژه کار می‌کنند را سرپرستی می‌نمایند، یک رابطه‌ی انجمنی سه‌سه‌یی یا سه‌جانبه است که اشیای سه کلاس در آن درگیر هستند. شکل ۵-۳-ب برخی از نمونه‌های ممکن از این رابطه‌ی سه‌سه‌یی را شرح می‌دهد.

همان‌طور که در مثال‌های بالا شرح داده‌شد، یک رابطه‌ی انجمنی دودویی می‌تواند با شکل V(X,X) ویا V(X,Y)، ویک رابطه‌ی انجمنی سه‌سه‌یی با شکل V(X,Y,Z) نشان داده شود که در آن V نام انجمن و X,Y,Z کلاس‌ها می‌باشند. این نماد در زمان تعریف انجمن‌ها در این کتاب مورد استفاده قرار خواهد گرفت. وقتی ریسک اشتباه و سردرگمی وجود نداشته باشد، پرانتزها و آرگومان‌ها حذف می‌شود و رابطه‌ی Own(Customer,Account) به‌شکل Own نوشته می‌شود.

۵-۳-۵- تعدد[۳] و نقش[۴]

رابطه‌ی انجمنی Own(Customer, Account) بیان می‌کند که مشتریان می‌توانند صاحب حساب‌ها باشند. بااین‌حال، این رابطه نشان نمی‌دهد که هر مشتری می‌تواند چند حساب داشته باشد، ویا چند مشتری می‌توانند حساب مشترک داشته باشند. معمولاً یک مشتری می‌تواند در یک بانک بیش از یک حساب داشته باشد. برخی از بانک‌ها، مالکیت مشترک یک حساب

---

[۱] Binary Association
[۲] Ternary Association
[۳] Multiplicity
[۴] Role

را اجازه می‌دهند، ولی برخی دیگر لازم می‌دانند که یک حساب متعلق به «یک و فقط یک» مشتری باشد. به‌عنوان یک مثال دیگر، یک دانشجو می‌تواند روی یک یا چند پروژه کار کند، ولی یک پروژه ممکن است دارای صفر یا بیشتر دانشجوی فعال باشد. یک پروژه ممکن است دانشجویی نداشته باشد، مانند وقتی که پروژه جدید است و یا خاتمه یافته است. دانشجویی که روی یک پروژه کار می‌کند ممکن است توسط یک یا چند استاد سرپرستی گردد. این مثال‌ها نشان می‌دهند که لازم است تعدد، شناسایی و توصیف گردد. تعدد به شکل زیر تعریف می‌گردد:

تعریف ۵-۷

تعدد یک کلاس در یک رابطه‌ی انجمنی، تعداد نمونه‌هایی از آن کلاس است که می‌توانند با هر ترکیبی از یک نمونه از هر یک از کلاس‌های دیگر آن رابطه در ارتباط باشند.

رابطه‌ی انجمنی Own(Customer, Account) را در نظر بگیرید. فرض کنید که بانک اجازه‌ی مالکیت مشترک بر یک حساب را می‌دهد، ولی لازم می‌داند که هر حساب حداقل یک مالک داشته باشد. تعدد Customer نسبت به رابطه‌ی انجمنی Own(Customer, Account)، یک یا چند است زیرا هر شئ Account می‌تواند متعلق به یک یا چند مشتری باشد. در این حالت، شکل ۵-۳-الف دیگر معتبر نیست زیرا برای حساب a5 یک مشتری وجود ندارد. اضافه کردن زوج (c4,a5) به جدول شکل ۵-۳-الف این قید را برآورده خواهد کرد. اگر بانک محدودیت جدیدی اضافه کند که هر حساب تنها می‌تواند متعلق به یک مشتری باشد، تعدد Customer یک خواهد بود. در این حالت شکل ۵-۳ -الف دیگر معتبر نیست. حذف زوج (c1,a2) از جدول این قید را برآورده خواهد کرد.

برای رابطه‌ی انجمنی سه‌سه‌یی بین Professor، Student، و Project، اگر هر دانشجوی درحال‌کار بر روی یک پروژه بتواند فقط توسط یک استاد راهنمایی شود، دراین‌صورت تعدد Professor نسبت به رابطه‌ی سه‌سه‌یی، یک است. همچنین در شکل ۵-۳-ب ، دانشجویی به‌نام چن دو استاد مختلف دارد و بر روی دو پروژه مختلف نیز کار می‌کند. بنابراین، رابطه‌ی نشان داده شده در شکل ۵-۳-ب هنوز قید را برآورده می‌کند. اگر یک یا چند استاد بتوانند یک دانشجوی درحال‌کار بر روی یک پروژه را راهنمایی کنند، آنگاه تعدد Professor نسبت به انجمن سه‌سه‌یی، یک یا چند است. در این حالت، جدول ۵-۳-ب هنوز معتبر است زیرا هر دانشجوی درحال‌کار بر روی یک پروژه، یک استاد راهنما دارد. بااین‌وجود، اگر یک دانشجو فقط بتواند روی یک پروژه کار کند و توسط یک استاد راهنمایی شود، دراین‌صورت یکی از دو چندتایی موجود در جدول شکل ۵-۳-ب باید حذف شود.

شکل ۵-۴ نمادهای UML موجود برای مقادیر مختلف تعدد را نشان می‌دهد. در یک نمودار کلاس، تعدد با تقاطع انجمن و لبه‌های کلاس نشان داده می‌شود. برای مثال، شکل ۵-۲ نشان می‌دهد که هر کارمند بر روی صفر یا یک پروژه کار می‌کند و هرپروژه، صفر یا بیشتر کارمند دارد.

اشیا نقش خاصی در رابطه‌های انجمنی ایفا می‌کنند. برای مثال، در رابطه‌ی work-supervised-by بین Student، Project، و Professor یک استاد نقش یک راهنما را ایفا می‌کند و یک دانشجو نقش یک دستیار پژوهشی را ایفا می‌کنند. در مثال Own(Customer, Account)، یک مشتری نقش صاحب حساب را ایفا می‌کند. نقش‌ها نیز مانند تعدد در نمودار کلاس نمایش داده می‌شوند. شکل ۵-۲ نشان می‌دهد که Employee و Project در رابطه‌ی work-on به‌ترتیب نقش‌های عضو و پروژه را ایفا می‌کنند.

0..1	zero or one	m..n	m to n
0..m	zero to m	m..*	m or more
,0..	zero or more	m	exactly m
1	exactly one (default)	1..*	one or more
i, j, k	explicitly enumerated		

شکل ۵-۴: نمادهایی برای بیان شکل‌های مختلف تعدد

۵-۳-۶- تجمیع

انجمن، رابطه‌ای عمومی بین یک یا چند کلاس است. کاربردهای عملی، به‌تناوب با رابطه‌های معروف به «بخشی از» یا Part-of برخورد می‌نمایند. برای مثال، یک موتور «بخشی از» یک خودرو است، یک کتاب از چند فصل تشکیل می‌شود که هریک شامل چند بخش هستند. رابطه‌ی part-of یک نوع خاص از رابطه‌ی انجمنی است و شایسته است که نام و مفهوم مدل‌سازی خاصی برای خود داشته باشد. به‌همین دلیل، جامعه‌ی مدل‌سازی نام معروف رابطه‌ی «تجمیع» را برای آن برگزیده است.

تعریف ۵-۸

تجمیع، یک رابطه‌ی دودویی بین دو کلاس است. این رابطه نشان می‌دهد که اشیای یک کلاس، بخش‌هایی از اشیای کلاس دیگر هستند.

چون تجمیع حالت خاصی از رابطه‌ی انجمنی است، به‌صورت Part-of(P,C) یا Part-of(P1,P2,…,Pn,C) نشان داده می‌شود تا بگوید که کلاس C تجمیعی از کلاس‌های P1,P2,…,Pn است. رابطه‌ی Part-of(P1,P2,…,Pn,C) درواقع نمایش اختصاری Part-of(P1,C) ، Part-of(P2,C) ،... و Part-of(Pn,C) است. به‌همین ترتیب، مفاهیم تعدد و نقش که برای رابطه‌ی انجمنی تعریف شده بود، برای تجمیع نیز تعریف می‌گردد. به‌عنوان مثال، هر خودرو، دو یا چهار در دارد. یعنی، تعدد door نسبت به رابطه‌ی Part-of(Door, Car) دو یا چهار است. نماد UML برای تجمیع، یک خط با یک لوزی است، که لوزی در طرف کلاس تجمیعی قرار می‌گیرد. شکل ۵-۵-الف نشان می‌دهد که یک Car (ماشین)، اجتماعی از یک Engine (موتور)، یک Brake (ترمز) و یک Transmission (میل‌لنگ) است.

۵-۳-۷- ارث‌بری[۱]

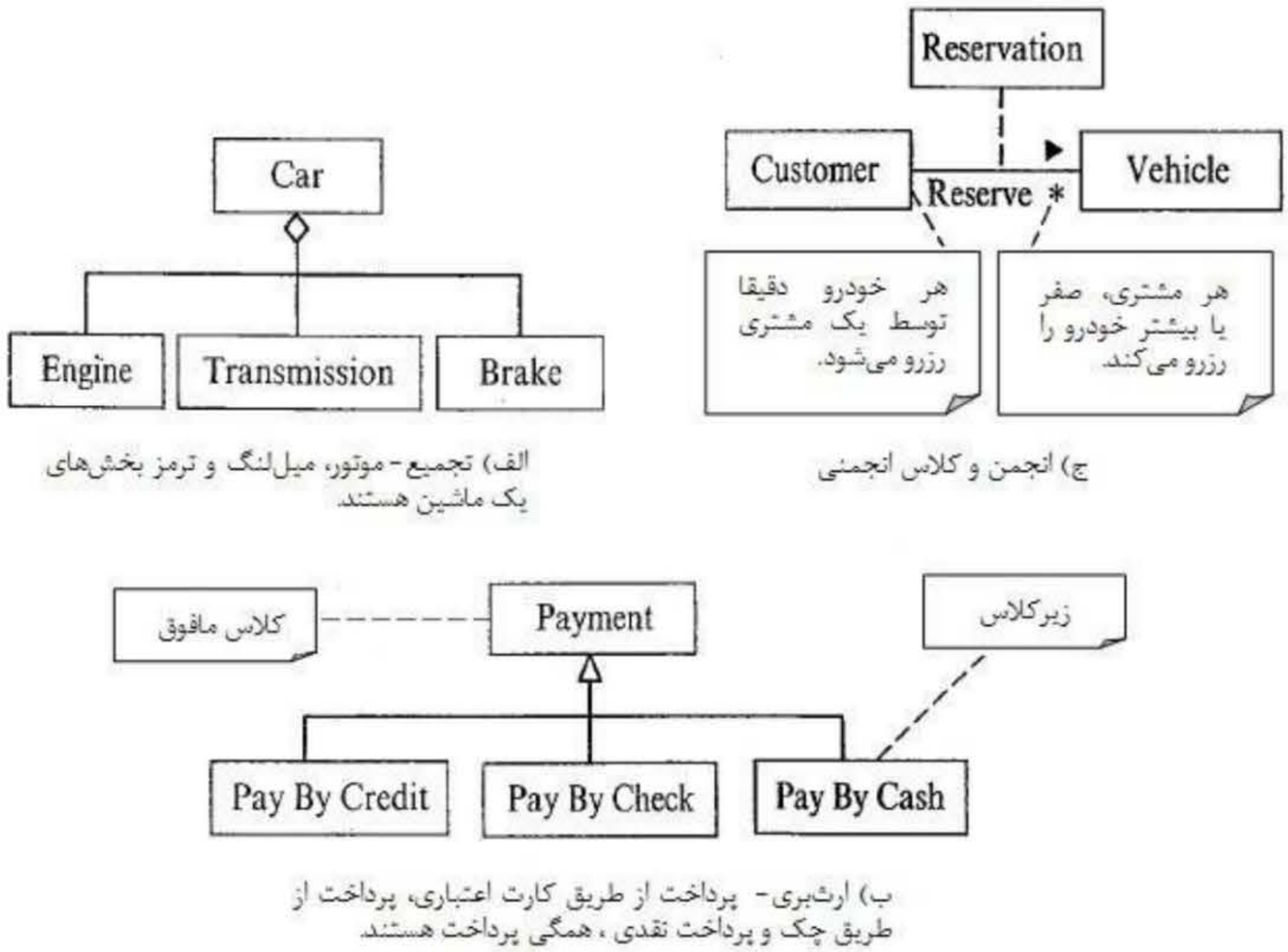
تعریف ۵-۹

ارث‌بری یک رابطه‌ی دودویی بین دو مفهوم یا کلاس است، به‌طوری‌که یک مفهوم یا کلاس، تعمیم‌یافته (یا خاص‌شده‌ی) کلاس دیگر باشد.

مثال‌های بسیاری از ارث‌بری در دامنه‌های کاربردی مختلف وجود دارند. در یک سیستم بانک‌داری، حساب جاری[۲] و حساب پس‌انداز[۳] وجود دارد که خاص‌شده‌ی حساب (Account) هستند. یک شرکت واسطه‌گری[۴] نیز میان انواع حساب

---

[۱] Inheritance
[۲] Checking Account
[۳] Savings Account
[۴] Brokerage



شکل ۵-۵: نمایش روابط و کلاس تجمیع

استقراضی[۱] و حساب اختیاری[۲] تفاوت قایل می‌شود. این‌ها همه انواع مختلفی از حساب هستند که مشابه یکدیگرند ولی رفتارهای مختلفی دارند. برای مثال، برای حساب جاری سود پرداخت نمی‌شود، ولی به حساب پس‌انداز سود تعلق می‌گیرد. یعنی Checking Account و Savings Account خاص‌شده‌ی Bank Account هستند و Bank Account تعمیم‌یافته‌ی این دو کلاس می‌باشد. Margin Account و Option Account، خاص شده‌ی Brokerage Account هستند و Brokerage Account تعمیم‌یافته‌ی این دو کلاس می‌باشد.

همان‌طور که در شکل ۵-۵-الف دیده می‌شود، در یک رابطه‌ی ارث‌بری، کلاسی که تعمیم‌یافته‌ی دیگر کلاس‌ها می‌باشد، کلاس مافوق[۳]، و کلاسی که خاص‌شده‌ی کلاس دیگر است، زیرکلاس[۴] نامیده می‌شود. رابطه‌ی ارث‌بری، رابطه‌ی IS-A هم نامیده می‌شود. این بدان علت است که هر نمونه از یک زیرکلاس، یک نمونه از کلاس مافوق آن است. برای مثال هر حساب پس‌انداز یا هر حساب جاری، یک حساب بانکی است. این مطلب نشان می‌دهد که صفت‌ها و رفتارهای تعریف‌شده برای کلاس مافوق، توسط زیرکلاس به ارث برده می‌شوند. یعنی استفاده از واژه‌ی «ارث‌بری»، برای اشاره به رابطه‌های عمومیت‌یافتن یا خاص‌شدن بین کلاس‌ها بوده است.

در مطلب قبل، نیاز به تعریف زیرکلاس‌های Savings Account و Checking Account از کلاس مافوق Bank Account، به‌دلیل تفاوت‌های رفتاری بین اشیای این زیرکلاس‌ها می‌باشد. یک دلیل دیگر برای تعریف زیرکلاس‌ها، تفاوت در رابطه‌های آن‌ها است. برای مثال، دانشجوی کارشناسی (Undergraduate) و دانشجوی تحصیلات تکمیلی (Graduate) را به‌عنوان

---

[۱] Margin Account
[۲] Option Account
[۳] Super Class
[۴] Sub Class

زیرکلاس‌های دانشجو (Student) درنظر بگیرید. یک دانشجوی کارشناسی باید از برنامه‌ی مربوط به مقطع کارشناسی برای انتخاب درس‌های خود استفاده کند، در حالی که یک دانشجوی تحصیلات‌تکمیلی، برنامه‌ی خاص مقطع خود را نیاز دارد. از سوی دیگر، یک دانشجوی کارشناسی ممکن است محدودیت‌های متفاوتی داشته باشد. تعریف دو زیرکلاس، دست‌کاری و تغییر این رابطه‌های انجمنی را بسیار ساده خواهد کرد. ارث‌بری نیز یک حالت خاص از رابطه‌ی انجمنی است. بنابراین، ISA(S,C) یا ISA(S1,S2,…,Sn,C) برای نشان دادن این‌که کلاس‌های S,S1,S2,...,Sn زیرکلاس‌های کلاس C هستند، به‌کار می‌رود. درواقع ISA(S1,S2,…,Sn,C) شکل اختصاری ISA(S1,C),ISA(S2,C),...,ISA(Sn,C) می‌باشد.

۵-۳-۸- ارث‌بری وچندریختی

در تفکر شیءگرا، ارث‌بری وچندریختی[1] مانند دو برادر و یا حتی دوقلو هستند. این‌دو آن‌قدر شبیه به‌نظر می‌رسند که برخی از نویسندگان آن‌ها را یکسان در نظر می‌گیرند، ولی در واقع این طور نیست. برای درک تفاوت آن‌ها، تعریف زیر ارائه می‌گردد:

تعریف ۵-۱۰

چندریختی به این معناست که یک چیز بتواند شکل‌های مختلفی به‌خود بگیرد.

با نگاهی به تعریف‌های ۵-۹ و ۵-۱۰، تفاوت این دو مفهوم دیده می‌شود. ارث‌بری، یک رابطه بین دو مفهوم یا کلاس را طوری تعریف می‌کند که یک کلاس عمومی‌تر (یا خاص‌تر) از دیگری باشد. با توجه به این رابطه‌ی عمومی‌کردن (یا خاص‌کردن)، یک متغیر از نوع کلاس مافوق می‌تواند به یک شئ از یک زیرکلاس رجوع نماید. به‌این‌دلیل، متغیر کلاس مافوق می‌تواند رفتارهای مختلفی از خود نشان دهد و به چندریختی برسد.

از سوی دیگر، چندریختی با توانایی یک چیز برای پذیرفتن شکل‌های مختلف تعبیر می‌شود. از آن‌جا که «یک چیز» مقید نشده است، می‌تواند نه‌تنها کلاس و شئ، بلکه هر چیزی باشد. قابلیت پذیرش شکل‌های مختلف تنها به زمینه‌های ارث‌بری محدود نمی‌شود، بلکه می‌تواند هر زمینه‌ی دیگری را شامل باشد. برای مثال، دو یا چند تابع عضویت می‌توانند دارای یک نام مشترک باشند ولی امضاهای متفاوت داشته باشند. یک اشاره‌گر به تابع در C++ می‌تواند در زمان اجرا به توابع مختلفی اشاره کند. این‌ها مثال‌هایی از چندریختی می‌باشند ولی در زمینه‌ی ارث‌بری نیستند.

۵-۳-۹- کلاس انجمنی

گاهی لازم است که صفت‌های روابط انجمنی شرح داده شوند. برای مثال، رابطه‌ی انجمنی «دانشجویان در درس‌ها ثبت‌نام می‌کنند» را در نظر بگیرید. یک دانشجو می‌تواند در یک درس یا بیشتر ثبت‌نام کند و برای هر درس ثبت‌نام‌شده یک نمره بگیرد. روشن است که نمره‌ی درس، یک صفت است زیرا وجود مستقل ندارد. سوال این است که کدام کلاس شئ باید دارای نمره باشد؟ اگر نمره، یک صفت از کلاس Student باشد، آن‌گاه کلاس Student باید چندین صفت نمره داشته باشد زیرا یک دانشجو می‌تواند در بیشتر از یک درس ثبت‌نام کند. اگر این‌طور باشد، چگونه می‌توان بین نمرات درس‌های مختلف تمایز قائل شد؟ چند راه سریع، ولی نه لزوماً خوب، را در ادامه می‌بینید.

یک راه‌حل ممکن، این است که بین هر نمره با یک درس رابطه‌ی انجمنی برقرار شود. این‌کار به دو راه مختلف می‌تواند انجام گردد. راه نخست، استفاده از یک جدول درهم‌سازی در کلاس Student است که از طریق آن نمرات دروس ذخیره‌سازی می‌شود. این روش نامطلوب است، زیرا جدول درهم‌سازی و رابطه‌ی انجمنی Enroll(Student,Course) باید یک تناظر

[1] Polymorphism

یک‌به‌یک داشته باشند. اما این تناظر تلویحاً درنظر گرفته شده است. برنامه‌ای که آن‌ها را پردازش می‌کند لازم است این تناظر را نگهداری نماید. این بدان معنی است که برای هر نمره‌ی ذخیره‌شده در جدول درهم‌سازی، برنامه باید تضمین کند که دانشجو واقعاً در درس ثبت‌نام کرده است. برنامه باید مطمئن شود که برای هر درسی که یک دانشجو در آن ثبت‌نام کرده، یک ورودی در جدول درهم‌سازی وجود دارد که نمره‌ی درس را ثبت می‌نماید.

علاوه بر مسئله‌ی تناظر ذکرشده، معمول است که شماره‌ی نیم‌سالی که دانشجو در آن برای یک درس مشخص نمره‌ای گرفته است ثبت گردد. این‌کار، لازم است زیرا اگر یک دانشجو درسی را سال‌ها پیش گرفته باشد، معمولاً پیش از آن‌که نمره‌ی درس بتواند نیازمندی‌های فارغ‌التحصیلی را برآورده سازد، دانشگاه خواهد خواست که دانشجو دوباره درس را بگیرد. در این حالت، استفاده از جدول درهم‌سازی برای ذخیره‌ی نمره و شماره‌ی نیم‌سال برای هر درس، راه خوبی نیست.

راه‌حل دیگر این است که کلاس Course-Grade برای ذخیره‌ی نمره‌ی هر درس تعریف شود و یک رابطه‌ی انجمنی یک‌به‌چند، مانند Receive(Student,Course-Grade) معرفی گردد. این روش می‌تواند نیاز به ثبت شماره‌ی نیم‌سال به‌همراه نمره‌ی هر درس را پوشش دهد. بااین‌حال، درنظرگرفتن تناظر یک‌به‌یک بین نمونه‌های رابطه‌های انجمنی Receive و Enroll، هنوز نمی‌تواند مشکل فرض‌کردن تلویحی آن را حل کند. یعنی برای ضمانت آن‌که هر نمونه از رابطه‌ی انجمنی Receive، نمونه متناظری از Enroll دارد و هر نمونه از Enroll، نمونه متناظری از Receive دارد، کار اضافی موردنیاز است. به‌همین شکل، اگر کسی نمره را به‌عنوان یک صفت از Course تعریف کند، مشکلاتی مشابه مشکلات فوق خواهد داشت. یک راه سریع دیگر پیشنهاد می‌کند که یک کلاس Grade و دو رابطه‌ی انجمنی اضافه شوند: یک رابطه‌ی انجمنی یک‌به‌چند از Student به Grade، مانند Has(Student,Grade)، و یک رابطه‌ی انجمنی یک‌به‌یک از Grade به Course، مانند Is-for(Grade,Course). ولی این راه‌حل نیز هنوز مشکل در نظرگرفتن تلویحی تناظر را که قبلاً شرح داده شد، حل نمی‌کند.

تحلیل دقیق مسائل مطرح‌شده نشان می‌دهد که صفت نمره، در واقع تکه‌ای از اطلاعات در مورد رابطه‌ی انجمنی Enroll(Student,Course) می‌باشد. به‌همین‌شکل، نیم‌سالی که در آن دانشجو درس را گرفته تا نمره بگیرد نیز تکه‌ای از اطلاعات در مورد Enroll(Student,Course) است. راه‌حل‌های سریع فوق سعی می‌کنند که این تکه‌های اطلاعات را بدون خود رابطه‌ی انجمنی ذخیره کنند که این کار، مسئله‌ی تناظر را ایجاد می‌نماید. یک راه سرراست و طبیعتاً بهتر می‌تواند ذخیره‌کردن اطلاعات در رابطه با انجمن، با خود انجمن باشد. این‌کار مسئله‌ی تناظر را حل خواهد کرد. پرسشی که باقی می‌ماند این است که چگونه این کار را انجام می‌دهیم. تعریف زیر به یک پاسخ اشاره می‌کند.

تعریف ۵-۱۱

یک کلاس انجمنی، کلاس خاصی است که برای نمونه‌های یک انجمن، صفت‌ها و رفتارها را تعریف می‌نماید.

بار دیگر رابطه‌ی Enroll(Student,Course) را در نظر بگیرید. در این مثال، می‌توان یک کلاس انجمنی به نام Enrollment برای رابطه‌ی Enroll بین Student و Course تعریف نمود. Enrollment دارای صفت‌های Grade و Semester ، و عملیات setGrade(…)، getGrade()، setSemester(…)، و getSemester() خواهد بود. اگر یک دانشجو سه درس بگیرد، تنها لازم است که سه نمونه از کلاس Enrollment (یک نمونه برای هر درسی که دانشجو گرفته است) ایجاد شود. شکل ۵-۶ نشان می‌دهد که چگونه نمونه‌های یک کلاس انجمنی می‌توانند با فرمت جدولی نشان داده شوند، به طوری‌که دو ستون اول نمونه‌هایی از طرفین رابطه را نشان دهند و دو ستون آخر نمایانگر ویژگی‌های این نمونه‌ها باشند.

در شکل ۵-۵-ج کلاس‌های انجمنی با یک خط نقطه‌چین نشان داده شده‌اند که کلاس انجمنی را به خط انجمنی وصل

Student	Course	Semester	Grade
Bachman	AI	Spr 07	A
Backman	DB	Spr 07	A
Backman	SE	Spr 07	A
Chang	AI	Spr 07	B
Chang	DB	Spr 07	A
Chang	SE	Spr 07	A
Chang	Compiler	Fal 06	B
Chang	Algorithms	Fal 06	A
Chang	Programming	Fal 06	B

شکل ۵-۶: نمایش جدولی اشیای یک کلاس انجمنی

می‌کند. در این شکل، یک رابطه‌ی انجمنی به نام Reserve بین Customer و Vehicle وجود دارد. کلاس Reservation اطلاعات مربوط به این‌که کدام مشتری کدام وسیله را رزرو می‌کند نگهداری می‌نماید. همان‌طور که خط نقطه‌چین از Reservation به خط انجمنی Reserve نشان می‌دهد، این یک کلاس انجمنی است.

۵-۴ گام‌های مدل‌سازی دامنه

با توجه به مفاهیم شئ‌گرایی که در بخش‌های قبلی معرفی شد، زمان آن رسیده که گام‌های مدل‌سازی دامنه شرح داده شوند. شکل ۵-۷ این گام‌ها را به‌همراه ورودی و خروجی آن‌ها نشان می‌دهد. برای ایجاد یک مدل خوب دامنه، ممکن است لازم باشد که این گام‌ها تکرار شوند. خلاصه‌ی این گام‌ها را در ادامه می‌بینید و شرح تفصیلی هر یک در بخش‌های آتی خواهد آمد.

گام۱. جمع‌آوری اطلاعات دامنه‌ی کاربرد

گام اول برای مدل‌سازی دامنه، جمع‌آوری اطلاعات دامنه‌ی کاربرد است. در بخش‌های قبل، تکنیک‌هایی برای جمع‌آوری اطلاعات دامنه‌ی کاربرد مطرح شده‌است که در بخش ۵-۴-۱ مرور خواهد شد. خروجی این گام، شامل همه‌ی اطلاعات و مستندات مربوطه در زمینه‌ی کاربرد خواهد بود.

گام۲. طوفان فکری

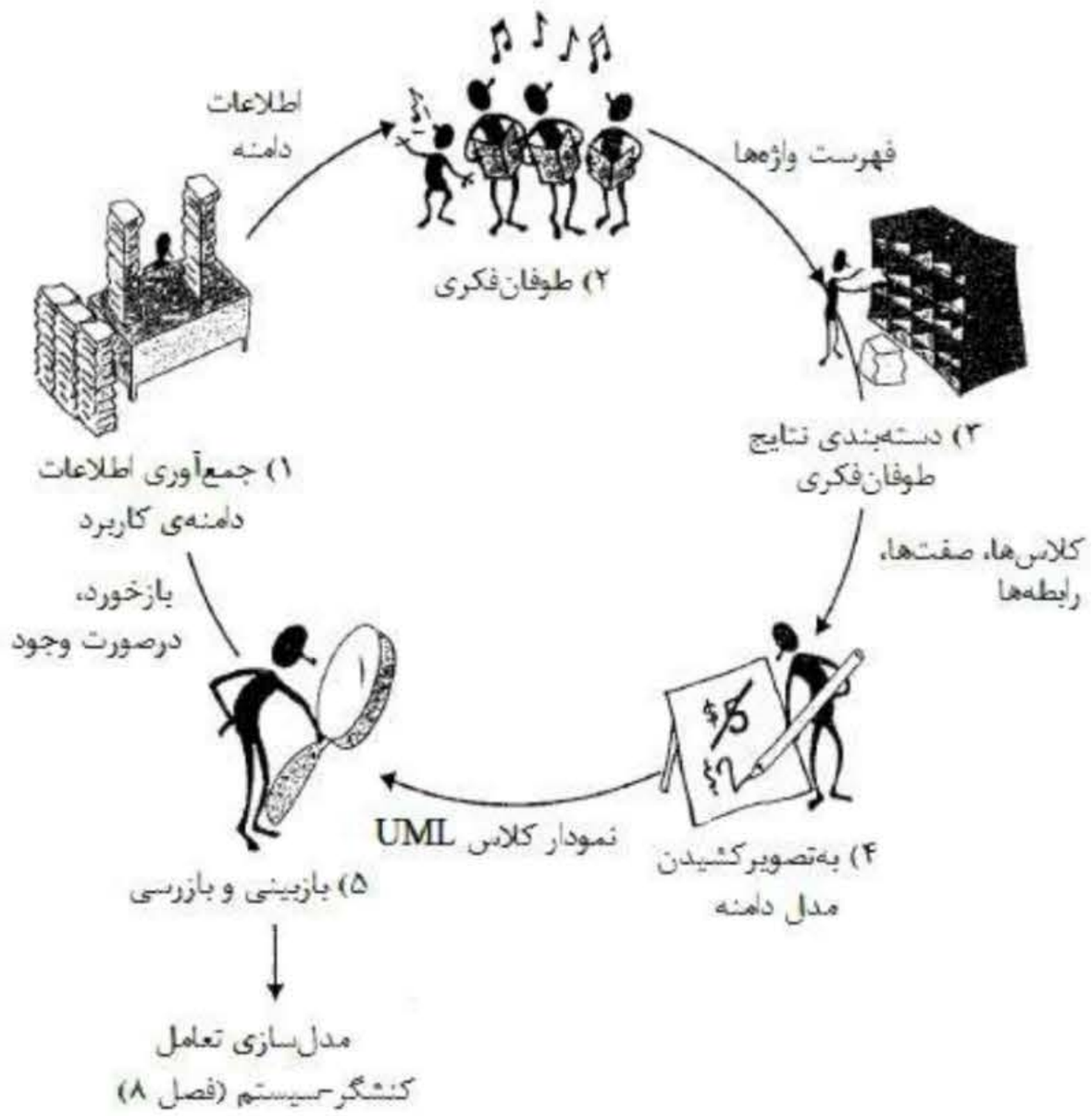
همان‌طور که در بخش ۵-۴-۲ شرح داده خواهد شد، پس از جمع‌آوری اطلاعات در مورد دامنه‌ی کاربرد، اعضای تیم توسعه گردهم می‌آیند تا مفاهیم مهم دامنه‌ی کاربرد را شناسایی و فهرست نمایند.

گام۳. دسته‌بندی نتایج طوفان فکری

سپس نتایج طوفان فکری به کلاس‌ها، صفت‌ها، روابط ارث‌بری، روابط انجمنی، و روابط اجتماع دسته‌بندی می‌شود. این موارد، خروجی‌های این گام را تشکیل می‌دهند. تکنیک‌های دسته‌بندی در بخش ۵-۴-۳ شرح داده شده است.

گام۴. به‌تصویرکشیدن مدل دامنه

نتایج دسته‌بندی به‌کمک یک نمودار کلاس UML به تصویر کشیده می‌شوند تا دیدی کلی و مجتمع از کلاس‌ها و صفت‌های آن‌ها، و روابط بین کلاس‌ها فراهم نمایند. قوانین تبدیل نتایج دسته‌بندی به یک نمودار کلاس در بخش ۵-۴-۴ نشان داده شده است.



شکل ۵-۷: گام‌های مدل‌سازی دامنه

گام۵. مرور مدل دامنه

مدل دامنه، برای شناسایی خطاها و موارد غیرمعمول مرور می‌شود. بسته به نتایج این بازبینی، برخی از گام‌های قبل تکرار می‌شوند و مدل دامنه اصلاح می‌گردد. یک نمونه از فهرست بررسی مدل دامنه در بخش ۵-۴-۵ ارائه شده است.

۵-۴-۱- جمع‌آوری اطلاعات دامنه‌ی کاربرد

چون مقصود اصلی از مدل‌سازی دامنه، فهم مفاهیم دامنه و چگونگی ارتباط آن‌ها با یکدیگر است، تیم توسعه باید مستندات یا توضیحات موجود در مورد کسب‌وکار را به‌دست آورد. این مستندات شامل موارد زیر می‌باشند:

- شرح فرایندهای کسب‌وکار
- رویه‌های عملیاتی، کتابچه‌های راهنما، نظام‌نامه‌ها، سیاست‌ها، آیین‌نامه‌ها، و قوانین
- شرح نیازمندی‌های نرم‌افزار، که باید در مرحله‌ی اکتساب نیازمندی‌ها ایجاد شده باشد.
- فرم‌های مختلف کسب‌وکار (مثلاً فرم‌هایی برای درخواست‌های مختلف، صورت‌حساب‌ها، سفارش‌های خرید، و غیره)

ممکن‌است مستندات، توضیحات، و فرم‌های فوق وجود نداشته باشند و لازم باشد تحلیل‌گر یا تیم توسعه، از تکنیک‌های جمع‌آوری اطلاعات برای به‌دست آوردن آن‌ها استفاده نماید. این تکنیک‌ها عبارتند از:

۱. معرفی مشتری
۲. مصاحبه با نمایندگان مشتری، کاربران، و متخصصین دامنه

۳. مطالعه‌ی نوشته‌ها و ادبیات مرتبط با کسب‌وکار
۴. مطالعه‌ی پروژه‌های مشابه
۵. مطالعه‌ی مستندات و فرم‌های کسب‌وکار
۶. مطالعه‌ی سیاست‌ها و آیین‌نامه‌های دولتی
۷. مطالعه‌ی استانداردهای صنعتی
۸. طراحی و استفاده از پرسش‌نامه‌ها

این تکنیک‌ها به‌تفصیل در فصل ۴ شرح داده شده و در این‌جا تکرار نمی‌شوند.

۵-۴-۲- طوفان فکری

پس از گام جمع‌آوری اطلاعات، اعضای تیم عباراتی که در مستندات مربوطه کاربرد را شرح می‌دهند مشخص نموده در کنار هم قرار می‌دهند، و مفاهیم خاص دامنه را شناسایی می‌نمایند. محصول نهایی این گام، فهرستی از عبارت‌های شناخته‌شده است. مفاهیم دامنه با به‌کاربردن قوانین زیر شناسایی می‌شوند. یعنی اعضای تیم، عبارت‌های خاص‌دامنه یا مربوط به دامنه را به شرح زیر شناسایی می‌نمایند:

۱. اسم‌ها یا عبارت‌های اسمی
۲. عبارت‌های «x از y»، یا «y ِ x» (به‌عنوان مثال رنگ ماشین، موتور ماشین)
۳. افعال متعدی
۴. صفات، قیدها، و اقلام شمارشی
۵. ارقام و اعداد، و کمیت‌ها
۶. عبارت‌های مالکیت (مانند «دارد»)
۷. اجزای سازنده، عبارت‌های «بخشی از» و «تشکیل‌شده از»
۸. عبارت‌های مربوط به دربرداشتن[۱] و محدود نگاه داشتن[۲]
۹. عبارت‌های «X یک Y است»، یا مفاهیم خاص کردن/تعمیم دادن

در زمان به‌کارگیری این قواعد، اعضای تیم باید رهنمون‌های زیر را به خاطر داشته باشند:

رهنمون ۵-۴

برروی مفاهیم خاص‌دامنه یا مربوط به دامنه، و روابط بین آن‌ها تمرکز نمایید.

اعضای تیم باید فقط مفاهیم و عبارات خاص‌دامنه، و یا مربوط‌به‌دامنه را شناسایی کنند، در غیر این‌صورت، کلمات و عبارات بسیاری وارد مدل دامنه خواهندشد و آن را بی‌جهت بزرگ، پیچیده، و کم‌فایده خواهند نمود. این بدان معنی است که قبل از هر یک از قوانین فوق، می‌تواند عبارات «خاص‌دامنه» یا «مربوط به دامنه» به‌کار برود. به‌عنوان می‌توانیم بگوییم «افعال متعدی خاص‌دامنه» ، یا «اعداد و کمیت‌های مربوط به دامنه». بااین‌وجود، برای سادگی، این قوانین به شکل مختصرتر نوشته شده‌اند.

[۱] Containing
[۲] Containment

رهنمون ۵-۵

از مفاهیم مربوط به طراحی و پیاده‌سازی چشم‌پوشی کنید.

مفاهیم مربوط به طراحی و پیاده‌سازی، متعلق به مدل دامنه نیستند. این مفاهیم، برای طراحی و پیاده‌سازی سیستم نرم‌افزاری معرفی شده‌اند. واسط گرافیکی کاربر، مفاهیم مربوط به پایگاه‌داده، ثبت تراکنش‌ها[۱]، و لیست پیوندی مثال‌هایی از این مفاهیم هستند. یک راه برای تشخیص این‌که آیا این مفهوم متعلق به دامنه هست یا نه، پرسیدن این سوال است : «اگر هیچ سیستم کامپیوتری برای این کاربرد ساخته نشود، آیا این مفهوم همچنان در دامنه‌ی کاربرد باقی می‌ماند؟» اگر پاسخ مثبت باشد، این یک مفهوم متعلق به دامنه است، و در غیر اینصورت، خیر.

مثالی که در ادامه می‌آید از بالانویس[۲] استفاده می‌کند تا به شرح این‌که کدام‌یک از عبارت‌های بالا شناسایی شده‌اند، کمک کند. برای نمونه، کلمه‌ای که زیر آن خط کشیده شده و بالانویس ۱ دارد نشان می‌دهد که قانون شماره‌ی یک طوفان فکری استفاده شده، و یا یک عبارت اسمی شناسایی شده است. در بخش بعد، از بالانویس‌ها برای نشان دادن چگونگی دسته‌بندی عبارات شناخته‌شده به مفاهیم مدل‌سازی استفاده می‌شود.

مثال ۵-۱

با استفاده از شرح زیر برای سیستم مدیریت تحصیل در خارج از کشور (SAMS)، مفاهیم مهم کاربرد را شناسایی نمایید.

«هر دانشجوی کارشناسی یا دانشجوی مقطع تحصیلات‌تکمیلی می‌تواند برای حداکثر دو برنامه‌ی تبادل دانشجو در هر نیم‌سال تحصیلی درخواست بدهد. یک درخواست شامل یک فرم درخواست، دو توصیه نامه از اعضای هیئت‌علمی، و یک فرم تطبیق دروس است که توسط یک مشاور دانشگاهی تأیید شده باشد.

...

یک برنامه‌ی تبادل دانشجو دارای نام برنامه، نوع برنامه، گروه دانشگاهی، موضوع دانشگاهی، کشور، ناحیه، ترم تحصیلی، و زبان است.»

جواب: شکل ۵-۸ برای نشان دادن عبارت‌هایی که شناسایی شده‌اند، از کلماتی که زیر آن‌ها خط کشیده شده و دارای بالانویس هستند استفاده می‌کند.[۳]

۵-۴-۳- دسته‌بندی نتایج طوفان فکری

درسومین گام از مدل‌سازی دامنه، عبارت‌های فهرست شده، به کلاس‌ها، ویژگی‌ها، مقادیر ویژگی‌ها، و روابط دسته‌بندی می‌شوند. این کار با اِعمال قوانین دسته‌بندی که در شکل ۵-۹ نشان داده شده است، انجام می‌گردد. ستون سمت چپ شکل ۵-۹، عبارات و ستون سمت راست آن، مفاهیم مدل‌سازی شیءگرای مربوطه را نشان می‌دهد.

محصول نهایی این گام، فهرستی از کلاس‌ها و ویژگی‌های آن‌ها، و ارتباطات بین‌کلاس‌ها می‌باشد.

---

[۱] Transaction Log

[۲] Superscript

[۳] مترجم: با توجه به این‌که در ادامه این اصطلاحات به‌عنون نام کلاس انتخاب خواهند شد، از ترجمه شکل خودداری شده است.



شکل ۵-۸: نتیجه‌ی طوفان فکری از یک قطعه شرح SAMS

مثال ۵-۲

نتایج طوفان فکری در مثال ۵-۱ را دسته‌بندی کنید.

جواب: شکل ۵-۱۱ نتایج دسته‌بندی را نشان می‌دهد و به شرح زیر است:

- ستون ۱، عبارت‌های شناسایی‌نشده در گام طوفان فکری را به همراه بالانویس‌ها و یا عددهای شناسایی عبارات نشان می‌دهد.
- ستون ۲، کلاس‌های مربوطه و صفت‌های آن‌ها را به همراه روابط بین کلاس‌ها نشان می‌دهد. این موارد با استفاده از کدهای دسته‌بندی شکل ۵-۱۰ مشخص شده است.
- بنابر یک قرارداد صنعتی، حرف اول هر کلمه از نام یک کلاس با حروف بزرگ نوشته شده است.
- کلاس‌ها به همراه صفت‌هایشان نشان داده شده‌اند. برای مثال، صفت‌های کلاس (Exchange Program) در زیر نام آن آورده شده‌اند.
- در ستون ۲، یک رابطه‌ی دودویی با استفاده از نماد زیر نشان داده شده است:
  relation-name ‘(‘ class-name1,class-name2’)’

شماره قانون	عبارت شناسایی‌شده	مفهوم متناظر مدل‌سازی
۱	اسم / عبارت اسمی الف- به‌طور مستقل وجود دارد. ب- نقشی است که توسط یک شئ بازی می‌شود. ج- یک رابطه‌ی چندبه‌چند را شرح می‌دهد. د- یک تعمیم یا خاص‌سازی است. ه- در کاربرد / دامنه به‌طور مستقل وجود ندارد.	کلاس نقش در انجمن کلاس انجمنی سوپرکلاس/ زیرکلاس صفت یک کلاس
۲	عبارت «X از Y» الف- X به‌طور مستقل در دامنه / کاربرد وجود دارد. ب- X به‌طور مستقل در دامنه / کاربرد وجود ندارد. ج- X نقشی را نشان می‌دهد که توسط یک شئ بازی می‌شود.	X بخشی از Y است، یا Y تجمیعی از X است. X یک صفت از Y است. X یک نقش در یک انجمن است.
۳	فعل متعدی	رابطه‌ی انجمنی
۴	صفت / قید / شمارش	مقدار صفت
۵	عددی الف- مفهوم مربوطه یک صفت است. ب- مفهوم مربوطه یک شئ است.	مقدار صفت تعدد
۶	عبارت مالکیت (مثلاً Y دارای X است.) الف- X به‌طور مستقل در دامنه / کاربرد وجود دارد. ب- درغیر این‌صورت	X یک تجمیع از Y است. X یک صفت از Y است.
۷	عبارت« تشکیل‌می‌شوداز / بخشی است از / شامل می‌شود بر»	رابطه‌ی تجمیع
۸	عبارت «دربردارد» الف- اشیای داخلی می‌توانند بدون تاثیر بر جامعیت شئ دربردارنده، حذف شوند. ب- درغیر این صورت	انجمن تجمیع
۹	«X، Y است» یا عبارت تعمیم‌دادن / خاص‌کردن	ارث‌بری

توجه: ۱) X یک صفت از Y است، اگر X به‌طورمستقل در دامنه وجود نداشته باشد.

۲) X و Y از طریق یک رابطه‌ی ارث‌بری، تجمیع، یا انجمنی در رابطه‌اند، اگر هر دو X و Y در کاربرد به‌طور مستقل وجود داشته باشند.

شکل ۵-۹: قوانین دسته‌بندی برای دسته‌بندی نتایج طوفان فکری

(A)	صفت (از یک کلاس)
(AC)	کلاس انجمنی (از یک انجمن)
(AG)	تجمیع
(AS)	انجمن
(C)	کلاس، ممکن است زیرکلاس کلاس دیگری باشد.
(I)	رابطه ارث‌بری
(m,n)	تعدد هر کلاس در یک انجمن دوطرفه
(r1,r2)	نام نقش هر کلاس در یک انجمن دوطرفه
(V)	مقدار صفت (یک صفت از یک کلاس)

شکل ۵-۱۰: کدهای دسته‌بندی

['(' [m],[n]')'] ['([role-name1] , [role-name2])']

که در آن m و n عبارت‌های تعدد، و role-name1 و role-name2 ، نام‌های نقش دو کلاس شرکت‌کننده می‌باشند. این نماد می‌تواند به‌همین‌ترتیب برای نمایش رابطه‌های nتایی بسط داده شود. مرتبه‌ی تعدد و شرح نقش اهمیتی ندارد. شکل ۵-۴ راه‌های مختلفی برای بیان یک تعدد مفروض را نشان می‌دهد.

- تعدد و نقش، با استفاده از نماد فوق، بعد از رابطه آورده شده است. برای مثال، انجمن یا رابطه‌ی Part-of(Faculty Recommendation Letter, Application)، دارای تعدد (۲،۱) می‌باشد که در ستون دوم، مقابل مقدار two از ستون اول نوشته شده است. این بدان معنی است که یک درخواست به دو توصیه‌نامه از اعضای هیئت‌علمی نیاز دارد، که از عدد ۲ درستون ۱ نتیجه گرفته می‌شود.

  به‌عنوان یک مثال دیگر، رابطه‌ی انجمنی apply-to (Student, Exchange Program)، درسطر بعد از رابطه، دارای تعدد (۲..۰،۱) می‌باشد. این بدان علت است که هر دانشجو می‌تواند در هر ترم به «کمتر یا برابر دو» برنامه‌ی تبادل دانشجو درخواست بفرستد.
- یک رابطه بدون ذکر اندازه‌ی تعدد، به‌طور پیش‌فرض دارای تعدد (۱،۱) خواهد بود. به‌عنوان مثال، تعدد دو رابطه‌ی انجمنی دیگر ذکر نشده است. این بدان معنی است که هردوی آن‌ها دارای تعدد پیش‌فرض (۱،۱) می‌باشند.
- کلاس‌های انجمنی، به‌همراه رابطه‌های انجمنی خود که در یک زوج پرانتز محصور می‌باشند، نشان داده شده‌اند. به‌عنوان مثال، Application با رابطه‌ی مربوط به خود، (apply-to)، دیده می‌شود.
- برای کمک به فهم این‌که کدام قانون دسته‌بندی مورد استفاده قرار گرفته است، شماره‌ی قانون در ستون سوم آمده است.

دانشجو ممکن است بپرسد «چرا گروه دانشگاهی و کشور، صفت هستند نه کلاس؟» این بدان سبب است که اگرچه یک کشور در حالت عمومی وجود مستقلی دارد و یک گروه دانشگاهی در دانشگاه متبوع خود دارای وجود مستقل است، ولی این‌دو در دامنه/کاربرد SAMS وجود مستقلی ندارند.

یافتن اطلاعات ناقص

در این گام، تیم توسعه نتیجه‌ی دسته‌بندی را امتحان می‌کند و اطلاعات ناقص آن را با استفاده از استدلال، حس عمومی، دانش دامنه که از جاهای دیگر به‌دست‌آمده، و تجربیات قبلی شناسایی می‌نماید. اطلاعات ناقص شامل موارد زیر است:

۱. کلاس‌های جاافتاده، که می‌تواند ازکلاس‌های موجود و روابط آن‌ها استنباط گردد. برای مثال، ازکلاس‌های Undergraduate Student و Graduate Student می‌توان کلاس Student را به‌عنوان یک کلاس مافوق استنباط کرد، زیرا یک دانشجو می‌تواند برای یک برنامه‌ی تبادل دانشجو درخواست بدهد.
۲. ویژگی‌ها و انواع جاافتاده، که می‌تواند از یک کلاس یا کلاس انجمنی که هیچ صفتی ندارد، شناسایی شود. در مثال ۵-۲، Undergraduate Student و Graduate Student دارای هیچ صفت مشترکی نیستند. صفت‌ها و انواع جاافتاده می‌تواند از فرم‌های کسب‌وکار، مانند فرم‌های درخواست، صورت‌حساب‌ها، فرم‌های سفارش، و فرم‌های رزرواسیون شناسایی گردند. برای مثال، صفت‌های Undergraduate Student و Graduate Student می‌توانند از فرم‌های ثبت‌نام دانشجو گرفته شوند.
۳. رابطه‌های جاافتاده، که می‌توانند از کلاس‌های منفرد[1]، یعنی کلاس‌هایی که با هیچ‌یک از کلاس‌های دیگر رابطه‌ای ندارند، به‌دست بیاید. در مثال ۵-۲، Faculty با هیچ کلاس دیگری در رابطه نیست. با توجه به شرح مسئله، Faculty توصیه‌نامه‌ها را فراهم خواهد کرد، بنابراین، یک رابطه‌ی انجمنی می‌تواند شناسایی گردد.

[1] Isolated

Brainstorming List	Classification Result	Rule
undergraduate student[1]	(C) Undergraduate Student	1 (a)
graduate student[1]	(C) Graduate Student	1 (a)
apply to[3]	(AS) apply to (Student, Exchange Program)	3
no more than two[5]	(1, 0..2)	5
exchange programs[1]	(C) Exchange Programs	1 (a)
has[6]		6 (b)
program name[1]	(A) program name	1 (e)
program type[1]	(A) program type	1 (e)
academic department[1]	(A) academic department	1 (e)
academic subject[1]	(A) academic subject	1 (e)
country[1]	(A) country	1 (e)
region[1]	(A) region	1 (e)
term of study[1]	(A) term of study	1 (e)
language[1]	(A) language	1 (e)
semester[1]	(A) Semester // same as term of study	1 (e)
application[1]	(AC) Application (of apply to)	1 (c)
consists of[7]	(AG) Part-of (Application Form, Application)	7
	(AG) Part-of (Faculty Recommendation Letter, Application)	7
two[5]	(2, 1)	5
	(AG) Part-of (Course Equivalency Form, Application)	7
application form[1]	(C) Application Form	1 (a)
faculty[1]	(C) Faculty	1 (a)
faculty recommendation letter[1]	(C) Faculty# Recommendation Letter	1 (a)
course equivalency form[1]	(C) Course Equivalency Form	1 (a)
approved by[3]	(AS) approved by (Course Equivalency Form, Academic Advisor)	3
academic advisor[1]	(C) Academic Advisor	1 (a)
	(C) Student // added	
	(I) ISA (Undergraduate Student, Student) // added	
	(I) ISA (Graduate Student, Student) // added	

شکل ۵-۱۱: دسته‌بندی مفاهیم طوفان فکری برای SAMS

۴. تعددهای جاافتاده. در برخی از موارد اطلاعات تعدد مهم هستند، زیرا قوانین دامنه و محدودیت‌های خاص کاربر را مشخص می‌نمایند. برای مثال، هر درخواست باید دو معرفی‌نامه را شامل گردد. برای شناسایی این موارد، هریک از رابطه‌های انجمنی و... به‌دقت بررسی می‌گردند تا اطمینان حاصل شود که تعدد مهمی جانیفتاده است.

۵-۴-۴- تصویرسازی از مدل دامنه

در این گام، نتیجه‌ی کلاس‌بندی، با استفاده از یک نمودار کلاس، تصویرسازی می‌شود. وقتی کلاس‌ها، صفت‌ها، و روابط، مانند آن‌چه در شکل ۵-۱۱ می‌بینید، شناسایی و مستندسازی شدند، تبدیل نتیجه به یک نمونه کلاس کار ساده‌ای خواهد بود. شکل ۵-۱۲ قوانین تبدیل و مثال‌هایی از بخش‌های قبل را نشان می‌دهد. از آنجا که این تبدیل کاری جزیی است، انجام آن می‌تواند به دستیاران یا افراد دیگر سپرده شود تا هزینه‌ها کاهش یابد.

مثال ۵-۳

نتیجه‌ی کلاس‌بندی به‌دست‌آمده از مثال ۵-۲ را به یک نمودار کلاس تبدیل کنید.

جواب: شکل ۵-۱۳ نمودار کلاس به دست آمده را نشان می‌دهد.

مدل دامنه و نمودار کلاس

مدل دامنه با یک نمودار کلاس نشان داده می‌شود که در آن کلاس‌ها هیچ عملیاتی را نشان نمی‌دهند. به این ترتیب، نمودار کلاس، یک مدل ایستا نامیده می‌شود. این بدان علت است که یک مدل دامنه می‌تواند به صورت تعدادی اعلان عمومی دیده شود که جملاتی در مورد کلاس‌ها و روابط آن‌ها می‌باشند که اگر براساس قوانین دامنه‌ی کاربرد ارزیابی گردند، در همه‌ی حالت‌های آن صحیح هستند.

برای مثال، «مشتری دارای نام و آدرس است» ، و «مشتری وسیله‌ی نقلیه را رزرو می‌کند»، در یک دامنه‌ی کاربردی اجاره‌ی خودرو صحیح هستند. در مقابل، «اعلان‌های خاص» جملاتی در مورد نمونه‌های خاصی از کلاس‌ها و روابط آن‌ها می‌باشند. برای مثال، «مشتری C1 دارای نام 'دیوید کونگ' و آدرس 'تگزاس آرلینگتون، خیابان یاتس، شماره‌ی ۴۱۶' می‌باشد»، و «مشتری C1 وسیله‌ی نقلیه V1 را رزرو می‌کند»، اعلان‌های خاص می‌باشند. مدل‌هایی که اعلان‌های خاص را مشخص می‌کنند، نمونه‌هایی از مدل‌هایی هستند که اعلان‌های عمومی را مشخص می‌کنند. برای مثال، یک نمودار شئ، نمونه‌ای از یک نمودار کلاس است. در یک نمودار شئ، مقادیر صفت‌ها و روابط واقعی بین اشیا نشان داده می‌شود. مطالعه‌ی نمودار شئ، فراتر از این کتاب است.

اگرچه یک نمودار کلاس می‌تواند عملیات یک کلاس را نشان دهد، یک مدل دامنه نباید این موارد را نمایش دهد. این بدان دلیل است که تخصیص مسئولیت‌ها یا توابع عضویت به کلاس‌ها، یک تصمیم طراحی است که باید در زمان طراحی گرفته شود. تخصیص عملیات به کلاس‌ها در مدل‌سازی دامنه، بدون درنظرگرفتن چگونگی تعامل اشیای نرم‌افزاری، یک تصمیم عجولانه است. بنابراین، خودداری‌کردن از تخصیص عملیات به کلاس‌ها در این گام، دارای اهمیت است.

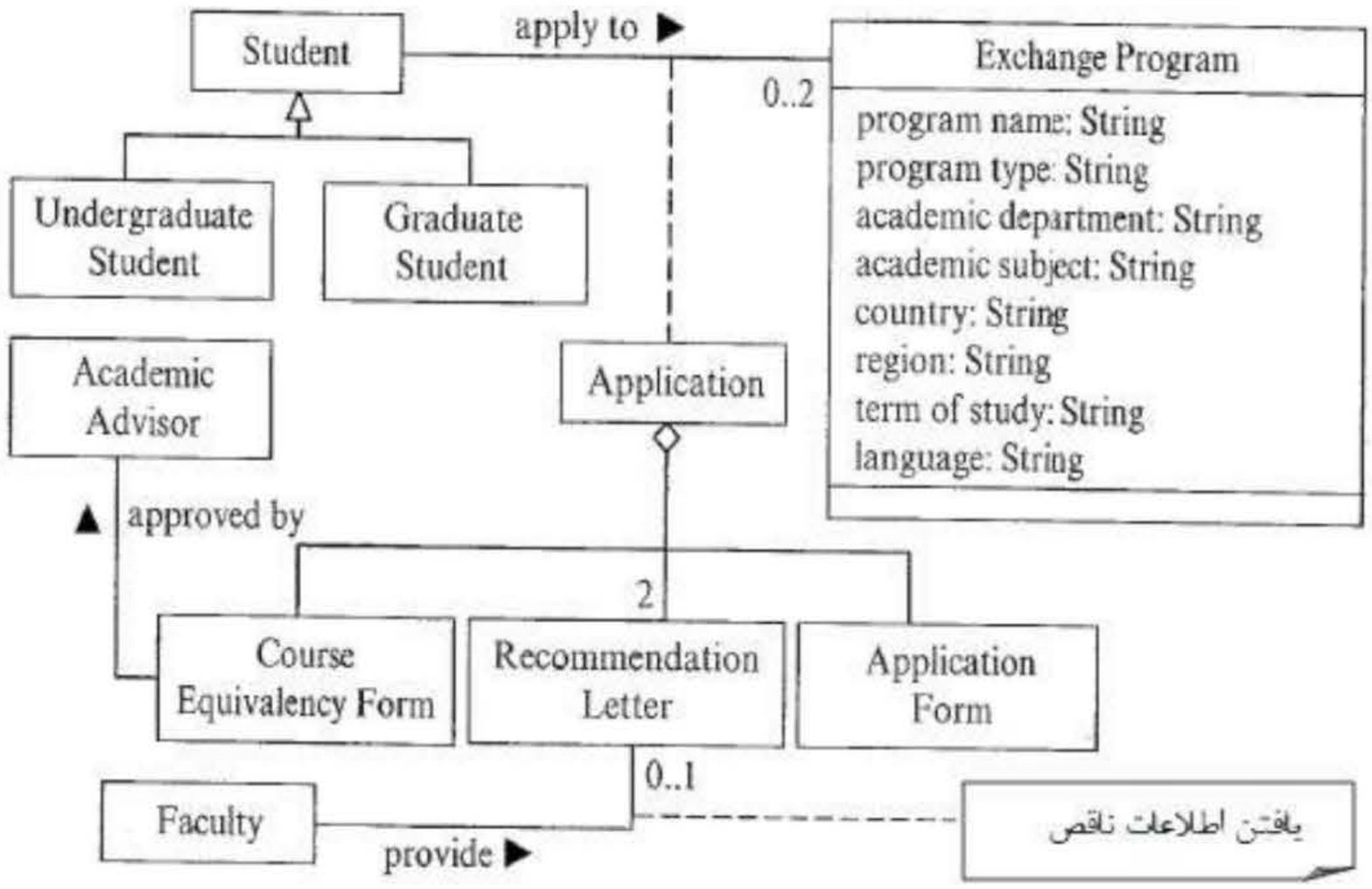
استفاده از اجتماع

یک اشتباه بسیار رایج در زمان استفاده از اجتماع، شناسایی نادرست کلاس‌ها به‌عنوان انواعی از صفت می‌باشد. برای مثال Engine، Transmission و Brake کلاس هستند، ولی در شکل ۵-۱۴-الف به‌عنوان انواعی از ویژگی برای کلاس Car مورداستفاده قرار گرفته‌اند. همان‌طورکه در رهنمون ۵-۱ گفته شد، یک شئ وجود مستقلی در کاربرد یا دامنه‌ی آن دارد، ولی در مورد یک صفت این‌گونه نیست. بنابراین، تلقی‌نمودن یک شئ به‌عنوان یک مقدار صفت از شئ دیگر، یک پارادوکس را ایجاد می‌نماید. این پارادوکس از این قرار است که چیزی وجود مستقل دارد و هم‌زمان وجود ندارد، که این غیرممکن است.

Classification Result	UML Class Diagram Representation
(C) Class Name (A) attr 1: type 1 (A) attr 2: type 2	Class Name attr 1: type 1 attr 2: type 2
Example: (C) Exchange Programs (A) program name: String (A) program type: String (A) academic department: String (A) academic subject: String (A) country: String (A) region: String (A) term of study: String (A) language: String	Exchange Program program name: String program type: String academic department: String academic subject: String country: String region: String term of study: String language: String
(AS) transitive-verb (Class 1, Class 2) (m, n) (role 1, role 2) (AC) Class 3 (transitive verb)	transitive-verb Class 1 — m — n — Class 2 role 1 / role 2 Class 3
Example: (AS) apply to (Student, Exchange Program) (1,0..2) (AC) Application (apply to)	Student — apply to ▶ — 0..2 — Exchange Program Application
Part-of (Class 1, Class 2) (m, n) (role 1, role 2)	Class 1 — role 1 / m — role 2 / n — Class 2
Example: Part-of (Faculty Recommendation Letter, Application) (2, 1)	Faculty Recommendation Letter 2 Application
ISA (Class 1, Class 2)	Class 1 → Class 2
Example: ISA (Undergraduate Student, Student)	Undergraduate Student → Student

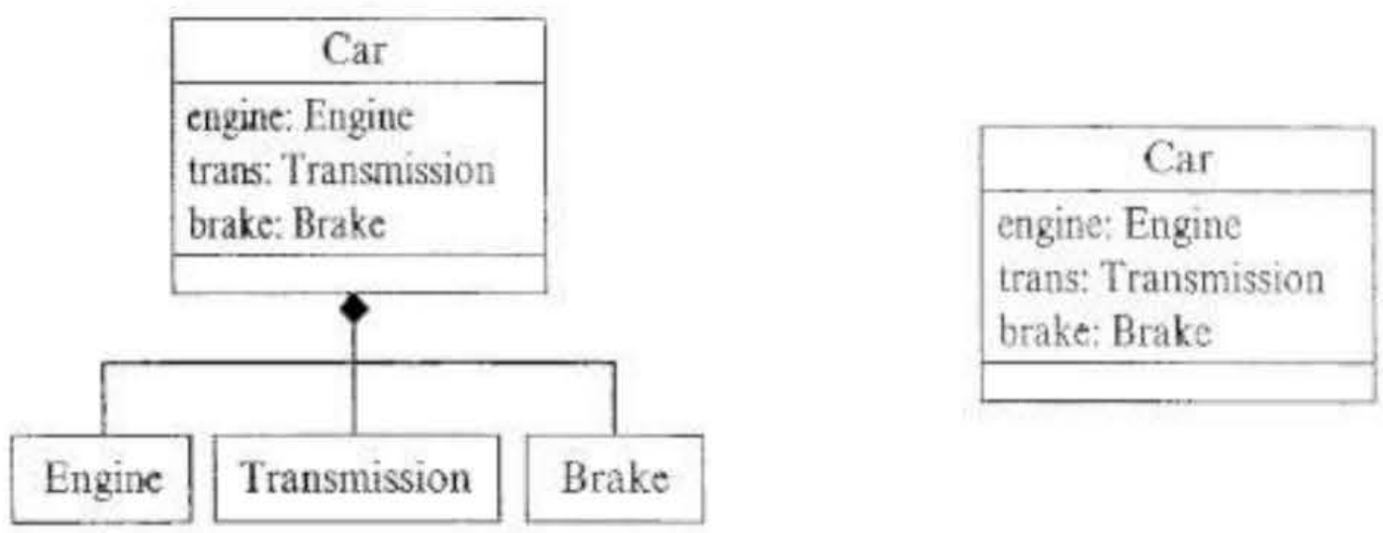
شکل ۵-۱۲: قوانین تبدیل: از نتیجه‌ی کلاس‌بندی‌شده به نمودار کلاس

شکل ۵-۵-الف مدل‌سازی کلاس Car، تشکیل‌شده از یک Engine، یک Transmission، و یک Brake را نشان می‌دهد. حس عمومی ما این شکل مدل‌سازی را صحیح می‌داند. در شکل ۵-۱۴-الف، گفته شده است که یک شئ Engine، یک شئ Transmission، و یک شئ Brake، خصوصیات یا صفت‌های یک شئ Car هستند. این صحیح نیست، زیرا نمونه‌هایی از Transmission،Engine ، Brake شئ هستند و نه صفت. قسمت صفت از نماد کلاس باید فقط صفت‌ها را نشان بدهد و نه اشیا را. اگر کسب‌وکار مورد نظر، اجاره‌ی خودرو یا فروش آن باشد که نیازمند نگهداری اطلاعاتی در مورد خودروهای مختلف است (مانند نوع موتور (دیزلی یا بنزینی)، جعبه دنده (دستی یا اتوماتیک)، و ترمز (استوانه‌ای یا صفحه‌ای))، دراین‌صورت



شکل ۵-۱۳: تبدیل نتیجه‌ی کلاس‌بندی به یک نمودار کلاس

انواع این صفت‌ها باید به‌صورت رشته‌های متنی و نه کلاس‌ها، ذکر شوند. اگر کاربرد مورد نظر، تولیدکننده‌ی خودرو و یا فروشگاه لوازم یدکی آن باشد که با اشیایی همچون موتور، جعبه دنده، و ترمز سروکار دارد، دراین‌صورت باید از نموداری مشابه شکل ۵-۵-الف استفاده شود.

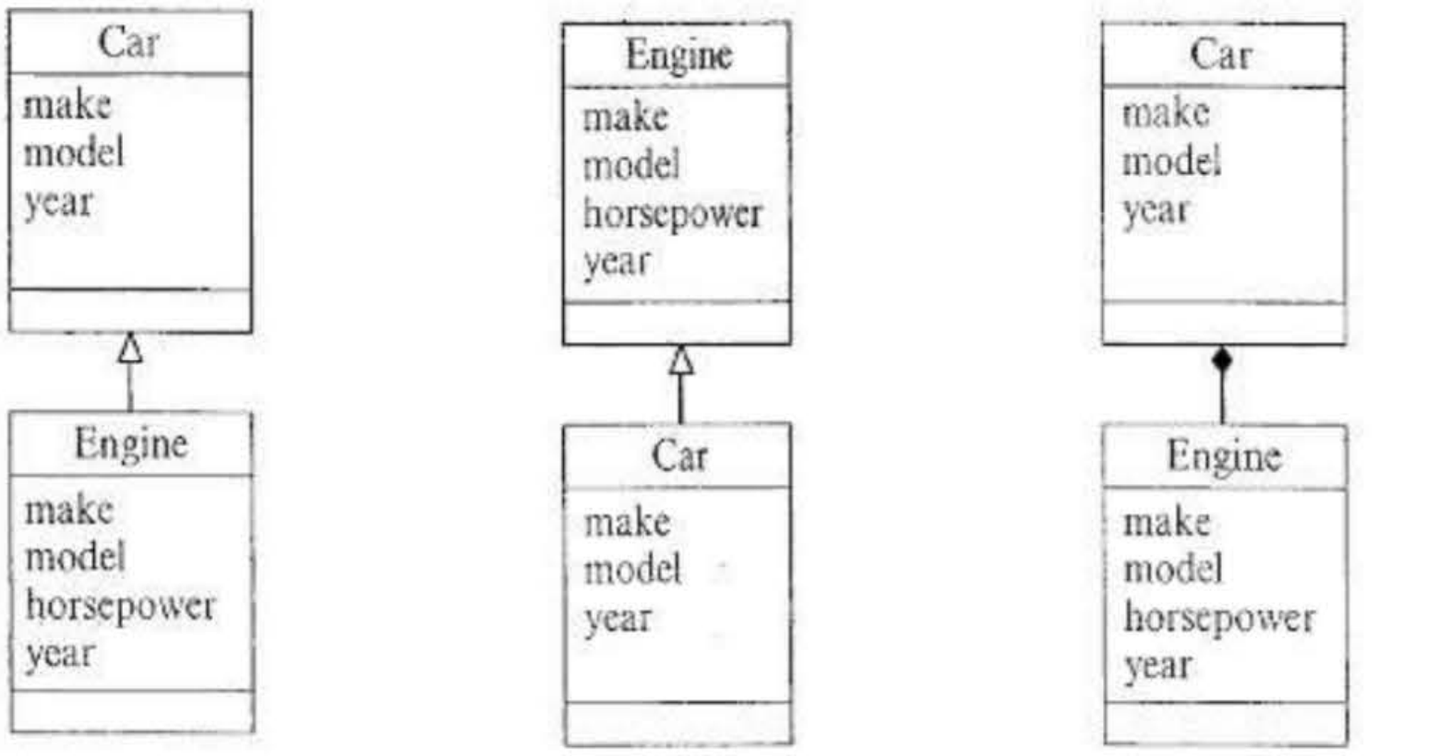


الف) نادرست: اشیای موتور، میل‌لنگ و ترمز صفت‌های ماشین هستند.

ب) نادرست: موتور، میل‌لنگ و ترمز هم به‌عنوان اجزای ماشین و هم به‌عنوان صفت‌های آن هستند.

شکل ۵-۱۴: کاربردهای نادرست اجتماع

شکل ۵-۱۴-ب، Engine، Transmission، و Brake را هم به‌عنوان کلاس و هم به‌عنوان صفت نشان می‌دهد. همان‌طور که قبلاً گفته شد، این یک پارادوکس است. دو عامل می‌تواند منجر به کاربردهای نادرست شکل ۵-۱۴ شود. اول این‌که یک قطعه از یک شئ، مانند موتور یک ماشین، معمولاً در برنامه‌نویسی شئ‌گرا به‌عنوان یک فیلد یا صفت از کلاس Car پیاده‌سازی



الف) موتور یک جزء انحصاری ماشین است - درست

ب) ماشین، موتور را به ارث می‌برد – نادرست

ج) موتور، ماشین را به ارث می‌برد - نادرست

شکل ۵-۱۵: توصیف نادرست روابط

می‌شود. دوم این‌که برخی از ابزارهای IDE، مدل‌سازی و تولید کد را جابه‌جا و یا حتی برابر در نظر می‌گیرند و روابط اجتماع را مشابه شکل ۵-۱۴-ب تولید می‌کنند. بدتر اینکه در برخی از موارد، تنها راهی که میتوان از ابزار IDE استفاده کرد، همان راه نادرست نمایش روابط اجتماع است.

استفاده از ارث‌بری

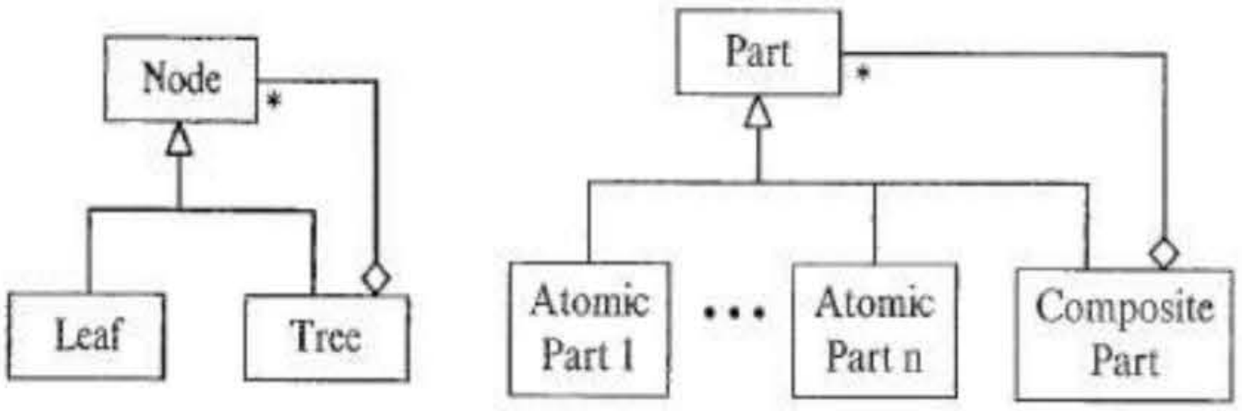
روابط ارث‌بری (و روابط اجتماع) گاهی به‌اشتباه به‌جای یکدیگر به‌کار برده می‌شوند. شکل ۵-۱۵-الف، یک تصویرسازی درست را نشان می‌دهد. در این تصویر، یک خودرو فقط از یک موتور تشکیل‌شده‌است. به علاوه، نمودار نشان می‌دهد که Car و Engine صفت‌های مشابهی همچون make، model، year دارند. این واقعیت که توان یک ماشین بر حسب اسب بخار، به توان موتور آن بستگی دارد باعث شده است که بسیاری از مدل‌سازان از رابطه‌ی ارث‌بری استفاده کنند (شکل ۵-۱۵-ب). این کار حتماً نادرست است، زیرا یک خودرو، موتور نیست. هیچ رابطه‌ی تعمیم یا خاص‌سازی بین این دو مفهوم وجود ندارد. نمودار شکل ۵-۱۵-ج نیز نادرست است، زیرا Engine، یک Car نیست.

توضیحاتی که داده شد، رهنمون زیر را مطرح می‌نماید:

رهنمون ۵-۶

فقط در صورتی از ارث‌بری استفاده کنید که هر نمونه از زیرکلاس، یک نمونه از کلاس مافوق هم باشد. این راهبرد، ISA نامیده می‌شود.

در شکل‌های ۵-۱۵-ب و ۵-۱۵-ج، ارث‌بری بین Engine و Car مورد استفاده قرارگرفته است. ولی نمونه‌های Engine، نمونه‌هایی از Car نیستند و بالعکس. بنابراین در اینجا نباید از ارث‌بری استفاده‌گردد. گاهی راهبرد ISA باید در یک زمینه‌ی عمومی به‌کار گرفته شود. برای مثال، یک شرکت تولیدی را در نظر بگیرید که قطعات در آن بتوانند به دو دسته‌ی قطعات ساده



الف) یک قطعه مرکب، قطعه‌ای است که شامل قطعات دیگر می‌شود.

ب) یک درخت، نودی است که خود ترکیبی از چند نود است.

شکل ۵-۱۶: قطعه و قطعه‌ی مرکب

و قطعات مرکب تقسیم شوند. قطعات ساده، شامل قطعات دیگر نیستند، ولی یک قطعه‌ی مرکب می‌تواند از قطعات ساده یا قطعات مرکب دیگر تشکیل گردد. همان‌طور که در شکل ۵-۱۶-الف نشان داده شده است، این وضعیت می‌تواند با استفاده از الگوی ترکیبی[1] مدل‌سازی شود. در مثال شکل ۵-۱۶-ب، یک Tree، یک Node و یا ترکیبی از چند Node است. الگوی ترکیبی به‌طور کامل در فصل ۱۶ شرح داده خواهد شد. راهبرد ISA در این زمینه‌های تعمیم‌یافته مورد استفاده قرار می‌گیرد.

رهنمون ۵-۷

فقط درصورتی از ارث‌بری استفاده کنید که همه‌ی روابط کلاس مافوق، روابط زیرکلاس هم باشد. این راهبرد، هم‌نوایی (Conformance) نامیده می‌شود.

از آنجا که هر نمونه از زیرکلاس باید یک نمونه از کلاس‌مافوق باشد، روابط کلاس مافوق نیز باید روابط زیرکلاس باشد. برای مثال، Student به Department تعلق دارد، یعنی Belong-to(Student, department). Undergraduate Student و Graduate Student هم Student هستند. یعنی Belong-to(Undergraduate Student, Department) و Belong-to(Graduate Student, Department).

حال شکل ۵-۱۷-الف را در نظر بگیرید که نشان می‌دهد «پیمانکاران» ، «کارمند» هستند. بنابر رهنمون ۵-۷، چون «کارمندان» سود دریافت می‌کنند، «پیمانکاران» هم سود دریافت می‌کنند. ولی برای بسیاری از شرکت‌ها، این امر صدق نمی‌کند. بنابراین، Contractor نباید زیر کلاس Employee باشد. بااین‌حال، برای بسیاری از شرکت‌های دیگر، یک Contractor، یک Employee هم هست. برای پوشش دادن این دو حالت، در شکل ۵-۱۷-ب، یک زیرکلاس دیگر از Employee به‌نام Full Time برای دریافت‌کردن Benefits تعریف شده است.

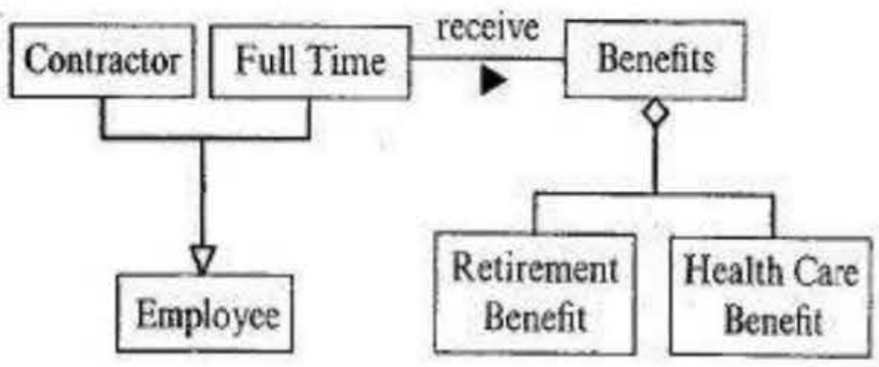
۵-۴-۵- فهرست مروری مدل دامنه

تیم توسعه باید مدل دامنه را مرور کند تا خطاها و موارد غیرعادی را شناسایی و تصحیح نماید. در ادامه، یک فهرست مروری برای مدل دامنه ارائه می‌گردد:

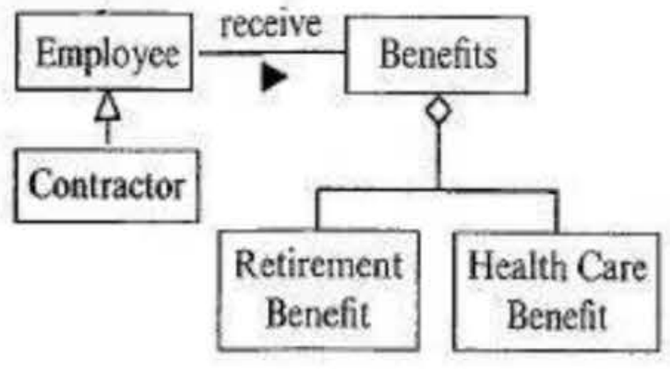
۱. آیا مدل دامنه، بیشتر کلاس‌های مهم دامنه‌ی کاربرد را دربردارد؟

---

[1] Composite Pattern



ب) تنها کارکنان تمام وقت، سود دریافت می‌کنند.



الف) اگر پیمانکاران سود دریافت نکنند، نادرست است.

شکل ۵-۱۷: زیرکلاس، روابط را از کلاس مافوق به ارث می‌برد.

۲. آیا مدل دامنه، همه‌ی روابط مهم دامنه‌ی کاربرد را نشان می‌دهد؟
۳. آیا هر رابطه در مدل دامنه، به‌درستی یک رابطه‌ی متناظر در جهان واقع را نشان می‌دهد؟
۴. آیا قیود تعدد مهمی وجود دارند که از قلم افتاده باشند و یا به غلط توصیف شده باشند؟
۵. آیا مدل دامنه شامل یک یا چند کلاس طراحی یا پیاده‌سازی است؟

توجه: هدف از ساختن مدل دامنه، کمک به تیم توسعه برای فهم دامنه‌ی کاربرد است. مدل دامنه، یک سند طراحی یا پیاده‌سازی نیست. بنابراین مدل دامنه باید تنها شامل کلاس‌های دامنه‌ی کاربرد باشد، نه کلاس‌های طراحی و پیاده‌سازی. اضافه کردن کلاس‌های واسط گرافیکی کاربر، پایگاه‌داده‌ها و کلاس‌های دسترسی به آن‌ها، الگوهای طراحی (به‌جز تعداد اندکی مانند الگوی ترکیبی)، سوابق تراکنش‌ها، و مانند آن به مدل دامنه نادرست است.

۶. آیا هر یک از کلاس‌های مدل دامنه، همه‌ی صفت‌های مهم را نشان می‌دهند؟

توجه: صفت‌های یک کلاس، اشیای کلاس را توصیف یا مشخص می‌کنند، و یا اطلاعات حالت اشیا را ذخیره می‌نمایند. وجود کلاس‌های بدون صفت به وضوح نشان می‌دهد که صفت هایی از قلم افتاده است، و یا کلاس‌ها ضروری نیستند.

۷. آیا نام‌گذاری هر کلاس، صفت، و یا رابطه، مناسب و قابل فهم است؟
۸. آیا کلاس‌هایی وجود دارند که به‌عنوان نوع یک صفت مورد استفاده قرار گرفته باشند؟

توجه: در نمودار کلاس UML، استفاده از کلاس‌ها یا اشیا به‌عنوان صفت، نادرست است.

۹. آیا هیچ یک از کلاس‌ها عملیاتی را نشان می‌دهند؟

توجه: تخصیص مسئولیت به کلاس‌ها در گام مدل‌سازی دامنه، یک کار عجولانه است. تخصیص مسئولیت، مربوط به مرحله‌ی طراحی است و باید در خلال طراحی شیءگرا انجام گردد.

۵-۵ جمع‌بندی

در این بخش، گام‌های مدل‌سازی دامنه برای یک کاربرد سیستم اجاره‌ی خودرو (CRS) انجام شده است. توصیف این کسب و کار در شکل ۵-۱۸ نشان داده شده و پس از طوفان فکری، زیر عبارات مهم آن خط کشیده شده است. در گام کلاس‌بندی، عبارات با استفاده از قوانین کلاس‌بندی (شکل ۵-۹)، کلاس‌بندی شده‌اند. نتیجه‌ی کلاس‌بندی در شکل ۵-۱۹ نشان داده شده است.

کلاس‌ها، صفت‌ها، و روابط اضافی (مانند حالت بدنه، تعداد درها، و وضعیت) در شکل‌ها نشان داده شده است. این صفت‌ها از مقادیر شناخته‌شده‌ی صفت ، مانند «سدان» یا «هاچ بک» ،«دو در» یا «چهار در»، و مقادیر وضعیت خودرو مانند

Description of Car Rental Business

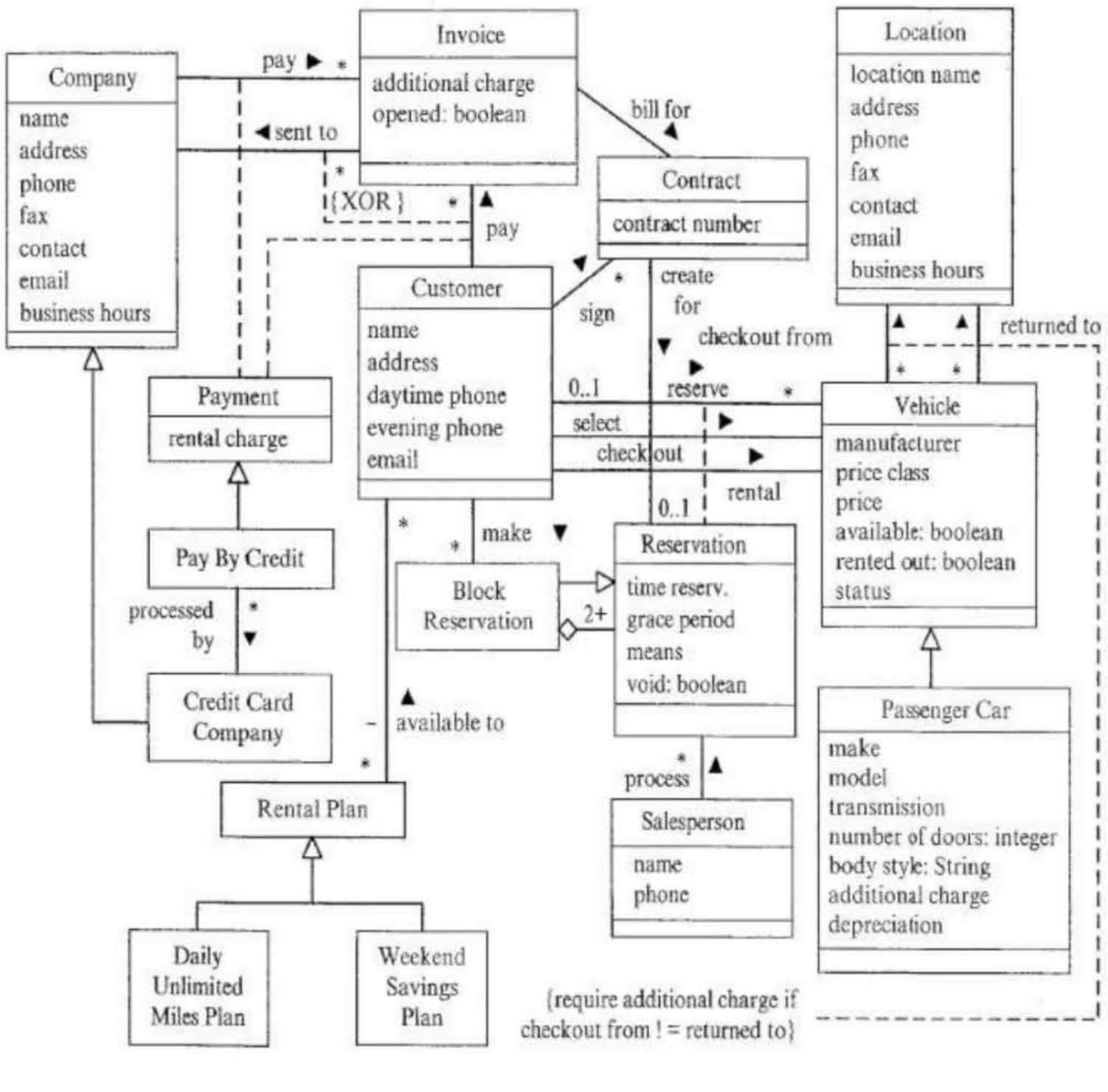
(1) Vehicles[1] can be taken from[3] one location[1] and returned to[3] the same
(2) location[1] or a different location with an additional charge[1]. Although
(3) the company is, at present, concerned only with passenger cars[1],
(4) it may branch out into other forms of vehicle rental[1] in the future
(5) and would like to be able to use the same reservation system.
(6) The company has several different makes of car[2] in its rental fleet,
(7) from different manufacturers[1]. Each make may have several models. For
(8) example, Toyota has Corolla, Camry, etc. The models are grouped
(9) into a small number of price classes[1]. The customer[1] must be able to
(10) select[3] the make and the model he/she wants to rent. If the selected
(11) car is not available[4], the system must display a message telling
(12) the customer that the car is rented out[4] and let the customer
(13) select another make and model or have the system suggest
(14) similar models of a different make. The company has a number
(15) of different rental plans[1] available to customers. For example,
(16) there are "daily unlimited miles plan[1]" and "weekend savings plan[1]."
(17) The company finds it important to have information available on
(18) the models of car[2], automatic[4] or manual[4] transmission[2], two[5]
(19) or four[5] doors, and sedan[1] or hatchback[1]. The rental prices[2] may be
(20) different for different options[2] and a customer will want to know
(21) such information when reserving[3] a car. Currently, customers make[3]
(22) reservations[1] directly with the car rental company either in person[4]
(23) or by phone[4]. The salespersons[1] process[3] the reservations[1]
(24) manually using a reservation form[1] and archive[3] them in the file
(25) cabinet. No deposit is required at the time of reservation[2]. The
(26) reservation is voided[4] if the customer does not show up to sign[3] the
(27) contract[1] for more than a given period of time[2]. Such reservation is
(28) honored only if there are still cars available[4] to satisfy request.
(29) Sometimes a customer wishes to make[3] a block reservation[1] for several[5]
(30) cars and to have the invoices[1] for all rentals on the reservation
(31) handled together. As soon as a car[1] is checked out[3] to a customer,
(32) an invoice is opened[4]. A single invoice[1] may cover[3] one or more[5]
(33) rentals[1]. Normally a customer[1] will pay[3] the invoice[1] when the car is
(34) returned[3] but, in some cases, the invoice[1] may be sent to[3] a company[1]
(35) (such as the customer's employer). When the customer[1] pays[3] by a
(36) credit card[4], the rental charge[1] will be processed[3] through a credit
(37) card processing company[1]. A car may or may not be available[4] for
(38) rental on a given day. Rental cars need frequent preventive
(39) maintenance and, in addition, any damage to a car has to be repaired
(40) as soon as possible. The company wants to keep track of the rental car
(41) purchase[4], repair[4], maintenance[4], and disposal[4] information for
(42) business and tax purposes (for example, depreciation of the rental
(43) cars[2]).

شکل ۵-۱۸: توصیف جزیی CRS، با عبارات خط‌کشی شده

«تعمیری» و مانند آن گرفته شده است. سپس همان‌طور که در شکل ۵-۲۰ دیده می‌شود، نتیجه‌ی کلاس‌بندی به یک نمودار کلاس تبدیل می‌شود و اطلاعات از قلم افتاده اضافه می‌گردد. بسیار طبیعی است که تحلیل‌گران مختلف، نتایج کار متفاوتی داشته باشند. کلاس‌ها، صفت‌ها، روابط، و کلاس‌های انجمنی ممکن است متفاوت باشند. این مسئله، تفاوت‌های موجود در احساس و ادراک تحلیل‌گر از دامنه‌ی کاربرد را نشان می‌دهد که امری غیرقابل‌اجتناب است. بخشی از این تفاوت، به‌دلیل

Brainstorming List	Classification Result	Rule
vehicle[1]	(C) Vehicle	1(a)
manufacturer[1]	(A) manufacturer	1(e)
price class[1]	(A) price class	1(e)
rental price[1]	(A) price	1(e)
available[4], not available[4]	(A) available: boolean	4
rented out[4]	(A) rented out: boolean	4
	(A) status	4
purchase[4]	(V) purchase	4
repair[4]	(V) repair	4
maintenance[4]	(V) maintenance	4
disposal[4]	(V) disposed	4
car[1], passenger car[1]	(C) Passenger Car	1(a)
makes of car[2]	(A) make	2(b)
model of car[2]	(A) model	2(b)
transmission[1]	(A) transmission	1(e)
automatic[4]	(V) automatic	4
manual[4]	(V) manual	4
	(A) number of doors: integer	5(a)
two[5] or four[5] doors[1]	(V) 2, 4	5(a)
	(A) body style: String	1(e)
sedan[1]	(V) "sedan"	1(e)
hatchback[1]	(V) "hatchback"	1(e)
options[1]	(same as transmission, # doors, body style)	
additional charge[1]	(A) additional charge	1(e)
depreciation of the rental cars[2]	(A) depreciation	2(b)
location[1]	(C) Location	1(a)
taken from[3]	(AS) taken from (Vehicle, Location)	3
returned to[3]	AS) returned to (Vehicle, Location)	3
other forms of vehicle[1]	(TBD)	
customer[1]	(C) Customer	1(a)
select[3]	(AS) select (Customer, Vehicle)	3
rental plan[1]	(C) Rental Plan	1(d)
daily unlimited miles plan[1]	(C) Daily unlimited Miles Plan	1(d)
weekend savings plan[1]	(C) Weekend Savings Plan	1(d)
reserving[3]	(AS) reserve (Customer, Vehicle)	3
reservation[1]	(AC) Reservation (reserve)	1(c)
time of reservation[2]	(A) time of reservation	2(b)
period of time[2]	(A) grace period	2(b)
	(A) means	4
in person[4]	(V) in person	4
by phone[4]	(V) by phone	4
voided[4]	(A) void: boolean	4
salesperson[1]	(C) Salesperson	1(a)
process[3]	(AS) process (Salesperson, Reservation)	3
archive[3]	(AS) archive (Salesperson, Reservation)	3
reservation form[1]	(same as Reservation)	
file cabinet[1] (not used)		
sign[3]	(AS) sign (Customer, Contract)	3
contract[1]	(C) Contract	1(a)
block reservation[1]	(C) Block Reservation	1(a)
make[3]	(AS) make (Customer, Block Reservation)	3
invoice[1]	(C) Invoice	1(a)
opened[4]	(A) opened: boolean	4
cover[3]	(AS) bill for (Invoice, Contract) (1, 1..*)	3
one or more[5]		5(b)
rentals[1]	(same as contract)	
checked out[3]	(AS) check out (Customer, Vehicle)	3
pay[3]	(AS) pay (Customer, Invoice)	3
sent to[3]	(AS) sent to (Invoice, Company)	3
company[1]	(C) Company	1(a)
	(AC) Payment	1(c)
rental charge[1]	(A) rental charge	1(e)
credit card[1]	(AC) Pay by Credit Card	1(d)
processed[3]	(AS) processed by (Pay by Credit Card, Credit Card Company)	3
credit card processing company[1]	(C) Credit Card Company	1(a)
	(I) ISA (Credit Card Company, Company)	9
	(I) ISA (Pay by Credit Card, Payment)	9
	(I) ISA (Daily unlimited Miles Plan, Rental Plan)	9
	(I) ISA (Weekend Savings Plan, Rental Plan)	9
	(I) ISA (Passenger Car, Vehicle)	9
several[5]	(AG) Part-of (Reservation, Block Reservation) (2+, 1)	7

شکل ۵-۱۹: کلاس‌بندی مفاهیم کاربرد CRS



شکل ۵-۲۰: تصویرسازی مدل دامنه‌ی اجاره‌ی خودرو

اختلاف اطلاعات جمع‌آوری شده از دامنه توسط تحلیل‌گر، سابقه، و تجربیات او ایجاد می‌شود، و بخشی به نحوه‌ی شناسایی مفاهیم دامنه و استفاده از قوانین کلاس‌بندی مربوط می‌شود.

۵-۶ رهنمون‌هایی برای مدل‌سازی دامنه

رهنمون ۵-۸

اعضای تیم باید طوفان فکری و کلاس‌بندی را به‌عنوان فعالیت‌های تیمی انجام دهند و نه به‌صورت فعالیت شخصی.

طوفان فکری و کلاس‌بندی با یک گروه سه تا پنج نفری از توسعه‌دهندگان می‌تواند به برانگیخته‌شدن ایده‌ها و کامل‌کردن محدودیت‌های یکدیگر کمک کند. ولی نکته‌ی مهم‌تر این است که جلسات طوفان فکری و کلاس‌بندی گروه به اعضای تیم اجازه می‌دهد که عقاید خود را برای تعریف و کلاس‌بندی یک مفهوم، تبادل نمایند. چنین جلساتی برای رسیدن به فهم مشترک دارای اهمیت است. این درک مشترک، پایه‌ای برای فعالیت‌های آینده‌ی طراحی و پیاده‌سازی می‌باشد. تجربه نشان می‌دهد که تیم‌هایی که مدل‌سازی دامنه را انجام می‌دهند، طراحی و پیاده‌سازی بهتری نیز خواهند داشت. از سوی دیگر، تیم‌هایی که

این‌کار را تقسیم می‌کنند و هر بخش از آن‌را به برخی از اعضای تیم واگذار می‌نمایند، در زمان طراحی و پیاده‌سازی ضعیف‌تر عمل خواهند کرد. مهم‌ترین دلیل این مسئله، عدم وجود درک مشترک از کاربرد است.

راهنمای ۵-۱

قوانین طوفان فکری را بین اعضای تیم تقسیم کنید تا روی آن تمرکز نمایند. برای اطمینان از این‌که چیزی از مفاهیم خاص دامنه از قلم نیفتاده است، این کار را دوباره با افراد متفاوت تکرار کنید.

اجرای راهنمای فوق، از این‌که تنها اسم‌ها و عبارت‌های اسمی به‌عنوان مفاهیم خاص دامنه شناسایی شوند، جلوگیری می‌کند.

رهنمون ۵-۹

ابتدا طوفان فکری به راه بیاندازید، سپس کلاس‌بندی کنید.

طوفان فکری و کلاس‌بندی را هم‌زمان انجام ندهید تا تیم بتواند در هر زمان بر روی یک موضوع تمرکز کند. این، یک کاربرد از قوانین جداسازی دغدغه‌هاست.

رهنمون ۵-۱۰

هر جلسه‌ی طوفان فکری و کلاس‌بندی باید یک تا دو ساعت طول بکشد.

طوفان فکری به زمانی برای آماده‌شدن احتیاج دارد. معمولاً چیزی در حدود ۱۵ تا ۳۰ دقیقه طول می‌کشد که جلسه‌ی طوفان فکری به اصطلاح گرم شود و کار مفید شروع شود. به‌دلایل مختلفی مانند خستگی شرکت‌کنندگان، بعد از دوساعت میزان تاثیر و کارایی جلسه کاهش می‌یابد.

راهنمای ۵-۲

اعضای تیم را وادار کنید که تکالیف خود را قبل از جلسه‌ی طوفان فکری و کلاس‌بندی انجام دهند. هر یک از اعضای تیم باید قبل از جلسه به تنهایی گام‌های طوفان فکری و کلاس‌بندی را انجام دهد و حاصل کار خود را به جلسه بیاورد. این کار کیفیت و کارایی جلسه را افزایش می‌دهد. به‌جای این کار، اعضای تیم می‌توانند از ۲۰ دقیقه‌ی اول جلسه برای کار انفرادی استفاده کنند و در ادامه، کار را به‌صورت تیمی دنبال نمایند.

راهنمای ۵-۳

از یک تخته‌سفید و یک دوربین دیجیتال استفاده کنید.

اعضای تیم باید به نوبت، مفاهیم شناسایی‌شده و کلاس‌بندی شده را روی تخته‌سفید ثبت کنند. برای مرتبط کردن این مفاهیم، خطوطی رسم کنید. برای مثال، قبل از «Model» یک 'a' بگذارید و آنرا به Vehicle وصل کنید تا نشان دهید که مدل، یک صفت از وسیله‌ی نقلیه است، یا با استفاده از خط ارتباطی ارث‌بری در UML، Weekend Savings Plan را به Rental Plan وصل کنید تا یک رابطه‌ی ارث‌بری را نشان دهد.

تخته سفید می‌تواند اعضای تیم را از آنچه موضوع صحبت است، مطلع و برآن تمرکز نماید. در پایان جلسه، یکی از اعضای تیم می‌تواند یک عکس از تخته‌سفید بگیرد و آن‌را برای سایر اعضای تیم بفرستد. یک یا دو نفر از اعضا می‌توانند مدل دامنه را رسم کنند و آن را برای مرور به دیگران بفرستند. به‌عنوان جایگزینی برای تخته‌سفید و دوربین دیجیتال، می‌توان از یک لپ‌تاپ که به پروژکتور متصل شده است، استفاده کرد.

رهنمون ۵-۱۱

هر جلسه‌ی طوفان فکری و کلاس‌بندی باید به‌طور کارآمدی هماهنگ شود.

یکی از وقت‌گیرترین و ناکارآمدترین کارهایی که می‌توان در یک جلسه انجام داد، بحث‌های دوره‌ای در مورد قرارگرفتن یا قرار نگرفتن یک مفهوم در مدل دامنه، و یا چگونگی کلاس‌بندی آن است. هیچ‌کدام از تصمیم‌هایی که شما در جریان توسعه‌ی نرم‌افزار می‌گیرید، کاملاً سیاه یا سفید نیستند. بسیاری از این تصمیم‌ها بر اساس فرضیات قبلی گرفته می‌شوند و به پیش‌زمینه و تجربیات تصمیم‌گیرنده بستگی دارند. تصمیم‌گیری در مورد وجود داشتن یا نحوه‌ی کلاس‌بندی یک مفهوم در مدل دامنه، همیشه علمی نیست. بنابراین، جلسات طوفان فکری و کلاس‌بندی باید به دقت هماهنگ شود. هماهنگ‌کننده‌ی جلسه باید آن را متمرکز بر بحث اصلی نگه دارد و اطمینان یابد که از وقت به‌خوبی استفاده می‌شود. جلسه باید بر شناسایی و سپس کلاس‌بندی مفاهیم مرتبط با دامنه و روابط آن‌ها متمرکز شود. بحثهای نامربوط و طولانی در مورد نکات طراحی و پیاده‌سازی باید متوقف، و درصورت لزوم به بعد از جلسه موکول شود. اگر پیشرفت کار به‌طورمرتب نشان داده شود، از وقت گروه بهتر استفاده خواهد شد. اگر وقت بیش از اندازه‌ای بر روی یک مفهوم یا موضوع صرف شود، هماهنگ‌کننده باید با رعایت ادب بحث را پایان دهد، موضوع را از دستور خارج کند، و راه‌های جایگزینی برای حل اختلافات پیشنهاد دهد.

رهنمون ۵-۱۲

در زمان جلسات طوفان فکری و کلاس‌بندی، نمودارهای کلاس UML را رسم نکنید.

رسم یک نمودار کلاس UML، درحالی‌که همزمان در حال تلاش برای شناسایی کلاس‌ها، و صفت‌ها و روابط آن‌ها هستید، ایده‌ی بدی است که متاسفانه بسیاری از تیم‌ها آن را اجرا می‌کنند. اعضای تیم باید طبق گام‌های شرح داده شده در بخش‌های ۵-۴-۲ و ۵-۴-۳ ، بر طوفان فکری و کلاس‌بندی متمرکز باشند. اگر آن‌ها به اطلاعات دیگری نیاز داشتند، رسم نمودار کلاس نسبتاً ساده است. بدون انجام یک کار خوب طوفان فکری و کلاس‌بندی، کیفیت نمودار کلاس UML بسیار پایین خواهد بود.

۵-۷ به‌کارگیری قوانین چابکی

در ادامه، رهنمون‌هایی را برای توسعه‌ی چابک مشاهده می‌کنید:

رهنمون ۵-۱۳

در کنار مشتری و کاربران کار کنید تا کاربردها و دامنه‌ی کاربرد آن‌ها را درک نمایید.

تیم توسعه باید کاربردی که نرم‌افزار برای آن ساخته می‌شود را درک کند. این بدان دلیل است که فهم نیازمندی‌های نرم‌افزار، به میزان درک تیم از کاربرد و دامنه‌ی آن بستگی دارد. بنابراین تیم نیازمند است که نزدیک به مشتری و کاربران کار کند. تعامل با مشتری و کاربران، به تیم در بیرون کشیدن اطلاعات لازم دامنه، رفع نقاط ابهام، و برطرف‌کردن سوء تفاهم‌ها کمک می‌کند.

رهنمون ۵-۱۴

تنها در جایی دامنه را مدل‌سازی کنید که تیم نیازمند فهم دامنه‌ی کاربرد باشد. مدل دامنه را ساده نگه دارید و آن را به‌طور افزایشی توسعه دهید.

تیم نباید با افراط در اجرای طوفان فکری، سعی کند که همه‌ی کلاس‌ها، صفت‌ها، و روابط را به یکباره شناسایی نماید. در عوض، تیم باید با توجه به موردهای کاربرد تخصیص‌یافته به دور اول، یک مدل دامنه‌ی مقدماتی با هدف کمک به فهم مفاهیم دامنه بسازد. مدل دامنه در هر یک از دوره‌های بعدی به آرامی تقویت خواهد شد. تلاش برای واردکردن همه‌ی کلاس‌ها، صفت‌ها، و روابط در مدل دامنه، منجر به مستنداتی بسیارسنگین خواهد شد و این یکی از معایب روش‌های سنتی است که روش‌های چابک از آن جلوگیری می‌نمایند.

رهنمون ۵-۱۵

مدل‌سازی دامنه می‌تواند به‌طور همزمان با مدل‌سازی تعامل کنش‌گر با سیستم، مدل‌سازی تعامل اشیا، مدل‌سازی حالت اشیا، و یا مدل‌سازی فعالیت‌ها انجام گردد.

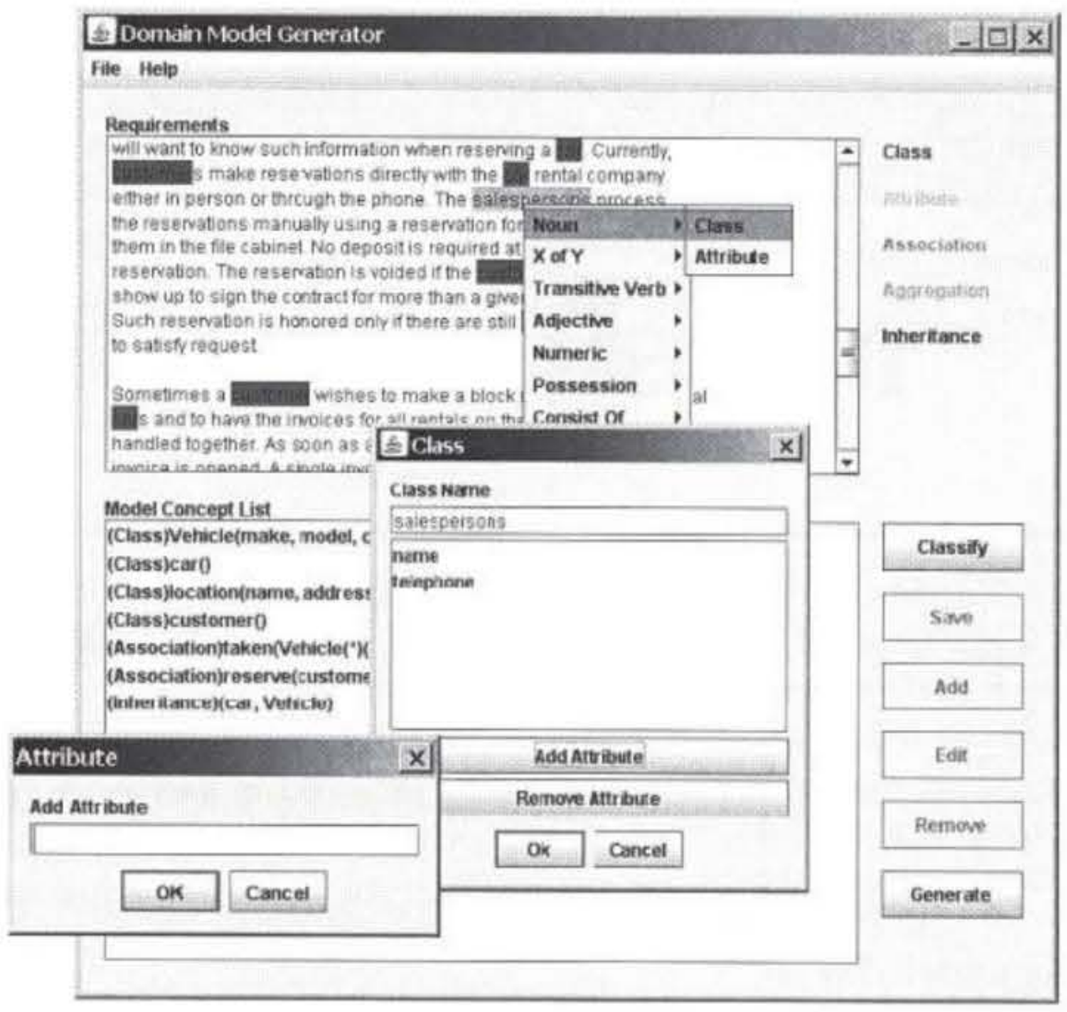
دانستن و درک هر بخش از دانش دامنه در زمان مدل‌سازی دامنه، کاری دشوار است. پافشاری بر به دست آوردن همه‌ی دانش لازم از دامنه، کاری وقت‌گیر و غیرضروری است، زیرا گام‌های آینده، قطعات از قلم افتاده‌ی اطلاعات را پیدا خواهند کرد. این بدان معناست که فرایند مدل‌سازی یک فرایند تکراری است که شامل بازگشت به عقب نیز می‌شود. به این ترتیب، فعالیت‌های مختلف مدل‌سازی می‌توانند با هم تعامل داشته باشند و به طور همزمان پیشرفت نمایند.

۵-۸ ابزارهای پشتیبان مدل‌سازی دامنه

ایجاد و مدیریت نمودارهای UML، کاری خسته‌کننده و وقت‌گیر است. وجود ابزارهای پشتیبان برای این کار، امری خوشایند است. ابزارهای بسیاری شامل محصولات عمومی، متن باز، و تجاری در این زمینه وجود دارند. برخی از شناخته‌شده‌ترین این ابزارها عبارتند از: IBM Rational Modeler، Microsoft Visio، Argo UML، Net Beans UML Plugin. این ابزارها،

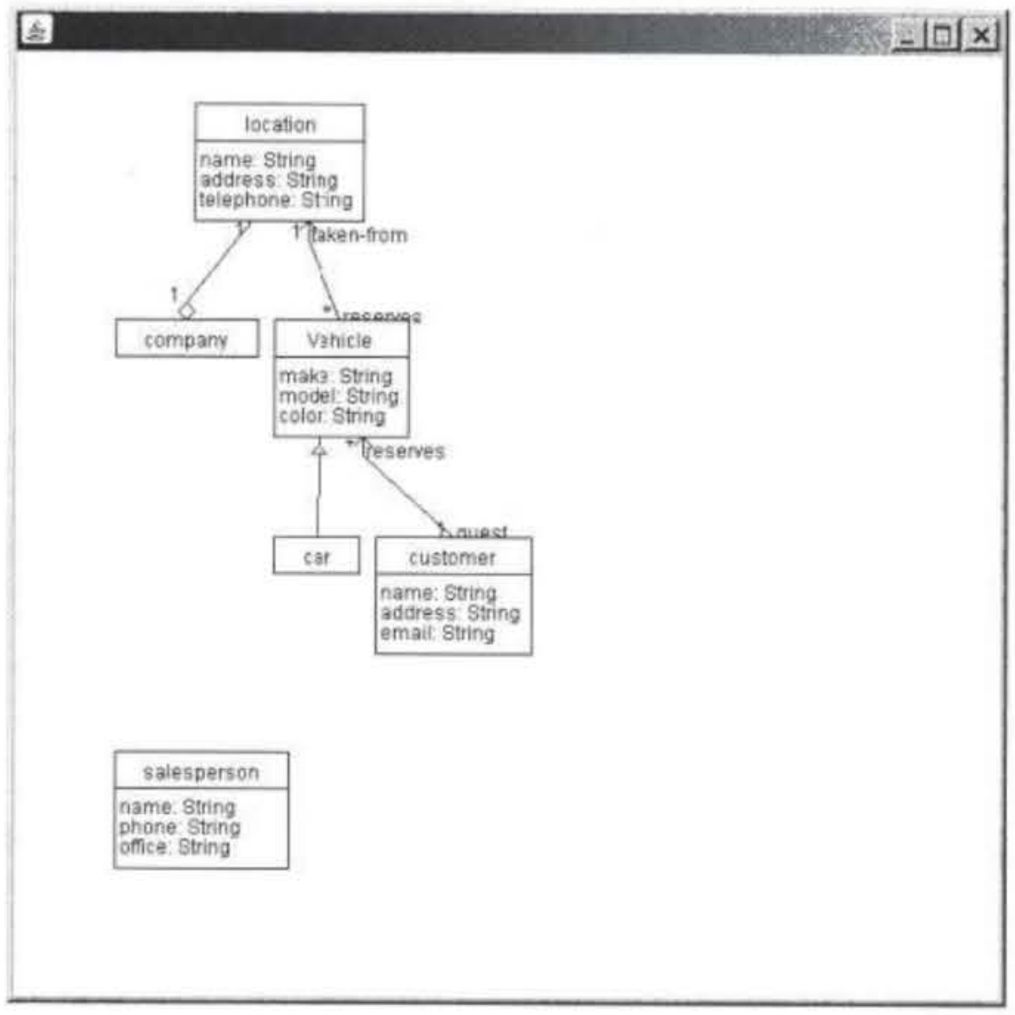
ویرایش‌گرهایی برای نمودارهای UML را به همراه امکانات دیگر در اختیار ما قرار می‌دهند. این ابزارها می‌توانند با استفاده از نمودارهای طراحی، شالوده‌ی کد را برای زبان‌های برنامه‌نویسی مختلف ایجاد نمایند. بیشتر این ابزارها، مهندسی معکوس، که تولید نمودارهای UML از کد است را نیز پشتیبانی می‌نمایند. این امکانات، تنها تعدادی از خصوصیات این ابزارها می‌باشد.

یک مجموعه ابزار که در دانشگاه تگزاس در آرلینگتون توسعه‌یافته است، متدولوژی نشان داده شده در این فصل را پشتیبانی می‌کنند. این ابزارها گونه‌های متفاوتی از عملیات را پشتیبانی می‌نمایند. با استفاده از امکان Manual Identification و Manual Classification، مهندس نرم‌افزار می‌تواند به‌طور دستی مفاهیم دامنه را شناسایی کند و آن‌ها را در قالب مفاهیم مدل‌سازی دسته‌بندی نماید. شکل ۵-۲۱ چگونگی انجام این کار را نشان می‌دهد. ابتدا مهندس نرم‌افزار شرح متنی را در قاب[1] نیازمندی‌ها باز می‌کند. سپس هر یک از مفاهیم دامنه را انتخاب کرده و با کلیک در بخش Model Concept List نمایش می‌دهد. مهندس نرم‌افزار می‌تواند پس از کلاس‌بندی، ویژگی‌های یک مفهوم مدل را شرح دهد. او می‌تواند به‌جای این کار، مفهوم را از Model Concept List انتخاب نماید و با فشردن کلید Edit ویژگی‌های موردنظر را ویرایش نماید. وقتی همه‌ی این کارها انجام شد، مهندس روی دکمه‌ی Generate کلیک می‌کند و ابزار، همان‌طور که در شکل ۵-۲۲ نشان داده شده است، نمودار UML را به‌طور خودکار رسم می‌نماید. برای فعال‌سازی قابلیت شناسایی و کلاس‌بندی خودکار، مهندس روی دکمه‌ی Classify کلیک می‌کند، و ابزار مفاهیم دامنه را شناسایی و دسته‌بندی می‌نماید. این مفاهیم رنگی می‌شوند و در قاب Model Concept List فهرست می‌گردند. مهندس نرم‌افزار می‌تواند با استفاده از بخش Model Concept List نتایج را ویرایش نماید و تغییر دهد. با کلیک کردن بر روی دکمه‌ی Generate، ابزار به‌طور خودکار نمودار کلاس را رسم می‌کند.



شکل ۵-۲۱: شناسایی دستی و کلاس‌بندی دستی

[1] Pane



شکل ۵-۲۲: نمودار کلاس تولیدشده برای مدل دامنه

خلاصه

این فصل کاربرد مدل دامنه و گام‌های ایجاد آن برای یک کاربرد موردنظر را نشان می‌دهد. مهم است که تشخیص داده شود که مدل‌سازی دامنه فقط به معنای رسم نمودار کلاس UML نیست. سه‌گام اول و گام آخر از مدل‌سازی دامنه دارای اهمیت بیشتری هستند و به تصمیم‌گیری دقیق‌تری نیاز دارند. این گام‌ها شامل جمع‌آوری اطلاعات در مورد دامنه‌ی کاربرد، طوفان فکری، کلاس‌بندی و یافتن اطلاعات جاافتاده می‌باشند. به‌این‌ترتیب، رسم مدل دامنه ساده‌ترین گام است. این فصل، تعریف مفهومی و تعریف توسعه‌ای را به‌عنوان دو فن مهم تعریف شرح می‌دهد.

همچنین مفاهیم مهم شئ‌گرایی شامل کلاس، شئ، ارث‌بری، چندریختی، اجتماع، و انجمن در این فصل مرور می‌شود. این فصل مفاهیم مهم شئ‌گرایی شامل کلاس UML را به همراه رهنمون‌هایی برای استفاده از این اجزای مدل‌سازی نشان می‌دهد. باید توجه داشت که هدف از این فصل، پوشش کامل نمودار کلاس UML نیست. بلکه قصد بر آن است که تنها زیرمجموعه‌ای از آنچه معمولاً در مدل‌سازی دامنه انجام می‌شود، ارائه گردد. کاربرد کرایه‌ی خودرو به‌عنوان مثالی برای شرح متدولوژی مدل‌سازی دامنه و نمودار کلاس UML مورد استفاده قرار گرفته است. این مثال با وجود کوچک بودن، به‌اندازه‌ی کافی برای نمایش متدولوژی و بیشتر خصوصیات پرکاربردتر نمودار کلاس UML فراگیر است.

برای مطالعه‌ی بیش‌تر

مرجع [۳۶] یک منبع خوب برای یادگیری نمودار کلاس UML است. شرح جامعی از UML نیز می‌تواند در وبگاه گروه مدیریت شئ در آدرس www.omg.org پیدا شود. نمادهای نمودار کلاس UML در OMT [۱۳۱] معرفی شده است که علاوه بر این، شامل یک متدولوژی برای شناسایی کلاس‌ها، ویژگی‌ها و روابط انجمنی است. بلاها و رامبو [۳۰] این متدولوژی را برای

UML 2.0 مورد تجدیدنظر قرار داده‌اند.

سوال‌های مروری فصل

۱. هدف از مدل‌سازی دامنه چیست؟
۲. گام‌های مدل‌سازی دامنه کدام است؟
۳. قوانین طوفان فکری در مدل‌سازی دامنه چیست؟
۴. قوانین کلاس‌بندی مفاهیم دامنه چیست؟
۵. مفاهیم مدل‌سازی اجزای نمودار کلاس UML را شرح دهید.
۶. رهنمون‌های مدل‌سازی دامنه را به‌طور خلاصه ذکر نمایید.
۷. چگونه قوانین چابکی را در مورد مدل‌سازی دامنه اعمال می‌نمایید؟

تمرین‌ها

۵-۱ یک گراف جهت‌دار یا دی‌گراف G = (V,E,L) از یک مجموعه گره‌ی V = {V1,V2,...,Vn}، یک مجموعه برچسب لبه‌ی L و یک مجموعه از لبه‌های جهت‌دار $E \in V \times V \times L$ تشکیل شده است. فرض کنید که شما برای طراحی و پیاده‌سازی یک ویرایشگر گراف انتخاب شده‌اید. این ویرایش‌گر برای رسم و ویرایش یک دی‌گراف مورد استفاده قرار می‌گیرد. شما باید یک مدل دامنه بسازید. در این تمرین باید کارهای زیر را انجام دهید:

الف) یک توصیف متنی برای گراف تهیه کنید. این توصیف باید اجزای یک گراف مانند گره‌ها، لبه‌ها و روابط بین اجزا را شرح دهد.

ب) قوانین طوفان فکری را، به همان صورتی که در بخش ۵-۴-۲ آمده است، در مورد توصیف متنی خود اعمال نمایید و حاصل طوفان فکری را ایجاد کنید.

ج) نتایج طوفان فکری را، به همان صورتی که در بخش ۵-۴-۳ گفته شده است، کلاس‌بندی کنید و نتیجه‌ی کلاس‌بندی را ایجاد نمایید.

د) نتیجه کلاس‌بندی را به یک نمودار کلاس UML تبدیل نمایید.

۵-۲ یک شرح یک صفحه‌ای از عملیات کسب‌وکار برای یک سیستم رزرواسیون هتل تهیه کنید. فرضیاتی که برای عملی شدن این سیستم لازم و ضروری می‌دانید بیان نمایید.

۵-۳ با استفاده از توصیف به دست آمده در تمرین ۵-۲، مرحله‌ی طوفان فکری را برای سیستم رزرواسیون هتل انجام دهید.

۵-۴ با استفاده از نتیجه‌ی طوفان فکری انجام شده در تمرین ۵-۳، مرحله‌ی کلاس‌بندی را برای سیستم رزرواسیون هتل انجام دهید.

۵-۵ براساس نتایج به دست آمده در تمرین ۵-۴، مدل دامنه را با استفاده از نمودار کلاس UML رسم نمایید.

۵-۶ نمودار کلاس مدل دامنه‌ی ایجادشده در تمرین ۵-۵ را با استفاده از یک فهرست بررسی مروری که در بخش ۵-۴-۴ توضیح داده شد، مرور نمایید. گزارشی از این مرور تهیه کنید که به سؤالات مطرح‌شده در فهرست بررسی پاسخ دهد.

۵-۷ یک کاربرد دیگر انتخاب نمایید و تمرین فوق را برای آن کاربرد انجام دهید.

۵-۸ فرض کنید که شما برای توسعه‌ی یک ویرایشگر گرافیکی برای یک نمودار کلاس UML انتخاب شده‌اید. در این تمرین شما باید مراحل مدل‌سازی دامنه را انجام دهید و یک مدل دامنه ایجاد کنید که مفاهیم مدل‌سازی که در شکل ۵-۱ نشان داده شده است را شرح دهد.

۵-۹ مقاله‌ای بنویسید که توضیح دهد کار اکتساب نیازمندی‌ها می‌تواند به ترتیب با استفاده از فرایند آبشاری، و فرایند چابک انجام گردد. تفاوت بین دو روش را بنویسید و به طور مختصر شرح دهید.
راهنمایی: این تمرین مشابه ولی کمی پیچیده‌تر از تمرین ۵-۱ است. مدل دامنه‌ی حاصل، یک نمودار کلاس UML است که یک نمودار کلاس UML را توصیف می‌نماید.

۶

طراحی معماری

نکات کلیدی

- معماری نرم‌افزاری یک سیستم یا یک زیرسیستم، به سبک طراحی ساختار سیستم، شامل برقراری ارتباط و تعامل بین زیرسیستم‌ها و اجزای آن، اطلاق می‌گردد.
- سیستم‌های مختلف، روش‌های طراحی و سبک‌های معماری[۱] متفاوتی را طلب می‌کنند.

سیستم‌های نرم‌افزاری برای انجام عملیات گوناگونی طراحی و ساخته می‌شوند. این عملیات توسط زیرسیستم‌ها و اجزای مختلفی انجام می‌گردد. از بسیاری از سیستم‌های نرم‌افزاری انتظار می‌رود که سال‌ها و بلکه دهه‌ها کار کنند. برای مثال، بسیاری از سیستم‌های بانکی در دهه‌ی ۱۹۶۰ ساخته شده‌اند و هنوز در حال کارند. البته این دستگاه‌ها در این مدت، بارها مورد ارتقا، بهبود امکانات، و مهندسی مجدد قرار گرفته‌اند. این تجربیات نگهداری نرم‌افزار نشان می‌دهد که ساختار یا معماری یک سیستم نرم‌افزاری، بر تعدادی از ویژگی‌های سیستم، شامل کارایی، بهره‌وری، امنیت، و قابلیت نگهداری بسیار مؤثر است. بنا به گفته‌ی بوچ[۲]، توسعه‌ی نرم‌افزار باید «بر توسعه‌ی اولیه و پی‌ریزی یک معماری نرم‌افزار تمرکز نماید، سپس از معماری سیستم، به‌عنوان یک دست‌ساخته اصلی، برای تصویرسازی، ساخت، مدیریت، و رشد سیستم تحت‌توسعه استفاده نماید» [۳۶]. به‌عبارت‌دیگر معماری یک سیستم نرم‌افزاری، نقش اصلی را در طول چرخه‌ی عمر آن ایفا می‌نماید. با مطالعه‌ی این فصل مطالب زیر را خواهید آموخت:

- معماری نرم‌افزار چیست؟ طراحی معماری چیست؟
- چرا طراحی معماری دارای اهمیت است؟
- قوانین طراحی نرم‌افزار کدامند و ویژگی‌های آن‌ها چه هستند؟
- سبک معماری چیست و پرکاربردترین سبک‌های معماری کدامند؟
- ارتباط بین انواع مختلف سیستم و سبک‌های مختلف معماری چیست؟

---

[۱] Architectural Style

[۲] Booch

۶-۱ طراحی معماری چیست؟

طراحی معماری برای یک سیستم نرم‌افزاری، مانند طراحی معماری یک ساختمان است. یک ساختمان از تعدادی اتاق و زیرسیستم‌هایی مانند گرمایش و تهویه‌ی هوا، تأسیسات برق، گاز، آب و لوله‌کشی‌های مختلف تشکیل می‌شود. هر یک از این اجزا منظور خاصی را برآورده می‌سازد و در عملکرد نهایی و سودمندی ساختمان موثر است. طراحی معماری یک ساختمان، امکانات اصلی و نیز زیرسیستم‌ها و چگونگی ارتباط و همکاری آن‌ها را برجسته می‌نماید. طراحی معماری نرم‌افزار نیز مشابه همین است و به شکل زیر تعریف می‌گردد:

تعریف ۶-۱

به سبک طراحی ساختار سیستم، شامل ارتباط و تعامل بین اجزای اصلی آن، معماری سیستم گفته می‌شود.

تعریف ۶-۲

طراحی معماری نرم‌افزار و یا به بیان ساده‌تر طراحی معماری، یک فرایند تصمیم‌گیری برای تعیین معماری نرم‌افزار سیستم تحت‌توسعه است.

معماری نرم‌افزار، به‌عنوان یک سبک طراحی، نشان می‌دهد که قبلاً تصمیم‌های طراحی مشخصی گرفته شده و معماری، نتیجه‌ی این تصمیم‌های طراحی است. بدین‌ترتیب، معماری نرم‌افزار می‌تواند به‌عنوان مجموعه‌ای از تصمیم‌های طراحی نیز تعریف گردد. تفاوت‌های اندکی بین نگاه به معماری به‌عنوان یک سبک طراحی یا یک مجموعه از تصمیم‌های طراحی وجود دارد. نگاه اول ملموس‌تر است و فهم را آسان می‌کند. این نگاه به ما امکان می‌دهد که معماری یک سیستم را به‌عنوان تعدادی زیرسیستم و ارتباط بین آن‌ها به‌تصویر بکشیم. تصمیم‌های طراحی به‌عنوان استدلالی برای استفاده از معماری به‌کار می‌روند.

معماری Nلایه، معماری مشتری-خادم و معماری مدل-نما-کنترلگر[۱] مثال‌هایی از معماری نرم‌افزار می‌باشند. معماری Nلایه، ساختار سیستم را به‌عنوان تعدادی لایه می‌بیند که هرکدام از خدمات لایه‌ی پایین‌تر استفاده می‌کنند. در معماری مشتری-خادم، یک جزء خادم، خدماتی را برای اجزای دیگر که با عنوان مشتری شناخته می‌شوند فراهم می‌نماید. مشتری‌ها خادم را می‌بینند، ولی خادم مشتری‌ها را نمی‌شناسد. معماری MVC از یک مدل داده، یک کنترلگر و تعدادی دیدگاه تشکیل می‌شود. کنترلگر، دیدگاه‌ها را از مدل داده جدا می‌نماید و اجازه می‌دهد که داده‌ها با استفاده از دیدگاه‌های مختلفی مانند جدول، نمودار دایره‌ای[۲]، یا نمودار هیستوگرام نشان داده شوند.

۶-۲ اهمیت طراحی معماری

همان‌طور که در داستان زیر که سال‌ها پیش اتفاق افتاده است می‌بینیم، از اهمیت طراحی معماری هرگز نمی‌توان چشم‌پوشی کرد. این داستان در مورد توسعه‌ی یک سیستم نرم‌افزاری با مقیاس متوسط تا بزرگ به زبان C++ است که به یک پایگاه‌داده‌ی شئ دسترسی پیدا می‌کرد. بسیاری از دانشجویان سیستم مدیریت پایگاه‌داده‌ی Ontologies را نمی‌شناسند، ولی این سیستم در حدود ۱۹۹۰ بسیار معروف بود. Ontologies یک زبان پرس‌وجوی شبیه C++ را برای کاربر فراهم می‌کرد، یعنی برنامه‌های C++ می‌توانستند مستقیماً به پایگاه‌داده دسترسی پیدا کنند. این بدان معنا بود که اشیای کسب‌وکار و پایگاه‌داده به‌خوبی با

[۱] Model-View-Controller (MVC)

[۲] Pie Chart

یکدیگر یکپارچه شده بودند. اما این یکپارچه‌سازی یا اتصال محکم، اگر با آن‌ها به‌درستی برخورد نشود، می‌تواند مشکل‌ساز شود. برای مثال کد برنامه‌ی کاربردی به‌شدت به پایگاه‌داده وابسته می‌شود. اگر پایگاه‌داده تغییر کند، مقدار زیادی از کد باید دوباره نوشته شود. برای جلوگیری از این مشکل، پیشنهاد شد که روکشی برای پایگاه‌داده نوشته شود که آن را از دید اشیای کسب‌وکار مخفی نماید. این‌کار، اشیای کسب‌وکار را در زمان تغییر یا تعویض پایگاه‌داده حفاظت می‌کند. اگر تغییری رخ دهد، فقط آن روکش باید تغییر کند و نه اشیای کسب‌وکار. تغییر روکش بسیار ساده‌تر از تغییر اشیای کسب‌وکار است. در فصل ۱۷ خواهیم دید که این کار در واقع طراحی یک چارچوب ماندگاری[۱] است. متاسفانه تیم توسعه به‌دلیل این‌که این پیشنهاد، پیچیده و ناکارامد به‌نظر می‌رسید، آن را رد کردند. این، ابتدای پروژه بود.

ده ماه بعد، پس از آن‌که بیشتر سیستم پیاده‌سازی شده بود، شایعه‌ای به‌گوش رسید که فروشندگان Ontologies بر سر قدرت درگیر مشکلات داخلی شده‌اند. پیامد این ماجرا این بود که شرکای اصلی این شرکت که نیروهای کلیدی و فنی آن بودند، تهدید کردند که شرکت را ترک کرده و شرکت جدیدی راه‌اندازی خواهند کرد. تیم توسعه که به Ontologies تکیه کرده بود، عصبی و ناراحت شده و آرزو می‌کرد که کارمندان Ontologies سر کارهای خود بمانند. یک ماه بعد، شایعه به واقعیت تبدیل شد و مؤسسان اصلی Ontologies آن را ترک کرده و شرکت جدیدی تأسیس نمودند. این مسأله برای پایان موفقیت‌آمیز پروژه یک فاجعه بود. تیم توسعه شروع به پیاده‌سازی روکش پایگاه‌داده کرد، ولی دیگر خیلی دیر شده بود. پروژه به‌شدت از زمان‌بندی خود عقب بود. چند ماه بعد، شرکت فروشنده‌ی Ontologies ورشکسته شد و چندی پس از آن، تیم توسعه از کار برکنار شدند.

۶-۳ فرایند طراحی معماری

فرایند طراحی معماری برای یک سیستم یا زیرسیستم نرم‌افزاری، یک فرایند شناختی تصمیم‌گیری است. این فرایند باید عوامل زیادی را در نظر بگیرد. نوع سیستمی که می‌خواهد توسعه داده شود، یک عامل مهم است. تجربه نشان داده است که نوع سیستم بر انتخاب سبک معماری موثر است. برای مثال، سیستم‌های تعاملی[۲] معمولاً از معماری Nلایه استفاده می‌کنند، سیستم‌های رویداد-رانده[۳] از معماری سیستم توکار استفاده می‌نمایند و سیستم‌های تبدیلی[۴] معماری برنامه‌ی اصلی و زیربرنامه‌ها را به‌کار می‌گیرند. اهداف طراحی نیز بر انتخاب سبک معماری موثرند. برای مثال، اگر امنیت، یک هدف مهم طراحی باشد، باید یک سبک معماری امنیتی مورد استفاده قرار گیرد. گاهی برای سیستم در حال توسعه یک سبک معماری مشخص وجود ندارد. در این حالت، یک طراحی مطابق با قواعد عرف برای معماری نرم‌افزار انجام می‌گیرد.

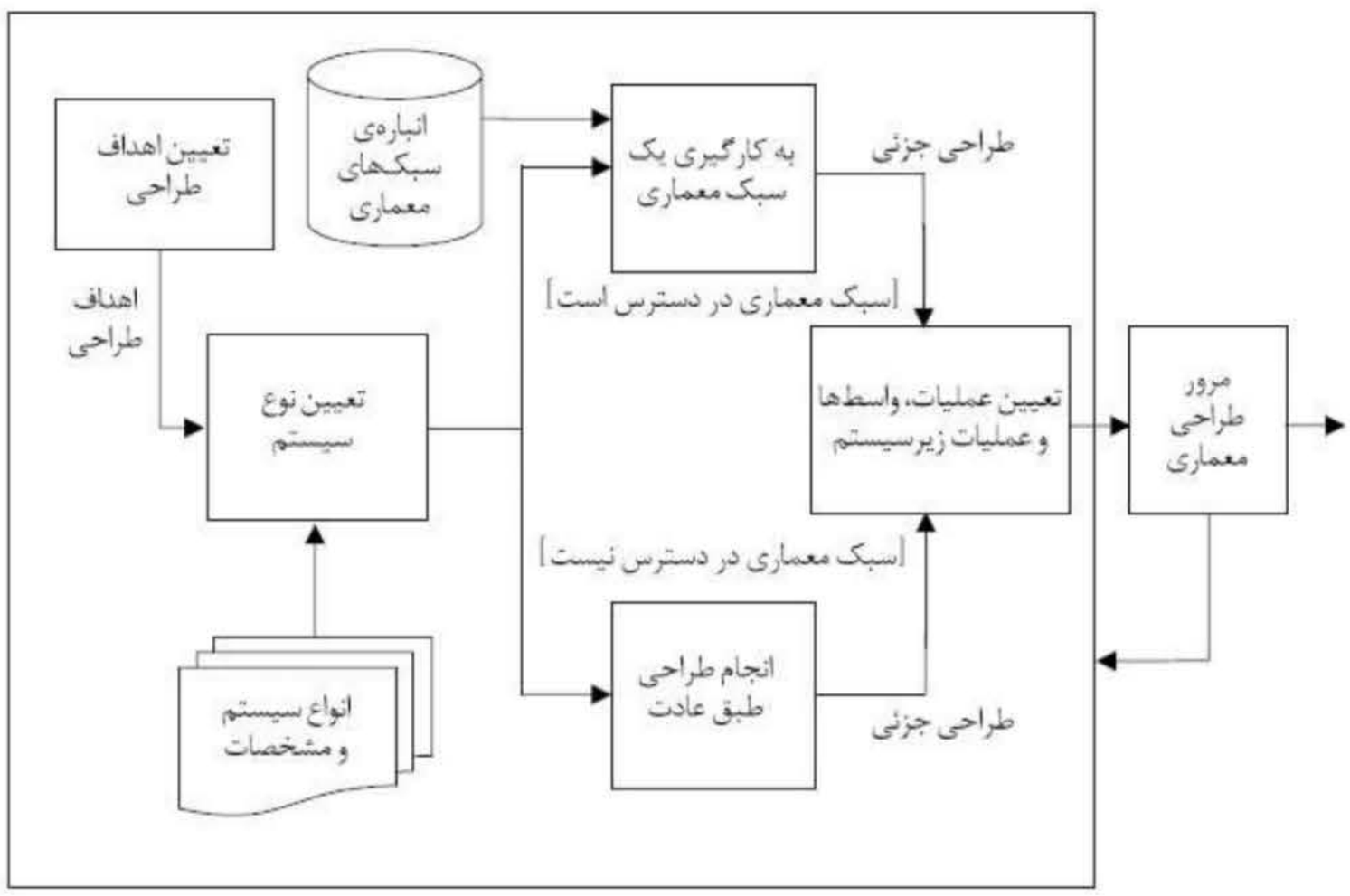
طراحی معماری یک فرایند بازگشتی است. این بدان دلیل است که یک سیستم از تعدادی زیرسیستم تشکیل می‌شود که آن‌ها نیز خود از زیرسیستم‌ها یا اجزای سطوح پایین‌تر تشکیل می‌شوند. لازم است که فرایند طراحی به‌طور بازگشتی تا همه‌ی سطوح پایین‌تر این سلسله مراتب انجام بگیرد، تا جایی که اجزای قرار گرفته در برگ‌های سلسله مراتب به‌راحتی طراحی و پیاده‌سازی گردند. متأسفانه هیچ قانون روشنی برای این کار وجود ندارد. پایان این فرایند به عوامل مختلفی مانند اندازه و پیچیدگی سیستم، تجربه تیم توسعه و اهداف طراحی بستگی دارد.

شکل ۶-۱ فرایند طراحی معماری را نشان می‌دهد. این فرایند باید به‌طور بازگشتی برای سیستم و زیرسیستم‌های شناسایی‌شده در طی فرایند طراحی انجام گردد.

۱. تعیین اهداف طراحی. در این گام، اهداف کلی طراحی شناسایی و تبیین می‌گردند. برای مثال، طراحی با هدف تحمل‌پذیری خطای نرم‌افزار، امنیت و ایمنی، و بیشینه کردن کارایی انجام می‌گیرد. اهداف کلی طراحی می‌تواند از شرح نیازمندی‌ها استخراج شود.

---

[۱] Persistence Framework
[۲] Interactive
[۳] Event-Driven
[۴] Transformational



شکل ۶-۱: فرایند طراحی معماری

۲. تعیین نوع سیستم. در این گام، نوع سیستم یا زیرسیستم‌ها مشخص می‌گردد. نوع سیستم و اهداف طراحی برای انتخاب یک سبک معماری از یک مخزن[1] مورد استفاده قرار می‌گیرد.
۳. به‌کارگیری یک سبک معماری و یا انجام یک طراحی معماری طبق عادت. اگر یکی از سبک‌های معماری موجود می‌تواند مورد استفاده قرار گیرد، این گام از آن سبک برای ایجاد یک طراحی معماری «استاندارد» استفاده می‌کند، در غیر این‌صورت یک طراحی به‌دلخواه انجام می‌گردد.
۴. تبیین عملیات، واسط‌ها و رفتار تعاملی زیرسیستم‌ها. در این گام، واسط‌های بین زیرسیستم‌ها تعریف می‌شوند و رفتار تعاملی بین زیرسیستم‌ها توصیف می‌گردد.
۵. بازبینی طراحی معماری. در این گام، طراحی معماری مورد بازبینی قرار می‌گیرد تا اطمینان حاصل گردد که نیازمندی‌ها، اهداف طراحی و قوانین طراحی نرم‌افزار برآورده شده است.

۶-۳-۱- تبیین اهداف طراحی معماری

طراحی معماری با هدف ایجاد یک ساختار کلی برای سیستم نرم‌افزاری انجام می‌گردد؛ اما این هدف، مجردتر از آن است که بتواند مفید باشد. یک طراحی معماری خوب برای یک سیستم، لزوماً برای سیستم دیگر مناسب نیست. بنابراین، اهداف طراحی معماری برای سیستم در حال توسعه باید مشخص شود و برای هدایت فرایند طراحی به‌کاربرده شود. یک هدف طراحی معماری، یک ویژگی یا جنبه‌ای از سیستم را که باید در زمان طراحی موردنظر قرار بگیرد مشخص می‌کند. کاربردهای مختلف، اهداف و اولویت‌های کسب‌وکاری متفاوت دارند که بر تصمیم‌های طراحی تاثیر می‌گذارد و بنابراین، باید در زمان تصمیم‌گیری در مورد اهداف طراحی درنظر گرفته شوند. گاهی اهداف طراحی معماری می‌تواند از نیازمندی‌های نرم‌افزار شامل نیازمندی‌های کیفی و امنیتی، استخراج شود. در ادامه، یک لیست جزیی از چنین نیازمندی‌ها یا عواملی که باید درنظر گرفته شوند، دیده

[1] Repository

می‌شود:

۱. سادگی تغییر و نگهداری. آیا برنامه‌ی کاربردی برای پاسخ به تغییرات، مانند تغییر نیازمندی‌ها، نیازمند تغییرات مکرر در سیستم است؟
۲. کاربرد قطعات تجاری. آیا پروژه، استفاده از COTS را ممنوع کرده و یا به آن نیاز دارد؟ و دامنه‌ی چنین بازاستفاده‌هایی چه قدر است؟
۳. کارایی سیستم. آیا کاربرد برای مواردی مانند پردازش داده‌ی بلادرنگ و یا حجم بسیار بالای تراکنش‌ها نیازمند کارایی بالا است؟
۴. قابلیت اطمینان. کاربرد مورد نظر تا چه حدی نیاز دارد که سیستم وظایف مورد انتظار خود را تحت شرایط فرضی به‌درستی انجام دهد؟
۵. امنیت. کاربرد تا چه اندازه به حفاظت داده‌ها و منابع برنامه نیاز دارد؟
۶. تحمل‌پذیری خطا[۱]. کاربرد مورد نظر تا چه حدی نیاز دارد که سیستم پس از بروز یک مشکل به‌کار خود ادامه دهد؟
۷. ترمیم. تا چه حدی کاربرد نیاز دارد که سیستم پس از یک فروپاشی[۲] به وضعیت قبلی برگردد؟

۶-۳-۲- تعیین نوع سیستم

نوع یک سیستم، مدل‌سازی، تحلیل، طراحی، پیاده‌سازی، و آزمون سیستم را به‌شدت تحت تاثیر خود قرار می‌دهد. به‌همین دلیل باید در زمان طراحی معماری نرم‌افزار برای یک سیستم، به نوع آن توجه نمود. اثر نوع سیستم بر انتخاب روش طراحی و معماری، در بخش‌هایی که در ادامه می‌آید شرح داده خواهد شد. از میان انواع مختلف سیستم‌ها، چهار نوع سیستم کاربردی که بیشتر دیده شده‌اند، به همراه رفتار آن‌ها مورد بحث قرار خواهد گرفت (شکل ۶-۲ را ببینید). این چهار نوع عبارتند از: سیستم‌های تعاملی، سیستم‌های رویداد-رانده، سیستم‌های تبدیلی و زیرسیستم‌های پایگاه‌داده. عبارت «نوع سیستم کاربردی» برای تمایز این سیستم‌ها با انواع سیستم‌های زیرساختی، مانند سرویس‌دهنده‌های فایل و سیستم‌های انتقال فایل، مورد استفاده قرارگرفته است. برای سادگی، در ادامه از عبارت «انواع سیستم» به‌جای «انواع سیستم کاربردی» استفاده می‌شود.

سیستم‌های تعاملی با تعامل بین سیستم و یک کنشگر برای انجام یک وظیفه از کسب‌وکار مشخص می‌شوند. کنشگرها، نقش‌هایی هستند که توسط موجودیت‌های خارجی در تعامل با سیستم، ایفا می‌شود. سیستم‌های رویداد-رانده با رفتار تعاملی و وابسته‌به‌حالت سیستم در پاسخ به محرک‌های خارجی یا رویدادها مشخص می‌شوند. سیستم‌های تبدیلی با فعالیت‌های پردازش اطلاعات شناخته می‌شوند که نوعاً شامل شبکه‌ای از فرایندهاست که ورودی را به خروجی کاملاً متفاوتی تبدیل می‌کند. زیرسیستم‌های پایگاه‌داده از داده‌های ماندگار برای انواع دیگر زیرسیستم‌ها پشتیبانی می‌کنند.

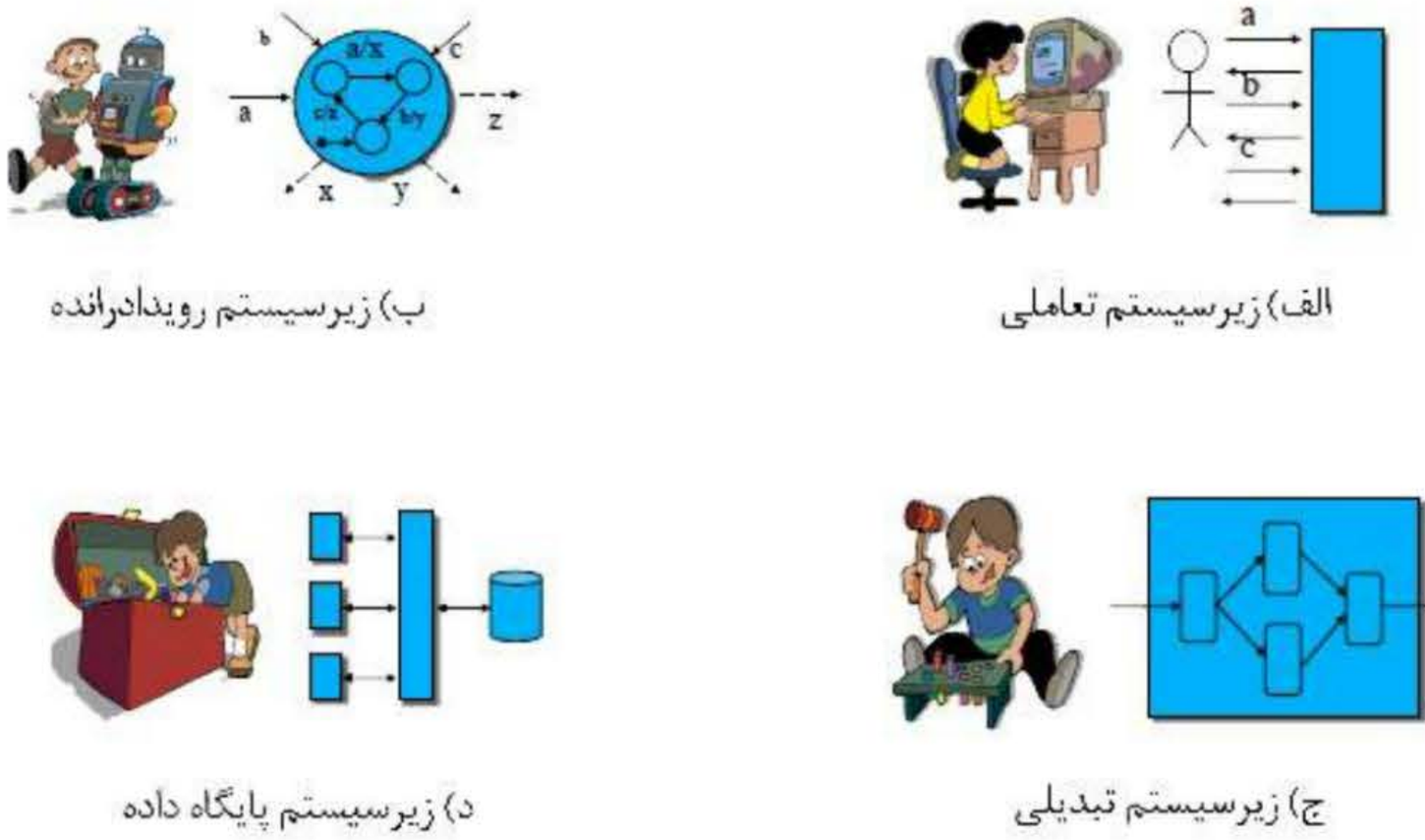
اثر بر روش طراحی و معماری

گاهی یک روش مدل‌سازی و طراحی، برای یک پروژه به‌خوبی کار می‌کند، ولی برای پروژه‌ی دیگر مناسب نیست. یک دلیل ممکن این است که نوع سیستم و معماری نرم‌افزار با هم متناسب نیستند. برای مثال، روش تحلیل و طراحی ساخت‌یافته (SA/SD) را درنظر بگیرید. این روش برای سیستم‌های نوع تبدیلی به‌خوبی کار می‌کند. سیستم‌های نوع تبدیلی می‌توانند با یک شبکه از تبدیل‌ها یا فرایندها، که با جریان‌های داده‌ی ورودی یا خروجی به‌هم وصل شده‌اند، مدل گردند. این بدان معناست که اثر سیستم، تبدیل داده‌ی ورودی اصلی به خروجی نهایی است. سیستم‌های تبدیلی با معماری برنامه‌ی اصلی و زیربرنامه‌ها همبستگی و تناسب خوبی دارند. روش SA/SD برای تحلیل و طراحی سیستم‌های جاسازی‌شده‌ی بلادرنگ مناسب نیست.

---

[۱] Fault Tolerance

[۲] Crash



الف) زیرسیستم تعاملی

ب) زیرسیستم رویدادرانده

ج) زیرسیستم تبدیلی

د) زیرسیستم پایگاه داده

شکل ۶-۲: چهار نوع مختلف سیستم و رفتار

این نوع سیستم، به‌جای رفتار تبدیلی، رفتار وابسته‌به‌حالت از خود نشان می‌دهد. رفتار وابسته‌به‌حالت به یک روش متفاوت طراحی نیاز دارد و به یک معماری متفاوت نرم‌افزار به نام معماری سیستم رویداد-رانده منجر می‌شود.

سیستم‌های تعاملی

یک سیستم تعاملی ویژگی‌های زیر را از خود نشان می‌دهد:

۱. تعامل بین سیستم و کنشگر برای انجام یک فرایند کسب‌وکار، از یک دنباله‌ی تقریباً ثابت از درخواست‌های کنشگر و پاسخ‌های سیستم تشکیل می‌شود. همان‌طور که در شکل ۶-۲-الف نشان داده شده است.
۲. سیستم باید هرگونه در خواست از طرف کنشگر را پردازش‌کرده و به آن پاسخ دهد.
۳. در بیشتر اوقات سیستم‌ها در فرایند مربوط به یک مورد کاربرد، تنها با یک کنشگر تعامل می‌کنند.
۴. کنشگر معمولاً یک انسان است، ولی می‌تواند یک وسیله یا زیرسیستم دیگر نیز باشد.
۵. تعامل از کنشگر شروع و به او ختم می‌شود.
۶. کنشگر وسیستم نوعی رابطه‌ی مشتری-خادم را از خود نشان می‌دهند، یعنی کنشگر خدماتی را درخواست می‌کند و سیستم این خدمات را فراهم می‌نماید. این رابطه‌ی مشتری-خادم، در زمانی که کنشگر یک سیستم دیگر است، بیشتر مشخص می‌شود.
۷. حالت سیستم، پیشرفت فرایند کسب‌وکار که با مورد کاربرد نشان داده شده است را منعکس می‌کند.

به‌دلیل استفاده‌ی گسترده از نرم‌افزارهای رومیزی و مبتنی بر وب، سیستم‌های تعاملی از رایج‌ترین انواع سیستم‌ها هستند. دنباله‌ی تقریباً ثابت تعامل‌ها در این سیستم‌ها می‌تواند از یک فرایند معمول ورود[1] مشاهده شود. برای ورود به یک سیستم،

[1] Log-on

کاربر روی لینک ورود کلیک می‌کند، سیستم صفحه‌ی ورود را نمایش می‌هد، کاربر شناسه و رمزعبور را وارد می‌کند، سیستم صفحه‌ی خوش‌آمدگویی را نمایش می‌دهد و یا در صورت معتبر نبودن شناسه و یا رمزعبور، یک پیغام خطا نشان می‌دهد.

برای مشاهده‌ی این‌که حالت سیستم پیشرفت فرایند کسب‌وکار را منعکس می‌کند، فرایند خرید برخط را درنظر بگیرید. مشتری به سیستم وارد می‌شود، اقلام مورد نظر خود را انتخاب می‌کند و سپس خارج می‌شود. اگر مشتری به سیستم وارد نشده باشد نمی‌تواند از آن خارج شود، زیرا سیستم حالت پیشرفت کار را نگهداری می‌کند. در زمان خروج از سیستم، مشتری باید آدرس ارسال کالا، نوع پرداخت وجه، و اطلاعات مربوط به پرداخت را به‌درستی مشخص نماید. این بار هم حالت سیستم، وضعیت پیشرفت فرایند کسب‌وکار را منعکس می‌نماید. سیستم از این حالت استفاده می‌کند تا اجازه ندهد که مشتری، عملیاتی به جز گام‌های طراحی‌شده برای فرایند کسب‌وکار را انجام دهد.

مدل‌سازی و طراحی سیستم‌های تعاملی، با شناسایی و توصیف موارد کاربرد شروع می‌شود. موارد کاربرد، فرایندهای کسب‌وکار را نشان می‌دهند که توسط یک کنشگر آغاز شده و توسط او پایان می‌یابد. سپس موارد کاربرد در زمان مدل‌سازی تعامل اشیا پالایش می‌شوند تا نشان‌دهند که چگونه اشیای یک سیستم، برای انجام وظایف کسب‌وکار با یکدیگر تعامل می‌کنند. مدل‌سازی تعامل در فصل ۹ مورد مطالعه قرار خواهد گرفت.

سیستم‌های رویداد-رانده

سیستم‌های رویداد-رانده، از خود رفتارهای واکنشی[۱] و وابسته به حالت نشان می‌دهند و دارای ویژگی‌های زیر هستند:

۱. سیستم، رویدادهایی را از موجودیت‌های خارجی دریافت نموده و آن‌ها را کنترل می‌کند.
۲. در حالت عمومی، سیستم‌های رویداد-رانده یک رشته‌ی ثابت از درخواست‌های ورودی را دریافت نمی‌کنند، بلکه درخواست‌ها به‌طور تصادفی به سیستم می‌رسند.
۳. سیستم‌های رویداد-رانده ناچار نیستند که به همه‌ی رویدادهای دریافتی پاسخ دهند. پاسخ آن‌ها وابسته‌به‌حالت است، یعنی یک رویداد می‌تواند بسته به حالت‌های مختلفی که ممکن است سیستم در آن‌ها قرار داشته باشد، به پاسخ‌های متفاوتی منجر گردد. سیستم می‌تواند برخی از رویدادهای ورودی را نیز نادیده بگیرد.
۴. سیستم‌های رویداد-رانده معمولاً در یک زمان با بیش از یک موجودیت خارجی تعامل می‌نمایند.
۵. موجودیت‌های خارجی معمولاً دستگاه‌های سخت‌افزاری و یا نرم‌افزاری به‌غیراز انسان هستند.
۶. حالت سیستم ممکن است پیشرفت محاسبات را منعکس نکند. حالت‌ها ممکن است رخ دادن مجدد برخی شرایط در سیستم را نشان دهند. به عنوان مثال، سیستم می‌تواند در حالت بیکار[۲] باشد، سپس به حالت پردازش رویداد A وارد شود، و دوباره به حالت بیکار برگردد.
۷. ممکن است سیستم ناچار باشد محدودیت‌های زمان‌بندی، زمانی، و زمان‌بندی موقتی را برآورده سازد. مثال‌هایی از این نوع محدودیت‌ها را در ادامه می‌بینید:

- زمان پردازش برای هر رقمی که شماره‌گیری می‌شود، نباید از N میلی‌ثانیه بیشتر شود (یک محدودیت زمانی[۳]).
- پس از ارسال تعداد مشخصی بوق تلفن، پیغام ضبط شده روی پیغام‌گیر پخش شود (یک محدودیت زمانی که دنباله‌ی مورد انتظاری از رویدادها را مشخص می‌کند).
- پس از رساندن سیگنال فشردن برای صحبت[۴]، شماره‌ی شناسایی شبکه[۵] باید در N میلی‌ثانیه ارسال شود (یک محدودیت زمانی زمان‌بندی‌شده، که یک محدودیت زمانی بین دو رویداد را نشان می‌دهد).

---

[۱] Reactive
[۲] Idle
[۳] Temporal
[۴] Push to talk-PTT
[۵] Network Identification

زیرسیستم‌های رویداد-رانده اغلب در سیستم‌های جاسازی‌شده دیده می‌شوند، یعنی در جایی که زیرسیستم نرم‌افزاری درون یک سیستم کلی متشکل از سخت‌افزار و نرم‌افزار قرار گرفته است. زیرسیستم نرم‌افزاری رویدادهایی که به‌طور تصادفی از جانب دستگاه‌ها می‌رسد را دریافت می‌کند، رویدادها را تفسیر و پردازش می‌نماید، و دستوراتی را برای به‌کار انداختن و یا کنترل دستگاه‌های مناسب صادر می‌کند. همان‌طور که در شکل ۶-۲-ب دیده می‌شود، سیستم‌های رویداد-رانده با نمودارهای حالت مدل می‌شوند. a، b،و c رویدادهایی که به‌طور تصادفی می‌رسند را نشان می‌دهند، و x، y، و z پاسخ‌های متناظر سیستم برای کنترل دستگاه‌ها را نشان می‌دهند. مدل‌سازی و طراحی زیرسیستم‌های رویداد-رانده، عناوین مورد بحث در فصل ۱۳ (مدل‌سازی حالت شئ برای سیستم‌های رویداد-رانده) می‌باشند.

سیستم‌های تبدیلی

سیستم‌های تبدیلی معمولاً ویژگی‌های زیر را از خود نشان می‌دهند:

۱. سیستم‌های تبدیلی می‌توانند متشکل از یک شبکه از فعالیت‌های پردازش اطلاعات در نظر گرفته شوند که هرکدام ورودی خود را، مانند آن‌چه در شکل ۶-۲-ج نشان داده شده است، به خروجی تبدیل می‌کنند.
۲. شبکه‌ی فعالیت‌ها می‌تواند با جریان‌های کنترلی درگیر باشد که علاوه بر رفتارهای همگام و ناهمگام[۱]، عملیات ترتیبی، انشعاب شرطی، و موازی نیز از خود نشان می‌دهند.
۳. عملیات تبدیل ورودی به خروجی کم‌وبیش یک عملیات دسته‌ای[۲] است و در خلال آن تعامل قابل‌توجهی بین سیستم و کنشگر وجود ندارد.
۴. سیستم‌های تبدیلی معمولاً بدون حالت[۳] هستند.
۵. سیستم‌های تبدیلی ممکن است به الگوریتم‌هایی با محاسبات بسیار زیاد و سنگین نیاز داشته باشند.
۶. کنشگر یک سیستم تبدیلی می‌تواند انسان، دستگاه‌های جانبی و یا سیستم‌های دیگر باشد.

سیستم‌های تبدیلی اغلب اوقات در محاسبات علمی مهندسی دیده می‌شوند، ولی می‌توانند در کاربردهای حرفه‌ای مثل مدیریت جریان کار[۴] و داده‌کاوی در کسب‌وکار نیز به چشم بیایند. یک مثال ساده و آشنا کمپایلر است که طی چهار مرحله، یک برنامه‌ی مبداء را به کد قابل اجرا تبدیل می‌کند. این مراحل عبارتند از: (۱) تحلیل لغوی[۵]، که برنامه‌ی مبدأ را به یک رشته کلمه‌ی توکن[۶]، مانند شناسه‌ها، عملگرها[۷]، عملوندها[۸] و کلمات ذخیره‌شده، تبدیل می‌کند. (۲) تحلیل نحوی[۹]، که رشته‌ی کلمات را به جملات برنامه تبدیل می‌کند. (۳) تولید کد، که برای هر جمله‌ی برنامه کد قابل اجرا تولید می‌کند. (۴) بهینه‌سازی کد که کارایی و کارآمدی کد قابل اجرا را بهبود می‌بخشد. در طول این فرایند، کنشگر و کامپایلر تعامل قابل‌توجهی ندارند.

یک مثال پیچیده‌تر مدیریت جریان کار است که در آن درخواست‌ها یا کارها ممکن است از مراحل متعدد پردازشی که توسط بخش‌های مختلف انجام می‌شود، عبور کنند، و ممکن است به تصمیم‌گیری، همگام‌سازی و پردازش همزمان نیاز داشته باشند. در هر بخش، کارمندان ممکن است با زیرسیستمی که فعالیت‌های آن بخش را خودکارسازی می‌نماید، تعامل داشته باشند. به‌این‌ترتیب این زیرسیستم یک زیرسیستم تعاملی از سیستم تبدیلی خواهد بود. مدل‌سازی و طراحی سیستم‌های تبدیلی در فصل ۱۴ (مدل‌سازی فعالیت برای سیستم‌های تبدیلی) خواهد بود.

---

[۱] Asynchronous
[۲] Batch
[۳] Stateless
[۴] WorkFlow Management
[۵] Lexical Analysis
[۶] Token
[۷] Operators
[۸] Operands
[۹] Syntax Analysis

زیرسیستم‌های ماندگاری شئ

زیرسیستم ماندگاری شئ زیرسیستمی است که قابلت‌هایی برای ذخیره و بازیابی اشیا از یک پایگاه‌داده یا سیستم فایل را فراهم می‌سازد، به‌طوری‌که رسانه‌ی ذخیره‌سازی پنهان باشد. به‌دلیل کاربرد آن در یک پایگاه‌داده، معمولاً برای سادگی به چنین سیستمی سیستم پایگاه‌داده گفته می‌شود. یک زیرسیستم پایگاه‌داده ویژگی‌های زیر را از خود نشان می‌دهد:

۱. پایگاه‌داده را از کل سیستم پنهان کرده، سیستم را از تغییرات در پیاده‌سازی پایگاه‌داده محافظت می‌کند.
۲. برخلاف سه نوع زیرسیستم دیگر، زیرسیستم پایگاه‌داده تنها مسئول ذخیره و بازیابی اشیایی از پایگاه‌داده است. این زیرسیستم، به جز در موارد معدود، پردازش قابل‌توجهی برای کسب‌وکار مربوطه انجام نمی‌دهد و بنابراین در مواردی مشابه وقتی تعداد زیادی رکورد باید به‌روز شوند، می‌تواند کارایی را ارتقا دهد.
۳. یک زیرسیستم پایگاه‌داده قادر است حجم عظیمی از داده‌های ساخت‌یافته و پیچیده را به شکل کارا ذخیره، بازیابی، و اصلاح نماید.

سیستم و زیرسیستم

مفاهیم سیستم و زیرسیستم به‌هم مربوط هستند. یک سیستم، زیرسیستمی از یک سیستم بزرگ‌تر است. یک زیرسیستم برای خودش یک سیستم است و ممکن است از انواع مختلفی از زیرسیستم‌ها تشکیل شده باشد. برای مثال یک سیستم تعاملی می‌تواند شامل یک زیرسیستم رویداد-رانده، یک زیرسیستم تبدیلی، و یک زیرسیستم پایگاه‌داده باشد. یک زیرسیستم رویداد-رانده ممکن است شامل یک زیرسیستم تعاملی، یک زیرسیستم تبدیلی، و یک زیرسیستم پایگاه‌داده باشد. این بدان معناست که باید برای هر نوع زیرسیستم تحت توسعه، سبک طراحی متناسب با آن انتخاب شود.

۶-۳-۳- استفاده از سبک‌های معماری

در بخش ۶-۳-۲، از بین همه‌ی انواع سیستم‌های کاربردی، چهار نوع آن‌ها معرفی شد. در این بخش نشان داده شد که انواع مختلف سیستم‌ها، به معماری‌های متفاوت نرم‌افزار نیازمندند. بنابراین انتخاب یک سبک معماری که با سیستم تحت‌توسعه سازگار باشد، از اهمیت ویژه‌ای برخوردار است. یک سبک معماری، یک طراحی معماری عمومی است که می‌تواند برای یک سیستم پذیرفته شود و یا با آن سازگار گردد. در این بخش تعدادی از سبک‌های معماری شرح داده می‌شود و ارتباط این سبک‌ها با انواع سیستم‌ها مورد بررسی قرار می‌گیرد. برخی از سبک‌های معماری، الگوی طراحی نیز هستند که در فصل‌های آینده مانند فصل‌های ۱۰، ۱۳، ۱۵، ۱۶، ۱۷، و ۲۱ شرح داده شده‌اند. سبک‌های معماری مطرح‌شده در این بخش عبارتند از:

۱. معماری Nلایه. این سبک معماری، اجزای سیستم را به لایه‌های نسبتاً مستقل با اتصال ضعیف، مرتب می‌نماید. هر لایه یک وظیفه و عملکرد خوش‌تعریف دارد و تأثیرات بر لایه‌های دیگر را کاهش می‌دهد. این سبک معماری برای طراحی سیستم‌های تعاملی مناسب است.
۲. معماری مشتری-خادم. این سبک معماری از یک سرویس‌دهنده (خادم) تشکیل می‌شود که خدماتی را برای تعدادی مشتری فراهم می‌سازد. مشتری‌ها خادم را می‌شناسند، خادم مشتری‌ها را نمی‌شناسد، و مشتری‌ها یکدیگر را نمی‌شناسند. به‌این‌ترتیب این معماری اتصال خادم و مشتری را کاهش می‌دهد.
۳. معماری برنامه‌ی اصلی و زیربرنامه‌ها. این سبک معماری اجزای سیستم را در قالب یک ساختار درختی سازمان‌دهی می‌کند که در آن محاسبات از ریشه یا برنامه‌ی اصلی شروع می‌شود و به‌طور بازگشتی و به طرف پایین درخت، توسط فرزندان ریشه انجام می‌گیرد. این سبک معماری برای طراحی سیستم‌های تبدیلی و یا مبتنی بر جریان‌کار مناسب است.

۴. معماری سیستم رویداد-رانده. این سبک معماری از یک کنترلگر مبتنی بر حالت تشکیل می‌شود که با تعدادی از اشیا تعامل نموده و آن‌ها را کنترل می‌کند. کنترلگر اجزا را می‌شناسد و اجزا نیز او را می‌شناسند، ولی اجزا یکدیگر را نمی‌شناسند. تعامل بین اجزا با میانجی‌گری کنترلگر انجام می‌شود. این سبک معماری برای طراحی سیستم‌های جاسازی‌شده و رویداد-رانده مناسب است.
۵. معماری چارچوب ماندگاری. این سبک معماری، پایگاه‌داده و سیستم فایل را از دید اشیایی که از آن‌ها استفاده می‌کنند مخفی نگاه می‌دارد. یعنی اشیا از وجود چنین دستگاه‌های ذخیره‌سازی مطلع نیستند، و همه‌ی تغییرات صورت‌گرفته برروی پایگاه‌داده و سیستم فایل اثری بر اشیا نخواهد گذاشت.

انواع سیستم و سبک معماری

همان‌طور که پیشتر گفته شد، انواع مختلف سیستم‌ها با سبک‌های مختلف معماری در ارتباط می‌باشند. این مطلب در شکل ۶-۳ نشان داده شده است. در این شکل، ویژگی‌های انواع سیستم و چگونگی تفاوت آن‌ها با انواع دیگر را نیز نشان می‌دهد. جدول یادشده، برای مشخص‌کردن یک سبک معماری با توجه به نوع سیستم درحال توسعه به‌کار می‌رود. در سال‌های اخیر، به‌دلیل توجه بیشتر به امنیت کامپیوترها، الگوهای امنیتی و سبک معماری امنیتی رواج یافته است. الگوهای امنیتی و سبک معماری در فصل ۲۴ مورد بررسی قرار گرفته است.

سبک معماری Nلایه

همان‌طور که در شکل ۶-۴-الف دیده می‌شود، در معماری Nلایه سیستم به تعدادی لایه‌ی سطح‌بندی‌شده یا طبقه تقسیم می‌گردد. درخواست‌های خدمت از یک لایه به لایه‌ی دیگر فرستاده می‌شود. ارسال درخواست از لایه‌ی پایین‌تر به لایه‌ی بالاتر مجاز نیست. شکل ۶-۴-ب مثالی را نشان می‌دهد که در آن کلاس‌ها و اشیا به پنج لایه دسته‌بندی شده‌اند. لایه‌ی واسط گرافیکی کاربر یک گروه از اشیا است که مسئول نمایش اطلاعات، منوها، و دکمه‌های عملیاتی به کاربر می‌باشند. این اشیا، عملیاتی از یک شئ کنترلگر در لایه‌ی کنترلگر را فعال می‌کنند. هر کنترلگر، مسئول برخورد با رویدادهای مربوط به یک مورد کاربرد مشخص می‌باشد. در بیشتر موارد یک تناظر یک‌به‌یک بین موردهای کاربرد و اشیای کنترلگر برقرار است و کلاس‌های کنترلگر، بعد از موردهای کاربرد نام‌گذاری می‌شوند. برای مثال یک سیستم اطلاعاتی کتابخانه، یک مورد کاربرد مربوط به خروج سند[1] دارد. این بدان معناست که یک کلاس کنترلگری خروج سند وجود دارد. اشیای این کلاس در زمان ورود اطلاعات از سوی کاربر و فشرده شدن یک کلید خاص، مسئول برخورد با رویدادهای مربوط به خروج سند می‌باشند.

وقتی کاربر دکمه‌ای را برای ارسال یک درخواست کلیک می‌کند، لایه‌ی GUI این درخواست را به کنترلگر مناسب مورد کاربرد منتقل می‌نماید. برای مثال اگر کاربر بخواهد یک سند را برگرداند، در خواست او توسط GUI به Checkout Document Controller فرستاده می‌شود. این شئ از لایه‌ی پایگاه‌داده درخواست می‌کند که اشیای User Document مربوطه را از پایگاه‌داده بازیابی کند. کنترلگر، شئ Loan را ایجاد می‌کند و حالت شئ Document را به حالت «خارج‌شده» تغییر می‌دهد. سپس این اشیا از طریق لایه‌ی پایگاه‌داده، در پایگاه‌داده ذخیره می‌شوند. اگر نیازی به ارتباط شبکه باشد، مثلاً اگر پایگاه‌داده در یک سایت دور قرار داشته باشد، لایه‌ی ارتباط شبکه نیز درگیر می‌شود.

معماری Nلایه که در شکل ۶-۴-ب نشان داده شده است، معمولاً به‌عنوان طراحی معماری برای سیستم‌های تعاملی مورد استفاده قرار می‌گیرد. چنین سیستم‌هایی طی موارد کاربردی مشابه مورد کاربرد Checkout-Document که قبلاً شرح داده شد، با کاربر تعامل می‌نمایند. معماری Nلایه، اشیای سیستم تعاملی را به لایه‌هایی تقسیم می‌کند و به هرکدام تعدادی مسئولیت

[1] Checkout Document

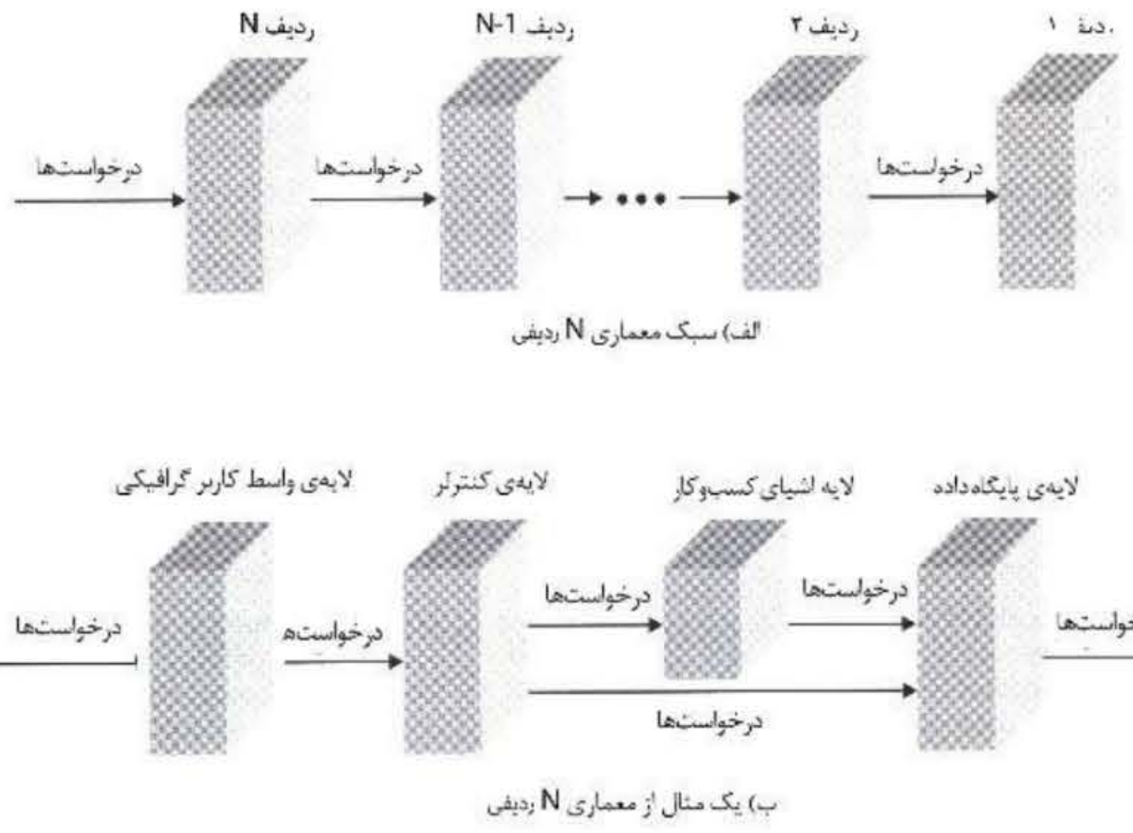
انواع سیستم/ زیرسیستم	ویژگی‌های برجسته‌ی زیرسیستم	سبک معماری	تفاوت‌ها
تعاملی	کنشگر، سیستم را به‌راه‌می‌اندازد و با آن تعامل می‌کند تا با استفاده از پروتکل‌های از پیش تعیین شده‌ی تعاملی، وبه‌همراه هم یک فرایند کسب‌وکار را انجام دهند.	چند لایه‌ای	در موارد زیر با سیستم رویداد-رانده متفاوت است: ۱) سیستم درهر جلسه با تنها یک کنشگر در تعامل است. ۲) رویدادهای سیستم با ترتیب ازپیش‌تعیین‌شده می‌رسند. ۳) سیستم باید به همه‌ی رویدادهای سیستم پاسخ دهد.
رویداد-رانده	سیستم از خود رفتارهای واکنشی وابسته به حالت نشان می‌دهد.سیستم رویدادهای دریافتی از موجودیت های خارجی را پردازش می‌کند و دستوراتی را برای موجودیت‌های خارجی صادر می‌کند.	معماری سیستم رویداد-رانده	با شیوه‌ی حل ابتکاری مسئله و مشتری-خادم و ماندگاری شئ در موارد زیر متفاوت است: ۱) رفتار واکنشی وابسته به حالت. ۲) موجودیت‌های خارجی می‌توانند با هم همکاری بکنند یا همکاری نکنند.
تبدیلی	یک مجموعه فعالیت‌های مرتبط پردازش‌اطلاعات، برای یک ورودی مشخص خروجی ایجاد می‌کند، یعنی ورودی به خروجی تبدیل می‌شود.	برنامه‌ی اصلی و زیربرنامه	با سایر سیستم‌ها در موارد زیر متفاوت است: ۱) تاکید بر یک مجموعه فعالیت‌های پردازش اطلاعات. ۲) تبدیل ورودی به خروجی
ماندگاری شئ	قابلیت ماندگاری اشیا و اشتراک‌گذاری بین کاربردهای مختلف را به همراه حفاظت نمودن اشیای کسب وکار در مقابل تغییرات در پیاده‌سازی پایگاه داده فراهم می‌نماید.	چارچوب ماندگاری	با حل ابتکاری مسئله در موارد زیر متفاوت است: ۱) قابلیت ماندگاری اشیا و به اشتراک‌گذاری را فراهم می‌کند. ۲) با تعداد زیاد رکوردها سر وکار دارد. ۳) پایگاه‌داده را از دید اشیای کسب‌وکار پنهان می‌کند.
مشتری-خادم	یک زیرسیستم اختصاصی با کارآیی بالا، خدماتی را که از جانب زیرسیستم‌ها یا اجزای دیگر از طریق شبکه درخواست می‌شوند را فراهم می‌نماید.	مشتری - خادم	با حل ابتکاری مسئله در موارد زیر متفاوت است: ۱) خادم،کار درخواست‌شده را انجام می‌دهد. ۲) مشتری‌ها چیزی را به اشتراک نمی‌گذارند. ۳) مشتری‌ها به دلایل متفاوتی از خدمات استفاده می‌کنند.
توزیع شده، غیرمتمرکز	زیرسیستم‌های توزیع‌شده و مستقل، از طریق یک شبکه با هم در ارتباطند تا از خدمات یکدیگر استفاده کنند و به یکدیگر خدمت ارائه کنند.	نظیر به نظیر	با مشتری-خادم و تخته سیاه در موارد زیر متفاوت است: ۱) مشتری‌ها و خادم ها نظیر یکدیگر هستند. ۲) طرف‌های نظیر، مشکلات شخصی خود را حل می‌کنند.
حل ابتکاری مسائل	اجزای هوشمند حل‌کننده‌ی مسئله، از طریق یک انباره‌ی داده‌ی مشترک با یکدیگر ارتباط و همکاری می‌نمایند تا راه‌حلی برای یک مسئله‌ی مشخص پیدا کنند.	تخته سیاه	با مشتری-خادم در موارد زیر متفاوت است: ۱) اجزا کار را انجام می‌دهند. ۲) تخته سیاه تنها یک انباره‌ی داده‌ای مشترک است. ۳) اجزا با کمک یکدیگر مسئله را حل می‌کنند.

شکل ۶-۳: نگاشت انواع سیستم به سبک‌های معماری

مرتبط اختصاص می‌دهد. برای مثال لایه‌ی GUI برای نمایش اطلاعات و اجزای GUI، لایه‌ی کنترلگر برای رسیدگی‌کردن به رویدادهای مربوط به موارد کاربرد، لایه‌ی اشیای کسب‌وکار به‌عنوان مجموعه‌ای از اشیای کسب‌وکار، لایه‌ی پایگاه‌داده برای ذخیره و بازیابی اشیا با یک پایگاه‌داده، و لایه‌ی ارتباطات شبکه برای ارتباط با یک سایت دور مورد استفاده قرار می‌گیرد.

معماری Nلایه، محدود به سیستم‌های تعاملی نیست. برای مثال، مدل هفت‌لایه‌ی ISO یک معماری هفت‌لایه است که برای ارتباطات شبکه به‌کار می‌رود. همچنین معماری Nلایه در طراحی سیستم‌های عامل و سیستم‌های ایمن نیز استفاده می‌گردد. یک سبک طراحی سیستم‌های ایمن، از هر لایه برای تصدیق دسترسی به لایه‌ی بعدی پایین‌تر استفاده می‌کند. این کار، در مقابل یک مهاجم، N مانع ایجاد می‌کند و داده‌های ارزشمند و منابع برنامه را حفاظت می‌نماید. معماری چندلایه‌ی فوق قوانین متعددی را برای طراحی نرم‌افزار تدوین می‌نماید که در بخش ۶-۵ نشان داده شده است.

برای مثال، قانون «جداسازی دغدغه‌ها»[۱] به معنای گروه‌بندی اشیای مربوط به دغدغه‌های مختلف در سیستم، به لایه‌های مختلف است . نتیجه‌ی این کار، چسبندگی بالا و طراحی اشیای نادان[۲] است. قوانین «پنهان‌سازی اطلاعات» و «طراحی برای تغییر» با این واقعیت پشتیبانی می‌شوند که هر لایه، لایه‌ی پایین‌تر را مخفی می‌کند و آن را از تاثیر تغییرات حفاظت می‌نماید.



شکل ۶-۴: معماری چندلایه

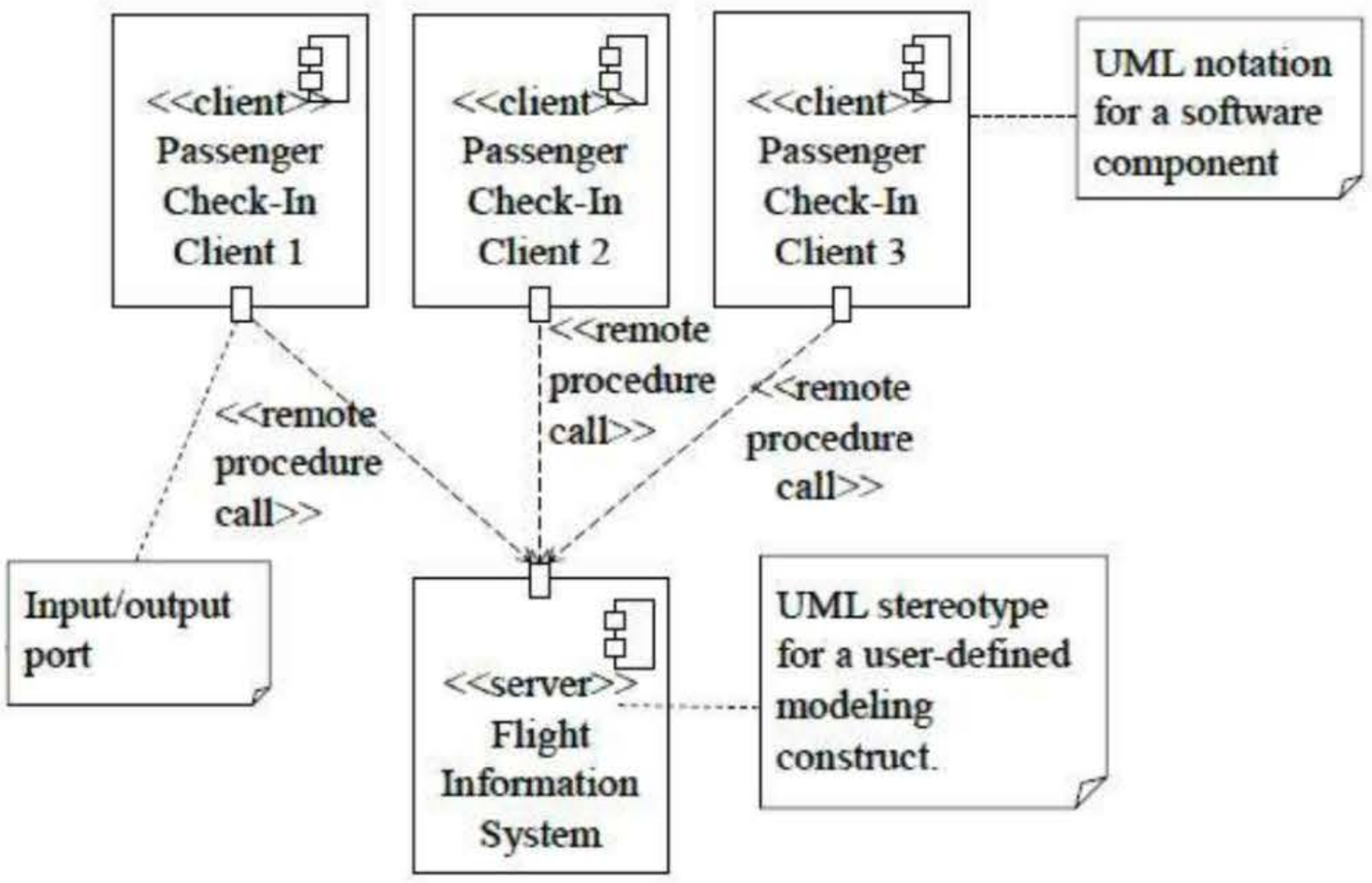
سبک معماری مشتری-خادم

همان‌طور که در شکل ۶-۵ نشان داده شده است، معماری مشتری-خادم از یک سرویس‌دهنده و تعدادی مشتری تشکیل می‌گردد. این نمودار، طراحی معماری برای بخشی از سیستم تحویل چمدان‌ها در فرودگاه (ABHS) که در فصل ۳ معرفی شد را نشان می‌دهد. ABHS، دارای کامپیوترهایی برای پذیرش مسافر است که در پایانه‌های مختلف فرودگاه قرار گرفته‌اند. این کامپیوترها باید برای بازیابی اطلاعات پرواز و به‌روزرسانی وضعیت پذیرش و پرواز مسافر، به سیستم اطلاعات پرواز دسترسی داشته باشند. در این حالت، کامپیوترهای پذیرش مسافر، مشتری‌ها هستند و سیستم اطلاعات پرواز، خادم است. مشتری‌ها درخواست‌های خدمت را از طریق فراخوانی از راه‌دور[۳] و یا پروتکل‌های ارتباطی دیگر به خادم می‌فرستند. خادم درخواست‌ها را پردازش می‌کند و نتایج را به مشتری‌ها ارسال می‌کند. مشتری‌ها خادم را می‌شناسند ولی خادم مشتری‌ها را نمی‌شناسد. این بدان معنی است که مشتری‌ها می‌توانند آزادانه و بدون تحت‌تاثیر قراردادن خادم (به جز اثر بر کارایی آن) به سیستم اضافه و یا از آن حذف شوند. معمولاً مشتری‌ها و خادم بر روی ماشین‌های مختلف در محل‌های متفاوت مستقر می‌باشند.

معماری مشتری-خادم برای کاربردهایی مفید است که به یک زیرسیستم برگزیده برای سرویس‌دهی به سایر زیرسیستم‌ها، که ممکن است بر روی ماشین‌های دیگر قرار گرفته باشند، نیاز دارند. اگر بیشتر سیستم‌های لازم برروی خادم قرار گیرد، مشتری‌ها می‌توانند سبک‌وزن ساخته شوند. برای مثال، در ABHS مشتری‌های پذیرش مسافر مشتری‌های سبک‌وزنی هستند. به جز ABHS مثال‌های بسیار دیگری از معماری مشتری-خادم را نیز می‌توان در کاربردهای جهان واقعی پیدا کرد. اغلب کاربردهای مبتنی‌بروب، سرویس‌دهنده‌های فایل، و پروتکل‌های انتقال فایل[۴] از جمله‌ی این نمونه‌ها می‌باشند.

---

[۱] Separation of Concerns
[۲] Stupid Object
[۳] Remote Procedure Call
[۴] File Transfer Protocol (FTP)



شکل ۶-۵: به‌کاربردن سبک معماری مشتری-خادم

سبک معماری برنامه‌ی اصلی و زیربرنامه‌ها

همان‌طور که در شکل ۶-۶-الف نشان داده شده است، معماری برنامه‌ی اصلی و زیربرنامه‌ها شکل درختی دارد. این ساختار، ازیک برنامه‌ی اصلی و تعدادی زیربرنامه تشکیل شده است. برنامه‌ی اصلی زیربرنامه‌های فرزند خود را فراخوانی می‌کند و این زیربرنامه‌ها نیز به نوبه‌ی خود زیربرنامه‌های سطوح پایین‌تر را فراخوانی می‌نمایند. معمولاً زیربرنامه‌های فرزند، از چپ به راست فراخوانی می‌گردند. اگر دو گره‌ی والد، یک زیربرنامه‌ی فرزند یکسان را فراخوانی کنند، معماری برنامه‌ی اصلی و زیربرنامه‌ها شکل شبکه‌ای خواهد داشت. افرادی که با یک زبان زیربرنامه‌ای مانند C یا فرترن برنامه‌نویسی کرده باشند، با این برنامه آشنا هستند.

معماری برنامه‌ی اصلی و زیربرنامه‌ها معمولاً برای طراحی معماری سیستم‌های تبدیلی مورد استفاده قرار می‌گیرد. معماری از یک نمودار جریان‌داده در SA/SD گرفته می‌شود. برای روشن‌کردن موضوع، بندهای زیر نشان می‌دهد که چگونه معماری شکل ۶-۶ -الف از نمودار جریان‌داده‌ی شکل ۶-۶ -ب به‌دست می‌آید. قبل از آن، بهتر است بدانیم که فرایندهای یک نمودار جریان‌داده می‌توانند به سه گروه اصلی دسته‌بندی گردند:

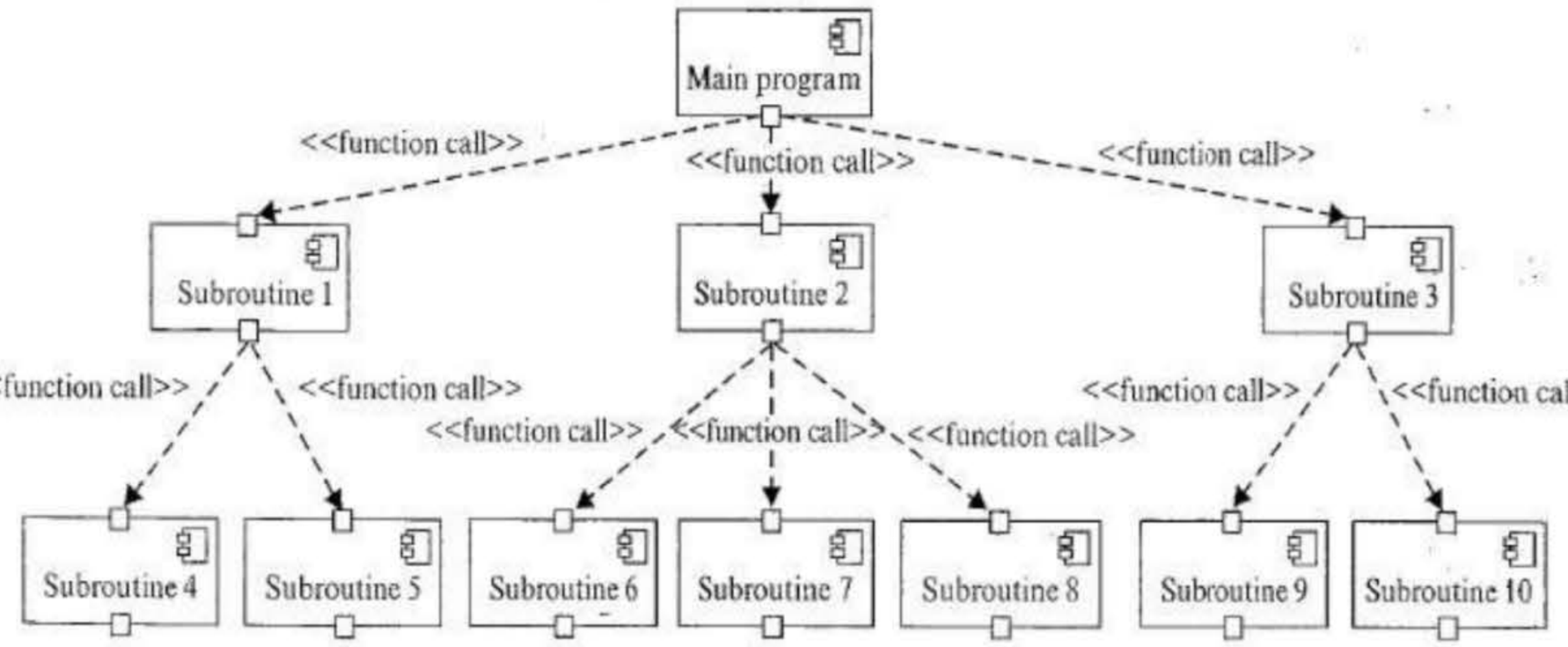
فرایندهای قالب‌بندی[۱]. هریک از آن‌ها ورودی خود را برای ایجاد یک خروجی با فرمت متفاوت، تغییر می‌دهد. برای مثال، یک شکل ارسال‌شده‌ی HTML به یک Struct تبدیل می‌شود. فرایندهای قالب‌بندی فقط ظاهر را تغییر می‌دهند، نه محتوا را.

فرایندهای تبدیل[۲]. هریک از آن‌ها یک پردازش مرتبط با کاربرد انجام می‌دهد که ورودی را به یک خروجی که از نظر معنایی متفاوت است، تبدیل می‌کند. برای مثال ورودی یک شرط، یک جستجو است و خروجی، یک فهرست از محصولاتی که شرط جستجو را برآورده می‌سازند.

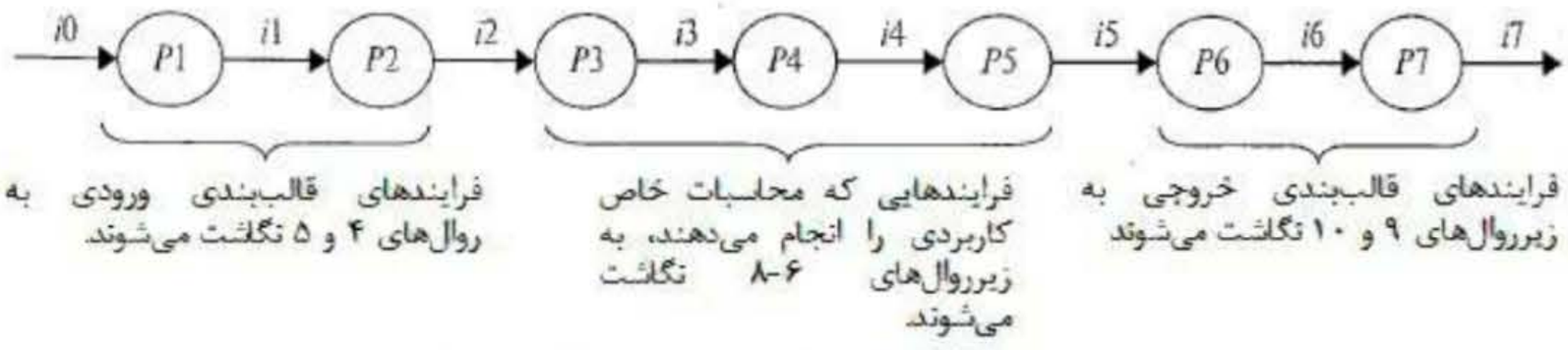
---

[۱] Formatting Processes

[۲] Transform Processes



الف) معماری یک برنامه اصلی و زیرروال‌ها



ب) یک مدل پردازش داده و نگاشت‌های آن به طراحی معماری

شکل ۶-۶: معماری برنامه‌ی‌اصلی و زیربرنامه‌ها

فرایندهای ارسال[۱]. یک فرایند ارسال، ورودی خود را تحلیل می‌کند و بسته به نوع تراکنش، ورودی را به یکی از فرایندهای بعدی که انواع متفاوتی از تراکنش‌ها را مدیریت می‌نماید، ارسال می‌کند.

استخراج معماری برنامه‌ی اصلی و زیربرنامه‌ها از شکل ۶-۶-ب از ورودی اولیه‌ی i0 شروع می‌شود و فرایندهای P1, P2,... را طی می‌کند تا یک فرایند ارسال یا تبدیل رخ دهد. برای مثال، فرض کنید که فرایند P3 یک تبدیل است. ردیابی تا جایی ادامه می‌یابد که یک فرایند قالب‌بندی مانند P6 پیدا شود. به‌این‌ترتیب ردیابی، فرایندهای قالب‌بندی ورودی P1 و P2، فرایندهای تبدیلی P3، P4 و P5، و فرایندهای قالب‌بندی خروجی P6 وP7 را شناسایی می‌نماید. این فرایندها به سطح پایین‌ترین زیربرنامه درشکل ۶-۶-الف نگاشت می‌گردند. توجه کنید که زیربرنامه‌های ۱ تا ۳ زیربرنامه‌های راه‌انداز[۲] هستند که زیربرنامه‌های قالب‌بندی ورودی، تبدیل، و قالب‌بندی خروجی را فعال می‌سازند. نمودار جریان‌داده‌ی شکل ۶-۶-ب، یک نمودار جریان‌داده‌ی تراکنش‌محور، نموداری است که با یک فرایند ارسال در ارتباط است. این فرایند ارسال، جریان‌های داده‌ی خروجی دارد که به فرایندهای تبدیلی وارد می‌شوند که انواع گوناگون تراکنش‌ها را انجام می‌دهند. استخراج یک معماری برنامه‌ی اصلی و زیربرنامه، از یک نمودار جریان‌داده‌ی تراکنش-محور مشابه است، با این تفاوت که فرایند ارسال، به برنامه‌ی اصلی یا زیربرنامه‌ای نگاشت می‌شود که بسته به نوع تراکنش یکی از زیربرنامه‌های فرزند خود را فرخوانی می‌نماید.

برای مثال طراحی یک محیط توسعه‌ی یکپارچه IDE را درنظر بگیرید. کار اصلی IDE شامل ترجمه‌ی برنامه‌های مبدا و مهندسی معکوس کد منبع برای تولید نمودارهای UML است. کاربر می‌تواند نمودارهایی که باید تولید شوند را انتخاب نماید.

[۱] Dispatching Processes

[۲] Driver

برای سادگی تنها یک زبان برنامه‌نویسی درنظر گرفته می‌شود. برای ترجمه‌ی کد منبع برای تولید کد قابل‌اجرا، کامپایلر عملیات زیر را انجام می‌دهد که توسط چهار مولفه‌ی نرم‌افزاری پیاده می‌شوند.

۱. تحلیل لغوی. این کار، کلمات برنامه‌ی مبدأ را تحلیل می‌نماید و آن‌ها را به انواع مختلف توکن‌ها دسته‌بندی می‌کند . برای مثال، «1stYear» معتبر نیست و «Year1»، «Static»، و «++» به‌عنوان شناسه، کلمه‌ی رزروشده و عملگر شناسایی می‌شوند. خروجی این کار یک رشته از توکن‌ها است که هریک دارای نوع، مقدار، و ویژگی‌های دیگر هستند. خروجی به تحلیل نحوی می‌رود که در ادامه شرح داده می‌شود.
۲. تحلیل نحوی. این کار ترکیب جملات برنامه را تحلیل می‌کند و یک درخت ترکیب برای نمایش برنامه ایجاد می‌نماید. خروجی تحلیل نحوی، ورودی تولید کد است.
۳. تولید کد. این کار یک پیمایش پس‌ترتیب[1] روی درخت ترکیب انجام می‌دهد و با مشاهده‌ی هرگره از درخت، کد اجرایی را تولید می‌کند. برای مثال، وقتی گره‌های «a» و «b» درهنگام پیمایش درخت دیده می‌شوند، مقادیر آن‌ها به نحوی بارگذاری می‌شوند. وقتی گره‌ی «+» دیده می‌شود، یک عمل جمع بر روی مقادیر دو فرزندش اجرا می‌گردد و حاصل ذخیره می‌شود. خروجی این مرحله به بهینه‌سازی کد می‌رود که در ادامه شرح داده می‌شود.
۴. بهینه‌سازی کد. این کار با استفاده از روش‌های بهینه‌سازی کد، کد اجرایی تولیدشده را بهبود می‌بخشد. کد بهینه‌شده حاصل کار کامپایلر است.

عملیات مهندسی معکوس نیز مشابه است، با این تفاوت که تولید کد و بهینه‌سازی کد، با تولید نمودار و نمایش نمودار جایگزین می‌گردد. بااین‌وجود، برای هر نوع از نمودارهای UML مورد پشتیبانی، یک ایجادکننده‌ی نمودار و یک نمایش‌دهنده‌ی نمودار وجود دارد. شکل ۶-۷-الف یک نمودار جریان‌داده برای IDE شرح داده شده را نشان می‌دهد. طراحی معماری برنامه‌ی‌اصلی و زیربرنامه برای این IDE نیز در شکل ۶-۷-ب دیده می‌شود. توجه داشته باشید که تنها یک ایجادکننده و یک نمایش‌دهنده‌ی نمودار در این شکل نشان داده شده است.

معماری سیستم رویداد-رانده

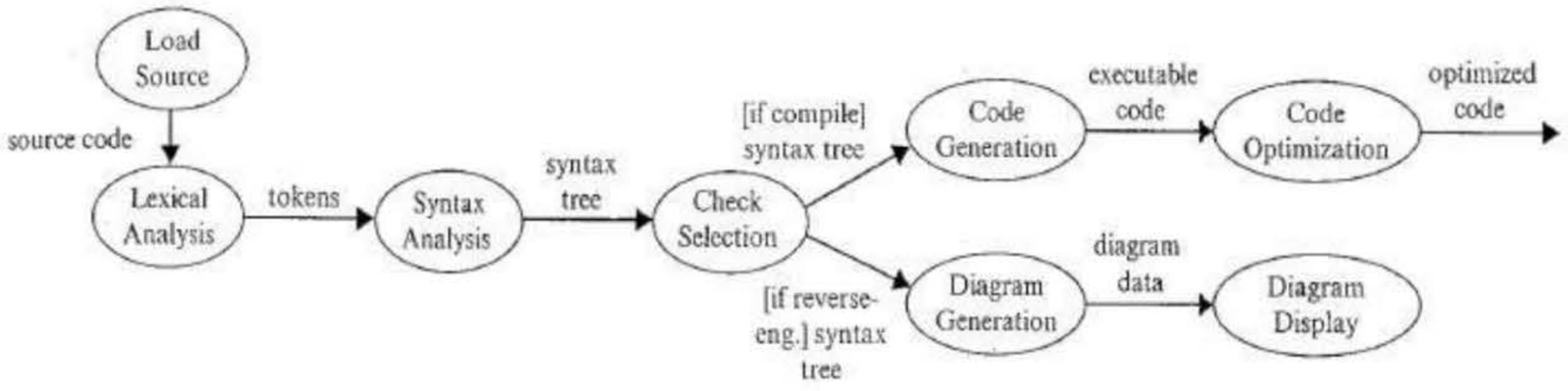
معماری سیستم رویداد-رانده[2] از یک کنترلگر مبتنی بر حالت و تعدادی مولفه‌ی تحت کنترل آن تشکیل می‌گردد. شکل ۶-۸ چنین معماری را نشان می‌دهد. همان‌طور که پیش‌تر گفته شد، یک سیستم رویداد-رانده رفتار وابسته‌به‌حالت از خود نشان می‌دهد. این رفتار توسط کنترلگر پیاده‌سازی می‌گردد. مولفه‌های تحت‌کنترل، رویدادها را به کنترلگر ارسال می‌کنند. کنترلگر، رویدادها را بسته به رفتار حالت پردازش می‌کند و دستورات را به مولفه‌های تحت‌کنترل می‌فرستد.

شکل ۶-۹ طراحی معماری سیستم نرم‌افزاری برای یک دستگاه تهویه‌ی هوای ابتدایی را نشان می‌دهد. نرم‌افزار، یک ماشین‌حالت ساده را پیاده‌سازی می‌نماید. این ماشین، رویدادهایی را از کلید واحد AC و حسگر حرارت اتاق می‌گیرد و دستوراتی را برای کنترل پنکه و کندانسور صادر می‌کند. کندانسور هوا را خنک می‌کند و پنکه هوا را برای خنک‌کردن اتاق به‌جریان می‌اندازد. نرم‌افزار بر روی یک تراشه‌ی مدار مجتمع میکروکنترلگر IC ذخیره و اجرا می‌گردد. تراشه‌ی IC دارای پین‌های ورودی و خروجی است. پین‌های ورودی به کلید واحد AC و حسگر دمای اتاق متصل هستند. پین‌های خروجی به پنکه و کندانسور متصل می‌باشند. هرپین، یک بیت را نشان می‌دهد که مقدار آن با ولتاژ مشخص می‌گردد و به‌عنوان مثال، ولتاژ بالا معادل ۱ و ولتاژ پایین معادل ۰ است. با خواندن و مقداردهی پین‌های تراشه‌ی IC نرم‌افزار داده‌ها یا رویدادهای ورودی را می‌گیرد و برای توقف کندانسور این پین ۰ می‌شود.

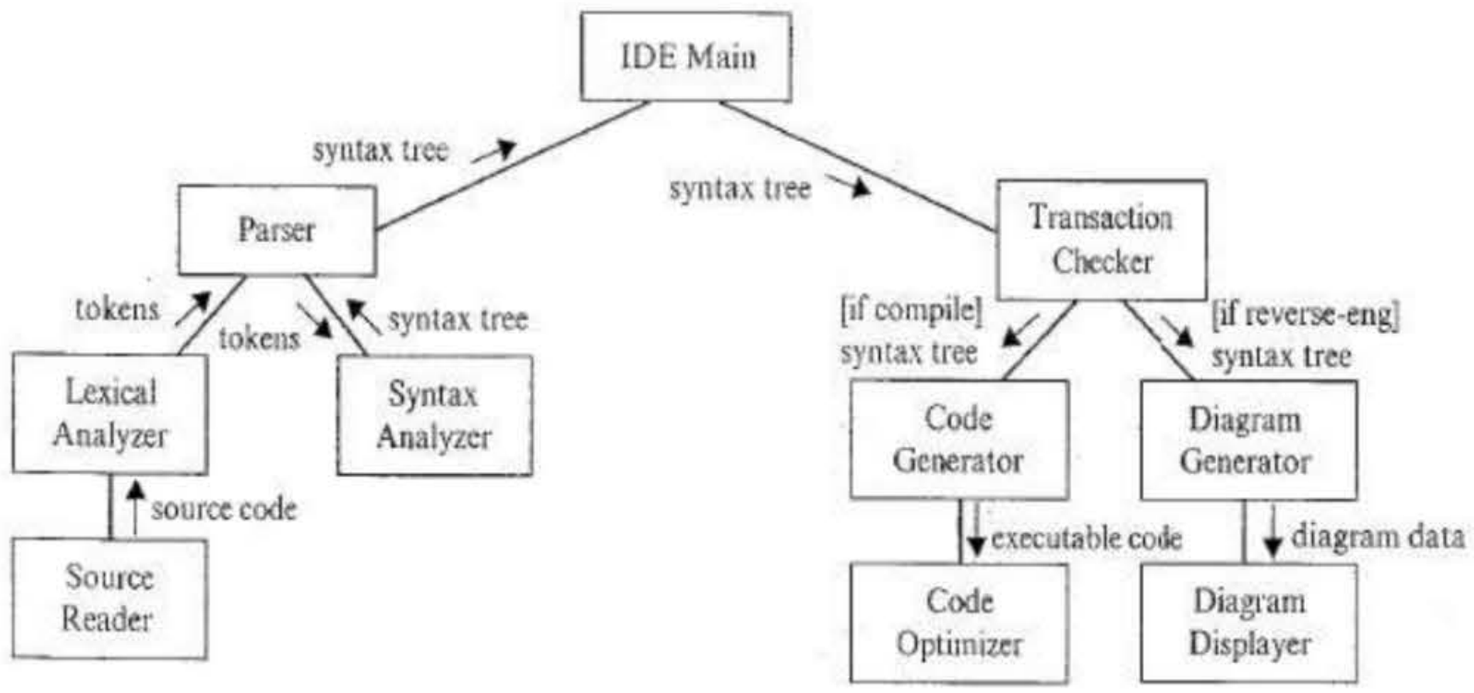
---

[1] Post-Order

[2] Event-Driven System Architecture (EDSA)



الف) مدل تحلیل IDE

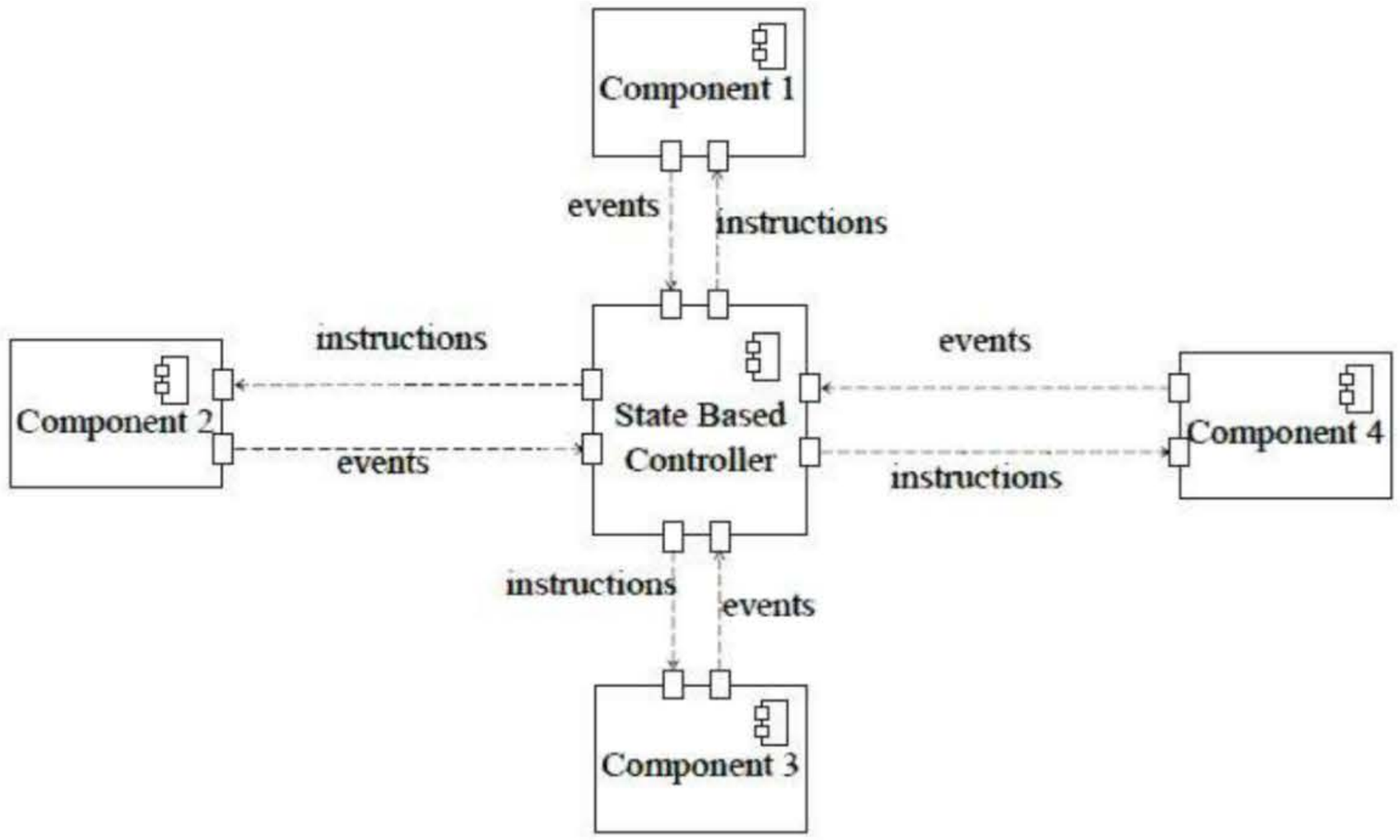


ب) برنامه اصلی و طراحی معماری زیرروال‌ها برای IDE

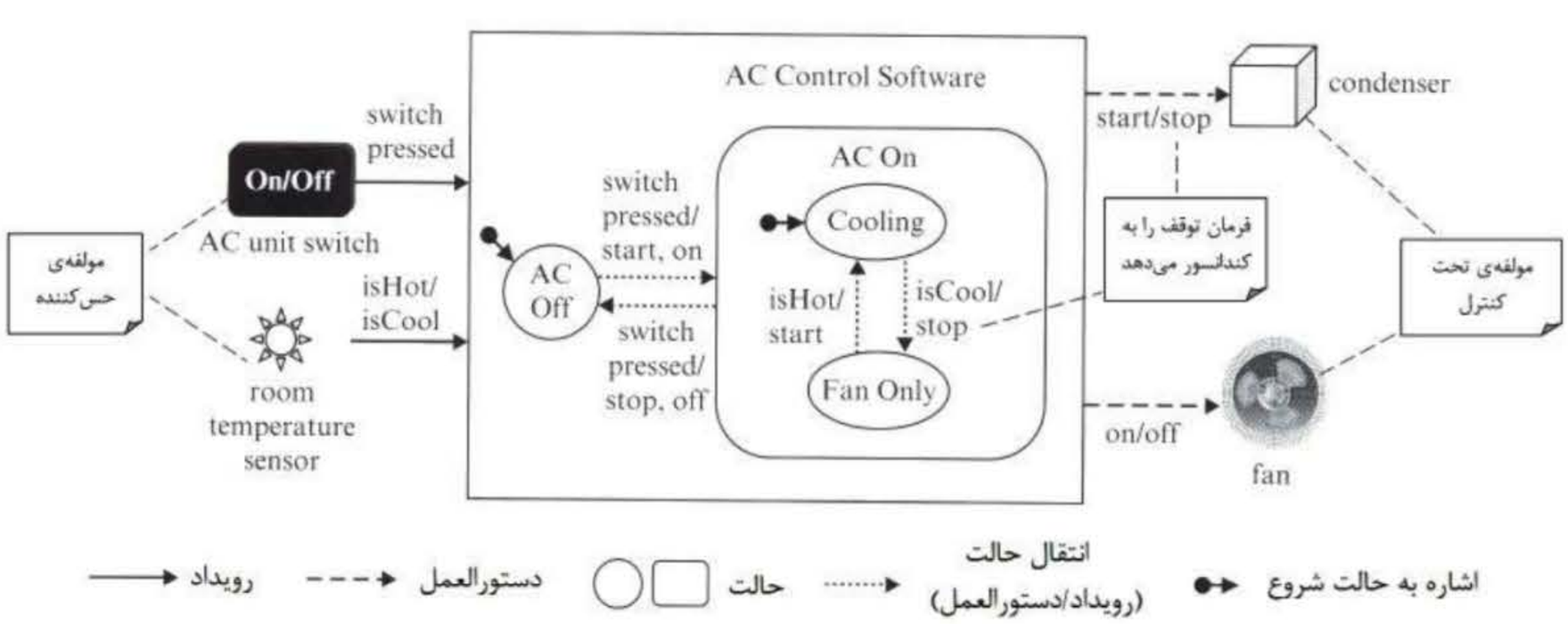
شکل ۶-۷: به‌کاربردن سبک طراحی معماری برنامه‌ی اصلی و زیربرنامه‌ها

هربار که کلید واحد AC فشرده می‌شود، نرم‌افزار حالت را از AcOff به AcOn و بالعکس تغییر می‌دهد. یعنی کلید بین دو حالت On و Off جابه‌جا می‌شود. به‌علاوه، نرم‌افزار برای کنترل پنکه و کندانسور، دستورات مناسب On/Off و Start/Stop صادر می‌کند. اگر نرم‌افزار در حالت Fan Only باشد و دمای اتاق گرم باشد، نرم‌افزار به حالت Cooling می‌رود و در همان زمان کندانسور را روشن می‌کند. اگر نرم‌افزار از قبل در حالت Cooling باشد، رویداد Is Hot نادیده گرفته می‌شود. انتقال‌های دیگر هم می‌توانند به روش مشابه تفسیر گردند.

نه معماری Nلایه و نه معماری خادم-مشتری، راه‌حل طراحی معماری مناسبی برای مسئله‌ی کنترل که توضیح داده شد، نیستند. علت این امر آن است که مسئله و نرم‌افزار، رویداد-رانده و وابسته‌به‌حالت هستند. برای مثال، برای معماری Nلایه، کنترل و مدیریت رویدادهای روشن و خاموش از کلید واحد AC دشوار است، زیرا سبک‌های طراحی Nلایه، حالت قبلی را به‌خاطر نمی‌سپارند و نمی‌دانند که حالت بعدی چیست. ممکن است تصمیم گرفته شود که یک متغیر حالت اضافه شود، ولی این کار معماری را تغییر خواهد داد و معماری جدید، معماری Nلایه نخواهد بود. در معماری خادم-مشتری، مشتری‌ها خادم را می‌شناسند ولی خادم مشتری را نمی‌شناسد. اگر این معماری به‌عنوان معماری نرم‌افزار AC مورد استفاده قرار گیرد، وقتی که حسگر دمای اتاق باعث یک تغییر حالت می‌شود، نمی‌داند که دستور کنترلی خود را به کدام وسیله ارسال نماید. نرم‌افزار کنترلی AC می‌تواند اگر رویدادی در حالت مناسب خود نبود، آن را نادیده بگیرد، بااین وجود، هر لایه از معماری Nلایه و خادم معماری مشتری-خادم باید بتواند به هر درخواستی که دریافت می‌کند، پاسخ دهد. درواقع آن‌ها نمی‌دانند که کدام رویداد باید پردازش شود و کدام باید صرف‌نظر گردد، آن‌ها باید همه‌ی درخواست‌ها را پردازش کنند.



شکل ۶-۸: معماری سیستم رویداد-رانده



شکل ۶-۹: نمایش معماری سیستم رویداد-رانده

سبک معماری چارچوب ماندگاری

سیستم‌های شئ‌گرا نیازمندند که بتوانند اشیا را در یک پایگاه‌داده ذخیره نمایند و در آینده آن‌ها را بازیابی کنند. به این ویژگی، ماندگاری شئ گفته می‌شود. معماری چارچوب ماندگاری، چنین توانمندی را در کنار ویژگی‌های جالب دیگر فراهم می‌نماید. این ویژگی‌ها شامل جداسازی دغدغه‌ها، پنهان‌سازی اطلاعات، طراحی برای تغییر، و طراحی ساده است. این سبک معماری که برای دسترسی به انواع مختلف پایگاه‌داده‌ها اعمال گردیده، درشکل ۶-۱۰ شرح داده شده است. انواع مختلف پایگاه‌داده‌ی مورد بررسی، شامل پایگاه‌داده‌های رابطه‌ای، شبکه‌ای، سلسله‌مراتبی، و شئ‌گرا می‌باشند. به‌علاوه، پایگاه‌داده‌ها ممکن است از سازنده‌های مختلف باشند. بدون استفاده از سبک معماری چارچوب ماندگاری، هریک از اشیای کسب‌وکار باید بدانند که

چگونه به انواع مختلف پایگاه‌داده از سازنده‌های مختلف دسترسی داشته باشند. همچنین، اشیای کسب‌وکار باید بدانند که داده‌ها در پایگاه‌داده‌های مختلف سازمان‌دهی می‌گردند. این امر، پیچیدگی اشیای کسب‌وکار را افزایش می‌دهد.

سبک چارچوب ماندگاری، برای فراهم‌کردن یک واسط شئ‌گرا برای اشیای کسب‌وکار، از مدیر پایگاه‌داده استفاده می‌نماید. مدیر پایگاه‌داده درخواست‌های رسیده از اشیای کسب‌وکار را به اشیای دسترسی به پایگاه‌داده محول می‌کند که با انواع مختلف پایگاه‌داده از سازنده‌های مختلف در ارتباطند. این کار، پیاده‌سازی، آزمون، و نگهداری اشیای کسب‌و کار را بسیار ساده می‌نماید.



شکل ۶-۱۰: نمایش سبک معماری چارچوب ماندگاری

سبک‌های معماری دیگر

علاوه بر سبک‌های معماری معرفی‌شده در بخش‌های قبل، سبک‌های معماری دیگری نیز در منابع مختلف معرفی‌شده است. برخی از این سبک‌ها در فهرست زیر نشان داده شده است. سبک‌های دیگر، در بخش ۴ (استفاده از الگوهای خاص موقعیت) معرفی گردیده‌اند.

- نظیربه‌نظیر[۱]. معماری نظیربه‌نظیر از اجزای مستقلی تشکیل می‌شود که از طریق پروتکل‌های شبکه با یکدیگر مرتبط می‌گردند. این معماری معمولاً در کاربردهای محاسباتی غیرمتمرکز یا توزیع‌شده مورد استفاده قرار می‌گیرد. سیستم‌هایی که بر اساس این معماری ساخته می‌شوند بسیار قدرتمندند، زیرا شکست یک جزء اثر کمی بر عملکرد سیستم دارد. به‌علاوه این سیستم‌ها بسیار قابل‌انعطاف و مقیاس‌پذیر هستند، زیرا اجزا می‌توانند به‌راحتی اضافه یا حذف گردند.
- لوله‌وصافی[۲]. معماری لوله‌وصافی از یک سری برنامه به نام صافی تشکیل می‌گردد که با جریان‌های داده‌ای به نام لوله، به‌شکل خط‌لوله با هم مرتبط می‌شوند. به‌همین‌دلیل به این سبک، معماری خط‌لوله‌ای[۳] نیز گفته می‌شود. این معماری

---

[۱] Peer-to-peer
[۲] Pipe and filter
[۳] Pipe Line Architecture

به‌طور گسترده‌ای در برنامه‌نویسی برای انجام پردازش‌های جمعی[۱] درجایی که خروجی یک برنامه‌ی قابل‌اجرا به برنامه‌ی دیگر منتقل می‌شود، به‌کار می‌رود. سادگی ارتباطات بین فرایندها باعث می‌شود که ترکیب‌های جدید برنامه‌ها برای انجام محاسبات دلخواه به‌راحتی صورت پذیرد.

- تخته سیاه[۲]. معماری تخته‌سیاه از یک مخزن داده‌ی سراسری به نام تخته سیاه و تعدادی مولفه‌ی مستقل تشکیل می‌گردد که از طریق این مخزن با هم درارتباطند. این مولفه‌های مستقل می‌توانند داده‌ها را به‌طور مستقیم و یا توسط سرویس‌هایی که تخته سیاه فراهم می‌نماید به‌روزرسانی کنند. از یک دیدگاه، این ساختار، تعامل بین مولفه‌های مستقل را ساده‌تر می‌کند ولی از سوی دیگر، اگر به‌خوبی طراحی و پیاده‌سازی نشده باشد، ممکن‌است به‌روزرسانی‌های همزمان، سبب بروز اثرات جانبی غیرمنتظره گردد.
- معماری سرویس‌گرا[۳]. SOA به‌عنوان ترکیبی از سیاست‌ها، تمرین‌ها، و چارچوب‌هایی دیده می‌شود که سبب می‌گردد عملیات کسب‌وکار بتوانند به‌عنوان سرویس‌هایی فراهم و استفاده گردند. یک SOA از فراهم‌کننده‌های سرویس، مصرف‌کننده‌های سرویس، و معماری سرویس (که سرویس‌های بین دوتای دیگر را مدیریت می‌نماید) تشکیل می‌شود. یک سرویس، یک مجرد و پیاده‌سازی از یک عمل کسب‌وکار در یک دامنه، و مشابه یک واسط و یک پیاده‌سازی است. برخلاف فراخوانی توابع، سرویس‌ها از راه استفاده از پروتکل http و XML فعال می‌گردند.
- معماری محاسبات ابری[۴]. محاسبات ابری، توسعه‌ای از SOA است که در آن، کل نرم‌افزار کاربردی به‌جای یک عمل کسب‌وکار به‌عنوان یک سرویس مطرح می‌شود[۵]. فراهم‌کننده‌ی سرویس اجازه می‌دهد که مصرف‌کننده‌ی سرویس در زمان نیاز به یک نرم‌افزار کاربردی به‌عنوان یک سرویس دسترسی پیدا کند. مصرف‌کننده‌ی سرویس برحسب میزان استفاده‌ی واقعی خود از نرم‌افزار، هزینه پرداخت می‌کند. Google Apps مثالی از SaaS است.

۶-۳-۴- انجام طراحی معماری لازم

استفاده‌ی مجدد از یک سبک معماری همواره در اولویت است، زیرا نه‌تنها باعث صرفه‌جویی در زمان و هزینه می‌شود، کیفیت را نیز به سیستم وارد می‌نماید. شرط لازم این است که سبک معماری صحیح انتخاب شود و به‌درستی اعمال گردد. با این وجود نمی‌توان همه‌ی توسعه‌ی سیستم کاربردی را هم با استفاده‌ی مجدد از سبک‌های معماری موجود انجام داد.

به‌عنوان مثال، طراحی یک زیرسیستم دارای مسئولیت بحرانی را درنظر بگیرید. یکی از اهداف طراحی این زیرسیستم این است که در زمان بروز یک خطا در نرم‌افزار، همچنان به کار خود ادامه دهد. اگر هیچ‌یک از سبک‌های موجود معماری این هدف طراحی را برآورده نکند، باید یک طراحی معماری برحسب نیاز انجام گردد. یک راه برای تحمل‌پذیری خطا در نرم‌افزار، استفاده از تنوع اشیا است، یعنی نسخه‌های بسیار متفاوتی از زیرسیستم طراحی و پیاده‌سازی شود. یکی از آن‌ها به‌عنوان نسخه‌ی فعال اجرا گردد، و وقتی که خطایی رخ می‌دهد یکی دیگر از نسخه‌های پشتیبان برگزیده و اجرا شود. برای این‌کار، مولفه‌ای از زیرسیستم که وظیفه‌ای بحرانی دارد شناسایی می‌گردد. این مولفه باید تحمل بروز خطا را داشته باشد. یک واسط عمومی برای این مولفه تعریف می‌شود. تیم‌های مختلفی این واسط را پیاده‌سازی می‌کنند تا نسخه‌های متفاوتی از یک مولفه ایجاد شود. هریک از پیاده‌سازی‌ها، از داده‌ساختارها و الگوریتم‌های متفاوتی استفاده می‌کنند. یکی از این نسخه‌ها انتخاب می‌شود و تا زمانی که باید جایگزین شود، اجرا می‌گردد.

الگوهای طراحی نرم‌افزار، برای طراحی دلخواه یک سیستم نرم‌افزاری مفید می‌باشند. برای مثال، برای طراحی معماری مولفه‌ای با مسئولیت بحرانی، که پیش‌تر به آن اشاره شد و باید قابلیت تحمل‌پذیری خطا داشته باشد، می‌توان از الگوهای پُل[۶]

---

[۱] Batch Processing
[۲] Blackboard
[۳] Service-Oriented Architecture (SOA)
[۴] Cloud Computing Architecture
[۵] SaaS
[۶] Bridge Pattern

و یادآور[1] استفاده کرد. این الگوها در فصل‌های ۱۶ و ۱۷ نشان داده شده‌اند. الگوی پُل سیستم را قادر می‌سازد که به‌طور شفاف و نامحسوس، نسخه‌ی متفاوتی از یک مولفه با مسئولیت بحرانی را جایگزین کند. یعنی مشتری این مولفه، از جایگزینی انجام‌شده مطلع نگردد. الگوی یادآور به سیستم اجازه می‌دهد که قبل از اجرای کاری که ممکن است منجر به بروز شکست در نرم‌افزار شود، حالت مولفه را ذخیره نماید. به این ترتیب، پیاده‌سازی جایگزین‌شده می‌تواند با حالت ذخیره‌شده آغازبه‌کار نموده، کار موردنظر را دوباره انجام دهد.

۶-۳-۵- تعیین واسطها و عملیات زیرسیستم

در این گام نیازمندی‌های نرم‌افزار و اهداف طراحی آن، به زیرسیستم‌ها و مولفه‌های معماری تخصیص داده می‌شود. گاهی لازم است که برخی از نیازمندی‌ها به نیازمندی‌های سطوح پایین‌تر شکسته شوند تا تخصیص آن‌ها به زیرسیستم‌ها ساده‌تر گردد. سپس، عملکرد هریک از زیرسیستم‌ها و مولفه‌ها، باتوجه به نیازمندی‌ها و اهداف طراحی آن‌ها مشخص می‌شود. برای مثال در شکل ۶-۷، نیازمندی مهندسی معکوس نمودار کلاس، نمودار توالی، نمودار فعالیت، و نمودار حالت UML به مولفه‌ی Diagram Generator (برای تولید نمودار) تخصیص داده می‌شود. یک نیازمندی کارایی، به یک یا چند زیرسیستم یا مولفه‌ی مربوط به کارایی اختصاص می‌یابد. یک هدف طراحی تحمل‌پذیر خطا به یک زیرسیستم با مسئولیت بحرانی که باید چنین قابلیتی را داشته باشد واگذار می‌گردد.

در این گام، واسطهای بین زیرسیستم‌ها مشخص می‌گردند. ورودی و خروجی هر زیرسیستم شامل تعداد، انواع، و ترتیب پارامترهای ورودی و خروجی در توصیف این واسطها تعریف می‌گردند. به‌علاوه، رفتار تعاملی بین زیرسیستم‌ها (به معنای رشته پیام‌هایی که باید بین آن‌ها تبادل گردد) در این مرحله تعیین می‌شود.

۶-۳-۶- بازبینی طراحی معماری

طراحی معماری مرور می‌شود تا از برآورده‌شدن اهداف طراحی و نیازمندی‌های نرم‌افزار اطمینان حاصل گردد. به‌علاوه، این مرور بررسی می‌کند که طراحی از اصول و قوانین طراحی، که در بخش ۶-۵ شرح داده شد، پیروی نماید.

۶-۴ سبک معماری و نمودار بسته

معماری نرم‌افزار، ساختار سیستم نرم‌افزاری را به‌صورت چند زیرسیستم و روابط داخلی بین آن‌ها تعریف می‌کند. همین معماری نرم‌افزار، محصولی اصلی برای مفهوم‌سازی، ساخت، مدیریت، و تکامل سیستم تحت توسعه است. این مفاهیم به معنای زیر هستند:

۱. مفهوم‌سازی. معماری، ساختار کلی یک سیستم را تعریف می‌کند. بنابراین، در فعالیت‌های آینده‌ی توسعه‌ی معماری به تیم توسعه کمک می‌نماید که به سیستم از نقطه‌نظر ساختار کلی آن بیاندیشند. برای مثال، معماری Nلایه را درنظر بگیرید. این معماری، سیستم را تشکیل‌شده از N لایه از اجزا می‌داند که هر لایه‌ی بالاتر، خدماتی را از لایه‌ی پایین‌تر بعدی درخواست می‌نماید. به این تصویر کلی از سیستم، تصویرسازی از سیستم گفته می‌شود. وقتی معماری نرم‌افزار تعریف می‌شود، به تیم کمک می‌کند که تصویر خود از ساختار سیستم را یکنواخت نماید.
۲. ساخت. معماری، ساخت سیستم نرم‌افزاری را آسان‌تر می‌نماید. دلیل این امر آن است که معماری به اعضای تیم اجازه می‌دهد که بدانند چگونه محصولات تولیدشده در خلال فرایند توسعه‌ی نرم‌افزار را سازمان‌دهی کنند. یک‌بار دیگر

[1] Memento

معماری Nلایه را برای یک سیستم تعاملی درنظر بگیرید. معماری این سیستم در حالت معمول از لایه‌های زیر تشکیل می‌شود که از بالا به پایین فهرست شده‌اند:

الف) لایه‌ی نمایش[۱]. این لایه مسئول نمایش واسط گرافیکی کاربر، و پاسخ‌های سیستم به کاربران است.

ب) لایه‌ی اشیای کسب‌وکار[۲]. این لایه مسئول پردازش تراکنش‌های کسب‌وکار است که با موارد کاربرد نشان داده شده‌اند.

ج) لایه‌ی انباره‌ی مانا[۳]. این لایه از اشیایی تشکیل می‌شود که عملیات مربوط به پایگاه‌داده، مانند ذخیره و بازیابی اشیا، را فراهم می‌نماید.

د) لایه‌ی ارتباط شبکه[۴]. این لایه، عملیات مربوط به ارتباطات شبکه را فراهم می‌سازد.

مسئولیت‌ها و وابستگی‌های بین لایه‌ها، به فرایند ساخت کمک می‌کند. برای مثال، کار توسعه‌ی یک لایه باید به افرادی از تیم سپرده شود که دانش و مهارت‌های انجام این‌کار را داشته باشند. لایه‌ها باید براساس وابستگی‌هایشان ساخته شوند. یعنی یک لایه‌ی پایین‌تر باید قبل از لایه‌ی بالاتر بعد از خود توسعه یابد تا پیاده‌سازی و آزمون کلاس‌های لایه‌ی بالاتر بتواند بدون نیاز به ساختن زیربرنامه‌هایی برای شبیه‌سازی اشیای ساخته‌نشده‌ی لایه‌ی پایین‌تر، انجام گردد.

۳. مدیریت. معماری نرم‌افزار، یک دیدگاه وابسته به معماری برای سازمان‌دهی محصولات میانی تولیدشده در طول فرایند توسعه، ارائه می‌نماید. برای مثال، محصولاتی چون کلاس‌ها، صفحات وب، و تصاویری که برای یک سیستم تعاملی ایجاد شده‌اند، می‌توانند برمبنای لایه‌های یک معماری Nلایه سازمان‌دهی گردند. به چنین سازمانی، سازمان یا معماری منطقی گفته می‌شود. به‌دلیل آن‌که المان‌های این سازمان، غیرقابل‌اجرا هستند، این سازمان منطقی است نه فیزیکی.

۴. رشددادن. معماری، پایه‌ای برای رشددادن و گسترش سیستم فراهم می‌نماید. برای مثال، یک سیستم اطلاعاتی کتابخانه، یک سیستم تعاملی است. اگر در ابتدای کار، و به‌دلیل محدودیت‌های بودجه‌ای، سیستم برای پشتیبانی امانت بین‌کتابخانه‌ای طراحی نشده باشد و مدتی بعد لازم گردد که این توانمندی فراهم گردد، معماری Nلایه با افزودن یک لایه‌ی ارتباطات شبکه، این ارتقای سیستم را امکان‌پذیر خواهدکرد.

برای استفاده از مزایای معماری نرم‌افزار برای فعالیت‌های توسعه، تیم نرم‌افزاری به راهی برای سازمان‌دهی مصنوعات تولیدشده در طول فرایند توسعه نیازمند است. نمودار بسته‌ی UML، سازوکاری برای این امر فراهم می‌نماید. نمادها و انگاره‌های یک نمودار بسته‌ی UML درشکل ۶-۱۱-الف دیده می‌شود.

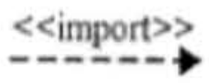
مثال ۶-۱

فرض کنید که یک سیستم اطلاعاتی کتابخانه، برای استفاده از معماری Nلایه طراحی شده است. چهار لایه‌ی معماری مشابه آنچه در نکته‌ی شماره‌ی ۲ این بخش شرح داده شد، به شکل زیر هستند: لایه‌ی نمایش، لایه‌ی اشیای کسب‌وکار، لایه‌ی پایگاه‌داده، و لایه‌ی شبکه. نمودار بسته‌ی مربوطه در شکل ۶-۱۱-ب نشان داده شده است. نمودار نشان می‌دهد که سیستم اطلاعاتی کتابخانه، دارای چهار بسته است که هریک، به یکی از لایه‌های معماری مرتبط است. بسته‌ی GUI دارای چهار کلاس است: Main، Main Frame، Checkout Dialog و Return Dialog. دو مورد اول، کلاس‌های عمومی (Public) هستند و دو مورد بعدی، کلاس‌های اختصاصی (Private). بسته‌ی کسب‌وکار دارای پنج کلاس است: Return Controller، Checkout Controller، Loan، Document،و Patron. دو کلاس اول، عمومی و سه تای آخر اختصاصی

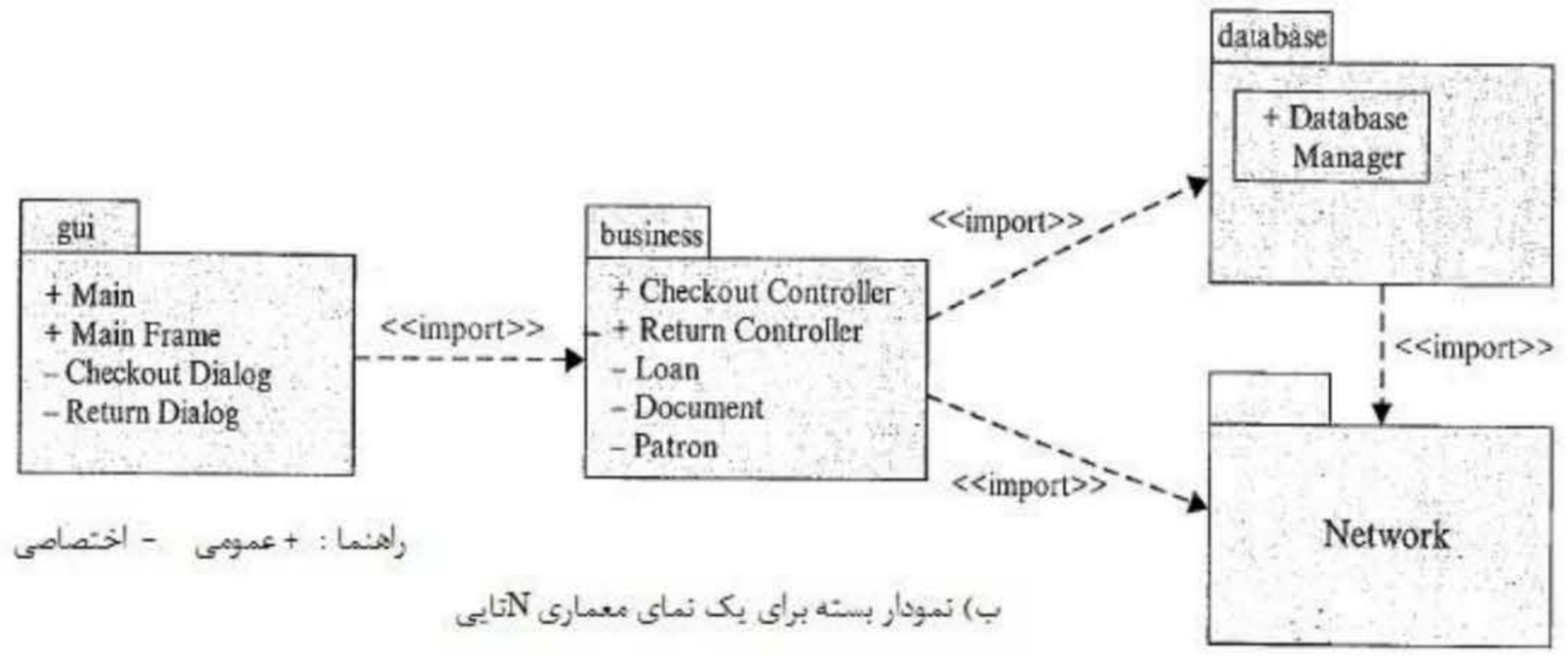
---

[۱] Presentation Layer
[۲] Business Objects Layer
[۳] Persistence Storage Layer
[۴] Network communication Layer

هستند. بسته‌ی GUI بسته‌ی کسب‌وکار را وارد می‌کند. این بدان معنی است که کلاس‌های عمومی بسته‌ی کسب‌وکار، نه کلاس‌های اختصاصی آن، وارد می‌شوند. بسته‌ی پایگاه‌داده تنها دارای یک کلاس است، Database Manager، که به بسته‌ی کسب‌وکار وارد می‌شود. کلاس‌های یک بسته می‌توانند با استفاده از فرمت متنی یا تصویری نشان داده شوند. با وجود آن‌که نام بسته‌های GUI، کسب‌وکار،و پایگاه‌داده در بخش فوقانی پوشه نشان داده می‌شود، نام بسته‌ی شبکه در بدنه‌ی پوشه دیده می‌شود. این بدان‌دلیل است که سه بسته‌ی اول، محتویات خود را نشان می‌دهند ولی بسته‌ی شبکه، محتویات خود را نمایش نمی‌دهد.

انگاره	معنا	نماد	روابط
Package	یک گروه‌بندی منطقی از دست‌ساخته‌های نرم‌افزاری شامل کلاس‌ها، بسته‌های دیگر، و سایر دست‌ساخته‌های نرم‌افزاری مدنظر		• یک بسته، می‌تواند سایر دست‌ساخته‌های نرم‌افزاری مانند کلاس‌ها یا بسته‌های دیگر را دارا باشد. • یک بسته می‌تواند بسته‌های دیگر را import کند.
Import	یک رابطه‌ی دارای استریوتایپ بین دو بسته. بسته‌ی مقصد در ابتدای پیکان، المان‌های عمومی بسته‌ی مبدا در انتهای پیکان را وارد می‌کند.		

الف- انگاره‌ها و نمادگان نمودار بسته



ب) نمودار بسته برای یک نمای معماری Nتایی

شکل ۶-۱۱: نمودار بسته برای یک سیستم اطلاعاتی کتابخانه

نمودار بسته در شکل ۶-۱۱-ب، یک ساختار با دیدگاه منطقی از کلاس‌های سیستم اطلاعاتی کتابخانه را تعریف می‌نماید. درطول فرایند توسعه، کلاس‌های متعدد و دست‌ساخته‌های نرم‌افزاری دیگر مانند صفحات وب و تصاویر تولید می‌شوند. نمودارهای بسته، به اعضای تیم کمک می‌کنند که بدانند کدامیک از این محصولات به کدام بسته تعلق دارد. به این ترتیب، بسته‌ها و محصولات مربوط به آن‌ها می‌توانند با هم به یک سیستم مدیریت پیکربندی (فصل ۲۲) وارد، و یا از آن خارج شوند. سلسله مراتب بسته‌ها (در جایی که یک بسته می‌تواند دارای بسته‌های دیگر باشد)، کنترل تغییرات را آسان می‌کند؛ زیرا بسته‌هایی که باید تغییر کنند و بسته‌هایی که از این تغییرات تاثیر می‌پذیرند، می‌توانند در هر سطح از سلسله‌مراتب شناسایی شوند. بدون چنین سازمان منطقی، مدیریت پیکربندی کلاس‌ها و سایر دست‌ساخته‌ها، دشوارتر خواهدبود.

۶-۵ اعمال قوانین طراحی نرم‌افزار

چندین دهه فعالیت در زمینه‌ی توسعه‌ی نرم‌افزار، به جامعه‌ی مهندسین نرم‌افزار، درس‌های بسیاری آموخته است. این درس‌ها منجر به تدوین یک مجموعه قوانین طراحی نرم‌افزار گردیده است. اعمال این قوانین می‌تواند طراحی معماری را ارتقا بخشد. به‌عنوان مثال، در دهه‌ی هفتاد میلادی قطعه برنامه‌های بسیاری نوشته شدند که هرکدام شامل هزاران خط کد و تعداد زیادی تابع متفاوت بودند. فهم، تغییر، آزمون، و استفاده‌ی مجدد چنین برنامه‌هایی بسیار دشوار است. به‌دلیل متمرکز نبودن هدف یک تابع، کسی متوجه نمی‌شود که این تابع دقیقاً چه کاری را انجام می‌دهد، بلکه به‌نظر می‌رسد که هرتابع تقریباً همه کاری انجام می‌دهد. به‌اصطلاح گفته می‌شود که چنین نوع برنامه‌هایی، از چسبندگی عملیاتی کمی برخوردارند. مشکلات مشابهی در طراحی معماری رخ می‌دهد. برای مثال، موارد کاربردی که از نظر عملیاتی نامرتبط‌اند، به یک زیرسیستم اختصاص می‌یابند. در برنامه‌نویسی شیءگرا اگر یک کلاس توابع یا متدهای نامرتبط بسیاری را شامل شود، چسبندگی عملیاتی پایینی خواهدداشت. فهم، آزمون، و نگهداری چنین کلاس‌هایی دشوار است. یک نوع دیگر از «برنامه‌ی بد» با مشکل جفت‌شدگی زیاد در ارتباط است. این بدان‌معنی است که توابع برنامه، یکدیگر را به هرشکل دلخواه فراخوانی می‌کنند که این امر منجر به یک گراف فراخوانی پیچیده خواهد شد. واضح است که فهم، آزمون، و نگهداری چنین برنامه‌ای دشوار است. مشکل مشابهی نیز در طراحی معماری، و طراحی و پیاده‌سازی اشیا وجود دارد. یعنی مشاهده‌ی روابط وابستگی پیچیده بین زیرسیستم‌ها، مولفه‌ها، و اشیا چندان معمول نیست. قوانین طراحی نرم‌افزار برای حل این مشکلات پیشنهاد شده‌اند. این بخش، قوانین طراحی نرم‌افزار را که می‌توانند در زمان طراحی مورد استفاده قرار گیرند، معرفی می‌نماید.

۶-۵-۱- قوانین طراحی نرم‌افزار کدامند؟

در بخش قبل، بسیاری از مشکلات مربوط به طراحی که در عمل رایج می‌باشند، شرح داده شد. این مشکلات بر بهره‌وری و کیفیت نرم‌افزار تاثیر منفی گذاشته، و هزینه‌های نگهداری نرم‌افزار را به‌شدت افزایش می‌دهند. یکی از راه‌حل‌های پیشنهادشده برای حل این مسائل، قوانین طراحی نرم‌افزار است که در ادامه تعریف می‌شود:

تعریف ۶-۳

قوانین طراحی نرم‌افزار، قواعد تاییدشده‌ای است که استفاده‌ی صحیح آن‌ها در طراحی نرم‌افزار، می‌تواند کیفیت نرم‌افزار را به‌شدت افزایش دهد.

قوانین طراحی نرم‌افزار، محصول خرد جمعی است که در طول دهه‌ها امر تحقیق و توسعه (R&D) توسط جامعه‌ی مهندسی نرم‌افزار ارزیابی شده است. این قوانین، دارایی‌های ارزشمند این جامعه هستند. در بخش‌های آینده به مطالعه‌ی قوانین طراحی شامل طراحی برای تغییر، جداسازی دغدغه‌ها، پنهان‌سازی اطلاعات، چسبندگی بالا، جفت‌شدگی کم، و ساده‌گیری[1] خواهیم پرداخت.

۶-۵-۲- طراحی برای تغییر

قانون طراحی برای تغییر، از این واقعیت که تغییر خط اصلی زندگی است، نشات گرفته است. رویدادهای زیادی می‌توانند باعث تغییر در یک سیستم بشوند. برخی از این رویدادها در ادامه آورده شده‌اند:

[1] Keep It Simple and Stupid (KISS)

- برای پاسخ‌گویی به تغییرات در محیط کسب‌وکار، لازم است نیازمندی‌های نرم‌افزار تغییر کند.
- برای رفع مشکلات سیستم، لازم است سیستم نرم‌افزاری تغییر داده شود.
- تغییرات در سخت‌افزار، سکو، محیط عملیاتی سیستم، و مانند آن تغییر در سیستم را لازم می‌کند.
- تغییر در نحوه‌ی مدیریت، صنعت، سیاست‌های شرکت، آیین‌نامه‌ها، فرایندهای عملیاتی، و مانند آن، تغییر در سیستم را لازم می‌نماید.
- برای ارتقای کارایی، قابلیت اطمینان، کاربرپسندی، بهره‌وری، امنیت، و مانند آن، به تغییر نیاز است.
- به‌دلیل پیشرفت تکنولوژی، تغییر موردنیاز است.
- تغییر در برنامه‌ریزی پروژه و بودجه‌بندی آن، باعث لزوم تغییر در سیستم می‌شود.

اگر پیاده‌سازی دقیقاً از روی یک طراحی خوب انجام شود، باید به سیستمی منجر گردد که بتواند با تغییرات سازگار شود و یا به راحتی تغییر داده شود. طراحی برای تغییر به این معناست که طراحی نرم‌افزار، باید براساس یک «سازوکار داخلی» برای سازگاری با تغییرات، و یا تسهیل تغییرات قابل‌پیش‌بینی انجام گردد. در اینجا منظور از طراحی نرم‌افزار، همه‌ی انواع طراحی مانند طراحی معماری، طراحی مولفه‌ها، طراحی بدنه‌ها، طراحی اشیا، و طراحی برنامه است.

در بخش ۶-۲، برای تاکید بر اهمیت طراحی معماری، داستان آنتولوژی‌ها شرح داده شده است. اگر از زاویه‌ی دیگری نگاه کنیم، خواهیم دید که این داستان اهمیت طراحی برای تغییر را نیز نشان می‌دهد. یعنی اگر تیم، قانون طراحی برای تغییر را اعمال کرده بود و رویه‌ی پنهان‌کننده‌ی پایگاه‌داده[۱] را پیاده‌سازی کرده بود، پروژه با شکست مواجه نمی‌شد. یک ماجرای دیگر در این زمینه را در ادامه می‌خوانید. سال‌ها پیش، نگارنده، قراردادی برای طراحی و پیاده‌سازی یک محصول تجاری مبتنی‌بروب منعقد کرد. درابتدا، نماینده‌ی مشتری درخواست کرد که محصول، تنها از پایگاه‌داده‌ی LDAP[۲] استفاده نماید. LDAP یک پایگاه‌داده‌ی سلسله‌مراتبی است که برای بازیابی داده‌ها بسیار کارآمد است و به‌طورگسترده‌ای برای کاربردهای مبتنی بر وب مورداستفاده قرار می‌گیرد.

قانون طراحی برای تغییر به ما توصیه می‌کرد که LDAP نباید تنها انتخاب پایگاه‌داده برای یک محصول تجاری باشد. مشتریان محصول باید بتوانند به انتخاب خود از انواع دیگر سیستم‌های مدیریت پایگاه‌داده استفاده نمایند و DBMS باید از دید کاربران مخفی باشد. بنابراین، برای مخفی‌کردن پایگاه‌داده از اشیای کسب‌وکار، و پشتیبانی از DBMSهای دیگر، از الگوهایی استفاده شد. بااین‌وجود، نماینده‌ی مشتری، طراحی را با استدلال غیرضروری بودن و ناکارآمدی این‌کار، رد کرد. ماه‌ها بعد، اولین نمونه از سیستم تحویل داده شد و بازاریاب‌های آن شرکت، شروع به عرضه‌ی کار برای فروش کردند. برخلاف آن‌که به‌نظر می‌رسید ویژگی‌های نوآورانه‌ی محصول، مشتریان را جذب کند، محصول فروش نرفت. بسیاری از مشتریان بالقوه‌ی محصول، هرگز نام LDAP را نشنیده بودند و نمی‌خواستند زیر بار ریسک معرفی آن به‌جای پایگاه‌داده‌های دیگر بروند. نماینده‌ی مشتری نزد ما برگشت و این بار محصولی با قابلیت پشتیبانی انواع مختلف DBMS موجود در بازار را درخواست کرد. او تصور می‌کرد که برای این‌کار باید هزینه‌ی زیادی بپردازد، ولی وقتی متوجه شد که محصول با این قابلیت طراحی و پیاده‌سازی شده است، بسیار خوشحال شد. سال‌ها گذشت و این محصول، همچنان در بازار فروخته می‌شود. ساختار معماری آن نیازی به تغییرات زیاد نداشته است.

استفاده از قانون طراحی برای تغییر به این معناست که تغییرات قابل‌پیش‌بینی، در طراحی معماری نرم‌افزار درنظر گرفته شود و این طراحی، سازوکارهایی را برای تطابق با این تغییرات شامل گردد. در مثال قبل، پیش‌بینی شده بود که محصول نیاز خواهد داشت که انواع دیگر DBMS به غیر از LDAP را هم پشتیبانی نماید. بنابراین، در طراحی معماری، یک مدیر پایگاه‌داده برای تسهیل این تغییر در نظر گرفته شده بود. به‌این‌ترتیب در زمان نیاز به پشتیبانی انواع دیگر DBMS، به هیچ کار اضافی و دوباره‌کاری نیاز نبود.

---

[۱] Database Wrapper

[۲] Lightweight Directory Access Protocol

۶-۵-۳- جداسازی دغدغه‌ها

ایده‌ی جداسازی دغدغه‌ها، به‌عنوان یک قانون حل‌مسئله توسط ادسگر دایکسترا[۱] مطرح گردید. براساس این ایده، به‌جای تمرکز یک‌باره و همزمان به همه‌ی جنبه‌های یک مسئله، هربار بر یکی از این جنبه‌ها و جدا از سایر آن‌ها تمرکز می‌شود. UML یک مثال عالی از به‌کارگیری قانون جداسازی دغدغه‌ها است. هریک از نمودارهای UML، بر مدل‌سازی یک جنبه از کاربرد یا سیستم تمرکز می‌کنند. برای مثال نمودار کلاس UML، با هدف مدل‌سازی و طراحی جنبه‌ی ساختاری مورد توجه قرار می‌گیرد. نمودار توالی، با هدف مدل‌سازی و طراحی تعامل اشیا موردتوجه قرار می‌گیرد. نمودار مورد کاربرد، به موارد کاربرد و زمینه‌های وقوع آن‌ها می‌پردازد.

طراحی نرم‌افزار، یک فعالیت حل‌مسئله است. باتوجه به قانون جداسازی دغدغه‌ها، باید مسئله‌ی طراحی نرم‌افزار را در دو سطح در نظر گرفت. در سطح بالاتر، مسئله، چگونگی انجام فرایند کلی طراحی است. در سطح پایین‌تر، مسئله، چگونگی طراحی اجزا و مولفه‌های سیستم نرم‌افزاری است. به عبارت دیگر، طراحی نرم‌افزار با هر دو دغدغه‌ی فرایند طراحی و محصول طراحی در ارتباط است. جداسازی دغدغه‌ها، یک قانون راهنما برای حل مسئله‌ی طراحی در هر دو سطح است. از دید فرایند کلی طراحی، جداسازی دغدغه‌ها به ما می‌گوید که طراحی باید بر یک جنبه از کل فرایند طراحی، و با چشم‌پوشی از جنبه‌های دیگر، تمرکز نماید. به‌عنوان مثال، سبک شرح داده شده در این کتاب را درنظر بگیرید. جداسازی دغدغه‌ها در گام‌های این متدولوژی منعکس می‌شود، یعنی هرگام بر یک جنبه از فرایند طراحی متمرکز می‌شود. مورد کاربرد گسترده، با جنبه‌ی مدل‌سازی و طراحی تعامل بین کنشگر و سیستم در ارتباط است. مورد کاربرد در سطح بالاتر، با توصیف دامنه‌ی کاربرد، یا زمان و مکان شروع و خاتمه‌ی مورد کاربرد در ارتباط است. مدل‌سازی دامنه، با به‌دست‌آوردن و مدل‌سازی دانش دامنه‌ی کاربرد در ارتباط است. مدل‌سازی تعامل اشیا، با چگونگی همکاری اشیا برای انجام یک فرایند کسب‌وکار در ارتباط است.

در مورد طراحی مولفه‌های مستقل نرم‌افزاری، جداسازی دغدغه‌ها به ما می‌گوید که هر مولفه باید بر یک جنبه از موضوع عملکرد نرم‌افزار تمرکز داشته باشد. برای مثال، مولفه‌ی GUI باید بر نمایش اطلاعات به کاربر نهایی تمرکز نماید. مولفه‌ی مربوط به پایگاه‌داده، باید بر ذخیره و بازیابی اطلاعات تمرکز کند. اشیای کسب‌وکار باید بر مسئولیت‌های خاص کاری خود متمرکز باشند. این نیازها با استفاده از سبک‌های معماری Nلایه برآورده می‌شوند. به‌عنوان مثال دیگر، هر مورد کاربرد باید یک و فقط یک فرایند کسب‌وکار را مدل‌سازی و پیاده نماید. فرایندهای مختلف کسب‌وکار باید با موارد مختلف کاربرد مدل‌سازی شوند. موردهای کاربرد یک سیستم نرم‌افزاری، باید بنابر عملکرد و وظیفه‌ی اصلی خود دسته‌بندی شوند و به زیرسیستم‌های مختلف تخصیص یابند. این‌ها همه کاربردهای نمونه‌ای از قانون جداسازی دغدغه‌ها می‌باشند.

استفاده از قانون جداسازی دغدغه‌ها برای طراحی معماری به این معناست که مسئولیت‌های مربوط به دغدغه‌های مختلف، به زیرسیستم‌های مختلف اختصاص داده شود. این کار، به چسبندگی عملیاتی بالا منجر خواهد شد و فهم و استفاده‌ی مجدد از زیرسیستم‌ها را آسان‌تر خواهد کرد. در سطح طراحی اشیا، این قانون توصیه می‌کند که مسئولیت‌های مربوط به دغدغه‌های مختلف، به اشیای مختلف اختصاص یابد. این موضوع به‌تفصیل در فصل ۱۰(استفاده از الگوهای واگذاری مسئولیت) شرح داده خواهد شد.

۶-۵-۴- پنهان‌سازی اطلاعات

قانون پنهان‌سازی اطلاعات، نخستین بار توسط دیوید پارناس[۲] به‌عنوان یک قانون طراحی معرفی گردید. مطابق این قانون، جزییات پیاده‌سازی یک بدنه‌ی نرم‌افزاری، برای کاهش اثرات تغییر آن بر سایر قسمت‌های سیستم نرم‌افزاری، محافظت می‌شود. طراحی زبان‌های برنامه‌نویسی شئ‌گرا مانند C++، جاوا و C از این قانون پشتیبانی می‌نماید. پنهان‌سازی اطلاعات،

---

[۱] Edsger Dijkstra

[۲] David Parnas

با اختصاصی‌کردن داده‌های یک کلاس و ثابت نگه‌داشتن واسط آن کلاس انجام می‌گردد. این کار به شکل کارآمدی، اثرات موجی و پیامدهای تغییرات صورت‌گرفته در داده‌ساختارها و پیاده‌سازی توابع را کاهش می‌دهد. با معرفی برنامه‌نویسی واسط، پنهان‌سازی اطلاعات جالب‌تر شده است. یک واسط می‌تواند با یک یا چند کلاس پیاده‌سازی شود و یک کاربر از این‌که بیش از یک پیاده‌سازی وجود دارد و اینکه کدام‌یک درحال‌حاضر درحال استفاده است، اطلاعی ندارد. این امر، مزایای قانون پنهان‌سازی اطلاعات را روشن‌تر می‌نماید.

به‌کارگیری قانون پنهان‌سازی اطلاعات در طراحی معماری، به معنای طراحی سیستم نرم‌افزاری برای محافظت‌کردن از جزییات پیاده‌سازی بخش‌هایی از سیستم از دید بقیه‌ی سیستم است. برای مثال طراحی یک سیستم نرم‌افزاری که از یک پایگاه‌داده استفاده می‌کند را درنظر بگیرید. قانون طراحی برای تغییر توصیه می‌کند که امکان تغییر سیستم مدیریت پایگاه‌داده باید در طراحی در نظر گرفته شود. طراحی باید این جزییات پیاده‌سازی را از دید کل سیستم مخفی نگه دارد. برای این کار طراحی باید از الگوی «پل» (فصل ۱۷) برای پنهان‌سازی جزییات پیاده‌سازی که نشان می‌دهد از کدام DBMS استفاده شده است، پیروی کند.

به‌عنوان یک مثال دیگر، فرض کنید که لازم است الگوریتمی روی عناصر یک داده‌ساختار اعمال شود. این داده‌ساختار می‌تواند یک درخت، یک جدول درهم‌سازی، یک لیست پیوندی، یا هر چیز دیگر باشد. بسیار مناسب است که الگوریتم بتواند بدون این‌که نگران نوع داده‌ساختار باشد، برای پردازش عناصر آن به‌کار رود. الگوی «تکرار کننده»[۱] که در فصل ۱۶ معرفی شده است، این نیاز را برآورده می‌سازد. یعنی یک واسط عمومی، که توابع لازم برای پیمایش یک مجموعه‌داده را فراهم می‌کند، به شکل زیر تعریف می‌گردد:

- *First()*: این تابع، نشانگر[۲] را طوری تنظیم می‌کند که به اولین عضو مجموعه داده اشاره کند.
- *Next()*: این تابع، نشانگر را به عضو بعدی مجموعه داده جابه‌جا می‌کند.
- *IsDone() : Boolean* این تابع، در صورتی‌که همه‌ی اعضای مجموعه داده پیمایش شده باشند، مقدار true را برمی‌گرداند.
- *getElement() : object* این تابع، عضوی را که نشانگر به آن اشاره می‌کند، برمی‌گرداند.

هر داده‌ساختار واقعی مانند درخت، لیست پیوندی، جدول درهم‌سازی، فرهنگ‌داده، و مانند آن، واسط را طوری پیاده‌سازی می‌کند که توابع واقعی برای پیمایش خود را فراهم سازد. به‌عنوان مثال یک لیست پیوندی، first() را طوری پیاده‌سازی می‌کند که نشانگر به اولین عضو لیست پیوندی اشاره کند، در حالی‌که یک درخت برای برای پیاده‌سازی این تابع، نشانگر را برای اشاره به ریشه‌ی درخت تنظیم خواهد کرد. پیاده‌سازی next() برای یک لیست پیوندی، نشانگر را به‌سادگی یک عضو جلو خواهد برد، ولی در یک درخت این کار بسته به نوع پیمایش درخت (پیش‌ترتیب، میان‌ترتیب، پس‌ترتیب) انجام خواهد شد. به این ترتیب، کاربر فقط مجموعه اعضای داده را می‌بیند و نوع پیاده‌سازی و جزییات سازمان‌دهی پیمایش داده‌ساختار از دید او پنهان خواهد ماند.

۶-۵-۵- چسبندگی زیاد

ایده‌ی قانون چسبندگی زیاد، از طراحی پیمانه‌ای در تحلیل و طراحی ساختمند سنتی گرفته شده است. در طراحی ساختمند، سیستم نرم‌افزاری به یک سلسله‌مراتب درختی از پیمانه‌ها تقسیم می‌شود که در آن، پیمانه‌های سطوح بالاتر، پیمانه‌های سطوح پایین‌تر را فراخوانی می‌کنند و نتایج بازگردانده شده از آن‌ها را ترکیب می‌نمایند. هر پیمانه، یک مولفه یا زیرسیستم را پیاده‌سازی می‌کند و از یک مجموعه داده تشکیل می‌شود. چسبندگی، میزان ارتباط این توابع با ماموریت اصلی آن پیمانه را اندازه‌گیری می‌کند. چسبندگی زیاد، قابلیت فهم، استفاده‌ی مجدد، و قابلیت نگهداری نرم‌افزار را افزایش می‌دهد. قانون چسبندگی زیاد

---

[۱] Iterator Pattern

[۲] Cursor

توصیه می‌کند که طراحی پیمانه‌ها باید طوری باشد که توابع هر پیمانه، بیشترین درجه‌ی ارتباط با مسئولیت اصلی پیمانه را داشته باشند. اعمال قانون چسبندگی زیاد در طراحی معماری به این معناست که مولفه‌ها و کلاس‌های هر زیرسیستم باید تا حدود زیادی به مسئولیت اصلی زیرسیستم مرتبط باشند.

۶-۵-۶- جفت‌شدگی کم

قانون جفت‌شدگی کم نیز از تحلیل و طراحی ساختمند نشأت گرفته است. در طراحی ساختمند، جفت‌شدگی درجه‌ی اثر یک پیمانه را در زمان اجرا بر پیمانه‌های دیگر اندازه می‌گیرد و این‌کار را با توجه به وابستگی‌ها و تعاملات بین پیمانه‌ها انجام می‌دهد. به‌عبارت دیگر، درجه تاثیر یک پیمانه‌ی مشخص در زمان اجرا بر رفتارهای پیمانه‌های دیگر سنجیده می‌شود. علاوه بر جفت‌شدگی از طریق رفتار زمان‌اجرا، قانون جفت‌شدگی کم، اثر تغییر در یک پیمانه بر پیمانه‌های دیگر را نیز اندازه‌گیری می‌کند. به عبارت دیگر، تعیین می‌نماید که وقتی پیاده‌سازی یک پیمانه‌ی مشخص تغییر می‌کند، چند پیمانه‌ی دیگر باید تغییر داده شوند.

جفت‌شدگی زیاد به‌دلیل بالابودن درجه‌ی وابستگی بین پیمانه‌ها، عدم‌قطعیت را در اثر زمان‌اجرا و تغییرات افزایش می‌دهد. به علاوه آزمون، استفاده‌ی مجدد، نگهداری، و تغییر پیمانه‌های چنین نرم‌افزاری دشوار است. دراین‌حالت، آزمون کاری دشوار است زیرا اگر پیمانه‌ی M1 به پیمانه‌ی M2 بستگی داشته باشد و M1 قبل از پیمانه‌ی M2 آزموده شود، آنگاه پیمانه‌ای که برای شبیه‌سازی M2 ساخته می‌شود، باید برای آزمون M1 هم ساخته و مورد استفاده قرار گیرد. ساختن این بدنه‌های آزمون[۱]، کاری پرهزینه و زمان‌بر است. به‌خصوص در زمانی‌که تعداد زیادی از این پیمانه‌ها مورد نیاز باشد. استفاده‌ی مجدد درصورت جفت‌شدگی زیاد، کاری دشوار است، زیرا درجه‌ی بالای وابستگی به این معناست که برای استفاده‌ی مجدد یک پیمانه، باید همه‌ی پیمانه‌هایی که این پیمانه به آن‌ها وابسته است نیز مورد استفاده‌ی مجدد قرار بگیرند. تغییر و نگهداری در این حالت دشوار است، زیرا تغییر یک پیمانه ممکن است بر بسیاری پیمانه‌های دیگر تاثیر داشته باشد. جفت‌شدگی کم، عدم‌قطعیت ذکرشده را کاهش می‌دهد و فهم، آزمون، استفاده‌ی مجدد، و نگهداری برنامه را آسان می‌نماید.

یکی از انواع جفت‌شدگی که معمولاً دیده می‌شود، جفت‌شدگی کنترل است. پیمانه‌ها از طریق یک متغیر به نام متغیر کنترلی با هم ارتباط برقرار می‌کنند. مقدار متغیر کنترلی نشان می‌دهد که کدام جریان کنترلی در زمان اجرا دنبال می‌گردد. برای مثال، پارامتری که در یک جمله‌ی Switch به‌عنوان Case استفاده می‌گردد، یک متغیر کنترلی است. این نشان می‌دهد که رفتار یک شئ توسط شئ دیگر کنترل می‌گردد. از آنجا که مقدار متغیر، تنها در زمان اجرا مشخص می‌شود، رفتار شئ که از این متغیر کنترلی استفاده می‌کند نمی‌تواند به‌سادگی پیش‌بینی شود، آزموده شود، و عیب‌یابی گردد. طراحی واقعی زیر به ما می‌گوید که یک متغیر کنترلی تا چه حد می‌تواند دردسرساز باشد.

داستان در مورد دو پیمانه است که توسط دو برنامه‌نویس توسعه‌یافته بودند. این دو برنامه‌نویس تصمیم گرفتند که دو پیمانه از طریق یک متغیر کنترلی با هم در ارتباط باشند. دو برنامه‌نویس در ملاقات‌های خود در زمان صرف نهار، درمورد مقادیر و این‌که قرار است هر مقدار به چه چیز اشاره کند به توافق رسیده بودند. در ابتدا قرار براین بود که این متغیر سه مقدار ممکن داشته باشد. با گذشت زمان، این تعداد به نُه مقدار افزایش یافت. یکپارچه‌سازی این دو پیمانه به یک کابوس شبانه تبدیل شده بود. بیش از یک ماه طول کشید تا روشن شد که این نُه مقدار در دو پیمانه معنی‌های متفاوتی دارند. در نهایت، برنامه‌نویس‌ها مجبور شدند که برای رفع مشکل، بیشتر کارهای انجام‌داده‌ی خود را دوباره انجام دهند. درسی که از این ماجرا می‌توان گرفت این است که تا حد امکان باید از استفاده از متغیرهای کنترلی اجتناب شود.

استفاده از قانون جفت‌شدگی کم در طراحی معماری، به معنای کاهش اثرات زمان‌اجرا و تاثیر تغییر در هر زیرسیستم بر

---

[۱] Test Stubs

زیرسیستم‌های دیگر است. به‌خصوص، طراحی باید از متغیرهای کنترلی دارای بیش از دو مقدار اجتناب نماید. به‌علاوه، برای کاستن تاثیر تغییر، می‌توان از قوانین طراحی برای تغییر و پنهان‌سازی اطلاعات استفاده کرد.

۶-۵-۷- ساده و احمقانه فرض کن

قانون ساده و احمقانه فرض کن، طراحی‌های ساده، سرراست، و قابل‌فهم را توصیه می‌نماید. از نگاه طراحی شیءگرا، این قانون می‌تواند به طراحی «اشیای نادان» تعبیر گردد. به‌طور خلاصه، یک «شیء نادان» شیئی است که به اندازه کافی ساده‌گیر، کندذهن و به همین دلیل دوست‌داشتنی است. یک شیء، ساده‌گیر است اگر و فقط اگر روش انجام یک کار، و نه چیز دیگر را بداند. برای مثال، یک ابزار نرم‌افزاری را درنظر بگیرید که معیارهای مختلفی از نرم‌افزار را، که کاربر با علامت‌زدن گزینه‌های مختلف، مشخص کرده است، محاسبه می‌نماید. یک روش این است که از یک سری جملات شرطی استفاده شود. یعنی اگر iامین گزینه انتخاب شده باشد، نرم‌افزار معیار iام را محاسبه کند. این شیء، ساده‌گیر نیست زیرا سؤال‌های مختلفی می‌پرسد، یعنی نیاز دارد که بررسی کند کدام گزینه‌ها انتخاب شده‌اند. این شیء کندذهن هم نیست، زیرا می‌داند که چگونه همه‌ی معیارها را محاسبه کند.

یک روش دیگر این است که یک کلاس مجرد معیار تعریف شود و تعداد زیادی کلاس معیار برای محاسبه‌ی معیارها معرفی شود. این راه، به طراحی «اشیای نادان» کمک می‌کند. به‌علاوه، برای هر هزینه، طراحی باید یک شیء معیار ایجاد کند که دائماً انتخاب گزینه‌ها را زیرنظر داشته باشد. وقتی یک گزینه انتخاب می‌شود، معیار مربوطه محاسبه می‌شود، و هیچ سؤالی پرسیده نمی‌شود. در این روش، هر شیء معیار، تنها می‌داند که چگونه یک معیار را محاسبه کند و چیز دیگری نمی‌داند. به‌این‌ترتیب، «اشیای نادان» طراحی شده‌اند.

به‌کارگیری این قانون در طراحی معماری، به‌معنای طراحی معماری برای استفاده از «اشیای نادان» است. قسمت چهار کتاب (استفاده از الگوهای خاص موقعیت)، مثال‌های کاربردی بیشتری، شامل طراحی یک چارچوب مانا، برای این قانون ارائه می‌دهد.

۶-۶ رهنمون‌هایی برای طراحی معماری

رهنمون‌های زیر با وجود آن‌که در گام‌های قبل نیز مورداشاره قرار گرفتند، ارزش تکرار مجدد را دارند.

۱. در صورت امکان، یک سبک معماری تعریف کنید. در بسیاری از کاربردها می‌توان از یک سبک معماری استفاده کرد و یا سبک جدید جدیدی تعریف نمود. این کار باعث صرفه‌جویی در زمان و کار لازم می‌گردد. با توجه به نوع زیرسیستم تحت‌توسعه، یک سبک معماری تعریف کنید.
۲. از قوانین طراحی نرم‌افزار استفاده کنید. همان‌طور که پیش‌تر گفته شد، قوانین طراحی نرم‌افزار، قوانین راهنمایی برای این کار هستند که کیفیت و ویژگی‌های مطلوب را به طراحی معماری می‌آورند. طراحی معماری یک نرم‌افزار باید برای طراحی برای تغییر، جداسازی دغدغه‌ها، چسبندگی‌بالا، جفت‌شدگی کم، پنهان‌سازی‌اطلاعات، و ساده‌گیری طراحی تلاش نمایند.
۳. از الگوهای طراحی استفاده کنید. الگوهای طراحی، در زمان تدوین قوانین طراحی نرم‌افزار، مشکلات معمول طراحی را حل می‌کنند. به‌این‌ترتیب، استفاده از الگوها در زمان طراحی معماری، کیفیت را به طراحی وارد کرده، باعث صرفه‌جویی در زمان و کار لازم می‌گردد. نکته مهم‌تر این است که الگوها می‌توانند برای حل مسائل جدید و چالش‌برانگیز طراحی معماری، به‌شکل‌های خلاقانه با هم ترکیب شوند. در قسمت چهار کتاب (استفاده از الگوهای خاص‌موقعیت) به این موضوع پرداخته شده است.

۴. طراحی را از دید اهداف و قوانین خاص آن بررسی کنید. طراحی معماری باید از لحاظ اهداف طراحی، مانند مشخص‌ساختن پایگاه‌داده از دید اشیای کسب‌وکار، فراهم‌ساختن یک نرم‌افزار تحمل‌پذیر خطا، و بیشینه کردن کارایی، و موارد مشابه این بررسی گردد. به‌همین‌ترتیب، طراحی معماری باید با توجه به قوانین معماری بررسی گردد تا اطمینان حاصل شود که قوانین مورد نظر طراحی رعایت شده‌اند.

۵. در صورت لزوم گام‌ها را تکرار کنید. طراحی معماری، اثر بسیار مهم و دنباله‌داری را بر کیفیت و بهره‌وری نرم‌افزار دارا می‌باشد. فرایند طراحی، یک فرایند شناختی است. به‌همین‌دلیل، صرف وقت برای تکرار گام‌ها جهت ایجاد یک طراحی خوب، ارزشمند است.

۶-۷ طراحی معماری و الگوهای طراحی

الگوهای طراحی، راه‌حل‌های ثابت‌شده‌ای برای مسائل رایج در طراحی می‌باشند. به‌همین‌دلیل است که این الگوها در طراحی معماری و سبک‌های معماری کاربرد گسترده‌ای دارند. برای مثال، معماری Nلایه که در شکل ۶-۴-ب نشان داده شده است، از الگوی کنترلگر که در فصل ۱۰ (استفاده از الگوهای واگذاری مسئولیت) آورده شده است استفاده می‌کند تا اشیای مربوط به نمایش و کسب‌وکار را ازهم جدا کند. چارچوب مانایی، الگوهای متعدد طراحی را ترکیب می‌نماید تا به اهدافی از طراحی مانند طراحی برای تغییر، چسبندگی‌زیاد، جفت‌شدگی کم، جداسازی دغدغه‌ها، و طراحی اشیای نادان برسد. الگوهای پل، دستور، پروکسی، و روش قالب[۱] برای این‌کار مورد استفاده قرار می‌گیرند. قسمت چهار کتاب (استفاده از الگوهای خاص‌موقعیت) این الگوها را به‌همراه طراحی چارچوب مانایی ارائه می‌دهد.

برخی از الگوهای طراحی به‌عنوان سبک‌های معماری درنظر گرفته می‌شوند. برای مثال الگوی مدل-دیدگاه-کنترلگر و الگوی مشاهده‌گر[۲]، سبک‌های معماری هستند. الگوی MVC از یک مدل داده، یک کنترلگر و تعدادی دیدگاه تشکیل می‌شود. کنترلگر، دیدگاه‌ها را از مدل داده جدا می‌کند و اجازه می‌دهد که داده از دیدگاه‌های مختلف نشان داده شود. الگوی مشاهده‌گر مشابه الگوی MVC است، ولی کنترلگر ندارد. این الگو از تعدادی مشاهده‌گر و یک مشاهده‌شونده تشکیل می‌شود. مشاهده‌شونده همان مدل داده در MVC است. مشاهده‌گرها همان دیدگاه‌ها هستند. مشاهده‌شونده واسطی فراهم می‌کند که مشاهده‌گرها بتوانند از طریق آن خود را برای رویدادهای موردعلاقه‌شان ثبت‌نام کنند. وقتی چنین رویدادهایی رخ می‌دهند، مشاهده‌شونده به مشاهده‌گرها اطلاع می‌دهد.

هر الگو مزایا و پیامدهایی دارد. اگر در یک معماری از یک الگو استفاده شود، در اکثر اوقات معماری می‌تواند از مزایای آن الگو استفاده کند. برای کم‌کردن پیامدهای منفی استفاده از یک الگو، می‌توان الگوها را با هم ترکیب کرد. برای‌مثال، به دلیل‌آن‌که یک مشاهده‌گر اطلاعی از وجود مشاهده‌گرهای دیگر ندارد، ممکن است یک مشاهده‌شونده در معرض به‌روزرسانی‌های همزمان قرار گیرد. این مشکل می‌تواند با ترکیب الگوی مشاهده‌گر با الگوی پروکسی، که به‌روزسازی مشاهده‌شونده را کنترل می‌کند، رفع گردد.

همان‌طور که گفته شد، دانستن ارتباط بین طراحی معماری و الگوهای طراحی می‌تواند به توسعه‌دهنده کمک کند که مزایا و معایب یک سبک معماری را درک کند. به علاوه، توسعه‌دهنده خواهد توانست معماری‌هایی طراحی کند که نیازهای کاربردهای خاص را برآورده سازد.

---

[۱] Template Method

[۲] Observer Pattern

۶-۸ به‌کارگیری قوانین چابکی

در زمان طراحی معماری، ارزش‌ها و قوانین چابکی زیر باید موردتوجه قرار گیرد:

رهنمون ۶-۱

نرم‌افزاری که به‌درستی کار می‌کند، ارزشمندتر از یک مجموعه‌ی وسیع از امکانات است.

معماری نرم‌افزار، با توجه به بازخوردهای دریافت‌شده و اعمال تغییرات درخواست‌شده در سیستم در دوره‌های متعدد تکرار رشد می‌یابد. اگرچه طراحی معماری امری مهم است، ولی باید از مستندسازی بیش از اندازه برای آن اجتناب گردد. مستندسازی وسیع، زمان و منابع باارزش را بیهوده مصرف می‌کند و فعالیت پیاده‌سازی را به‌تعویق می‌اندازد. در بسیاری از اوقات، بهترین تصمیم‌ها در دوره‌ی معماری سیستم در زمان پیاده‌سازی گرفته می‌شود. پیاده‌سازی بخش واقعی از فرایند حل مسئله است که طی آن درک تیم از مسئله افزایش می‌یابد و راه‌حل بهبود می‌یابد. این امر در بسیاری از مواقع سبب تغییر و بهبود معماری می‌گردد. به‌این‌ترتیب، با درنظرگرفتن اهمیت بیشتر برای نرم‌افزاری که کارمی‌کند نسبت به مستندات وسیع، طراحی معماری نیز ارتقا می‌یابد.

رهنمون ۶-۲

قانون ۲۰/۸۰ اجرا شود: چیزی که به اندازه‌ی کافی خوب است، کافی است.

هیچ معماری بهینه‌ای برای سیستم وجود ندارد. این به‌دلیل آن است که طراحی معماری یک مسئله‌ی بدرفتار است. همه‌ی ویژگی‌های یک مسئله‌ی بدرفتار در طراحی معماری دیده می‌شود، یعنی هیچ قانونی برای پایان کار وجود ندارد. یک طراحی همیشه می‌تواند ارتقا پیداکند. از سوی دیگر، از یک زمان به بعد، بازگشت‌ها برای ارتقای بیشتر طراحی کم می‌شود. بنابراین، آنچه به‌حدکافی خوب است، کافی است. برای بسیاری از پروژه‌های واقعی، می‌توان بسته به نوع سیستم و زیرسیستم‌های آن، به‌سرعت ترکیبی از چند سبک معماری و چند الگوی طراحی را انتخاب نمود و به‌کار برد. پس از این کار، معماری در دوره‌های تکرار بعدی تغییر خواهد کرد و بهتر خواهد شد.

خلاصه

این فصل، فرایند طراحی معماری و اهمیت طراحی معماری را نشان می‌دهد. به‌علاوه، چهار نوع اصلی از سیستم‌ها به‌همراه ویژگی‌های آن‌ها به‌تفصیل مورد بررسی قرار می‌گیرد. این چهار نوع سیستم عبارتند از: سیستم‌های تعاملی، سیستم‌های رویداد-رانده، سیستم‌های تبدیلی، و سیستم‌های انباره‌ی ماندگار. یک سیستم نرم‌افزاری می‌تواند از یک یا چند زیرسیستم از این انواع تشکیل گردد. روش‌ها و تکنیک‌های طراحی برای این چهار نوع سیستم متفاوت می‌باشند.

در این فصل، تاثیر نوع سیستم بر روش‌های طراحی مورد بحث قرار می‌گیرد. به‌علاوه، سبک‌های مختلف معماری نرم‌افزار و مزایا و محدودیت‌های هر یک در این فصل شرح داده می‌شود. سبک‌های مورد بحث شامل Nلایه، مشتری-خادم، برنامه‌ی اصلی

و زیربرنامه‌ها، رویداد-رانده، چارچوب ماندگاری، و سبک‌های دیگر معماری می‌باشند. نحوه‌ی انتخاب یک سبک معماری با توجه به نوع سیستم و یا زیرسیستم تحت توسعه نیز در این فصل مورد اشاره قرار می‌گیرد. علاوه بر موارد قبل، قوانین طراحی نرم‌افزار در این فصل ارائه شده‌اند. این قوانین شامل طراحی برای تغییر، جداسازی دغدغه‌ها، پنهان‌سازی اطلاعات، چسبندگی زیاد، جفت‌شدگی کم، و ساده‌گرفتن توسعه‌ی نرم‌افزار می‌باشند. قوانین طراحی نرم‌افزار، کل فرایند طراحی، و نه فقط طراحی معماری را هدایت می‌نمایند.

برای مطالعه‌ی بیش‌تر

کتاب تألیفی تیلور و همکاران [۱۴۸] یک مرجع عالی برای طراحی معماری نرم‌افزار است. فعالیت‌های تحلیلی و طراحی، به همراه شرح ساختمند سبک‌های مختلف معماری در این کتاب مورد توجه قرار گرفته است. در [۱۸]، بس، کلمنتز، و کازمن مفاهیم طراحی معماری و بهترین روش‌های آن را معرفی می‌نمایند. در این کتاب، زمینه‌ی کسب‌وکار برای طراحی معماری در نظر گرفته شده است. کتاب تالیفی کلمنت و همکاران [۴۸] بر مستندسازی یک معماری با هدف استفاده‌ی دیگران، نگهداری، و ساخت یک سیستم بر مبنای این معماری تمرکز می‌نماید. مراجع [۷] و [۱۵۳] روش‌هایی را برای بررسی تطابق کد با طراحی معماری ارائه می‌دهند. مرجع [۶۷] کتابی کلاسیک در زمینه‌ی الگوهای طراحی نرم‌افزار است که ۲۳ الگوی شناخته‌شده را معرفی می‌نماید. لارمن [۱۰۴] این الگوها را به همراه ۹ الگوی GRASP مورد بررسی قرار داده است. (GRASP مخفف معادل واژه‌ی الگوهای نرم‌افزاری عمومی تخصیص وظایف[1] می‌باشد.) کتاب‌های تالیفی بوچ، جاکوبسن، و رامبو [۳۶] و [۹۱] و [۱۳۲]، UML و فرایند یکنواخت را که فعالیت طراحی معماری را دربرمی‌گیرد، معرفی می‌نماید.

سوال‌های مروری فصل

۱. طراحی معماری چیست؟
۲. چرا طراحی معماری مهم است؟
۳. فرایند طراحی معماری چیست؟
۴. سبک معماری چیست؟
۵. ارتباط بین انواع سیستم و سبک‌های طراحی چیست؟
۶. قوانین طراحی نرم‌افزار کدامند؟
۷. چگونه قوانین طراحی نرم‌افزار برای طراحی سبک‌های معماری به‌کار گرفته می‌شوند؟

[1] General Responsibility Assignment Software Pattern

تمرین‌ها

۶-۱ جدولی تشکیل دهید که سطرهای آن نشان‌دهنده‌ی سبک‌های معماری در شکل ۶-۳، و ستون‌های آن نشان‌دهنده‌ی قوانین طراحی ذکرشده در این فصل باشد. این جدول را طوری تکمیل کنید که نشان دهد کدام سبک معماری کدام قوانین طراحی را به کار می‌گیرد.

۶-۲ برای هر یک از سبک‌های معماری فهرست‌شده در تمرین قبل، به‌اختصار شرح دهید که چگونه از قوانین طراحی استفاده می‌کند و به چه مزایایی می‌رسد؟

۶-۳ در تمرین ۴-۴، شرح نیازمندی‌های یک سیستم مدیریت تقویم را ایجاد نمودید. در این تمرین کارهای زیر را انجام دهید:

الف) نوع سیستم را شناسایی کنید و به‌اختصار پاسخ خود را توجیه نمایید.

ب) یک سبک معماری را انتخاب کنید و یک طراحی معماری برای سیستم ایجاد کنید.

ج) عملکرد و واسط هر یک از زیرسیستم‌ها و مولفه‌های طراحی معماری را شرح دهید.

د) توضیح دهید که کدام یک از قوانین طراحی نرم‌افزار مورد استفاده قرار گرفته‌اند، چگونه این‌کار انجام شده است، و مزایای استفاده از هر قانون چه بوده است. به‌علاوه درصورت وجود مشکلات احتمالی، این مشکلات را مشخص نمایید.

۶-۴ برای سیستم مدیریت تحصیل در خارج از کشور که در فصل ۴ معرفی شد، کارهای زیر را انجام دهید:

الف) نوع سیستم را تشخیص دهید و به‌اختصار پاسخ خود را توجیه نمایید.

ب) یک معماری برای سیستم ایجاد کنید.

ج) عملکرد و واسط هریک از زیرسیستم‌ها و مولفه‌های طراحی معماری را شرح دهید.

د) توضیح دهید که کدام یک از قوانین طراحی نرم‌افزار مورد استفاده قرار گرفته‌اند، چگونه این‌کار انجام شده است، و مزایای استفاده از هر قانون چه بوده است. به‌علاوه به مشکلات احتمالی مربوط به کاربرد قوانین طراحی اشاره نمایید.

۶-۵ سیستم حمل‌ونقل چمدان در فرودگاه، که در فصل ۳ معرفی شد، را درنظر بگیرید. زیرسیستم حامل از اسکنرهای بارکد و پیشرانه‌هایی برای هدایت چمدان‌ها در مسیرشان روی تسمه‌های متحرک استفاده می‌کند. یک زیرسیستم نرم‌افزاری وجوددارد که با این دو نوع وسیله کار می‌کند. نوع این زیرسیستم را مشخص کنید. توجه کنید که زیرسیستم ممکن است در بیش از یک نوع زیرسیستم درگیر باشد. سبک(های) معماری مورد استفاده را انتخاب کنید. به‌علاوه، طرحی از طراحی معماری تهیه کنید و عملکرد هر یک از زیرسیستم‌ها و مولفه‌های آن را مشخص نمایید.

۶-۶ در تمرین ۴-۵، برای یک سیستم رزرو چنداتاقه مبتنی بر وب، SRS تهیه نموده‌اید. در این تمرین، شما باید نوع سیستم را شناسایی کنید و طراحی معماری آن را رسم کنید. به‌علاوه به‌اختصار شرح دهید که عملکرد هر یک از زیرسیستم‌ها و اجزای طراحی معماری چیست.

سوم

مدل‌سازی و طراحی سیستم‌های تعاملی

فصل هفتم استنتاج مورد کاربردها از نیازمندی‌ها
فصل هشتم مدل‌سازی تعامل کنشگر-سیستم
فصل نهم مدل‌سازی تعامل شئ
فصل دهم اعمال الگوهای واگذاری مسئولیت
فصل یازدهم استنتاج نمودار کلاس طراحی
فصل دوازدهم طراحی واسط کاربر

۷

استنتاج مورد کاربردها از نیازمندی‌ها

نکات کلیدی

- مورد کاربرد یک فرایند کسب‌وکاری است؛ که با یک کنشگر شروع و با همان کنشگر تمام می‌شود و یک وظیفه‌ی کسب‌وکاری را برای آن کنشگر به انجام می‌رساند.
- مورد کاربردها از نیازمندی‌ها به‌دست می‌آیند (استنتاج می‌شوند) و نیازمندی‌ها را برآورده می‌سازند.
- برنامه‌ریزی برای توسعه و استقرار مورد کاربردها و زیرسیستم‌ها به‌گونه‌ای انجام می‌شود که بر طبق نیازهای کسب‌وکاری و اولویت‌های مشتری باشد.

فصل ۴ روش‌ها و تکنیک‌هایی را برای جمع‌آوری و تبیین نیازمندی‌ها معرفی می‌نماید. نیازمندی‌ها عبارتند از قابلیت‌هایی که سیستم باید ارائه دهد. بیان نیازمندی‌ها جملات اعلانی هستند که فقط می‌گوید سیستم باید چه قابلیت‌هایی ارائه نماید، ولی در خصوص چگونگی ارائه‌ی آن‌ها چیزی نمی‌گوید. این امر به برنامه‌ساز آزادی می‌دهد تا بهترین راه‌حل نرم‌افزاری را طراحی و پیاده‌سازی نماید. ولی به‌دلیل نبود نگرشی از کسب‌وکار و دانش قلمرو، امکان دارد بهترین راه‌حل نرم‌افزاری از نقطه نظر برنامه‌ساز، انتظارات کاربر را برآورده ننماید.

«مورد کاربرد» راه‌حلی برای این مشکل ارائه می‌کند. برای مثال، یک سیستم اطلاعات کتابخانه[۱] را در نظر بگیرید. شاید یکی از نیازمندی‌ها بگوید که LIS باید به عضو اجازه‌ی دریافت مستندات را بدهد. مورد کاربردی که از این نیازمندی به دست می‌آید مشخص می‌کند که چگونه سیستم با عضو تعامل خواهد داشت تا این قابلیت تأمین شود. علاوه بر این بخش که نشانگر تعامل کنشگر سیستم است، در هر مورد کاربرد یک بخش پردازش پیش‌زمینه نیز وجود دارد. برای مثال، برای دریافت یک مستند، باید یک شئ امانت ایجاد شده و در پایگاه‌داده ذخیره گردد. شئ سند و شئ عضو[۲] نیز باید به‌روزرسانی شده و مجدداً در پایگاه‌داده ذخیره گردند. مدل‌سازی و طراحی تعامل کنشگر سیستم در فصل ۸ و پردازش پیش‌زمینه یا مدل‌سازی تعامل اشیا در فصل ۹ معرفی شده‌اند. در این فصل، موارد زیر را یاد خواهید گرفت:

- چگونه مورد کاربردها از نیازمندی‌ها استنتاج شوند؟

---

[۱] Library Information System (LIS)

[۲] Patron

- چگونه قلمرو هر مورد کاربرد مشخص شود؟ یعنی، یک مورد کاربرد چه موقع و کجا شروع و چه موقع خاتمه می‌یابد؟
- چگونه مورد کاربردها و روابط آن‌ها با کنشگرها و زیرسیستم‌ها به تصویر کشیده شوند؟

۷-۱ کنشگر چیست؟

سیستم‌های نرم‌افزاری اطلاعات را برای کاربردهای مورد نظر پردازش می‌کنند. سیستم درخواست‌ها و ورودی‌ها را از کاربران دریافت کرده و نتایج را به کاربران تحویل می‌دهد. در برخی موارد، ممکن است یک سیستم نرم‌افزاری خود درون یک سیستم بزرگ‌تر، که می‌تواند یک سیستم نرم‌افزاری-سخت‌افزاری یا مرکب از دیگر زیرسیستم‌ها باشد، تعبیه شده باشد. در این موارد، سیستم در حال توسعه ممکن است درخواست‌هایی از دستگاه‌های سخت‌افزاری یا سایر زیرسیستم‌های سیستم کلی دریافت کرده و نتایجی را به آن‌ها برگرداند. از آن‌جا که رسم بر آن بوده است که اصطلاح کاربر یا «استفاده‌کننده» به کاربران انسانی اطلاق شود، نیاز به واژه‌ای خنثی با قلمروی وسیع‌تر بوده تا هم به کاربران انسانی و هم به کاربران غیرانسانی اشاره داشته باشد. تعریف زیر این واژه را معرفی می‌کند.

تعریف ۷-۱

کنشگر عبارت است از یک نقش (کسب‌وکاری) که به‌وسیله‌ی و به‌واسطه‌ی مجموعه‌ای از موجودیت‌های (کسب‌وکاری) یا ذی‌نفعان، که بیرون از سیستم بوده و با آن در تعامل هستند، بازی می‌شود.

در این تعریف می‌بینیم که کنشگر، نقشی است که توسط برخی موجودیت‌ها بازی می‌شود، نه خود موجودیت‌ها. این جداسازی وقتی مفید است که یک موجودیت در یک زمان دو یا چند نقش را بازی کند. مثلاً، یک کتابدار کتابی را برداشته و تراکنش خروج را اجرا می‌کند. در این جا، شخص دو نقش بازی می‌کند: یکی نقش کتابدار و دیگری نقش عضو.

۷-۲ مورد کاربرد چیست؟

کاربران با سیستم تعامل می‌کنند تا یک وظیفه‌ی کسب‌وکاری مشخص را انجام دهند. اغلب اوقات، تعامل از یک الگوی مشخص پیروی می‌کند. برای مثال، برای برداشت پول از یک دستگاه خودپرداز [۱] مشتری بانک، کارت بانکی‌اش را وارد دستگاه می‌کند، ATM صفحه‌ی ورود را نمایش می‌دهد، مشتری شماره کارت[۲] خود را وارد می‌کند، ATM فهرست اصلی را نشان می‌دهد و الی آخر. این قبیل دنباله‌ای از گام‌ها، یک فرایند کسب‌وکار، یا به عبارتی، چگونگی استفاده‌ی یک کاربر از سیستم برای انجام یک وظیفه‌ی کسب‌وکاری را مشخص می‌کنند. در توسعه‌ی سیستم باید تمرکز بر روی شناسایی چنین فرایندهای کسب‌وکاری باشد، که به‌وسیله‌ی مورد کاربردها مدل‌سازی می‌شوند. توصیف مورد کاربرد در ادامه می‌آید.

تعریف ۷-۲

مورد کاربرد یک فرایند کسب‌وکار است. مورد کاربرد با یک کنشگر شروع‌شده، با آن کنشگر خاتمه می‌یابد، و یک وظیفه‌ی کسب‌وکاری را برای آن کنشگر به انجام می‌رساند.

---

[۱] Automated Teller Machine (ATM)

[۲] Pin Number

تعریف ۷-۲ چهار صفت مهم را برای یک مورد کاربرد در نظر می‌گیرد:

۱. هر مورد کاربرد یک فرایند کسب‌وکاری است. این اصلی‌ترین ویژگی یک مورد کاربرد است. یک سیستم ATM را در نظر بگیرید که به مشتری اجازه می‌دهد وجهی را به حساب واریز نماید، موجودی را بررسی کند، وجهی از حساب برداشت کند، و بین حساب‌های بانکی وجهی را نقل و انتقال دهد. از این‌ها چهار فرایند کسب‌وکاری پیشنهاد می‌شود: واریز وجه[۱]، برداشت وجه[۲]، بررسی موجودی[۳]، و انتقال وجه[۴].
۲. هر مورد کاربرد باید توسط یک کنشگر آغاز گردد. مشتری ATM باید کارت بانکی‌اش را وارد کند تا فرایندهای کسب‌وکاری ATM شروع شوند.
۳. هر مورد کاربرد باید توسط کنشگر خاتمه یابد به‌گونه‌ای که کنشگر بداند که فرایند کسب‌وکاری به‌طور موفقیت‌آمیز پایان یافته است. برای مثال، فرایند کسب‌وکاری واریز وجه وقتی خاتمه می‌یابد که مشتری ATM رسید واریز خود را دریافت نماید. فرایند کسب‌وکاری بررسی موجودی وقتی تمام می‌شود که مشتری ATM دکمه‌ی OK را فشار دهد و تأیید کند که می‌خواهد موجودی خود را مشاهده نماید.
۴. هر مورد کاربرد باید یک وظیفه‌ی کسب‌وکاری را برای کنشگر انجام دهد. برای مثال، برداشت وجه به مشتری ATM اجازه می‌دهد پولی از حساب خویش برداشت نماید.

مباحث فوق نشان می‌دهند که واریز وجه، برداشت وجه، بررسی موجودی و انتقال وجه چهار صفت یک مورد کاربرد را دارند؛ بنابراین، این‌ها می‌توانند به‌عنوان مورد کاربردهایی برای برنامه‌ی ATM تلقی شوند.

۷-۳ فرایند کسب‌وکاری، عمل، و کنش

هر مورد کاربرد یک فرایند کسب‌وکاری[۵] است، نه یک عمل[۶] یا کنش[۷]. بار دیگر برنامه‌ی ATM را در نظر بگیرید: شخصی ممکن است فکر کند که مواردی چون «گرفتن موجودی از پایگاه‌داده» یا «برقراری ارتباط با سرور بانک» یا «وارد کردن رمز» می‌توانند مورد کاربرد تلقی شوند. ولی یک بررسی دقیق نشان می‌دهد که این‌ها فرایندهای کامل کسب‌وکاری نیستند؛ بلکه گام‌ها یا اعمالی از یک فرایند کسب‌وکاری هستند. برای مثال، «وارد کردن رمز» را در نظر بگیرید. این در واقع یک گام از فرایند ورود به سیستم است، که خودش یک مورد کاربرد خواهد بود. به همین منوال، «گرفتن موجودی از پایگاه‌داده» نیز یک فرایند کسب‌وکاری نبوده و با کنشگر شروع نمی‌شود. این در واقع یک عمل از مورد کاربرد بررسی موجودی یا برداشت وجه می‌باشد. احتمال دیگر این است که این در واقع قرار بوده همان بررسی موجودی باشد که یک فرایند کسب‌وکاری محسوب می‌شود. برای تمایز بین فرایند کسب‌وکاری، عمل، و کنش، تعاریف زیر ارائه می‌شوند.

تعریف ۷-۳

فرایند کسب‌وکاری عبارت است از دنباله‌ای از گام‌های پردازش اطلاعات که برای انجام یک وظیفه‌ی کسب‌وکاری کامل (با منظور مشخص)، لازم و کافی هستند.

---

[۱] Deposit Money
[۲] Withdraw Money
[۳] Check Balance
[۴] Transfer Money
[۵] Business Process
[۶] Operation
[۷] Action

در این‌جا عبارت «انجام یک وظیفه‌ی کسب‌وکاری کامل (با منظور مشخص)» ضروری است زیرا وجه تمایز بین فرایند کسب‌وکاری با عمل یا کنش همین است.

تعریف ۷-۴

عمل عبارت است از دنباله‌ای از کنش‌ها یا دستورالعمل‌ها که برای انجام یک گام از یک فرایند کسب‌وکاری انجام می‌شوند.

عمل، یک وظیفه‌ی کسب‌وکاری را انجام نمی‌دهد، بلکه تنها یک گام از یک فرایند کسب‌وکاری را اجرا می‌کند. برای مثال «گرفتن موجودی از پایگاه‌داده» یک وظیفه‌ی کسب‌وکاری انجام نمی‌دهد، گرچه بخشی از یک وظیفه‌ی کسب‌وکاری را اجرا می‌کند. به همین شکل، «وارد کردن رمز» تنها یک گام از فرایند ورود به سیستم را انجام می‌دهد. لذا، این‌ها مورد کاربرد نیستند.

تعریف ۷-۵

کنش عبارت است از یک حرکت، عمل، یا دستورالعمل تقسیم‌نشدنی که در حین اجرای یک عمل، انجام می‌شود.

مثال ۷-۱

طراحی و پیاده‌سازی یک ویرایشگر گرافیکی برای ترسیم نمودارهای کلاس UML را در نظر بگیرید. ویرایشگر امکان ایجاد نمودارهای جدید، ذخیره، حذف و ویرایش نمودارها را به کاربر می‌دهد. مورد کاربردهای این ویرایشگر کدامند؟

جواب: دامنه‌ی کاربردی این ویرایشگر، ابزار نرم‌افزاری/ویرایشگر گرافیکی/ ویرایشگر نمودارهای UML می‌باشد. در نگاه اول، ممکن است این مورد کاربردها شناسایی شوند: «ایجاد نمودار جدید»، «ذخیره‌ی نمودار»، «حذف نمودار» و «ویرایش نمودار». تعیین این‌که این‌ها واقعاً مورد کاربرد باشند نیاز به تحلیل دقیق دارد. ابتدا «ایجاد نمودار» را در نظر بگیرید. سؤالی که باید پرسیده شود این است: «آیا این یک فرایند کسب‌وکاری است؟» اگر هست، هدف کسب‌وکاری آن کدام است؟ آیا کاربر نمودار را فقط ایجاد می‌کند برای این‌که آن را ایجاد کرده باشد؟ معمولاً کاربر یک نمودار جدید ایجاد می‌کند، اعمال ویرایشی از قبیل افزودن، تغییر و حذف کلاس‌ها و روابط بین آن‌ها را انجام می‌دهد، و سپس آن را ذخیره می‌کند. بنابراین فرایند کسب‌وکار باید «ویرایش یک نمودار جدید» باشد، و در واقع «ایجاد نمودار جدید» و «ذخیره‌ی نمودار» اعمالی از مورد کاربرد «ویرایش یک نمودار جدید» هستند.

امّا، یک تحلیل‌گر ممکن است بگوید که «ایجاد نمودار جدید» و «ذخیره‌ی نمودار» فرایندهای کسب‌وکاری هستند. اشکالی در این برداشت نیست، چون افراد مختلف ممکن است درک متفاوتی از جهان داشته باشند. سؤال اینجاست که کدام بهتر است؟ برای یافتن پاسخ، بیایید فرض کنیم که «ایجاد نمودار جدید»، «ذخیره‌ی نمودار» و «ویرایش نمودار» هر سه به‌عنوان مورد کاربرد در نظر گرفته شوند. از آن‌جا که تحویل افزایشی به تیم این امکان را می‌دهد که برخی مورد کاربردها را قبل از دیگران تحویل دهد، بنابر این، فرض می‌کنیم که «ایجاد نمودار جدید» ابتدا تحویل گردد. در این حالت، کاربر نمی‌تواند هیچ کار معناداری، مانند ویرایش نمودار، را انجام دهد. این بدان معنی است که «ایجاد نمودار جدید» نباید به‌عنوان یک مورد کاربرد در نظر گرفته شود. به‌همین ترتیب، «ذخیره‌ی نمودار» هم یک مورد کاربرد نیست.

کاربر می‌تواند یک نمودار جدید یا یک نمودار موجود را ویرایش کند. این دو مورد را می‌توان به‌عنوان دو مورد کاربرد

جداگانه تلقی نمود: ۱) ویرایش یک نمودار جدید ۲) ویرایش یک نمودار موجود. راه دیگر این است که، این دو مورد را، به‌عنوان انتخاب‌های مورد کاربرد ویرایش نمودار در نظر بگیریم. اصل رعایت سادگی و سرراستی پیشنهاد می‌دهد که این دو را دو مورد کاربرد جداگانه در نظر بگیریم.

برای ویرایش یک نمودار کلاس، کاربر می‌تواند کلاس‌ها و رابطه‌ها را اضافه، اصلاح، یا حذف نماید. این کارها را نباید به‌عنوان مورد کاربرد در نظر گرفت زیرا این‌ها فرایندهای کسب‌وکار نیستند؛ بلکه این‌ها اعمالی از مورد کاربرد ویرایش نمودار کلاس هستند.

مثال ۷-۲

با توجه به مثال ۷-۱، اعمالی که برای افزودن یک کلاس به یک نمودار کلاس باید انجام شوند کدامند؟

جواب: برای افزودن یک کلاس به یک نمودار کلاس، اعمال زیر انجام می‌شود:

۱. کاربر در یک فضای خالی از محدوده‌ی ترسیم که به آن صفحه رسم (Canvas) می‌گویند کلیک راست می‌کند.
۲. سیستم یک فهرست بازشونده (Pop-up Menu) نشان می‌دهد که افزودن کلاس (Add class) یکی از گزینه‌های آن است.
۳. کاربر گزینه‌ی افزودن کلاس را انتخاب می‌کند.
۴. سیستم یک جعبه‌ی گفت‌وگو به کاربر نشان می‌دهد تا وی اطلاعاتی در خصوص کلاسی که باید اضافه شود را در آن وارد نماید.
۵. کاربر اطلاعات را وارد و روی دکمه‌ی OK کلیک می‌کند.
۶. سیستم اطلاعات واردشده را وارسی نموده و کلاس را ترسیم می‌نماید (یا یک پیغام خطا نشان می‌دهد).

این مثال نشان می‌دهد که هر عمل (مثل افزودن یک کلاس) از تعدادی کنش تشکیل می‌شود. شکل ۷-۱ مثال‌هایی از مورد کاربردها، اعمال و کنش‌های مربوطه را نشان می‌دهد.

۷-۴ گام‌های استنتاج مورد کاربردها از نیازمندی‌ها

همان‌گونه که قبلاً بحث شد، مورد کاربردها نیازمندی‌ها را پالایش کرده و یک طراحی از رفتار سیستم را مشخص می‌کنند. بنابراین، مورد کاربردها باید از نیازمندی‌ها استنتاج‌شده و نیازمندی‌ها را ارضا نمایند. در این بخش، گام‌های لازم برای استنتاج مورد کاربردها و زیرسیستم‌ها از نیازمندی‌ها و تخصیص آن‌ها به تکرارها را ارائه می‌نماییم. همان‌گونه که در شکل ۷-۲ نشان داده شده و در زیر نیز تشریح شده‌اند، پنج گام وجود دارد.

گام ۱- شناسایی مورد کاربردها. در این گام عبارت‌های فعل-اسم که نشان‌دهنده‌ی فرایندهای کسب‌وکار هستند از روی نیازمندی‌ها شناسایی می‌شوند. همچنین، کنشگرهایی که از مورد کاربردها، استفاده می‌کنند و زیرسیستم‌هایی که دربرگیرنده‌ی مورد کاربردها هستند به دست می‌آیند.

گام ۲- تعیین قلمرو مورد کاربردها. در این گام، مورد کاربردهای سطح بالا مشخص می‌شوند. تیم‌ها و اعضای آن‌ها از روی مورد کاربردهای سطح بالا می‌فهمند که هر مورد کاربرد چه موقع و کجا شروع‌شده یا خاتمه می‌یابد؟

گام ۳- ترسیم زمینه‌ی مورد کاربردها. در این گام، مورد کاربردها، کنشگرها، زیرسیستم‌ها، و روابط آن‌ها با استفاده از نمودارهای مورد کاربرد UML ترسیم می‌شوند.

برنامه (کاربرد)	مورد کاربرد	اعمال	کنش‌ها
ویرایشگر متن (Text Editor)	ویرایش گزارش (Edit Report)	بازکردن گزارش	روی File کلیک کن، Open را انتخاب کن، به پوشه‌ی موردنظر برو، فایل را انتخاب کن، روی دکمه‌ی OK کلیک کن
ویرایشگر نمودار کلاس (Class Diagram Editor)	ویرایش نمودار (Edit Diagram)	بازکردن نمودار	روی File کلیک کن، Open را انتخاب کن، به پوشه‌ی موردنظر برو، فایل را انتخاب کن، روی دکمه‌ی OK کلیک کن
		افزودن کلاس	روی محدوده‌ی ویرایش کلیک راست کن، Add Class را انتخاب کن، اطلاعات کلاس را تکمیل کن، روی دکمه‌ی OK کلیک کن
		ذخیره نمودار	File را انتخاب کن، Save را انتخاب کن
خودپرداز (ATM)	واریزوجه/برداشت‌وجه (Deposit Money/ Withdraw Money)	شروع	کارت بانکی را وارد کن
		احراز هویت	رمز را وارد کن، کلید Enter را فشار بده
		انجام تراکنش	نوع تراکنش را انتخاب کن ، مقدار واریز/برداشت را وارد کن، وجه را واریز کن/وجه را برداشت کن، رسید واریز/برداشت را بردار
		خاتمه	کلید Exit را فشار بده ، کارت خارج شده را بردار

شکل ۷-۱: مثالی از مورد کاربردها، اعمال و کنش‌ها

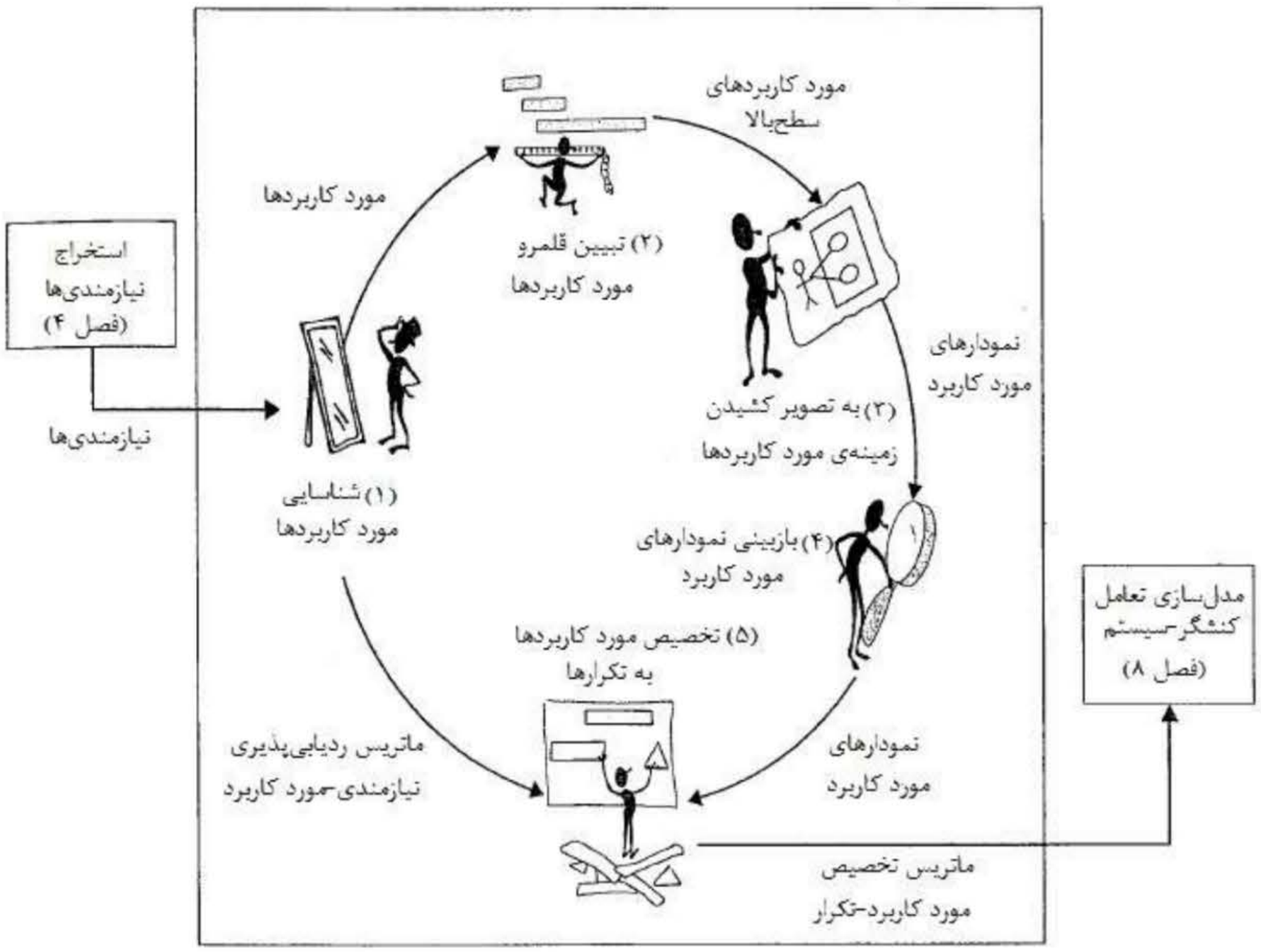
گام ۴- بازبینی مورد کاربردها و نمودارها. در این گام، تبیین مورد کاربرد و نمودار مورد کاربرد بازبینی می‌شوند.
گام ۵-تخصیص مورد کاربردها به تکرارها. در این گام، طول مدت هر تکرار تعریف‌شده و مورد کاربردها به تکرارها تخصیص داده می‌شوند.

۷-۴-۱- شناسایی مورد کاربردها

شناسایی مورد کاربردها شامل چهار فعالیت می‌شود. نخست، در نیازمندی‌ها به دنبال عبارت‌های فعلی-اسمی که مختص دامنه هستند بگردید. هر عبارت فعلی-اسمی مختص دامنه در صورتی که تعریف ۷-۲ را بر آورده نماید، یک مورد کاربرد تلقی خواهد شد. دوم، به دنبال عبارت‌های اسمی بگردید که کنشگر هستند، و نیز عبارت‌های اسمی که بیانگر زیرسیستم‌های دربرگیرنده‌ی مورد کاربردها هستند. سوم، مورد کاربردها را در بین زیرسیستم‌ها بازچینی کنید تا چسبندگی زیرسیستم بهتر شود. دست آخر، یک جدول ردیابی‌پذیری نیازمندی-مورد کاربرد تولید کنید که نشان دهد هر مورد کاربرد کدام یک از نیازمندی‌ها را محقق می‌کند. این فعالیت‌ها در بخش‌های زیر تشریح شده‌اند.

استنتاج مورد کاربردها، کنشگرها، و زیرسیستم‌ها

برای شناسایی مورد کاربردها، کنشگرها و زیرسیستم‌ها، اعضای تیم با همکاری، نیازمندی‌ها را یکی‌یکی خوانده و سعی می‌کنند عبارت‌های فعلی-اسمی را پیداکرده یا از متن برداشت نمایند. این عبارت‌ها باید بیان‌گر انجام دادن کاری، یا انجام شدن کاری،



شکل ۷-۲: گام‌های مدل‌سازی مورد کاربردها

یا اجرای یک کار مشخص در دامنه‌ی کاربرد یا برای بعضی از کنشگران کاربرد باشد. هر عبارت فعلی-اسمی که بدین شکل استنتاج شده باشد، به‌عنوان یک مورد کاربرد تلقی خواهد شد اگر پاسخ به تمام پرسش‌های زیر مثبت باشد:

۱. آیا این یک فرایند کسب‌وکاری (کامل) است؟ اگر فقط یک عمل یا کنش است، پس پاسخ منفی است.
۲. آیا با یک کنشگر آغاز می‌شود؟
۳. آیا با یک کنشگر تمام می‌شود؟
۴. آیا یک وظیفه‌ی کسب‌وکاری را برای یک کنشگر انجام می‌دهد؟

اعضای تیم در حین شناسایی مورد کاربردها، کنشگرها و سیستم‌ها/زیرسیستم‌ها را نیز شناسایی می‌کنند. برای شناسایی کنشگرها، به دنبال یافتن یا استنتاج اسم‌ها و عبارت‌های اسمی باشید. این‌ها در واقع نقش‌هایی در کسب‌و کار هستند که توسط موجودیت‌های خارجی بازی می‌شوند. این موجودیت‌های خارجی جرقه‌ی شروع مورد کاربرد را می‌زنند، یا این‌که وظیفه کسب‌وکاری برای آن‌ها انجام می‌شود. برای شناسایی سیستم‌ها یا زیرسیستم‌هایی که دربرگیرنده‌ی مورد کاربردها هستند، به دنبال یافتن یا استنتاج اسم‌ها و عبارت‌های اسمی باشید که بیان‌گر سیستم‌ها، زیرسیستم‌ها، یا جنبه‌هایی از کسب‌وکار که مورد کاربرد به آن تعلق دارند، هستند. توجه کنید که نیازمندی‌ها ممکن است همیشه به وضوح، به دقت، یا رسماً یک عبارت فعلی-اسمی را نشان ندهند. برای مثال، «راه‌اندازی سیستم» یا «خاموش‌کردن سیستم» را اغلب در مستندات تبیین نیازمندی‌ها به صورت «راه‌اندازی» یا «خاموش» خلاصه‌نویسی می‌کنند. گاهی اوقات ممکن است نیاز باشد که فرایندهای کسب‌وکاری را از

روی نیازمندی‌ها استنتاج نموده و سپس مورد کاربردها را استنتاج نمود. خوشبختانه، اگر تبیین نیازمندی‌ها به درستی نوشته شده باشد، در این صورت استنتاج مورد کاربردها دشوار نخواهد بود.

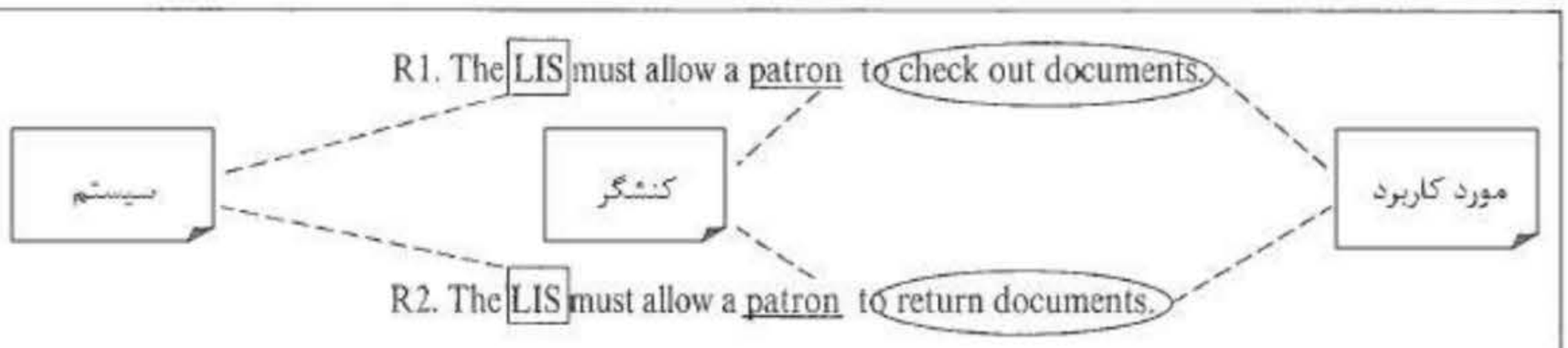
مثال ۷-۳

از روی نیازمندی‌های زیر که مربوط به یک سیستم اطلاعات کتابخانه (LIS) است مورد کاربردها را استنتاج نمایید.

R1) LIS باید به عضو امکان دریافت مستندات را بدهد.
R2) LIS باید به عضو امکان برگشت مستندات را بدهد.

جواب: شکل ۷-۳ چگونگی شناسایی مورد کاربردها، کنشگران، و زیرسیستم‌ها را نشان می‌دهد. جعبه‌ی گوش‌سگی (Dog-eared) نمادی است که UML برای توضیحات به کار می‌برد. جدول تصمیم در شکل ۷-۳ چگونگی تعیین این‌که هر عبارت فعلی-اسمی یک مورد کاربرد هست یا خیر را نشان می‌دهد. این به اعضای تیم کمک می‌کند تعیین کنند یک عبارت فعلی-اسمی یک مورد کاربرد هست یا خیر. برای مثال، این جدول نشان می‌دهد که «دریافت سند» و «برگشت سند» هر دو مورد کاربرد هستند چون پاسخ به هر چهار سؤال فوق برای آن‌ها مثبت خواهد بود. ولی «اجازه دادن به عضو» مورد کاربرد نیست چون تمامی جواب‌ها منفی است؛ و یک جواب منفی کافی است تا نتیجه بگیریم که این مورد کاربرد نیست. بنابراین، دو نیازمندی فوق، مورد کاربردها، کنشگرها، و زیرسیستم‌های زیر را تولید می‌کنند.

UC1) دریافت سند (کنشگر: عضو، سیستم: LIS)
UC2) برگشت سند (کنشگر: عضو، سیستم: LIS)



عبارت فعلی-اسمی	آیا یک فرایند کسب‌وکاری است؟	آیا با یک کنشگر آغاز می‌شود؟	آیا با یک کنشگر پایان می‌یابد؟	آیا برای کنشگر یک وظیفه‌ی کسب‌وکاری را انجام می‌دهد؟	آیا یک موردکاربرد هست؟	کنشگر	زیرسیستم
Checkout Documents	Y	Y	Y	Y	Y	Patron	**LIS**
Return Documents	Y	Y	Y	Y	Y	Patron	**LIS**
Allow a Patron	N	N	N	N	N	NA	NA

شکل ۷-۳: شناسایی مورد کاربردها برای LIS

مثال ۷-۴

اداره آموزش بین‌المللی (OIE) در یک دانشگاه می‌خواهد یک سیستم مدیریت آموزش برون‌مرزی (SAMS) مبتنی بر وب توسعه دهد. در ادامه سه نیازمندی عملکردی این سیستم نوشته شده است.

R1) SAMS باید بر اساس انواع معیارهای جست‌وجو، قابلیت جست‌وجو را برای برنامه‌های تبادل بین کشوری فراهم کند.

R2) SAMS باید نتایج جست‌وجو در خصوص یک برنامه‌ی تبادل برون‌مرزی را به صورت سلسله مراتبی نمایش دهد تا امکان مرور نتایج، از یک خلاصه سطح بالا گرفته تا جزییات، وجود داشته باشد.

R3) SAMS باید به دانشجویان امکان ارسال تقاضای برخط در خصوص برنامه‌های تبادل برون‌مرزی را بدهد.

از روی این نیازمندی‌های کارکردی، مورد کاربردها، کنشگرها، و زیرسیستم‌ها را شناسایی نمایید.

جواب: شکل ۷-۴ چگونگی شناسایی مورد کاربردها، کنشگرها، و زیرسیستم‌ها را نشان می‌دهد. از روی R1 و R2 نمی‌توان به وضوح کنشگرها را شناسایی نمود. در این حالت، نیازمندی‌ها به دقت بررسی و کنشگرها استنتاج می‌شوند. به نظر می‌رسد که هر کاربر وب می‌تواند جست‌وجو را به کار برده و وظایف ریز هر برنامه را مشاهده نماید؛ لذا، کنشگران این دو مورد کاربرد عبارتند از کاربران وب، که شامل دانشجویان می‌شود و نیز کارکنان OLE. از R3 می‌توان دانشجو (Student) را به‌عنوان کنشگری برای مورد کاربرد ارسال تقاضای برخط (Submit Online Application) شناسایی نمود. به‌طور خلاصه، از نیازمندی‌های فوق، مورد کاربردهای زیر استنتاج می‌گردند:

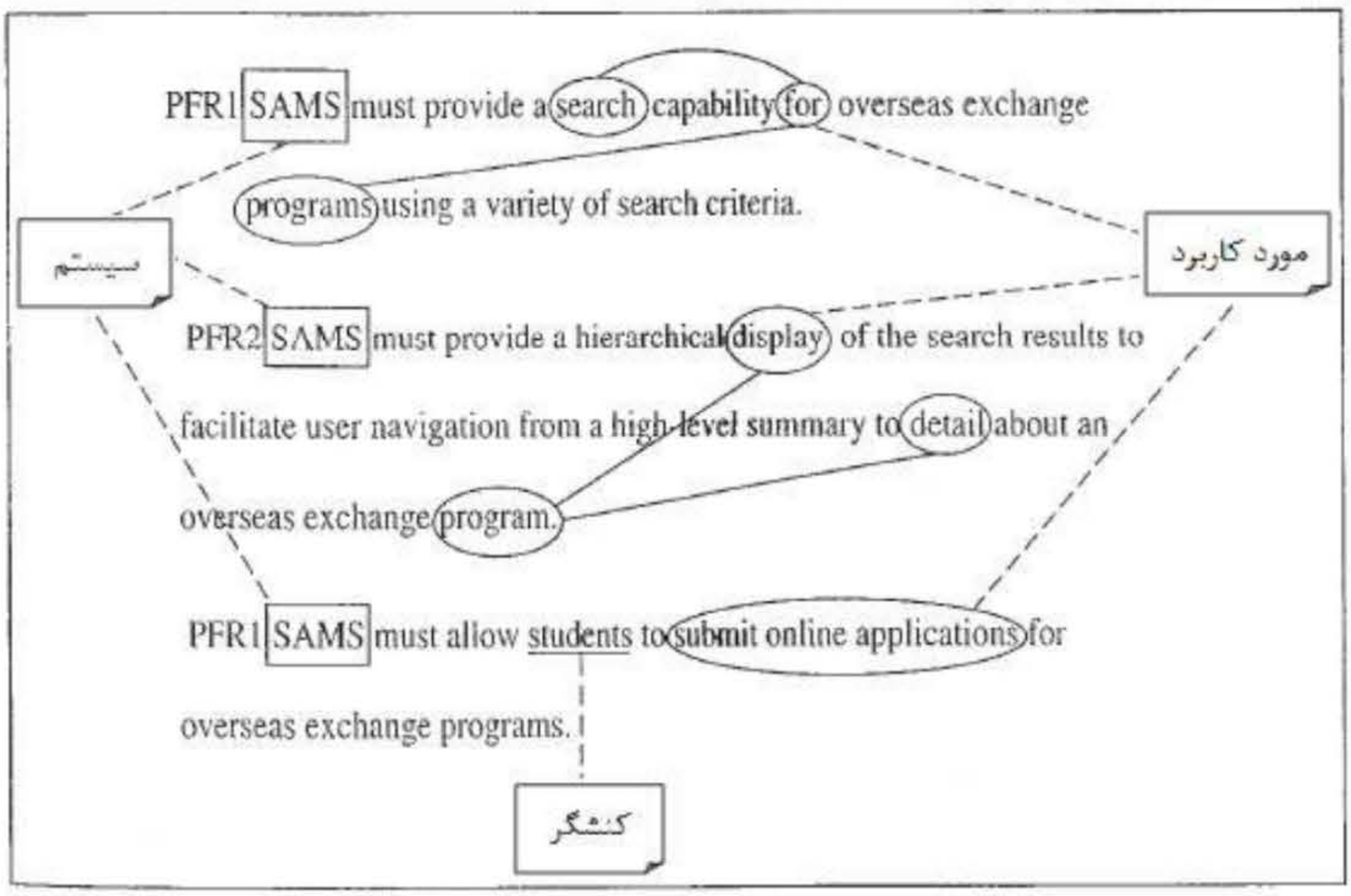
UC1) جست‌وجوی برنامه‌ها (کنشگر: کاربر وب، سیستم: SAMS)
UC2) نمایش جزییات برنامه‌ها (کنشگر: کاربر وب، سیستم: SAMS)
UC3) ارسال تقاضای بر خط (کنشگر: دانشجو، سیستم: SAMS)

این مثال می‌گوید این‌گونه نیست که هر نیازمندی به وضوح رسماً یک عبارت فعلی-اسمی را نشان دهد که مورد کاربرد، کنشگر، یا سیستم را مشخص کند؛ یعنی، برای استخراج این موارد از زمینه، نیاز به استنتاج است. توجه کنید که «ارائه‌ی امکان جست‌وجو» و «تسهیل کار کاربر» مورد کاربرد نیستند، زیرا پاسخ به سؤال‌های مذکور مثبت نیست.

مثال ۷-۵

کدام یك از موارد زیر مورد کاربرد نیستند و چرا؟

۱. فشردن دکمه‌ی ارسال (Press the Submit Button)
۲. جست‌وجوی یك فهرست متصل (Search a Linked List)
۳. پردازش داده‌ها (Process Data)
۴. پرس‌وجو از پایگاه‌داده (Query Database)
۵. راه‌اندازی سیستم (Startup System)



شکل ۷-۴: شناسایی مورد کاربرد برای SAMS

جواب: این عبارت‌ها به صورت زیر تحلیل شده‌اند:

۱. «فشردن دکمه‌ی ارسال» یك مورد کاربرد نیست زیرا یك فرایند کسب‌وکاری نیست، بلکه یك کنش از یك مورد کاربرد است.
۲. «جست‌وجوی یك فهرست متصل» یك مورد کاربرد نیست. این یک فرایند در حوزه‌ی علم کامپیوتر است.
۳. «پردازش داده‌ها» یك مورد کاربرد نیست. این خیلی کلی است؛ لذا نمی‌توان آن را به وضوح با یك فرایند کسب‌وکار مرتبط کرد. کنشگر کیست؟ وظیفه‌ی کسب‌وکاری چیست؟ نمی‌توان به این سؤال‌ها پاسخ داد.
۴. «پرس‌وجو از پایگاه‌داده» یك مورد کاربرد نیست. این یك عمل پایگاه‌داده‌ای است، نه یك فرایند کسب‌وکاری.
۵. «راه‌اندازی سیستم» یك مورد کاربرد هست. در نگاه اول، ممکن است فکر کنیم که این خیلی کلی است، ولی در واقع این‌طور نیست. سیستم خاصی که باید راه‌اندازی شود به‌طور ضمنی معلوم شده است. برای مثال، این‌کار ممکن است به‌صورت راه‌اندازی سیستم LIS، راه‌اندازی SAMS، ویا موارد مشابه باشد. به‌عنوان یك اصل کلی، هر سیستمی نیاز به یك مورد کاربرد راه‌اندازی و یك مورد کاربرد خاموش‌کردن (Shutdown) دارد.

بازچینی مورد کاربردها میان زیرسیستم‌ها

وقتی سیستمی از صفر ساخته می‌شود، معمولاً نیازمندی‌ها به سیستم به شکل کلان رجوع می‌کنند و کاری به زیرسیستم‌های دیگر ندارند. در این حالت، تمام مورد کاربردهایی که در بخش قبلی شناسایی شدند به سیستم واگذار می‌شوند. بنابراین، لازم است مورد کاربردها را بر حسب وظایف‌شان افراز نموده و براساس این افرازها، زیرسیستم‌ها را تشکیل داد. هدف این است که زیرسیستم‌هایی تشکیل گردند که چسبندگی عملیاتی زیاد از خود نشان دهند، یعنی مورد کاربردهای هر زیرسیستم باید یك وظیفه‌ی هسته‌ای داشته باشند. هیچ‌کدام از زیرسیستم‌ها نباید شامل تعداد زیادی مورد کاربرد باشد به‌گونه‌ای که طراحی،

پیاده‌سازی، آزمون، و نگهداری هر زیرسیستم آسان باشد. برای رسیدن به این اهداف، مورد کاربردها بین زیرسیستم‌ها بازچینی می‌شوند، برای کاهش پیچیدگی، یك زیرسیستم تجزیه می‌شود، و در صورت لزوم چند زیرسیستم با یك‌دیگر ادغام می‌شوند.

برحسب مشاهدات زیر، مورد کاربردها گروه‌بندی می‌شوند:

افراز مبتنی بر نقش. مورد کاربردهایی که مربوط به یك کنشگر می‌شوند معمولاً کارکردهای خاص آن نقش را از خود نشان می‌دهند. به این، افراز مبتنی برنقش (مورد کاربردها) گفته می‌شود. مثال‌ها عبارتند از: واریز وجه، برداشت وجه، انتقال وجه، و بررسی موجودی. مورد کاربردهای مربوط به مدیر سیستم معمولاً کارهای مدیریت سیستم از قبیل راه‌اندازی (سیستم)، خاموش‌کردن (سیستم)، ویرایش پیکربندی، و کارهای مشابه را انجام می‌دهد.

توجه کنید که افرازمبتنی بر نقش ممکن است زیرسیستم‌هایی ایجاد کند که شامل تعداد زیادی مورد کاربرد بشوند. این وقتی اتفاق می‌افتد که کنشگر بتواند مورد کاربردهای زیادی را اجرا کند. اغلب، مورد کاربردهای یك افراز بزرگ‌تر می‌تواند با استفاده از قوانین زیر یا بر اساس وظایف، تقسیم شوند.

افراز ارتباطی. مورد کاربردهایی که یك شئ مشترك را پردازش می‌کنند تمایل دارند وظایف خاص شئ را انجام دهند. به این افراز ارتباطی گفته می‌شود، زیرا مورد کاربردها از طریق شئ مشترکی که پردازش می‌کنند با یکدیگر ارتباط دارند. برای مثال، یك ویرایشگر گرافیکی برای نمودارهای UML را در نظر بگیرید. مورد کاربردهای این ویرایشگر عبارتند از پروژه‌ی جدید، ویرایش ویژگی‌های پروژه، بازکردن پروژه، بستن پروژه، حذف پروژه، ویرایش نمودار جدید، و ویرایش نمودار موجود. این مورد کاربردها را می‌توان به دو مجموعه مورد کاربرد افراز نمود، یکی با پروژه‌ها سروکار دارد و دیگری با نمودارها.

افراز مبتنی برگونه. برخی مواقع، می‌توان شیئی که نام مورد کاربرد بر اساس آن انتخاب شده است را ملاك افراز قرار داد. برای مثال، یك ویرایشگر نمودارهای UML را که شامل مورد کاربردهایی برای ویرایش گونه‌های مختلفی از نمودارهای UML مانند نمودارهای مورد کاربرد، نمودارهای کلاس، و نمودارهای توالی می‌شود را در نظر بگیرید. این مورد کاربردها را می‌توان برحسب گونه‌ی نمودار افراز نمود، که منجر به زیرسیستم‌هایی می‌شود که به ترتیب نمودارهای مورد کاربرد، نمودارهای کلاس، و نمودارهای توالی را تولید می‌کنند. به این‌کار افراز مبتنی بر گونه گفته می‌شود.

علاوه بر موارد بالا، می‌توان از رابطه‌ی ارث‌بری بین کنشگرها نیز برای بازچینی مورد کاربردها میان زیرسیستم‌ها بهره جست:

۱. برای کاهش تعداد مورد کاربردهای یك زیرسیستم، مورد کاربردهای مرتبط با یك کنشگر کلاس پایین‌تر را می‌توان از مورد کاربردهای کنشگر کلاس بالاتر آن جدا نمود تا زیرسیستمی تشکیل شود که مربوط به همان زیرطبقه باشد. این در واقع همان افراز مبتنی بر نقش است که در مورد ارث‌بری اعمال شده است.
۲. مورد کاربردهای یك کنشگر کلاس پایین را می‌توان با مورد کاربردهای یك کنشگر کلاس بالاتر ادغام نمود تا درصورت لزوم تعداد زیرسیستم‌ها کاهش یابند.

مثال ۷-۶

مورد کاربردهای زیر که در مورد سیستم مدیریت مطالعات برون‌مرزی (SAMS) می‌باشند را به‌گونه‌ای افراز کنید که زیرسیستم‌های مناسب تشکیل شوند:

UC01- جست‌وجو به دنبال برنامه (کنشگر: کاربروب، سیستم: SAMS)
UC02- نمایش جزییات برنامه (کنشگر: کاربروب، سیستم: SAMS)
UC03- ارسال برخط تقاضا (کنشگر: دانشجو، سیستم: SAMS)

-UC04 ورود به سیستم (کنشگر: دانشجو، سیستم: SAMS)
-UC05 خروج از سیستم (کنشگر: دانشجو، سیستم: SAMS)
-UC06 ویرایش تقاضای برخط (کنشگر: دانشجو، سیستم: SAMS)
-UC07 بررسی وضعیت تقاضا (کنشگر: دانشجو، سیستم: SAMS)
-UC08 ارسال توصیه‌نامه (کنشگر: هیئت‌علمی، سیستم: SAMS)
-UC09 تأیید فرم معادل‌سازی دروس (کنشگر: استادراهنما، سیستم: SAMS)
-UC10 فهرست‌گیری از تقاضاها (کنشگر: کارمند، سیستم: SAMS)
-UC11 ایجاد حساب کاربری (کنشگر: مدیر، سیستم: SAMS)
-UC12 حذف حساب کاربری (کنشگر: مدیر، سیستم: SAMS)
-UC13 ویرایش حساب کاربری (کنشگر: مدیر، سیستم: SAMS)
-UC14 ویرایش تنظیمات سیستم (کنشگر: مدیر، سیستم: SAMS)
-UC15 ورود کارمند به سیستم (کنشگر: کارمند، سیستم: SAMS)
-UC16 خروج کارمند از سیستم (کنشگر: کارمند، سیستم: SAMS)
-UC17 ویرایش ترجیحات فردی (کنشگر: کاربر، سیستم: SAMS)
-UC18 افزودن برنامه (کنشگر: کارمند، سیستم: SAMS)
-UC19 حذف برنامه (کنشگر: کارمند، سیستم: SAMS)
-UC20 ویرایش برنامه (کنشگر: کارمند، سیستم: SAMS)
-UC21 بارگذاری برنامه‌ها (کنشگر: کارمند، سیستم: SAMS)
-UC22 ارسال بازخورد (کنشگر: دانشجو، سیستم: SAMS)
-UC23 مشاهده بازخورد (کنشگر: کاربروب، سیستم: SAMS)
-UC24 راه‌اندازی سیستم (کنشگر: مدیر، سیستم: SAMS)
-UC25 خاموش‌کردن سیستم (کنشگر: مدیر، سیستم: SAMS)

جواب: همانگونه که مشاهده می‌شود، SAMS دارای ۲۵ مورد کاربرد می‌باشد. این مورد کاربردها خیلی به هم مرتبط نیستند. بنابراین، چسبندگی عملکردی سیستم پایین است. نمودار مورد کاربردی که تمام این مورد کاربردها را نشان دهد خیلی پیچیده شده و فهم آن سخت خواهد بود. برای افراز این مورد کاربردها، افراز مبتنی بر نقش را به کار می‌بریم. نتیجه در شکل ۷-۵ مشاهده می‌شود.

مثال ۷-۷

مورد کاربردهای زیر را افراز کنید:

-UC1 ایجاد پروژه جدید (کنشگر: کاربر، سیستم: GED)
-UC2 ویرایش پروژه (کنشگر: کاربر، سیستم: GED)
-UC3 بازکردن پروژه(کنشگر: کاربر، سیستم: GED)
-UC4 بستن پروژه (کنشگر: کاربر، سیستم: GED)
-UC5 حذف پروژه (کنشگر: کاربر، سیستم: GED)

کنشگر: مدیر ,SAMS/Admin	کنشگر: دانشجو ,SAMS/Student	کنشگر: کارمند ,SAMS/Staff
UC11-ایجاد حساب کاربری	UC03-ارسال برخط تقاضا	UC10- فهرست‌گیری از تقاضاها
UC12-حذف حساب کاربری	UC04- ورود به سیستم	UC15-ورود کارمند به سیستم
UC13-ویرایش حساب کاربری	UC05-خروج از سیستم	UC16-خروج کارمند از سیستم
UC14-ویرایش تنظیمات سیستم	UC06- ویرایش تقاضای برخط	UC18-افزودن برنامه
UC24-راه‌اندازی سیستم	UC07- بررسی وضعیت تقاضا	UC19-حذف برنامه
UC25-خاموش‌کردن سیستم	UC22-ارسال بازخورد	UC20-ویرایش برنامه
		UC21-بارگذاری برنامه‌ها

SAMS/Account User, کنشگر: کاربر حساب	SAMS/Reference, کنشگر: هیئت‌علمی، استادراهنما	SAMS/Web User, کنشگر: کاربر وب
UC17-ویرایش ترجیحات فردی	UC08-ارسال توصیه‌نامه	UC01-جستوجو بهدنبال برنامه
	UC09- تأیید فرم معادل‌سازی دروس	UC02-نمایش جزئیات برنامه
		UC23-مشاهده بازخورد

شکل ۷-۵: زیرسیستم‌هایی که توسط کنشگرها شناسایی شده‌اند.

-UC6 ویرایش نمودار جدید (کنشگر: کاربر، سیستم: GED)

-UC7 ویرایش نمودار موجود (کنشگر: کاربر، سیستم: GED)

جواب: دو جزء کنشگر و زیرسیستم در تمام این مورد کاربردها یکسان هستند. اعمال افراز ارتباطی منجر به تولید دو افراز می‌شود که در شکل ۷-۶ نشان داده شده است.

GED/Project Manager	GED/Diagram Manager
UC1- ایجاد پروژه جدید	UC6- ایجاد نمودار جدید
UC2- ویرایش پروژه	UC6- ویرایش نمودار جدید
UC3- بازکردن پروژه	
UC4- بستن پروژه	
UC5- حذف پروژه	

شکل ۷-۶: زیرسیستم‌های منتج از افراز ارتباطی

ساخت ماتریس ردیابی

این واقعیت که یك مورد کاربرد از یك نیازمندی داده‌شده بدست آمده است، درون یك ماتریس ردیابی نیازمندی مورد کاربرد[۱] وارد می‌شود. شکل ۷-۷ یك RUTM فرضی را نشان می‌دهد که در آن ردیف‌ها بیانگر نیازمندی‌ها و ستون‌ها نشان دهنده‌ی مورد کاربردها می‌باشند. دومین ستون (از چپ) اولویت نیازمندی‌ها را نشان می‌دهد که ۱ به معنی بالاترین و ۵ به معنی پایین‌ترین است. احتمال دارد اولویت‌ها در حین مرحله‌ی نیازمندها بدست آمده باشند. اگر یك مورد کاربرد از روی یك نیازمندی بدست آید، آن‌گاه خانه‌ی مرتبط با نیازمندی-مورد کاربرد علامت (ضربدر) می‌خورد. ردیف اولویت مورد کاربرد[۲] اولویت مورد کاربردها را نشان می‌دهد، که در واقع بالاترین اولویت نیازمندی مربوطه است. اولویت مورد کاربردها برای برنامه‌ریزی تکرارها مفید خواهد بود. بدین معنی که، مورد کاربردهای با اولویت بالاتر، اول از همه توسعه و تحویل خواهند شد.

توجه کنید که یك مورد کاربرد ممکن است بیش از یك نیازمندی را پشتیبانی نماید. برای مثال، مورد کاربرد جست‌وجوی برنامه‌ها از سیستم SAMS این نیازمندی را حمایت می‌کند: «SAMS (R1) باید با استفاده از معیارهای مختلف جست‌وجو، قابلیت جست‌وجو برای برنامه‌های تبادل بین کشوری را فراهم آورد.» این مورد کاربرد باید همان‌گونه که در R2 گفته شده، نتایج را به صورت سلسله‌مراتبی نمایش دهد. بنابراین، این مورد کاربرد R2 را نیز پشتیبانی خواهد کرد. فرض کنید که لازم باشد SAMS چندین سیستم مدیریت پایگاه‌داده را پشتیبانی نماید. در این صورت این حقیقت که جست‌وجوی برنامه‌ها باید به یك پایگاه‌داده دسترسی داشته باشد به این نیازمندی ربط خواهد داشت. در این حالت، خانه‌ی متناظر در جدول RUTM علامت می‌خورد.

استفاده از RUTM چند حسن دارد. می‌توان دید کدام مورد کاربردها یك نیازمندی خاص را محقق می‌کنند. همچنین می‌توان دید کدام نیازمندی‌ها توسط یك مورد کاربری خاص محقق می‌شوند. این قابلیت ردیابی دو طرفه سه حسن دارد. نخست، اطمینان می‌دهد که هر نیازمندی به هر حال توسط برخی از مورد کاربردها تحویل خواهد شد (یعنی، هیچ ردیفی از جدول خالی نیست). به سادگی می‌توان وارسی کرد که آیا مورد کاربردها، نیازمندی‌ها را برآورده می‌سازند، یعنی، آیا برخی مورد کاربردها از قلم افتاده‌اند. دوم، تضمین می‌کند که تمام مورد کاربردها لازم‌اند (یعنی، هیچ ستونی از جدول خالی نیست). آخر این‌که، اطمینان می‌دهد که مورد کاربردهای با اولویت بالاتر هر چه زودتر توسعه وتحویل خواهند شد.

ما برنامه‌سازان اغلب مایلیم خصیصه‌های زیبا یا مورد کاربردهای غیرضروری را پیاده‌سازی نماییم. تصور ما بر این است که مشتری از این که این امکانات را مجانی در اختیار خواهد داشت خوشحال می‌شود. ولی متأسفانه، اغلب مشتریان این‌طور فکر نمی‌کنند، زیرا مورد کاربردها و خصیصه‌های اضافی بار آموزشی بیشتری بر مشتری تحمیل خواهند کرد. مؤلف به مواردی برخورده است که مشتری صراحتاً درخواست کرده موارد اضافی حذف شوند، زیرا به آن‌ها احتیاجی نداشته و هرگز از آن‌ها استفاده نخواهد کرد. ماتریس ردیابی در شناسایی این قبیل مورد کاربردها به ما کمك خواهد نمود.

۷-۴-۲- تعیین قلمرو مورد کاربردها

توسعه‌ی سیستم در حالت کلی یك مسئله‌ی بدرفتار است (بخش ۴-۲ را ببینید). یکی از ویژگی‌های مسائل بدرفتار این است که «قانون توقفی وجود ندارد- همواره می‌توان کار را بهتر انجام داد». فکر کنید تا به حال چند بار تعطیلات خودتان را فدا کرده‌اید یا چقدر بی‌خوابی کشیده‌اید چون می‌خواسته‌اید یك طراحی یا پیاده‌سازی را حتی تا لحظات آخر بهبود بخشید. پس، تیم باید بداند چه موقع توقف کند. مورد کاربردهای سطح بالا یك راه‌حل برای این مشکل ارائه می‌دهند. مورد کاربرد سطح بالا، در واقع پالایش یا شرح مفصل‌تری از یك مورد کاربرد است. برای تشخیص این دو، مورد کاربردهایی که با عبارت‌های

---

[۱] Requirements Use Case Traceability Matrix (RUTM)

[۲] UC Priority

Requirement	Priority	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8
R01	1	*							
R02	4		*			*		*	
R03	3					*			
R04	5		*				*		
R05	5			*			*		*
R06	1			*					
R07	2				*				
R08	4	*	*		*				
R09	5	*						*	
R10	3								*
UC Priority		1	4	1	2	3	5	4	3

شکل ۷-۷: جدول نمونه‌ی ردیابی‌پذیری نیازمندی-مورد کاربرد

اسمی-فعلی شناخته می‌شوند را مورد کاربرد مجرد[۱] می‌نامیم. این بدین خاطر است که عبارت اسمی-فعلی، کارکرد یا فرایند کسب‌وکاری مورد کاربرد را به صورت مجرد مشخص می‌کند. برای مثال، «امانت‌گرفتن سند» نوعی بیان مجرد از یك فرایند کسب‌وکاری است. طبق تعریف ۷-۲، هر مورد کاربرد باید با یك کنشگر شروع و با همان کنشگر تمام شود. ولی مورد کاربرد کی و کجا شروع می‌شود؟ و کی تمام می‌شود؟ مورد کاربردی سطح بالا به این پرسش‌ها پاسخ می‌دهد:

تعریف ۷-۶

مورد کاربرد سطح بالا مشخص می‌کند یك مورد کاربرد کی و کجا شروع شده و کی تمام می‌شود. به عبارت دیگر، قلمرو مورد کاربرد را مشخص می‌کند.

تبیین قلمرو مورد کاربرد در واقع قانون توقف را تعریف می‌کند. برای هر مورد کاربرد، سه تصمیم گرفته می‌شود:

۱. مورد کاربرد کی شروع می‌شود؟ یعنی، کنش کنشگر[۲] که باعث می‌شود یك رویداد یا یك محرك تولید شده و به سیستم نرم‌افزاری در حال ساخت ارسال شود، چیست؟ برای مثال، کاربر ATM کارت بانکی خود را وارد می‌کند، کاربر SAMS روی پیوند[۳] «جست‌وجوی برنامه‌ها» کلیک می‌کند، یا تماس‌گیرنده گوشی را بر می‌دارد.
۲. کنش کنشگر کجا اتفاق می‌افتد؟ برای مثال، کاربر SAMS روی پیوند «جست‌وجوی برنامه‌ها» از صفحه‌ی خانگی SAMS کلیک می‌کند. اگر صفحه‌ی خانگی SAMS مشخص نشده باشد، در این‌صورت کاربر SAMS نمی‌داند پیوند «جست‌وجوی برنامه‌ها» را از کجا پیدا کند. برنامه‌ساز هم نمی‌داند پیوند جست‌وجو را کجا بگذارد.

---

[۱] Abstract Use Case
[۲] Actor Action
[۳] Link

۳. مورد کاربرد کی تمام می‌شود؟ باید به‌طور صریح یا ضمنی معلوم شود که کنش کنشگر چگونه از تکمیل مورد کاربرد آگاه می‌شود. برای مثال وقتی مورد کاربرد شروع تماس[۱] با موفقیت تمام شود، تماس گیرنده صدای بوق را می‌شنود. وقتی مورد کاربرد جست‌وجوی برنامه‌ها خاتمه می‌یابد، کاربر وب فهرستی از برنامه‌های تبادل برون‌مرزی را مشاهده می‌کند. به‌عنوان مثالی که نیاز به کنش صریح کنشگر دارد، سیستم ATM را در نظر بگیرید. ممکن است برای این‌که مشتری ATM پیغام «تراکنش با موفقیت انجام شد» را مشاهده نماید، لازم باشد که دکمه‌ی OK را فشار دهد.

قلمرو مورد کاربردها در قالب دو جمله‌ی اعلانی بیان می‌شود:

۱. اولین جمله به شکل «این مورد کاربرد شروع می‌شود با» است. (مترجم: این عبارت در انگلیسی به صورت «This Use Case Begins With» نوشته شده که سر کلمات آن می‌شود TUCBW و از این به بعد در متن از آن استفاده خواهد شد). پس از این عبارت می‌گوییم که کنشگر چه کاری (کنشی) انجام می‌دهد و کجا این کنش اتفاق می‌افتد. همواره از جملات سوم-شخص و زمان حال ساده استفاده کنید.
۲. دومین جمله به شکل «این مورد کاربرد خاتمه می‌یابد با» است. (مترجم: این عبارت در انگلیسی به صورت «This Use Case Ends With» نوشته شده که سر کلمات آن می‌شود TUCEW و از این به بعد در متن از آن استفاده خواهد شد). پس از این کنشگر به صراحت یا به‌طور ضمنی معلوم می‌کند که کار خواسته شده توسط مورد کاربرد انجام شده است. در این جا نیز از جملات سوم-شخص و زمان حال ساده استفاده کنید.

مثال ۷-۸

مورد کاربردهای سطح بالا را برای (۱) جست‌وجوی برنامه‌ها، (۲) نمایش جزییات برنامه از وب‌کاربرد SAMS مشخص کنید.

جواب: مورد کاربردهای سطح بالا عبارتند از:

UC1) جست‌وجوی برنامه‌ها

TUCBW کاربر وب روی پیوند «جست‌وجوی برنامه‌ها» از صفحه‌ی خانگی SAMS کلیک می‌کند.

TUCEW کاربر وب فهرست جدول‌گونه‌ای از برنامه‌هایی که با شرایط و معیارهای جست‌وجوی وی مطابقت دارند را مشاهده می‌کند.

UC2) نمایش جزییات برنامه

TUCBW کاربر وب بر روی پیوند «مشاهده‌ی جزییات» یك برنامه در نمایش جدول گونه‌ی برنامه‌ها کلیک می‌کند.

TUCEW کاربر وب جزییات برنامه‌ی انتخاب شده را مشاهده می‌نماید.

---

[۱] Initiate Call

مثال ۷-۹

مورد کاربردهای سطح بالا را برای (۱) شروع تماس، و (۲) دریافت تماس در یك سیستم تلفنی مشخص نمایید.

جواب: مورد کاربردهای سطح بالا عبارتند از:

UC1) شروع تماس

TUCBW تماس‌گیرنده گوشی را از روی تلفن برمی‌دارد.

TUCEW تماس‌گیرنده صدای بوق را می‌شنود.

UC2) دریافت تماس

TUCBW مخاطب (صدای زنگ را شنیده) و گوشی را از روی تلفن برمی‌دارد.

TUCEW یکی از طرفین تماس (تماس‌گیرنده یا مخاطب) گوشی را روی تلفن می‌گذارد.

۷-۴-۳- مصورسازی زمینه‌ی مورد کاربرد

توصیف مورد کاربردها، کنشگرها، و زیرسیستم‌ها به صورت متنی فقط برای کاربردهای کوچکی که تنها شامل تعداد اندکی مورد کاربرد، کنشگر و زیرسیستم باشند جواب می‌دهد. بسیاری از کاربردهای دنیای واقعی شامل تعداد زیادی مورد کاربرد کنشگر، و زیرسیستم می‌شوند. توسعه‌ی نرم‌افزار برای سیستم‌های بزرگ پیچیده نیاز به روشی برای سازمان‌دهی و مصورسازی این اطلاعات دارد. نمودار مورد کاربرد UML این نیاز را برآورده می‌سازد.

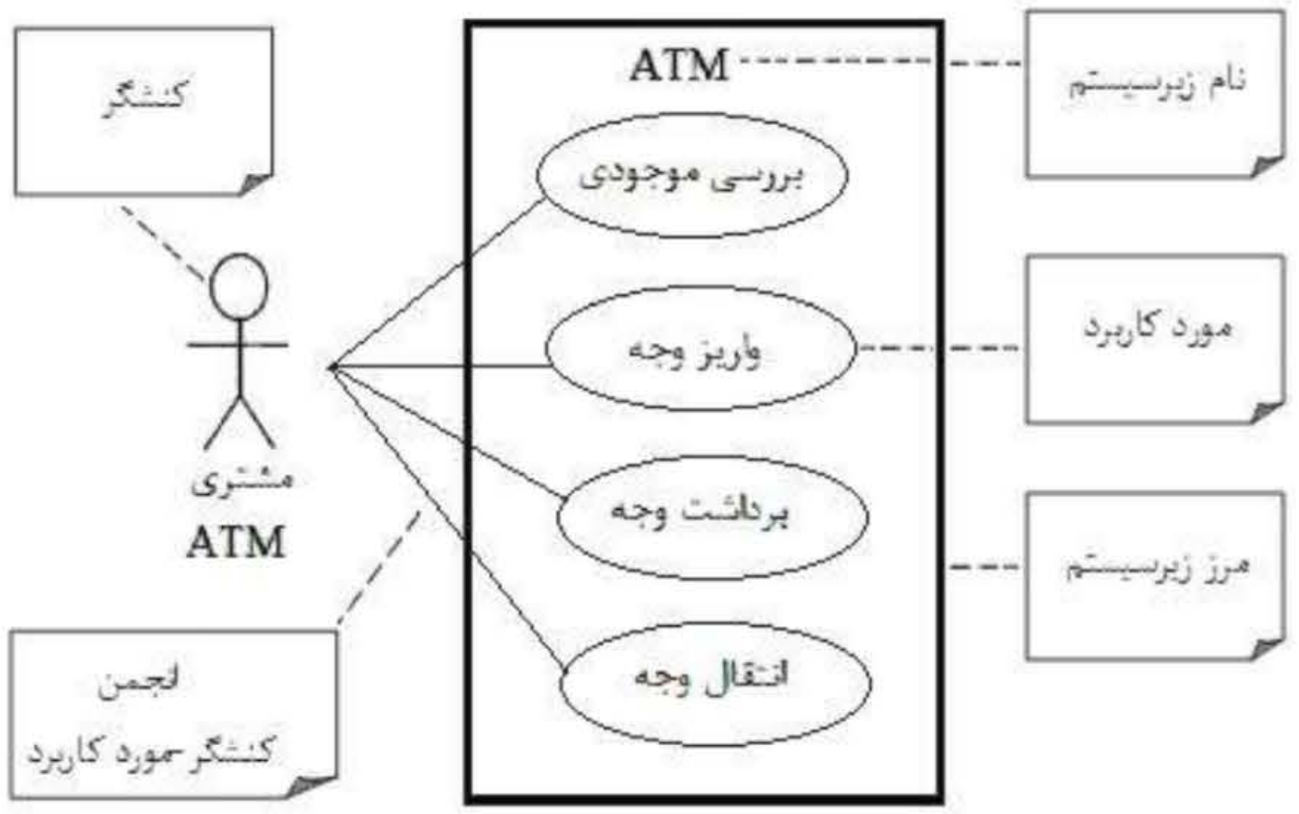
نمودار مورد کاربرد UML

تعریف ۷-۷

نمودار مورد کاربرد یك نمودار رفتاری UML است که موارد زیر را به تصویر می‌کشد:

- مورد کاربردهای سیستم یا زیرسیستم
- کنشگرهایی که از مورد کاربردها استفاده می‌کنند
- مرز سیستم یا زیرسیستم
- روابط ارث‌بری مابین کنشگرها
- روابط بین مورد کاربردها

شکل ۷-۸ یك نمودار مورد کاربرد را برای نرم‌افزار ATM نشان می‌دهد. این نمودار مورد کاربرد می‌گوید که ATM چهار مورد کاربرد دارد. کنشگر این مورد کاربردها، مشتری ATM است. مشتری ATM از این مورد کاربردها برای انجام اعمال زیر استفاده می‌کند: بررسی موجودی، واریز وجه، برداشت وجه، و انتقال وجه.



شکل ۷-۸: نمودار مورد کاربرد برای مثال ATM

انگاره‌ها و نمادگان در نمودارهای مورد کاربرد

انگاره و نمادگان دو مفهوم کاملاً مرتبط هستند. نمادگان در واقع نمادهایی هستند که در یك زبان برای نمایش مفاهیم مدل‌سازی (یا انگاره‌ها) پیشنهاد می‌شوند. انگاره‌ها مفاهیم زیربنایی یا معنی ساختارهای مدل‌سازی (یا نمادگان) هستند. متأسفانه بسیاری از نوآموزان UML تنها نمادگان را فرا می‌گیرند نه انگاره‌های زیربنایی را، و نتیجه‌ی کارشان نمودارهایی می‌شوند که به‌زیبایی رسم شده‌اند اما بی‌معنا هستند. به‌همین خاطر، در شکل ۷-۹ هم انگاره‌ها و هم نمادگان برای نمودارهای مورد کاربرد نشان داده شده‌اند. دانشجو باید گاه‌به‌گاه سری به این جدول بزند تا مطمئن شود که از نمادگان با معنی درست‌شان استفاده می‌کند.

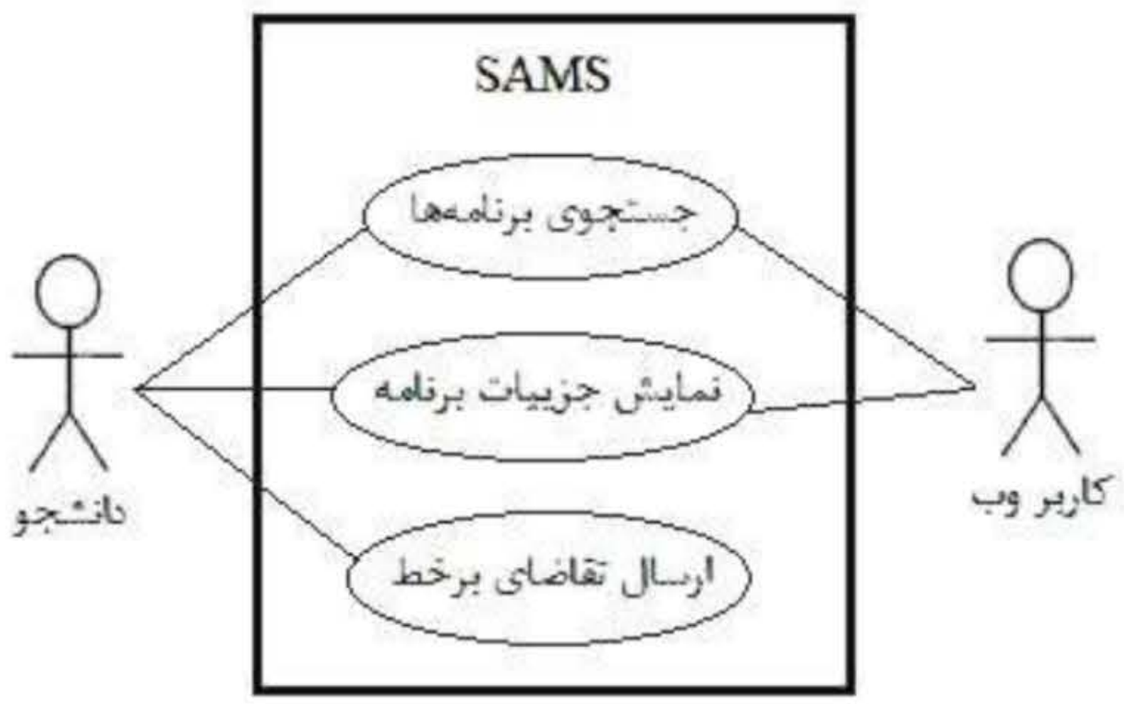
مثال ۷-۱۰

برای مورد کاربردهای SAMS مبتنی بر وب موضوع مثال ۷-۴، یك نمودار مورد کاربرد رسم نمایید.
جواب: مورد کاربردها، کنشگرها، و زیرسیستم‌ها قبلاً در مثال ۷-۴ شناسایی شدند. تبدیل آن‌ها به نمودار مورد کاربرد نسبتاً ساده بوده و در شکل ۷-۱۰ نشان داده شده است.

نمودار مورد کاربرد باید مرز زیرسیستم را نشان دهد، این‌کار چندین مزیت دارد. با بررسی این‌که مورد کاربردها چه کاری انجام می‌دهند، چسبندگی زیرسیستم ارزیابی می‌شود- یعنی، مورد کاربردها باید به‌جای این‌که چند وظیفه‌ی غیرمرتبط داشته باشند، یك وظیفه‌ی هسته‌ای را از خود به نمایش بگذارند. همچنین، می‌توان زیرسیستم‌های حساس و بحرانی را شناسایی نمود. این‌ها زیرسیستم‌هایی هستند که باید هر چه سریع‌تر توسعه و استقرار یابند به‌گونه‌ای که بتوان این زیرسیستم‌ها را به زودی و به دفعات آزمایش نمود. دیگر این‌که می‌توان تعیین نمود که آیا زیرسیستم را باید از صفر ساخت یا می‌توان مولفه‌های تجاری آماده فروش را بازاستفاده نمود.

انگاره	معنا	نمادگان
مورد کاربرد	مورد کاربرد عبارت است از یک فرایند کسب‌وکار که با یک کنشگر شروع شده و با آن کنشگر خاتمه‌یافته و یک وظیفه‌ی کسب‌وکار را برای کنشگر انجام می‌دهد.	نام مورد کاربرد
کنشگر	کنشگر عبارت است از نقشی که به‌وسیله‌ی یا به‌واسطه‌ی مجموعه‌ای از موجودیت‌ها یا ذی‌نفعان کسب‌وکار که خارج سیستم بوده و با سیستم در تعامل هستند، ایفا می‌شود.	نام نقش
مرز سیستم/ زیرسیستم	مرز سیستم/زیرسیستم مورد کاربردها را محصور کرده و قابلیت‌های سیستم/زیرسیستم را نشان می‌دهد.	نام سیستم
انجمن بین کنشگرها و مورد کاربردها	انجمن بین کنشگر و مورد کاربرد نشان می‌دهد که کنشگر مورد کاربرد را به‌کار می‌بندد.	———
ارث‌بری	یک رابطه‌ی دودویی میان دو مفهوم به‌گونه‌ای که یکی تعمیم (یا تخصیص) دیگری است.	از مفهوم خاص به مفهوم عام اشاره می‌کند.
رابطه‌ی توسعه بین مورد کاربردها	یک رابطه‌ی دودویی میان دو مورد کاربرد به‌گونه‌ای که یکی می‌تواند فرایند دیگری را ادامه دهد.	«extend» ----------→ از مورد کاربرد توسعه‌یابنده به‌سمت مورد کاربرد توسعه‌یافته
رابطه‌ی شمول بین مورد کاربردها	یک رابطه‌ی دودویی میان دو مورد کاربرد به‌گونه‌ای که یکی شامل دیگری است و قسمتی از فرایند کسب‌وکاری‌اش تلقی می‌شود.	«include» ----------→ از مورد کاربرد شامل به سمت مورد کاربرد مشمول

شکل ۷-۹: انگاره‌ها و نمادگان در نمودار مورد کاربرد



شکل ۷-۱۰: نمودار مورد کاربرد برای SAMS

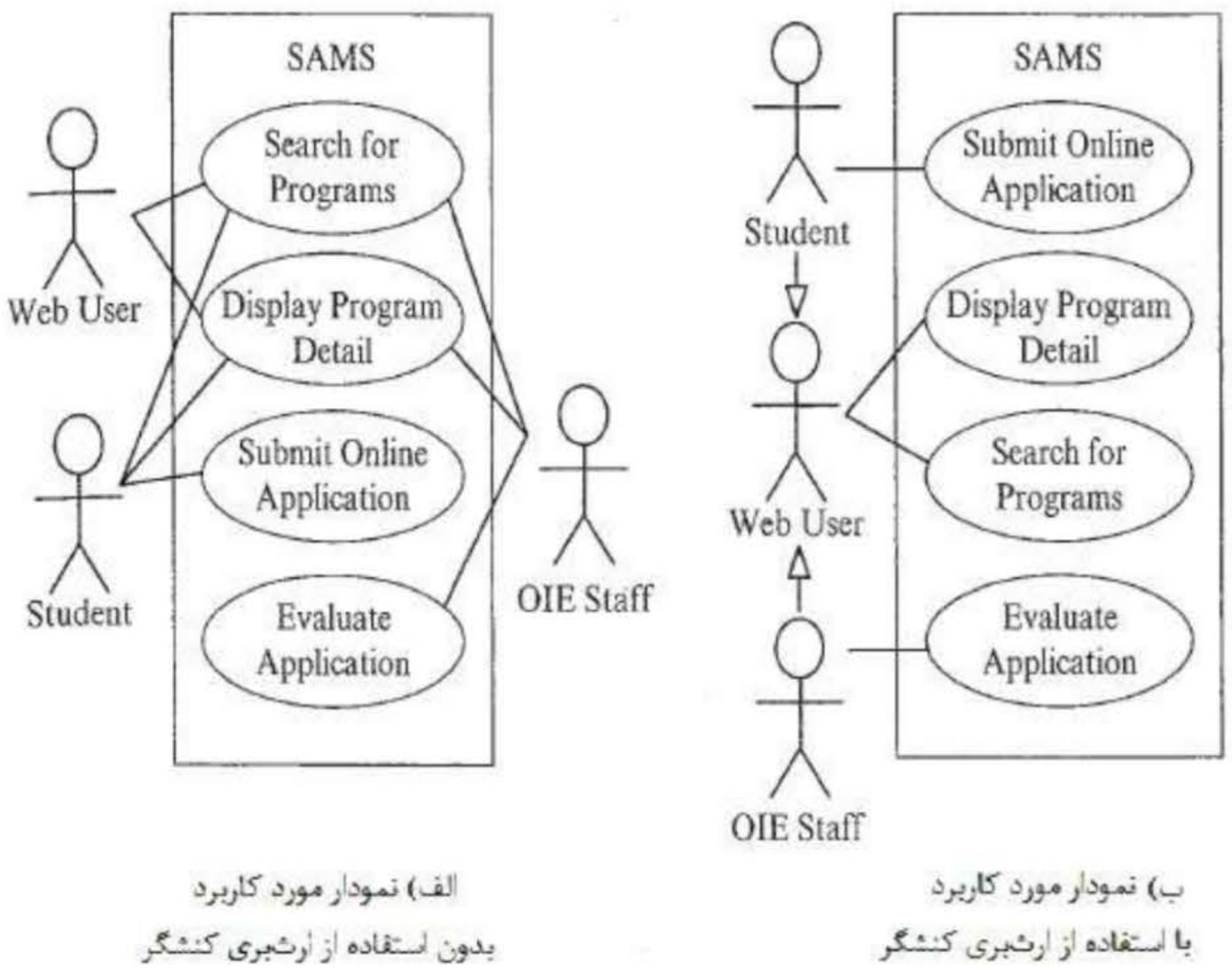
نمایش روابط ارث‌بری کنشگر

تعریف ۷-۸

ارث‌بری عبارت است از یك رابطه‌ی دودویی بین دو مفهوم به‌گونه‌ای که یك مفهوم تعمیم (یا تخصیص) مفهوم دیگر است.

رابطه‌ی ارث‌بری با عنوان رابطه‌ی «هست»[۱] نیز شناخته می‌شود؛ زیرا هر نمونه از مفهوم خاص در واقع نمونه‌ای از مفهوم عام خواهد بود. گام‌های مربوط به شناسایی روابط ارث‌بری در مبحث مدل‌سازی دامنه در فصل ۵ گفته شده است.

کشف رابطه‌ی ارث‌بری بین کنشگرها می‌تواند نمودار مورد کاربرد را ساده کند. شکل ۷-۱۱(الف) یك نمودار مورد کاربرد بدون استفاده از ارث‌بری کنشگرها و شکل ۷-۱۱(ب) با استفاده از ارث‌بری کنشگرها می‌باشد.



شکل ۷-۱۱: (الف) استفاده از ارث‌بری کنشگرها (ب) جهت ساده‌سازی نمودارهای مورد کاربرد

نمایش رابطه بین مورد کاربردها

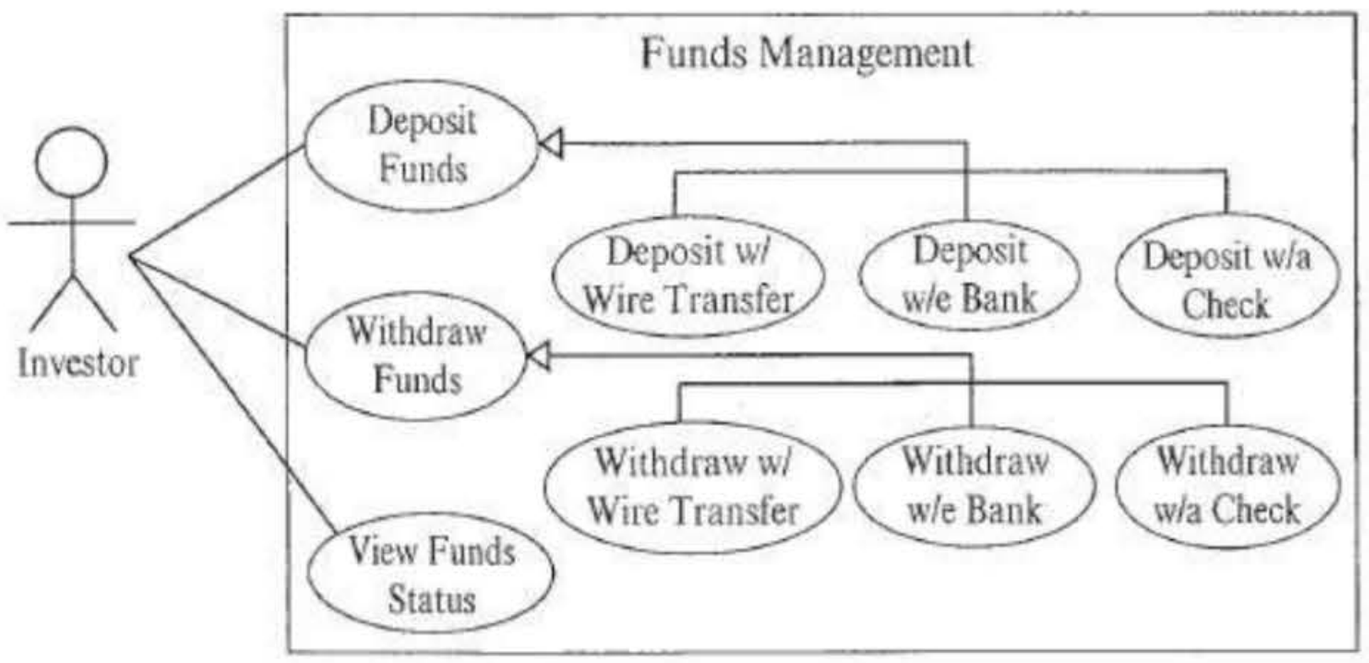
این بخش رابطه‌ی بین مورد کاربردها را معرفی می‌کند. افراد مبتدی باید از این بخش عبور کنند، زیرا نمایش رابطه بین مورد کاربردها همواره لازم نیست. مبتدیان اغلب روابط را اشتباه می‌فهمند و در مدل‌سازی نیز از این روابط استفاده‌ی اشتباه می‌کنند. این روابط در این بخش معرفی شده‌اند به این دلیل که در برخی موارد استفاده خواهند شد.

در برنامه‌سازی OO، رابطه‌ی ارث‌بری بین کلاس‌های اشیا وجود دارد. این رابطه بین مورد کاربردها نیز وجود دارد. از این

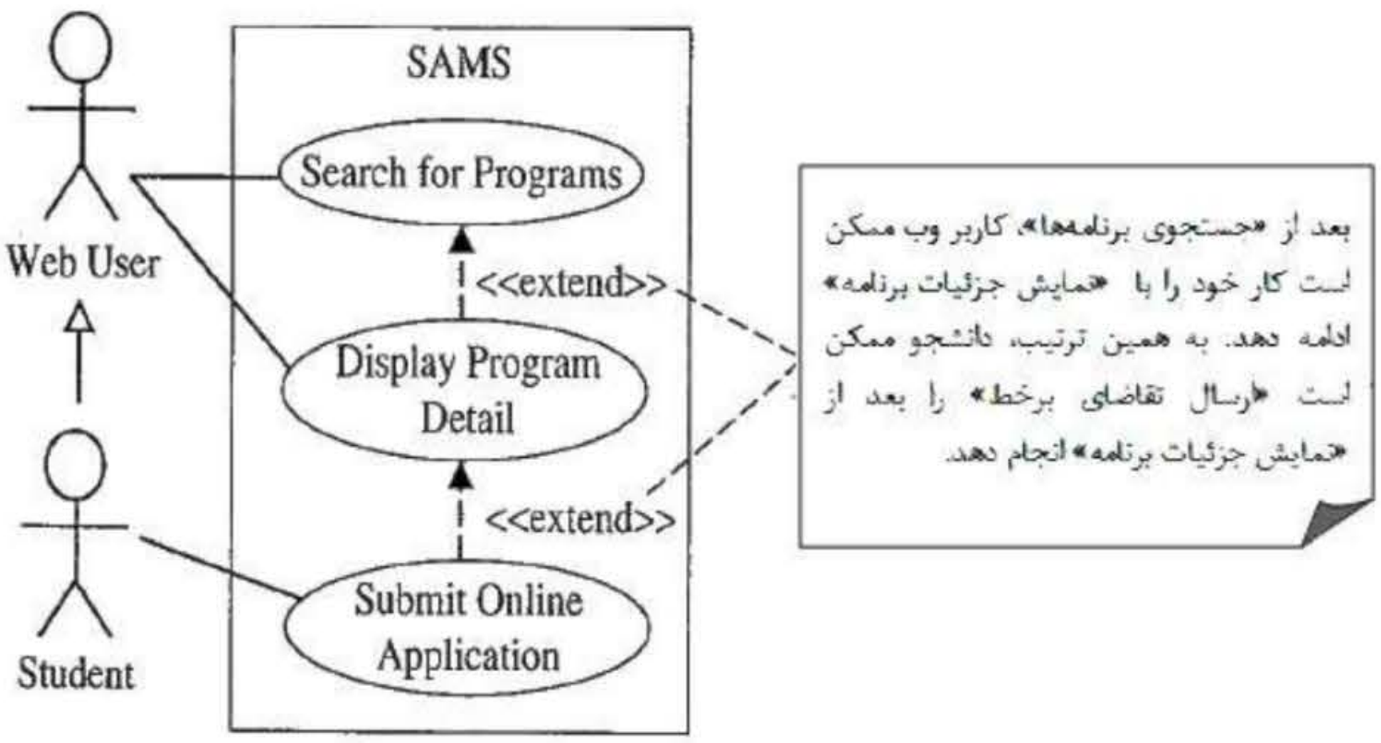
[۱] IS-A

رابطه می‌توان برای نشان دادن چند فرایند کسب‌وکار مختلف که روش‌های انجام یك کار یکسان هستند استفاده نمود. همان‌طور که در شکل ۷-۱۲ نشان داده شده است، مورد کاربردی واریز وجه در یك برنامه‌ی واسطه‌گری می‌تواند در قالب یکی از مورد کاربردهای زیر انجام شود: واریز از طریق انتقال سیمی، واریز از طریق چك، واریز از طریق بانك. به همین صورت، برداشت وجه نیز می‌تواند به سه شکل و در قالب سه مورد کاربرد انجام شود.

مورد کاربردها فرایندهای کسب‌وکار می‌باشند. گاهی اوقات اجرای یك فرایند کسب‌وکار می‌تواند در قالب یك فرایند کسب‌وکار دیگر ادامه یابد. در وب‌کاربرد SAMS، دانشجو ممکن است بخواهد مورد کاربرد نمایش جزییات برنامه را برای یکی از برنامه‌هایی که در اثر مورد کاربرد جست‌وجوی برنامه‌ها فهرست شده‌اند، اجرا کند. دانشجو ممکن است تصمیم بگیرد بعد از مرور جزییات یك برنامه‌ی تبادل برون‌مرزی ارسال تقاضای برخط را انجام دهد. این مثال‌ها رابطه‌ی گسترش بین مورد کاربردها را نشان می‌دهند. (شکل ۷-۱۳ را ببینید). یعنی، کنشگر ممکن است به دلخواه بخواهد فرایند کسب‌وکاری یك مورد کاربرد را در قالب یك فرایند کسب‌وکاری مورد کاربرد دیگر گسترش دهد.



شکل ۷-۱۲: نمایش فرایندهای جایگزین با استفاده از ارث‌بری مورد کاربردها

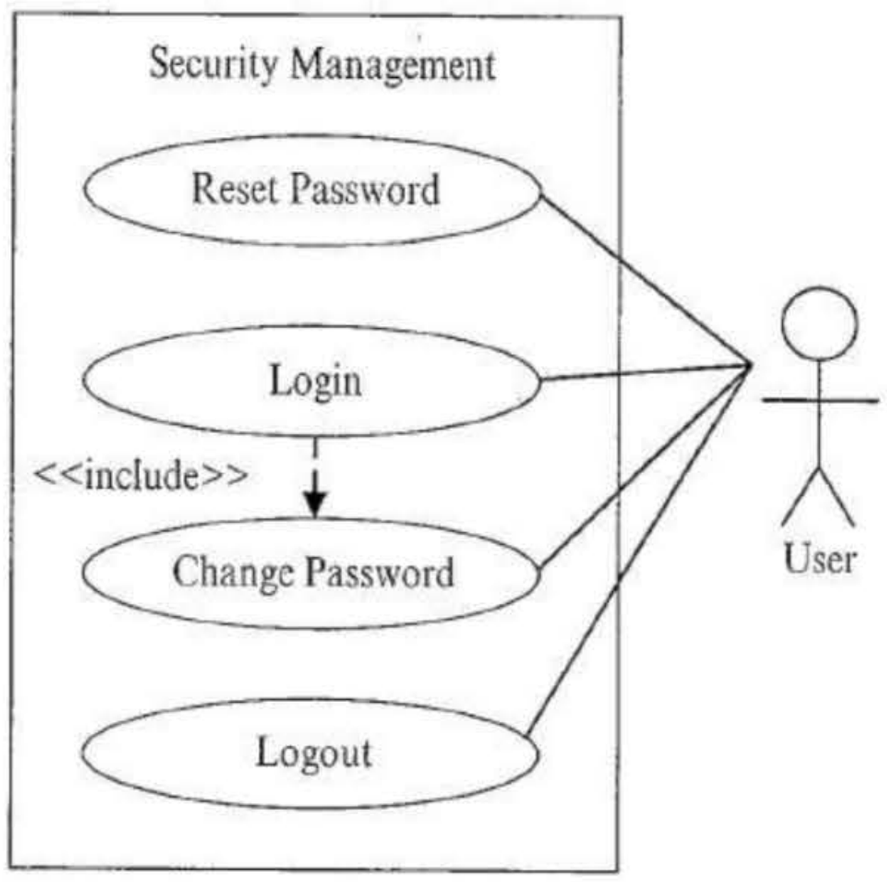


شکل ۷-۱۳: رابطه‌ی گسترش بین مورد کاربردها

بعد از جست‌وجوی برنامه‌ها، کاربر وب ممکن است با نمایش جزییات برنامه کار را ادامه دهد. به همین ترتیب، دانشجو ممکن است ارسال تقاضای برخط را پس از نمایش جزییات برنامه انتخاب کند.

توجه کنید که در این موارد، دانشجو می‌تواند است بعد از جست‌وجوی برنامه‌ها کار را با نمایش جزییات برنامه ادامه دهد، ولی مجبور نیست این کار را بکند. به‌همین ترتیب، دانشجو ممکن است نخواهد مورد کاربرد ارسال تقاضای برخط را بعد از مشاهده‌ی جزییات برنامه انجام دهد. بدین معنی که، رابطه‌ی گسترش نباید به‌معنی الزام در اجرای یك مورد کاربرد بعد از اجرای یك مورد کاربرد دیگر باشد.

یك فرایند کسب‌وکار ممکن است دربرگیرنده‌ی فرایند کسب‌وکار دیگری باشد، همین امر در خصوص مورد کاربردها نیز صادق است. به‌عنوان مثال، دو مورد کاربرد ورود به سیستم[۱] و تغییر رمز عبور[۲] را در نظر بگیرید. اگر کاربری با یك رمز عبور موقتی که از طریق رایانامه دریافت کرده است وارد شود، در این‌صورت لازم است رمز عبور موقتی خود را تغییر دهد. در این حالت، مورد کاربردی ورود به سیستم (به‌طور مشروط) شامل مورد کاربرد تغییر رمز عبور می‌شود. همان‌گونه که در شکل ۷-۱۴ نشان داده شده است.



شکل ۷-۱۴: نمایش رابطه‌ی شمول مورد کاربرد

۷-۴-۴- بازبینی مشخصات مورد کاربردها

در این گام، مورد کاربردهای مجرد، مورد کاربردهای سطح بالا، و نمودارهای مورد کاربرد طبق فهرست بررسی زیر مورد بازبینی قرار می‌گیرند:

مورد کاربردهای مجرد

۱. آیا نام هر کدام از مورد کاربردها از یك عبارت اسمی–فعلی تشکیل شده و بیان‌گر کاری که برای کنشگر انجام می‌شود هست؟
۲. آیا هر کدام از مورد کاربردها یك فرایند کسب‌وکار را نشان می‌دهد؟

---

[۱] Login
[۲] Change Password

۳. آیا می‌توان هر کدام از مورد کاربردها را به دو یا چند مورد کاربرد تجزیه کرد؟
۴. آیا می‌توان چند مورد کاربرد را با هم ادغام نمود؟

ماتریس ردیابی نیازمندی-مورد کاربرد

۱. آیا ماتریس ردیابی نیازمندی-مورد کاربرد ساخته شده است؟
۲. آیا هر کدام از نیازمندی‌ها در ماتریس ردیابی درج شده‌اند؟
۳. آیا در ماتریس ردیابی نیازمندی-مورد کاربرد ردیف یا ستون خالی وجود دارد؟
۴. آیا مورد کاربردهایی که برای هر کدام از نیازمندی‌ها فهرست شده‌اند، برای آن نیازمندی لازم هستند؟
۵. آیا مورد کاربردهایی که برای هر کدام از نیازمندی‌ها فهرست شده‌اند، برای آن نیازمندی کافی هستند؟

مورد کاربردهای سطح بالا

۱. آیا برای هر مورد کاربرد شناسایی شده، یك تبیین مورد کاربرد سطح بالا وجود دارد؟
۲. آیا عبارت TUCBW برای هر مورد کاربرد سطح بالا به وضوح می‌گوید که مورد کاربرد کی و کجا شروع می‌شود؟
۳. آیا عبارت TUCEW برای هر مورد کاربرد سطح بالا به وضوح می‌گوید که مورد کاربرد کی خاتمه می‌یابد و آیا با آن‌چه مد نظر کنشگر بوده است خاتمه می‌یابد؟
۴. آیا هر مورد کاربردی سطح بالا به درستی قلمرو فرایند کسب‌وکار را مشخص می‌کند؟

نمودار مورد کاربرد

۱. آیا نمودار مورد کاربرد مرز زیرسیستم را نشان می‌دهد؟
۲. آیا نام زیرسیستم به‌طور مناسب انتخاب شده است که بتواند عملکرد زیرسیستم را بیان کند؟
۳. آیا تناظر یك‌به‌یك بین مورد کاربردهای موجود در نمودار مورد کاربرد و ماتریس ردیابی نیازمندی-مورد کاربرد وجود دارد؟
۴. آیا هیچ نمودار مورد کاربردی هست که تعداد زیادی مورد کاربرد داشته باشد؟
۵. آیا هیچ نمودار مورد کاربردی هست که بدون هیچ دلیل موجهی، فقط شامل یك مورد کاربرد باشد؟
۶. آیا هیچ نمودار مورد کاربردی هست که شامل مورد کاربرد یا کنشگری شود که به هیچ کنشگر یا مورد کاربردی وصل نشده باشد؟
۷. آیا هیچ کنشگری هست که نام نقش نداشته باشد؟
۸. آیا همه‌ی مورد کاربردها به کنشگر مناسب وصل شده‌اند؟
۹. آیا هر کدام از مورد کاربردها به زیرسیستم متناسب اختصاص داده شده‌اند؟

۷-۴-۵- تخصیص مورد کاربردها به تکرارها

اکنون مورد کاربردها شناسایی و به‌تصویر کشیده شده‌اند و اولویت آن‌ها از روی اولویت نیازمندی‌ها به‌دست آمده است. گام بعدی این است که یك زمان‌بندی برای توسعه و تحویل این مورد کاربردها تولید شود. به این کار برنامه‌ریزی تکرارها بر اساس مورد کاربردها می‌گوییم، که خود شامل چند گام می‌شود. ابتدا، تلاش لازم برای توسعه و تحویل هر مورد کاربرد تخمین زده می‌شود. این کار بر اساس روش‌های تخمین که در فصل ۲۳ توضیح داده شده‌اند انجام می‌شود. سپس، وابستگی بین مورد کاربردها شناسایی می‌شود به‌گونه‌ای که تحویل آن‌ها بر طبق وابستگی‌هایشان انجام شود. برای مثال «امانت مدرك» نباید دیرتر از «بازگشت مدرك» تحویل شود. سرانجام، با استفاده از تکنیک‌های زمان‌بندی که در فصل ۲۳ گفته شده است، یك زمان‌بندی برای تکرار تولید می‌شود. در تولید این زمان‌بندی چند عامل باید مد نظر قرار گیرد که در زیر به ترتیب اهمیت تشریح شده‌اند:

۱. اولویت مورد کاربردها : مورد کاربردهای با اولویت بالاتر باید هر چه زودتر توسعه یافته و تحویل شوند تا اولویت‌ها و نیازهای کسب‌وکار مشتری را برآورده نمایید.
۲. وابستگی بین مورد کاربردها : اگر مورد کاربرد B به مورد کاربرد A وابسته باشد، در این‌صورت B نباید قبل از A تحویل شود زیرا بدون A کاربران قادر نخواهند بود از B استفاده کنند.
۳. توانایی تیم در توسعه و تحویل مورد کاربردها : تلاش مورد نیاز برای مورد کاربردهایی که به یك تکرار تخصیص یافته‌اند نباید از توانایی تیم بیشتر باشد.

شکل ۷-۱۵ یك زمان‌بندی نمونه را نشان می‌دهد که به آن جدول (ماتریس) تخصیص مورد کاربرد به تکرار می‌گویند. مورد کاربردها در ردیف، و تکرارها در ستون قرار می‌گیرند. خانه‌های جدول بیان‌گر توزیع تلاش مورد کاربردها بین تکرارها می‌باشد. باید مراقب بود که جمع تلاش‌ها در هر ستون از توانایی تیم، که در این مثال نه نفر در هفته است، بیشتر نشود. جمع تلاش‌های هر ردیف نیز نباید از تلاش لازم برای توسعه و تحویل آن مورد کاربرد کمتر باشد. وابستگی‌ها و اولویت‌های مورد کاربردها نیز باید برآورده شوند.

Use Case	Priority	Effort (person-week)	Depend On	Iteration1 3 wks 9/15/08-10/3/08	Iteration2 3 wks 10/6/08-10/24/08	Iteration3 3 wks 10/27/08-11/14/08	Iteration4 3 wks 11/17/08-12/5/08
UC1	1	5	None	5			
UC2	4	4	UC8			4	
UC3	1	2	UC1	2			
UC4	2	5	UC3	2	3		
UC5	4	*3	UC4			3	
UC6	5	*6	UC4				6
UC7	4	5	UC2		1	1	3
UC8	3	5	None		5		
Total Effort		35		9	9	8	9

Team size = 3 Persons

شکل ۷-۱۵: تخصیص مورد کاربردها به تکرارها

۷-۵ رهنمون‌هایی برای استنتاج مورد کاربردها

رهنمون ۷-۱

مورد کاربردها باید از نیازمندی‌های کارکردی استنتاج شده و آن‌ها را برآورده نمایند.

رهنمون ۷-۲

مورد کاربردها می‌توانند از نیازمندی‌های غیرکارکردی نیز استنتاج شوند. (مثلاً، نیازمندی‌های امنیتی)

رهنمون ۷-۳

در انتخاب عبارت‌های فعلی–اسمی دقت کنید به‌شکلی که این عبارت‌ها دقیقاً بگویند مورد کاربردها چه چیزی را برای کنشگر انجام می‌دهند.

رهنمون ۷-۴

تبیین مورد کاربرد سطح بالا با آن‌چه که کنشگر می‌خواسته انجام دهد خاتمه یابد.

رهنمون ۷-۳ و رهنمون ۷-۴ به تیم توسعه امکان می‌دهند که بر روی طراحی، پیاده‌سازی، و آزمون مورد کاربردهایی تمرکز کند که آن‌چه کنشگر انتظار دارد را انجام می‌دهند.

رهنمون ۷-۵

هر مورد کاربرد باید دقیقاً یك وظیفه‌ی کسب‌وکار را برای کنشگر انجام دهد (چسبندگی عملکردی).

برای مثال، «ورود به سیستم و انجام معامله» باید به دو مورد کاربرد تبدیل شود: «ورود به سیستم» و «انجام معامله». به همین ترتیب، «بررسی موجودی و انبار» باید بشود «بررسی موجودی» و «بررسی انبار».

رهنمون ۷-۶

مورد کاربردهای سطح بالا نباید فعالیت‌های پردازشی پس‌زمینه را تعیین کنند.

این رهنمون دو اصل طراحی نرم‌افزاری را حمایت می‌کند: ساده و احمقانه بگیر[۱]، و جداسازی دغدغه‌ها، که در فصل ۶ تشریح شده‌اند. منظور از یك مورد کاربرد سطح‌بالا این است که فقط قلمرو مورد کاربرد را تعیین کند یا بگوید کی وکجا شروع شده و کی خاتمه می‌یابد.

---

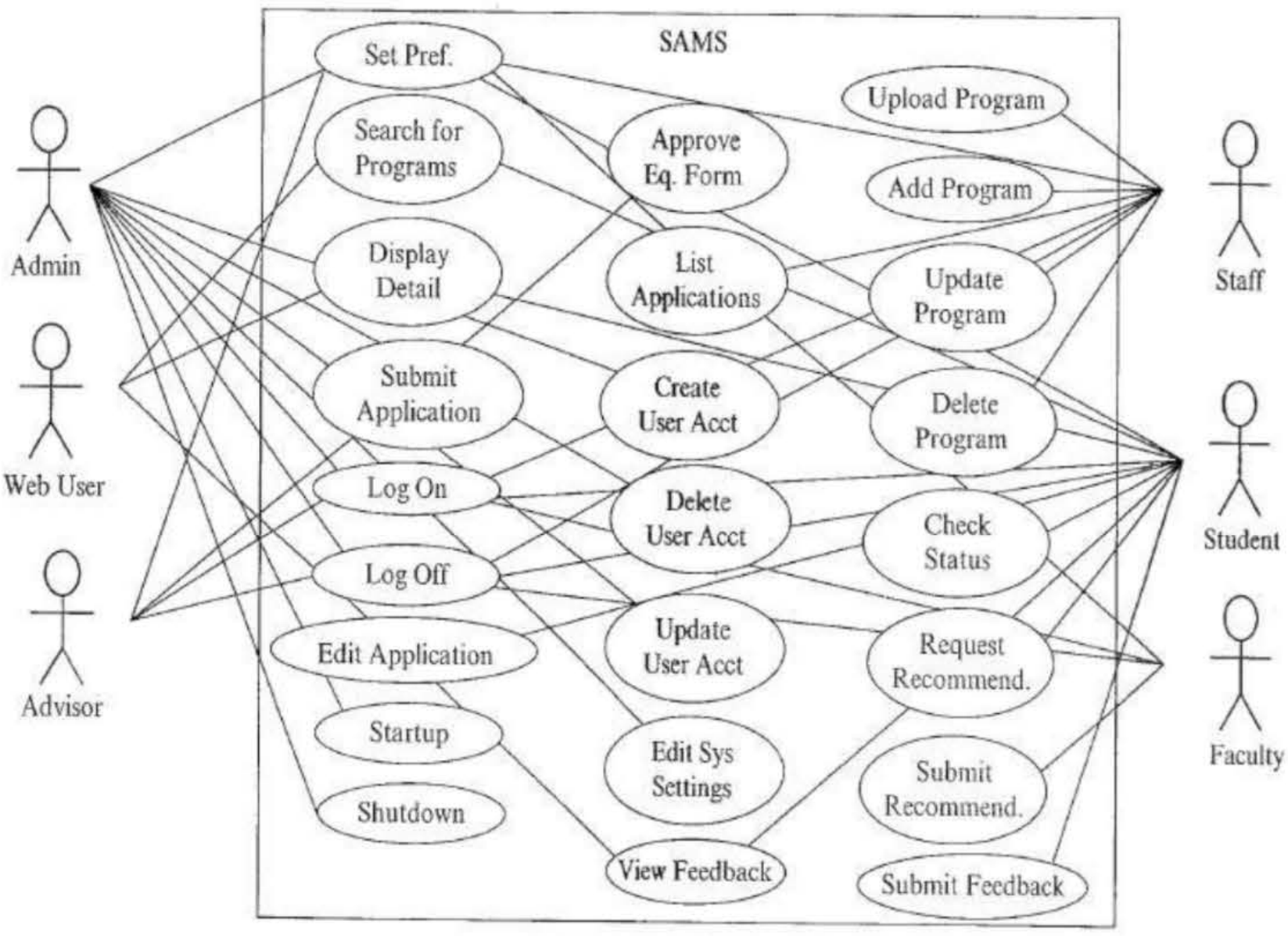
[۱] Keep It Simple and Stupid

رهنمون ۷-۷

از نمایش موارد زیر پرهیز کنید:

- چندین مورد کاربرد در یك نمودار.
- چندین نمودار مورد کاربرد که هر کدام تنها شامل یك مورد کاربرد است.
- روابط متعدد بین مورد کاربردها.
- نمودارهای مورد کاربرد کلاً پیچیده.

شکل ۷-۱۶ یك نمودار مورد کاربرد کلاً پیچیده را نشان می‌دهد که باید از آن پرهیز نمود.

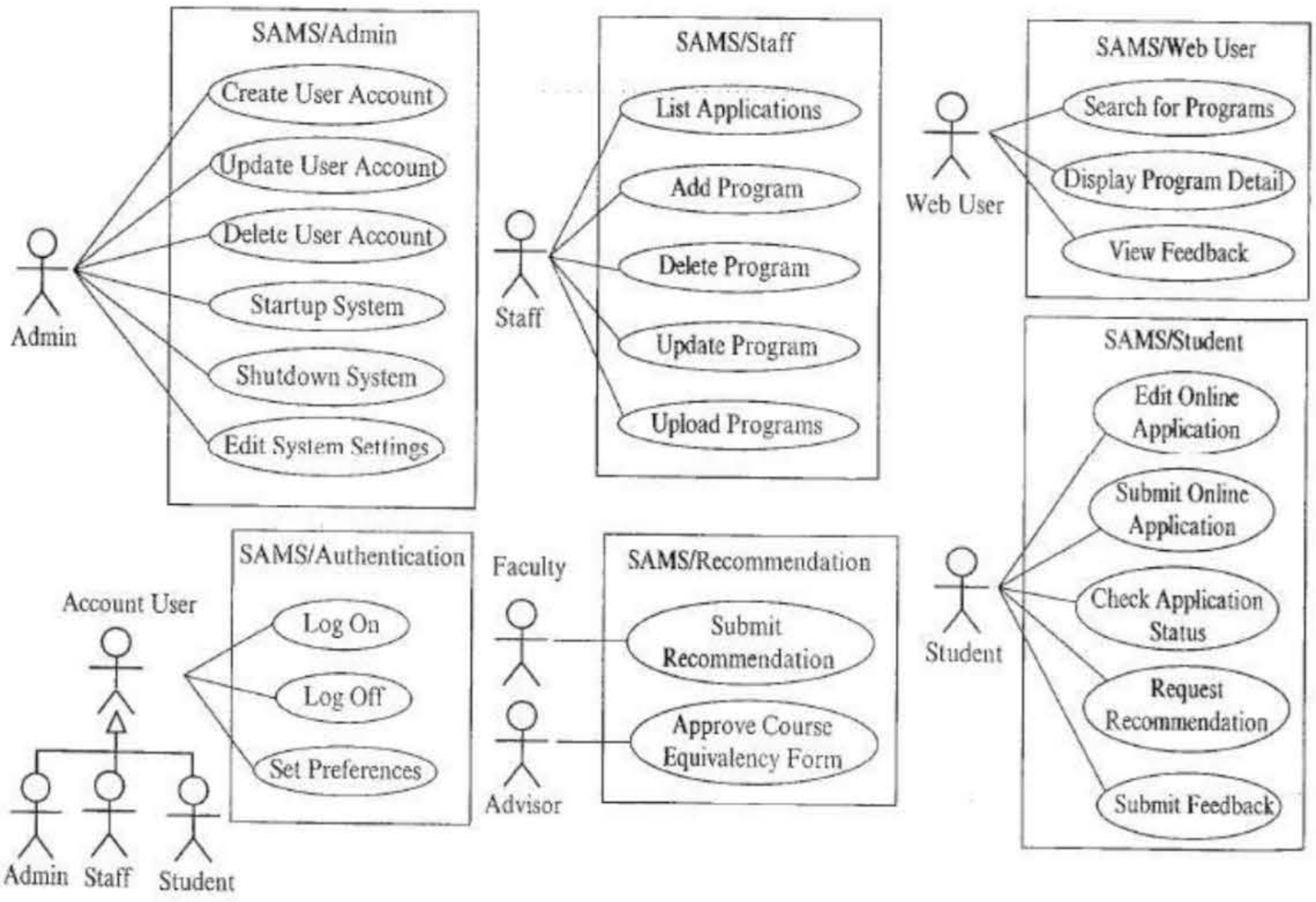


شکل ۷-۱۶: نمودار مورد کاربرد کلاً پیچیده که باید ازآن پرهیز نمود.

رهنمون ۷-۸

مورد کاربردها را با دقت به گروه‌هایی از مورد کاربردها که کاملاً به‌هم مرتبط هستند تقسیم نمایید.

برای دیدن برخی تکنیك‌های افرازبندی به بخش ۷-۴-۱ مراجعه کنید. به‌عنوان مثال، اعمال روش افراز مبتنی بر نقش بر روی نمودار نشان داده شده در شکل ۷-۱۶ منجر به تولید نمودار شکل ۷-۱۷ خواهد شد. این کار پیچیدگی نمودارهای مورد کاربرد را به شدت کاهش داده و در عین حال چسبندگی کارکردی زیرسیستم‌ها را نیز افزایش می‌دهد.



شکل ۷-۱۷: افراز مبتنی بر نقش باعث کاهش پیچیدگی و افزایش چسبندگی می‌شود.

رهنمون ۷-۹

فقط مورد کاربردها و کنشگرهایی را نشان دهید که به یك زیرسیستم یا یک جنبه مربوط هستند.

رهنمون ۷-۱۰

برای هر سیستم، بر اساس کارکرد آن، یك نام با معنی انتخاب کنید.

رهنمون ۷-۱۱

با استفاده از ارث‌بری کنشگر، تعداد پیوندهای کنشگر-مورد کاربرد را در نمودار مورد کاربرد کاهش دهید (همان‌گونه که در شکل ۷-۱۱ مشاهده می‌شود).

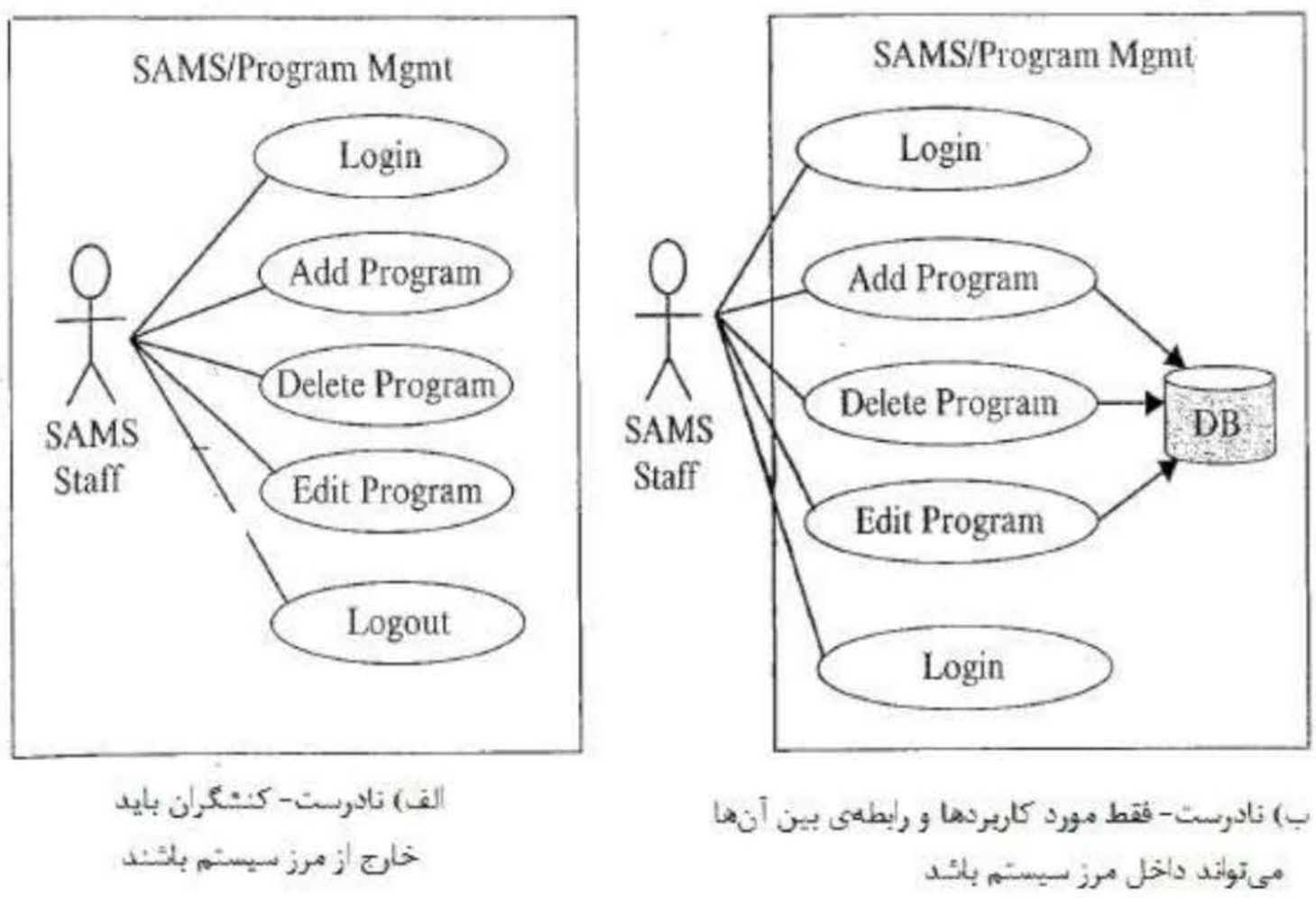
رهنمون ۷-۱۲

روابط کنشگر-مورد کاربرد همیشه روابط انجمنی هستند.

رهنمون ۷-۱۳

در مرز (محدوده) سیستم فقط می‌توان مورد کاربردها و روابط‌شان را نشان داد.

شکل ۷-۱۸ دو نمودار مورد کاربرد نادرست را نشان می‌دهد. یکی از نمودارها، کنشگر را داخل مرز سیستم قرارداده و دیگری یك پایگاه داده را درون مرز سیستم گذاشته است.



الف) نادرست- کنشگران باید خارج از مرز سیستم باشند

ب) نادرست- فقط مورد کاربردها و رابطه‌ی بین آن‌ها می‌تواند داخل مرز سیستم باشد

شکل ۷-۱۸: نمودارهای مورد کاربرد که نادرست رسم شده‌اند.

۷-۶ به‌کاربردن اصول چابکی

رهنمون ۷-۱۴

با مشتری و کاربران همکاری نزدیك داشته باشید تا فرایندهای کسب‌وکاری و اولویت‌های ایشان را بفهمید و به آن‌ها کمك کنید تا نیازهای واقعی‌شان را شناسایی کنند.

مورد کاربردها، فرایندهای کسب‌وکاری هستند. فقط مشتری و کاربران هستند که فرایند کسب‌وکار، چگونگی انجام و اولویت آن‌ها را می‌دانند. بنابراین، تیم توسعه باید با مشتری و کاربران از نزدیک کار کند تا بتواند مورد کاربردها را شناسایی و اعتبارسنجی نماید.

رهنمون ۷-۱۵

اعضای تیم باید با یکدیگر کار کنند تا مورد کاربردها، کنشگرها و زیرسیستم‌ها را شناسایی نموده و قلمرو مورد کاربردها را تعیین نمایند.

فرایند استنتاج و تبیین مورد کاربردها صرفاً به‌خاطر مستندسازی نیست. در این فرایند، اعضای تیم فرایندهای کسب‌وکار را می‌فهمند و برداشت‌شان را با یکدیگر به اشتراک می‌گذارند. این کار پایه‌ای فراهم می‌کند که باعث می‌شود قطعاتی که اعضای تیم ساخته‌اند با هم کار کنند. یکی از دلایلی که روش‌های چابک به افراد و تعاملات‌شان بها می‌دهند نیز همین است. تقسیم کار، بدون کارکردن افراد با هم برای شناسایی و تبیین مورد کاربردها، می‌تواند در مراحل بعدی منجر به تولید نتایج ضعیفی گردد.

رهنمون ۷-۱۶

نیازمندی‌ها تکامل می‌یابند ولی مقیاس زمان ثابت است.

توسعه‌ی چابک مایل است برای تکرارها از مقیاس زمانی ثابتی استفاده کند. این باعث می‌شود کارکردن اعضای تیم در تکرارها آسان‌تر شود. برای مثال، پس از چند تکرار، اعضای تیم می‌فهمند که در یک تکرار چقدر کار می‌تواند انجام شود. بنابراین، تیم می‌تواند با یک سرعت ثابت پیش برود. اگر تکرار نتواند تمام خصیصه‌ها را کامل کند، در این صورت با هم‌فکری مشتری و کاربران باید خصیصه‌های با اولویت پایین را حذف یا آن‌ها را به تکرار بعدی منتقل کرد.

رهنمون ۷-۱۷

بر روی تحویل متناوب افزایش‌های کوچک، که هرکدام تنها چند مورد کاربرد را تحویل می‌دهند، تمرکز کنید.

این کار قابلیت دید را افزایش می‌دهد. بدین معنی که تصمیم‌گیری در مورد این‌که چه میزان کار را می‌توان در دو هفته انجام داد، نسبت به تصمیم در مورد کار قابل انجام در شش ماه آسان‌تر خواهد بود. به‌علاوه، تحویل متناوب افزایش‌های کوچک، ریسک و درک نادرست از نیازمندی‌ها را کاهش می‌دهد. همچنین، این‌کار از آزمون مداوم توسط کاربران پشتیبانی نموده و منحنی یادگیری کاربران را کاهش می‌دهد.

رهنمون ۷-۱۸

سعی نکنید یک مجموعه‌ی بهینه از مورد کاربردها به‌دست آورید. به حد کفایت خوب، کافی است.

نخست این‌که، هیچ مجموعه‌ی بهینه‌ای از مورد کاربردها وجود ندارد زیرا نمی‌توان معیارها را تعریف نمود. علاوه براین، نیازمندی‌ها همواره در حال تغییر هستند، بدین معنی که مورد کاربردها نیز مجبورند تغییر کنند. بنابراین، تنها کافی است بیشتر نیازمندی‌ها و مورد کاربردهای با اولویت بالاتر را شناسایی و تبیین نمود. مهم‌تر این‌که به‌سوی پیاده‌سازی مورد کاربردها پیش

رفت و از کاربران بازخورد گرفت. یعنی، به حد کفایت خوب، کافی است.

۷-۷ پشتیبانی ابزاری برای مدل‌سازی مورد کاربردها

ابزارهای مختلفی برای ترسیم و مدیریت نمودارهای مورد کاربرد وجود دارند، مانند Microsoft، IBM Rational Modeler، Visio، Argouml، NetBeans UML Plugin. این ابزارها قابلیت‌های ویرایشی و سایر ویژگی‌های مربوطه را در اختیار می‌گذارند. اکثر این ابزارها مهندسی معکوس[۱] و تولید کد اسکلت از روی نمودار UML را نیز پشتیبانی می‌کنند. به‌منظور پشتیبانی متدولوژی ارائه شده در این فصل، ابزاری در دانشگاه تگزاس-آرلینگتون[۲] توسعه یافته است. این ابزار شبیه ابزار مدل‌سازی دامنه است که در انتهای فصل مدل‌سازی (فصل ۵) معرفی شد. بدین معنی که، این ابزار مهندس نرم‌افزار را در شناسایی عبارت‌های فعلی-اسمی که همان مورد کاربردهای بالقوه هستند، راهنمایی می‌کند. ابزار از مهندس نرم‌افزار می‌خواهد که تأیید کند که مورد کاربرد چهار خاصیت پیش‌تر گفته‌شده را برآورده می‌نماید: مورد کاربرد باید یك فرایند کسب‌وکار باشد، با یك کنشگر شروع شود، با کنشگر تمام شود، یك وظیفه‌ی کسب‌وکاری برای کنشگر به انجام رساند. به‌علاوه، ابزار مهندس نرم‌افزار را راهنمایی می‌کند تا برای مورد کاربردها، کنشگرها و سیستم‌ها را نیز شناسایی کند. مهندس می‌تواند در حین این فرایند، مورد کاربردها را ویرایش نماید. وقتی مورد کاربردها شناسایی شدند، مهندس نرم‌افزار بر روی دکمه‌ی تولید نمودار[۳] کلیک می‌کند، ابزار طرح نمودار را محاسبه می‌کند و به‌طور خودکار نمودار مورد کاربرد را ترسیم می‌نماید.

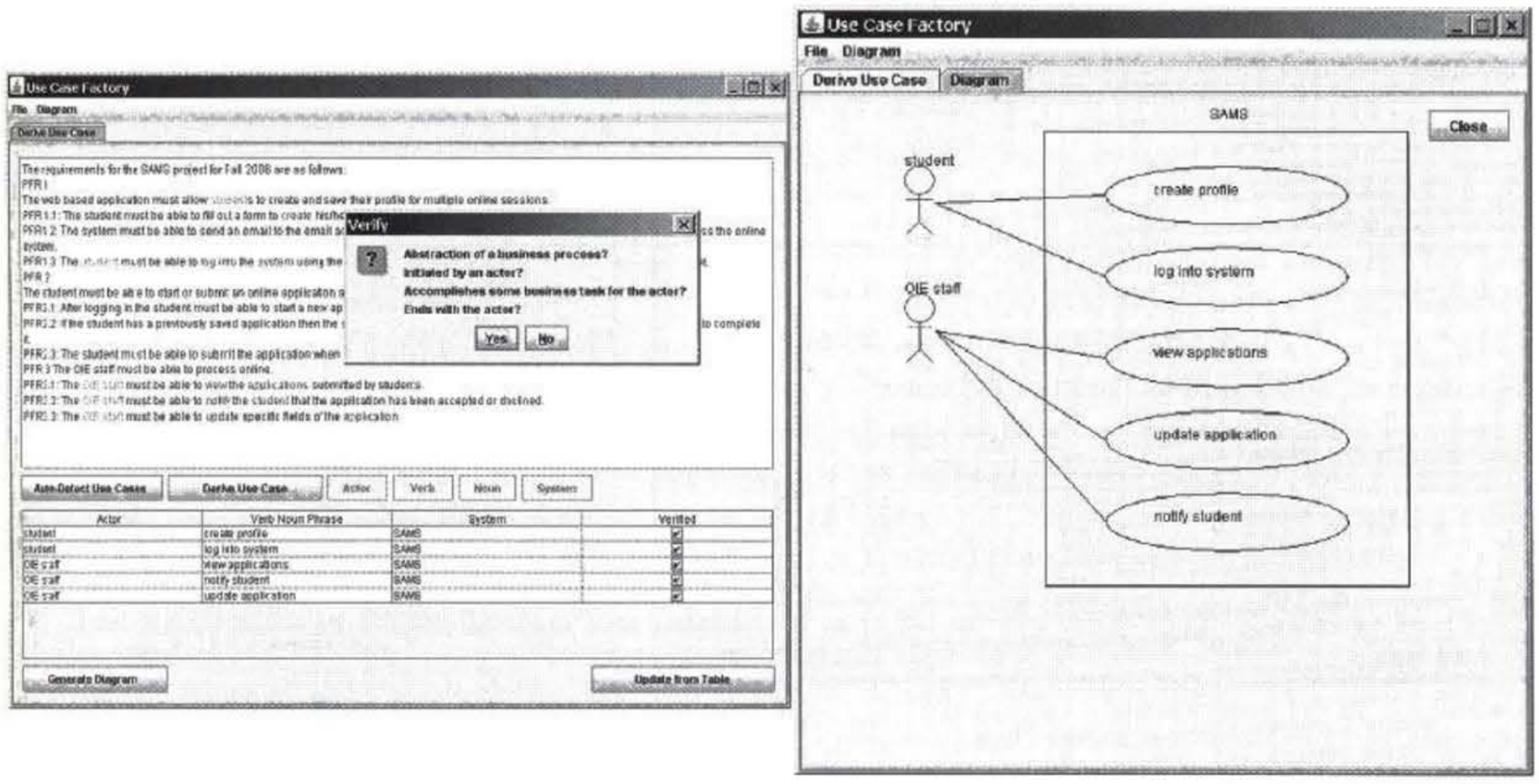
ابزار در چهار مد[۴] شبیه ابزار مدل‌سازی دامنه کار می‌کند. در مد شناسایی دستی و طبقه‌بندی دستی، مهندس نرم‌افزار به‌صورت دستی مورد کاربردها، کنشگرها، و سیستم‌ها را شناسایی می‌کند. ابزار به‌طور خودکار نمودارهای مورد کاربرد را ترسیم می‌نماید. شکل ۷-۱۹ نشان می‌دهد که این‌کار چگونه انجام می‌شود. بعد از خواندن متن تبیین نیازمندی‌ها، مهندس بر روی دکمه‌ی استنتاج مورد کاربرد[۵] کلیک می‌کند، و عبارت فعلی-اسمی، کنشگر و سیستم را به ترتیب برجسته[۶] کرده و برای طبقه‌بندی آن‌ها بر روی دکمه‌های متناظر کلیک می‌کند. ابزار پیغامی نمایش داده و از مهندس می‌خواهد تأیید کند که مورد کاربرد چهار ویژگی یك مورد کاربرد را برآورده می‌سازد. اگر مهندس روی بله کلیک کند، در این‌صورت مورد کاربرد در جدولی که در قسمت پایین صفحه است ظاهر می‌شود. مهندس بر روی دکمه‌ی تولید نمودار[۷] کلیک می‌کند تا نمودار مورد کاربرد نمایش داده شود. در مد شناسایی خودکار و طبقه‌بندی خودکار، ابزار تمام این کارها را به‌طور خودکار انجام می‌دهد. شکل ۷-۲۰ نشان می‌دهد که چگونه این‌کار انجام می‌شود.

خلاصه

این فصل چگونگی شناسایی مورد کاربردها، کنشگرها و زیرسیستم‌ها از روی نیازمندی‌ها، و نیز ثبت روابط در یك جدول (ماتریس) ردیابی نیازمندی-مورد کاربرد (RUTM) را ارائه می‌نماید. جدول (RUTM) اولویت نیازمندی‌ها و مورد کاربردها را نیز نشان می‌دهد. از این اطلاعات در تولید زمان‌بندی تکرار استفاده می‌شود به‌گونه‌ای که مورد کاربردهای با اولویت بالا هرچه زودتر توسعه یافته و تحویل شوند. تخصیص مورد کاربردها به تکرارها در قالب یك جدول تخصیص مورد کاربرد- تکرار نشان داده می‌شود. مورد کاربردهای سطح بالا، قلمرو مورد کاربرد یا این‌که کی و کجا شروع و کی تمام می‌شود را مشخص می‌کند. مورد کاربردها، کنشگرها و زیرسیستم‌ها و نیز روابط بین آن‌ها در قالب نمودارهای مورد کاربرد UML به‌تصویر کشیده می‌شوند.

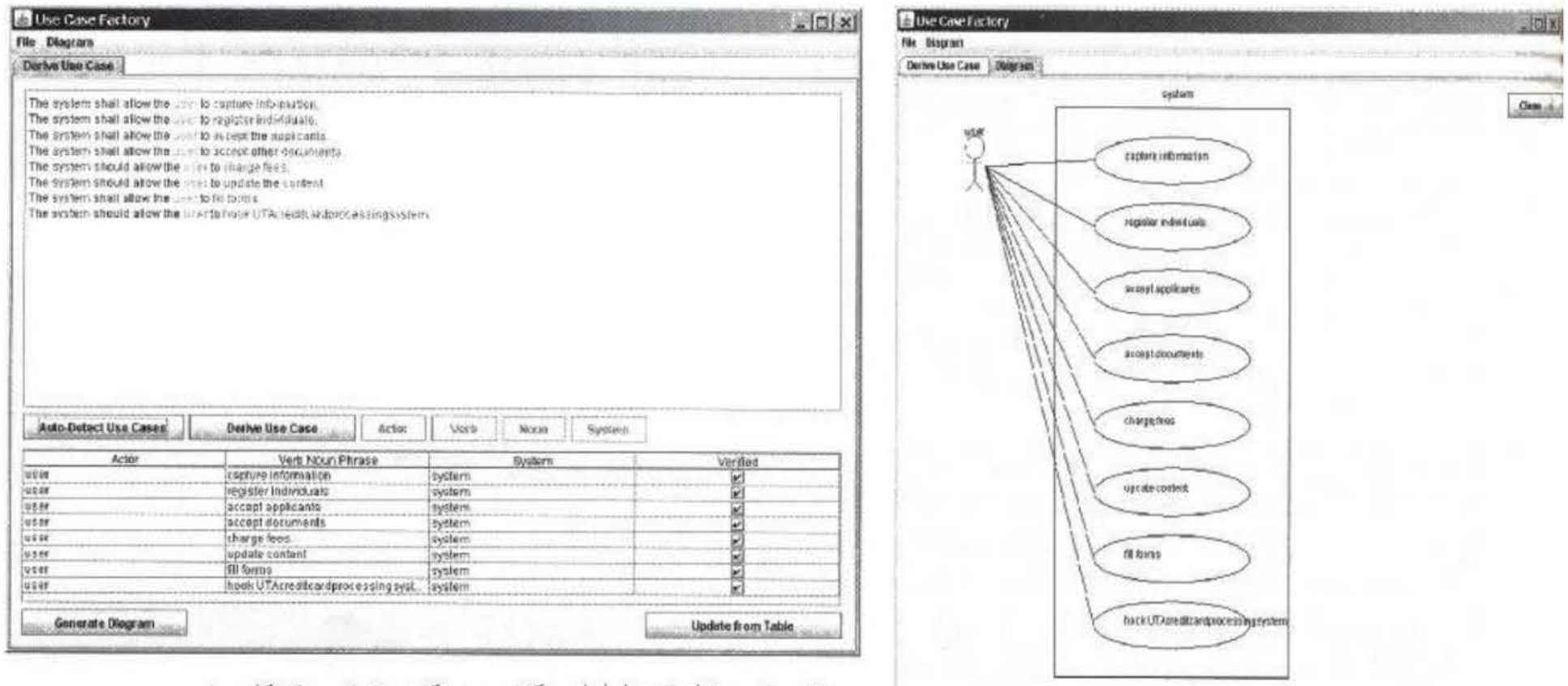
---

[۱] Reverse Engineering
[۲] University of Texas at Arlington
[۳] Generate Diagram
[۴] Mode
[۵] Derive Use Case
[۶] Highlight
[۷] Generate Diagram



مهندس بر روی دکمه‌ی استنتاج مورد کاربرد (Derive Use Case) کلیک می‌کند تا هر یک از مورد کاربردها را شناسایی کند. مهندس کنشگر، عبارت فعلی-اسمی، و سیستم را برجسته کرده و بر روی دکمه‌ی متناظر کلیک می‌کند تا مورد کاربرد وارد شود. ابزار پیغام تایید را نمایش می‌دهد. اگر مهندس بله را انتخاب کند، در این‌صورت مورد کاربرد در جدولی در قاب پایین صفحه ظاهر می‌شود. مهندس بر روی دکمه‌ی تولید نمودار (Generate Diagram) کلیک می‌کند تا نمودار مورد کاربرد نمایش داده شود.

شکل ۷-۱۹: شناسایی مورد کاربردها به صورت دستی



مهندس تبیین نیازمندی‌ها را باز می‌کند و روی دکمه‌ی تشخیص خودکار مورد کاربردها (Auto Detect Use Cases) کلیک می‌کند. ابزار به طور خودکار مورد کاربردها، کنشگرها و سیستم را شناسایی می‌کند، و آن‌ها را در جدول فهرست می‌کند. مهندس روی جعبه‌های مربوط کلیک می‌کند تا هر مورد کاربرد را تایید کند. سپس بر روی دکمه‌ی تولید نمودار (Generate Diagram) کلیک می‌کند تا از ابزار بخواهد نمودار مورد کاربرد را تولید کند.

شکل ۷-۲۰: استنتاج خودکار مورد کاربردها

این فصل پایان مرحله‌ی برنامه‌ریزی را نوید می‌دهد. در مرحله‌ی تکرار، مورد کاربردها بر طبق زمان‌بندی تکرارها، توسعه و تحویل می‌شوند.

برای مطالعه‌ی بیش‌تر

مفهوم مورد کاربرد توسط ایوار جیکسون[۱] در روشی که وی آن‌را مهندسی مورد کاربرد می‌نامید، معرفی شد [۹۰]. فرایند یکنواخت برای مدل‌سازی مورد کاربرد در [۹۱] تشریح شده است. آلیستر کوبرن[۲] [۴۹] بحث جامعی روی نوشتن مورد کاربرد دارد. نمونه‌ی مورد کاربرد توسط دیگر نویسندگان با عنوان سناریو مطرح شده است [۱۰۴،۱۳۱]. مرجع [۱۳۱] به صورت غیررسمی چگونگی آماده‌سازی سناریوها را بر اساس مثال ATM توضیح داده است. مرجع [۱۰۴] به‌طور غیررسمی سناریو را به‌عنوان نمونه‌ی یك مورد کاربرد و دنباله‌ی مشخصی از اعمال و تعاملات بین کنشگرها و سیستم تعریف می‌کند. ارائه‌ی مختصری درباره‌ی نمودار مورد کاربرد را در [۳۶] و ارائه‌ی جامع آن را در www.omg.org می‌توان یافت. درانتها، بیتنر و اسپنس[۳] یك روش مدل‌سازی مورد کاربرد را به کمك مثال‌های دنیای واقعی در [۲۹] ارائه نموده‌اند.

سوال‌های مروری فصل

۱. مورد کاربرد چیست؟
۲. چهار ویژگی یك مورد کاربرد کدامند؟
۳. آیا هر عبارت فعلی–اسمی یك مورد کاربرد است؟ چرا؟
۴. فرایند کسب‌وکاری، عمل و کنش چیست؟ این‌ها چه ربطی به هم دارند؟
۵. کنشگر چیست؟ آیا یک شخص یك کنشگر است؟
۶. مورد کاربردها را از کجا باید استنتاج کرد؟ چرا؟
۷. چگونه باید مورد کاربردها، کنشگرها، و زیرسیستم‌ها را استنتاج نمود؟
۸. مورد کاربرد سطح بالا چیست؟
۹. چگونه می‌توان یك مورد کاربرد سطح بالا را تبیین نمود؟
۱۰. نمودار مورد کاربرد چیست؟
۱۱. چگونه می‌توان مورد کاربردها را افراز نمود تا چسبندگی کارکردی زیرسیستم‌ها بهبود یافته و تعداد مورد کاربردهای هر زیرسیستم کاهش یابد؟
۱۲. ماتریس ردیابی نیازمندی-مورد کاربرد چیست؟ فایده‌ی چنین جدولی چیست؟
۱۳. ماتریس تخصیص مورد کاربرد به تکرار چیست؟ فایده‌ی چنین جدولی چیست؟

---

[۱] Ivar Jacobson
[۲] Alistair Cockburn
[۳] Bittner and Spence

تمرین‌ها

۱-۷ در تمرین ۴-۲، شما تبیین نیازمندی‌ها را برای یك ماشین‌حساب مجازی رومیزی، یك منشی تلفنی، و یك سیستم رایانامه‌ی برخط مبتنی بر وب تولید کردید. برای کاربردهای فوق، مورد کاربردها، کنشگرهای مربوطه، و زیرسیستم‌ها را بدست آورید. همچنین مورد کاربردهای سطح بالا را مشخص کرده و نمودارهای مورد کاربرد را رسم نمایید.

۲-۷ از روی نیازمندی‌های کارکردی سیستم مدیریت تقویم که در تمرین ۴-۴ تولید کردید، مورد کاربردها را استنتاج نمایید. همچنین مورد کاربردهای سطح بالا را مشخص کنید.

۳-۷ برای تبیین نیازمندی‌های خدمات نمایشگاهی ملی[۱] که در تمرین ۴-۶ تولید کردید، کارهای زیر را انجام دهید:

الف) از روی تبیین نیازمندی‌ها، مورد کاربردها را بدست آورید.

ب) مورد کاربردهای سطح بالا را مشخص کنید.

ج) نمودارهای مورد کاربرد را ترسیم نمایید.

د) درصورت لزوم، قوانین افراز مورد کاربرد را اعمال نموده و تعداد مورد کاربردهایی که به زیرسیستم‌ها تخصیص داده‌اید را کاهش دهید.

۴-۷ برای سیستم اجاره‌ی خودرو[۲] که در پیوست ت آمده است، اعمال زیر را انجام دهید:

الف) از روی نیازمندی‌هایی که در تمرین ۴-۸ تولید کردید، مورد کاربردها را استنتاج کنید.

ب) یک ماتریس ردیابی نیازمندی-مورد کاربرد بسازید.

ج) مورد کاربردهای سطح بالا را مشخص نمایید.

د) نمودارهای مورد کاربرد را ترسیم نمایید.

۵-۷ برای یك ابزار مدل‌سازی مورد کاربرد که متدولوژی تشریح شده در این فصل را پیاده‌سازی می‌کند، نیازمندی‌های کارکردی را تدوین کنید. حاصل کار نباید بیش از ده یا دوازده صفحه باشد. مورد کاربردها را بدست آورده، مورد کاربردهای سطح بالا را مشخص کرده، و نمودارهای مورد کاربرد را تولید کنید. طبق فهرست بررسی داده شده، این‌ها را بازبینی کرده و یك گزارش بازبینی تولید نمایید.

---

[۱] National Trade Show Services (NTSS)

[۲] Car Rental System (CRS)

۸

مدل‌سازی تعامل کنشگر-سیستم

نکات کلیدی

- مدل‌سازی تعامل کنشگر-سیستم عبارت است از مدل‌سازی و طراحی این‌که چگونه سیستم با کنشگرها تعامل می‌کند تا مورد کاربردها انجام شوند.
- مدل‌سازی تعامل کنشگر-سیستم با ایجاد یک جدول دو ستونی که برای هر تعامل، ورودی و خروجی کنشگر و نیز پاسخ سیستم را نشان می‌دهد انجام می‌شود.

در فصل ۷ چگونگی استنتاج مورد کاربردها از نیازمندی‌ها و چگونگی تبیین قلمرو هر مورد کاربرد معرفی گردید. نتایج تحت عنوان مورد کاربردهای مجرّد و مورد کاربردهای سطح بالا نام‌گذاری گردیدند. هر مورد کاربرد یک فرایند کسب‌وکاری است، که با یک کنشگر شروع و با همان کنشگر تمام شده و یک وظیفه‌ی کسب‌وکاری را برای وی به انجام می‌رساند. برای انجام این کار، سیستم باید با کنشگر تعامل داشته باشد تا آن مورد کاربرد عملی شود. به این تعامل، تعامل کنشگر-سیستم گوئیم. مدل‌سازی و طراحی این رفتار تعاملی را نیز مدل‌سازی و طراحی رفتار تعاملی کنشگر-سیستم می‌نامیم. این فصل بر روی مدل‌سازی و طراحی چنین رفتاری تمرکز دارد. همان‌گونه که در فصول قبلی گفته شد، مورد کاربردها در سه سطح تجرید مدل‌سازی می‌شوند:

۱. **مورد کاربرد مجرد:** هر مورد کاربرد مجرد یک عبارت فعلی-اسمی است که آن‌چه را مورد کاربرد برای کنشگر انجام می‌دهد، برجسته می‌کند.
۲. **مورد کاربرد سطح بالا:** هر مورد کاربرد سطح بالا شامل یک عبارت TUCBW می‌شود که تعیین می‌کند کی و کجا مورد کاربرد شروع شده و نیز شامل یک عبارت TUCEW می‌باشد که مشخص می‌کند مورد کاربرد کی تمام می‌شود. اصطلاحات TUCBW و TUCEW به ترتیب بیان‌گر جملات «این مورد کاربرد شروع می‌شود با» و «این مورد کاربرد تمام می‌شود با» هستند. به عبارت دیگر، مورد کاربرد سطح بالا محدودی مورد کاربرد را مشخص می‌کند.
۳. **مورد کاربرد گسترده:** مورد کاربرد گسترده چگونگی تعامل کنشگر با سیستم/زیرسیستم را مشخص می‌کند. این مورد کاربرد، ادامه و پالایش مورد کاربرد سطح بالاست. این مورد کاربرد، تعامل را در قالب یک جدول دو ستونی نمایش می‌دهد. ستون راست ورودی کنشگر و/یا کنش‌های کنشگر و ستون چپ پاسخ‌های سیستم را مشخص می‌کند.

برای سادگی، در این کتاب از مورد کاربردهای مجرد با عنوان مورد کاربرد یاد می‌کنیم. مورد کاربردهای مجرد و مورد کاربردهای سطح بالا در فصل ۷ (استنتاج مورد کاربردها از نیازمندی‌ها) تشریح گردیدند. این فصل تبیین مورد کاربردهای گسترده را معرفی می‌کند. وقتی این فصل را تمام کنید، این چیزها را یاد خواهید گرفت:

- مدل‌سازی تعامل کنشگر-سیستم
- اهمیت مدل‌سازی تعامل کنشگر-سیستم
- چگونگی انجام مدل‌سازی تعامل کنشگر-سیستم

۸-۱ مدل‌سازی تعامل کنشگر-سیستم چیست؟

مدل‌سازی تعامل کنشگر-سیستم عبارت‌است از مدل‌سازی و طراحی این‌که چگونه سیستم با کنشگرها تعامل می‌کند تا مورد کاربردها انجام شوند. همان‌گونه که در شکل ۸-۱ مشاهده می‌شود ستون چپ ورودی کنشگر و/یا کنش‌های کنشگر و ستون راست پاسخ‌های سیستم را مشخص می‌کند. به‌طور مشخص‌تر، تبیین یک مورد کاربرد گسترده در قالب یک جدول دو ستونی این نکات را مشخص می‌نماید:

۱. **شماره و نام مورد کاربرد:** این موارد در بالای جدول شکل ۸-۱ نشان داده شده‌اند.
۲. **وضعیت اولیه‌ی واسط کاربر:** گام ۰ در ستون راست[1]، وضعیت اولیه‌ی واسط کاربر را قبل از شروع مورد کاربرد نشان می‌دهد. تعیین وضعیت اولیه‌ی واسط کاربر مهم است زیرا: (الف) به برنامه‌ساز می‌گوید که سیستم چه چیزی را باید به کنشگر نشان دهد، و (ب) به کنشگر می‌گوید که قبل از شروع مورد کاربرد، او چه چیزی را خواهد دید.
۳. **مورد کاربرد را کی و کجا شروع کنیم:** گام ۱ در ستون چپ مشخص می‌کند که مورد کاربرد کی و کجا شروع می‌شود. این همان عبارت TUCBW از مورد کاربرد سطح بالاست.
۴. **ورودی کنشگر و کنش کنشگر در هر گام از تعامل:** این در ستون چپ از هر ردیف جدول دو ستونی مشخص می‌شود.
۵. **پاسخ متناظر سیستم:** این در ستون راست از هر ردیف جدول دو ستونی مشخص می‌شود.
۶. **مورد کاربرد کی تمام می‌شود؟** آخرین قلم در ستون چپ تعیین می‌کند که مورد کاربرد کی تمام می‌شود. این همان عبارت TUCEW از مورد کاربرد سطح بالاست.

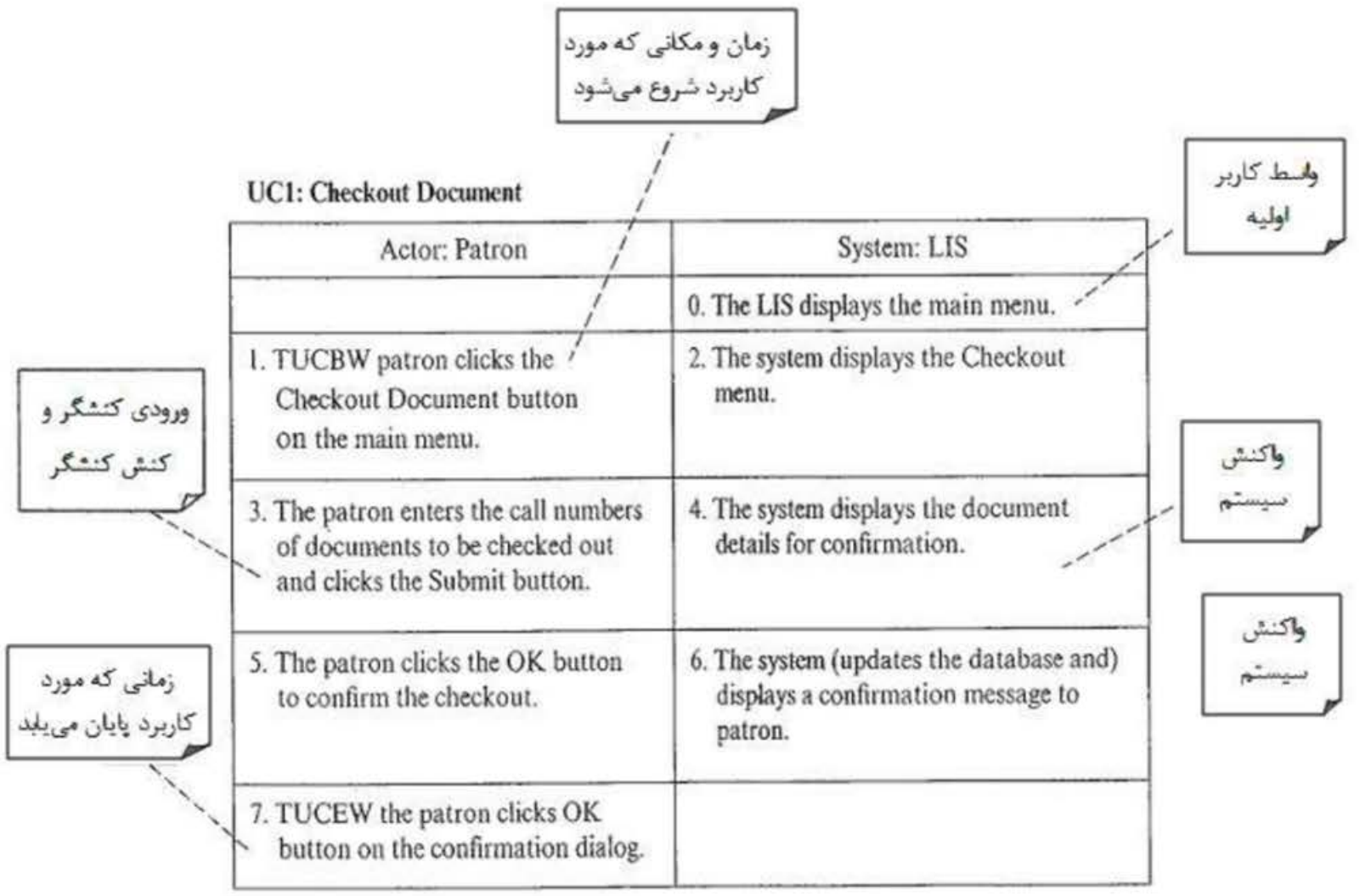
۸-۲ اهمیت مدل‌سازی تعامل کنشگر-سیستم

از جمله فواید تبیین مورد کاربرد گسترده می‌توان به موارد ذیل اشاره نمود:

۱. تعامل کنشگر-سیستم یا رفتار تعاملی سیستم را مشخص می‌کند که طراحی، پیاده‌سازی و آزمون می‌توانند از آن تبعیت کنند.
۲. برای تولید نسخه اولیه‌ی دفترچه راهنمای کاربران می‌توان از آن استفاده کرد. بدین‌خاطر که مورد کاربرد گسترده دقیقاً می‌گوید که کاربر چگونه از سیستم استفاده می‌کند تا یک وظیفه‌ی کسب‌وکاری را انجام دهد. نسخه‌ی اولیه‌ی دفترچه راهنمای کاربران به کاربر نوعی کمک می‌کند تا با پیش‌نمونه‌ی سیستم به آسانی کار کند.

---

[1] مترجم: توجه کنید که چنانچه جدول به فارسی تهیه شود جای ستون‌های چپ و راست عوض خواهد شد.



UC1: Checkout Document

Actor: Patron	System: LIS
	0. The LIS displays the main menu.
1. TUCBW patron clicks the Checkout Document button on the main menu.	2. The system displays the Checkout menu.
3. The patron enters the call numbers of documents to be checked out and clicks the Submit button.	4. The system displays the document details for confirmation.
5. The patron clicks the OK button to confirm the checkout.	6. The system (updates the database and) displays a confirmation message to patron.
7. TUCEW the patron clicks OK button on the confirmation dialog.	

شکل ۸-۱: مورد کاربرد گسترده برای سیستم اطلاعات کتابخانه

۳. چنان‌چه در حین مراحل بعدی طراحی و پیاده‌سازی برای نشان‌دادن تغییرات در تعامل کنشگر-سیستم، این تبیین مورد کاربرد گسترده به‌روزرسانی شد، می‌توان از آن برای تولید دفترچه راهنمای کاربران به‌روز شده نیز استفاده نمود. این‌کار باعث کاهش تلاش، هزینه، و زمان مربوط به توسعه‌ی سیستم خواهد شد.

۴. می‌توان از آن برای تولید موارد آزمون مبتنی بر مورد کاربرد نیز استفاده نمود. این موضوع در فصل ۲۰ (آزمون نرم‌افزار) مطرح خواهد شد.

۸-۳ گام‌های مدل‌سازی تعامل کنشگر-سیستم

فعالیت اصلی در مدل‌سازی تعامل کنشگر-سیستم، ساختن مورد کاربردهای گسترده برای آن مورد کاربردهایی است که به تکرار جاری تخصیص یافته‌اند. این کار شامل گام‌های زیر می‌شود:

- **گام ۱-** یک جدول دو ستونی برای مورد کاربرد گسترده که در حال ساخت است تشکیل دهید.
- **گام ۲-** هر کدام از گام‌های تعامل کنشگر سیستم را تعیین کنید تا سیستم پاسخی که در عبارت TUCEW مشخص شده را تولید کند.
- **گام ۳-** با کمک یک فهرست بررسی بازبینی، تعامل کنشگر-سیستم را بازبینی نمایید.

۸-۳-۱- ایجاد یک جدول دو ستونی

یک جدول دو ستونی رسم کرده و شماره و نام مورد کاربرد را بالای جدول بنویسید. عنوان ستون‌های چپ و راست را به ترتیب با نام نقش کنشگر و نام سیستم و زیرسیستم پُر کنید. عبارت‌های TUCBW و TUCEW مربوط به مورد کاربرد سطح بالای متناظر را به‌عنوان قلم‌های دوم و سوم ستون چپ وارد کرده و به‌ترتیب، به آن‌ها برچسب گام ۱ و گام ۳ بدهید. هرچه گام‌های بیشتری وارد جدول می‌شوند، شماره‌ی گام ۳ نیز افزایش می‌یابد. اولین قلم ستون چپ را خالی رها کنید. بدین معنی که خانه‌ای که دقیقاً زیر عنوان ستون چپ است خالی خواهد بود. سپس بر حسب عبارت TUCBW مورد کاربرد، اولین چیزی که سیستم نمایش خواهد داد را تشخیص داده و آن‌را در اولین خانه‌ی ستون راست بنویسید. برچسب این گام را «گام ۰» بگذارید. شکل ۸-۲ ، نتایج این گام را برای یک سیستم اطلاعات کتابخانه (LIS) نشان می‌دهد[1]. توجه کنید که در عبارت TUCBW در گام ۱ به فهرست اصلی رجوع شده است، بنابراین، گام ۰ بدین صورت خواهد بود: «۰- LIS فهرست اصلی را نشان می‌دهد.»

UC1: Checkout Document

کنشگر: عضو	سیستم: LIS
	۰- LIS فهرست اصلی را نشان می‌دهد.
۱- TUCBW عضو روی دکمه‌ی دریافت مدرک از فهرست اصلی کلیک می‌کند.	۲-
۳- TUCEW عضو برای تایید روی دکمه‌ی OK کلیک می‌کند.	

شکل ۸-۲: ایجاد یک مورد کاربرد گسترده

۸-۳-۲- تعیین گام‌های تعامل کنشگر-سیستم

در این مرحله، گام‌های تعامل کنشگر-سیستم مشخص می‌شوند. این‌کار با عبارت TUCBW در گام ۱ شروع می‌شود. پاسخ متناظر سیستم معلوم شده و به‌عنوان گام ۲ در ستون راست درج می‌شود. نتیجه به صورت «سیستم ... را نمایش می‌دهد» نوشته می‌شود. به‌طور کلی، سیستم نتایج را نشان می‌دهد یا یک جعبه‌ی گفت‌وگو را نشان می‌دهد تا ورودی کنشگر را بدست آورد. اگر نیاز به ورودی کنشگر باشد، در این‌صورت یک ردیف به جدول اضافه شده و ورودی کنشگر و کنش کنشگر در آن وارد می‌شوند. این فرایند برای سایر گام‌ها نیز تکرار می‌شود تا این‌که سیستم پاسخ مشخص شده در عبارت TUCEW را تولید نماید. نتیجه‌ی این گام در شکل ۸-۱ نشان داده شده است.

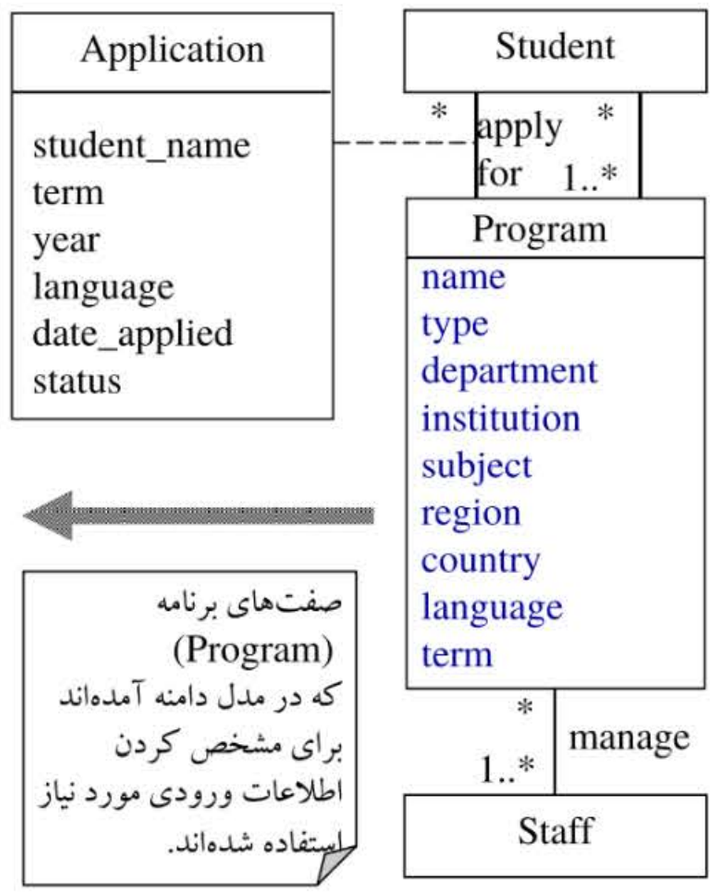
برخی مواقع، سیستم نیاز دارد که اطلاعاتی در خصوص یک مفهوم دامنه از سوی کنشگر وارد شود. این اطلاعات معمولاً در مدل دامنه یافت می‌شوند. شکل ۸-۳ این موضوع را نشان می‌دهد. برای مثال، در گام ۴ اطلاعات برنامه خواسته شده است؛ این اطلاعات را می‌توان از صفت‌های کلاس برنامه در مدل دامنه یافت. به همین ترتیب، ویژگی‌های کلاس انجمنی Application در شکل ۸-۳-(ب) می‌تواند در تبیین مورد کاربرد گسترده تقاضای برخط (Apply Online) به‌کار رود. چنان‌چه یک مفهوم دامنه در مدل دامنه پیدا نشود، باید مدل دامنه را اصلاح و بروزرسانی کرد. اگر مفهوم دامنه پیدا شود ولی اطلاعات آن ناقص باشد، باید فهرست صفت‌های آن را توسعه داد.

---

[1] مترجم: مجدداً یادآوری می‌شود که چون این جدول به فارسی تهیه شده است، جای ستون‌های راست و چپ عوض شده‌اند.

UC7: Add Program	
Actor : Staff User	System : SAMS
	(0) System displays the Welcome page.
(1) TUCBW staff user clicks on Program Management link on welcome page.	(2) System shows the Program Management submenu.
(3) Staff user clicks on Add Program link.	(4) System shows the Add Program Form with name, type, department, ...
(5) Staff user fills the form and clicks on the submit button.	(6) The System displays a success message.
(7) TUCEW staff user sees the program has been successfully added message.	

الف- مورد کاربرد گسترده



ب- بخشی از مدل دامنه

شکل ۸-۳: استفاده مجدد اطلاعات از مدل دامنه

۸-۳-۳- بازبینی مشخصات تعامل کنشگر-سیستم

مورد کاربردهای گسترده‌ای که در تکرار جاری تولید می‌شوند با کمک فهرست بررسی بازبینی زیر مورد بازبینی قرار می‌گیرند:

۱. آیا در مشخصات شماره مورد کاربرد و نام مورد کاربرد وجود دارد؟ آیا این شماره و نام با شماره و نامی که در ماتریس ردیابی‌پذیری نیازمندی- مورد کاربرد درج شده و نیز در مورد کاربرد سطح بالای متناظر با آن همخوانی دارد؟
۲. آیا کنشگر و سیستم به درستی شناسائی و مشخص شده‌اند و با آن‌چه در مورد کاربرد سطح بالا نوشته شده است مطابقت دارند؟
۳. آیا در گام ۰ مورد کاربرد گسترده، اولین چیزی که سیستم نمایش می‌دهد مشخص شده است؟
۴. آیا عبارت‌های TUCBW و TUCEW با آن‌چه که در مورد کاربرد سطح بالا نوشته شده مطابقت دارند؟
۵. آیا مورد کاربرد در ستون راست با کنشگر شروع و خاتمه می‌یابد؟
۶. آیا در جریان تعامل بین کنشگر و سیستم، خانه‌های خالی در جدول وجود دارند؟
۷. آیا مورد کاربرد گسترده به‌درستی و به اندازه‌ی کافی تعامل کنشگر-سیستم را مشخص می‌کند که بتواند فرایند کسب‌وکار را انجام دهد؟
۸. آیا گام‌های ستون راست به‌وضوح و به‌درستی ورودی کنشگر، و کنش‌های کنشگر، مانند کلیک‌کردن روی دکمه‌ی OK، را مشخص می‌کند؟
۹. آیا گام‌های ستون چپ به‌وضوح و به‌درستی عکس‌العمل‌های سیستم به کنشگر را مشخص می‌کنند؟

۸-۴ تبیین جریان‌های جایگزین

برای بسیاری از کاربردهای دنیای واقعی، تعامل بین کنشگر و سیستم شامل جریان‌های جایگزین نیز می‌شود. بدین معنی که، در یک گام مشخص از تعامل، ممکن است سیستم به کنشگر امکان انتخاب از میان تعدادی گزینه را بدهد. شکل ۸-۴ یک مثال از این مورد را نشان می‌دهد. در گام ۲، سیستم یک گفت‌وگو با دو انتخاب را نشان می‌دهد، همان‌گونه که از گام ۳ برداشت می‌شود. در گام ۳، تبیین مورد کاربرد گسترده از (الف) و (ب) استفاده کرده تا دو گزینه را نشان دهد. در گام ۴، عکس‌العمل‌های سیستم نیز با (الف) و (ب) نشان داده شده‌اند تا با گزینه‌های کاربر در قدم ۳ منطبق شوند.

کنشگر: کاربر	سیستم: مسیریاب
	۰- مرورگر صفحه‌ای را نشان می‌دهد.
۱- TUCBW کاربر "http://192.168.0.1" را در فیلد آدرس وارد کرده و کلید Enter را فشار می‌دهد.	۲- سیستم جعبه‌ی گفتگوی Enter Network Password را نمایش می‌دهد.
۳- کاربر الف) نام‌کاربری و کلمه‌ی‌عبور را وارد کرده و درصورت تمایل "save password" را علامت می‌زند و روی دکمه‌ی OK کلیک می‌کند تا وارد شود، یا ب) روی دکمه‌ی cancel یا abort کلیک می‌کند.	۴- سیستم به تناسب یکی از موارد زیر را نمایش می‌دهد الف) صفحه خوشامدگویی، یا ب) صفحه‌ی ورود غیرمجاز.
۵- TUCEW کاربر صفحه خوشامدگویی یا صفحه‌ی ورود غیرمجاز را مشاهده می‌کند.	

شکل ۸-۴: تبیین جریان‌های جایگزین در مورد کاربردهای گسترده

۸-۵ استفاده از پیش‌نمونه‌های واسط کاربر

«یک شکل بهتر از هزار کلمه است.» استفاده از پیش‌نمونه‌های واسط کاربر[1] مناسب موجب بهبود مشخصات مورد کاربردهای گسترده خواهد شد. منظور این است که پاسخ‌های سیستم در ستون راست مشخصات مورد کاربرد گسترده را می‌توان با به نمایش گذاردن پیش‌نمونه‌های واسط کاربر غنی‌تر ساخت، همان‌گونه که در شکل ۸-۵ نشان داده شده است. گام (۲) می‌گوید که «سیستم زیرفهرست مدیریت برنامه را نشان می‌دهد.» پیش‌نمونه‌ی واسط کاربر برای نمایش ظاهر زیرفهرست به‌کار می‌رود. به‌همین ترتیب، گام ۳ یک پیش‌نمونه‌ی واسط کاربر دیگر را نشان می‌دهد.

همان‌گونه که در شکل‌های ۸-۶ الی ۸-۸ نشان داده شده است، می‌توان برای افزایش خوانایی با استفاده از اشاره‌گر، پیش‌نمونه‌های واسط کاربر را نشان داد. همچنین می‌توان از سایر تکنیک‌های ارائه از قبیل پیوندهای URL نیز استفاده نمود.

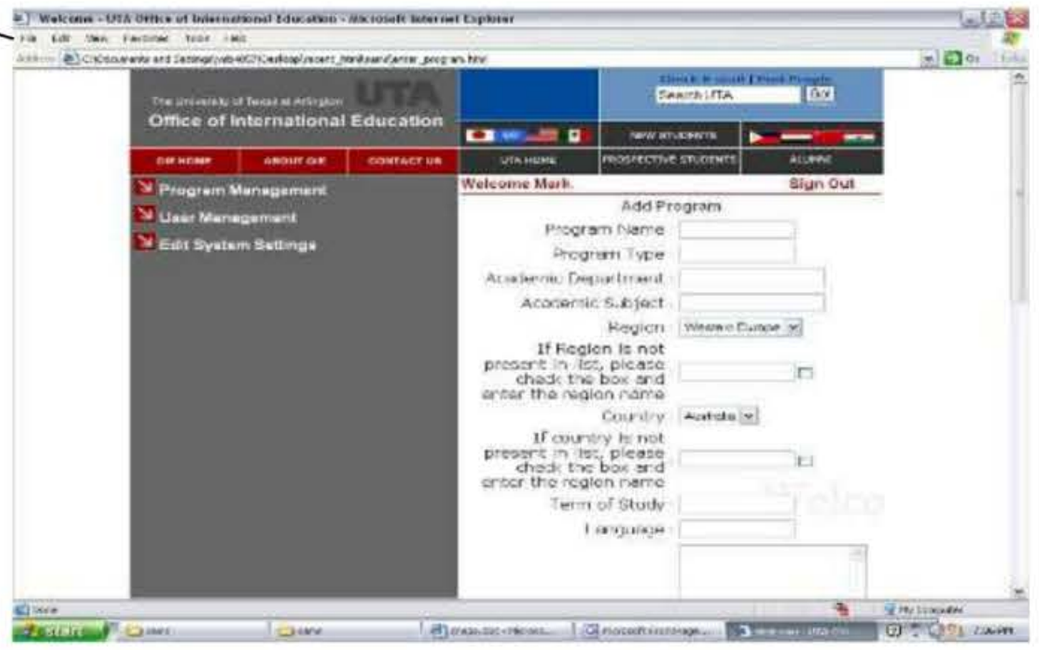
استفاده از پیش‌نمونه‌های واسط کاربر در تبیین مورد کاربرد گسترده چند حُسن دارد:

۱. در گرفتن بازخورد از کاربر و مشتری در مراحل اولیه‌ی چرخه‌ی حیات توسعه، بسیار مفید است.
۲. استفاده از پیش‌نمونه‌های واسط کاربر در تبیین مورد کاربرد گسترده موجب تسهیل در آماده‌سازی راهنمای کاربر می‌شود.
۳. استفاده از پیش‌نمونه‌های واسط کاربر کمک می‌کند تا مورد آزمون‌های مبتنی بر مورد کاربرد زودتر آماده گردند.

---
[1] User Interface (UI) Prototypes

Actor : Staff User	System : SAMS
(1) **TUCBW** staff member clicks on Program Management link on welcome page.	(2) System shows the submenu consisting of various operations that a staff user can do under Program Management.
(3) Staff member clicks on Add Program link.	(4) System shows the Add Program Form
(5) User fills the form and clicks on submit button.	(6) The System updates the database with the new program details and displays a success message.
(7) **TUCEW** staff member is shown a message that the program has been successfully added.	





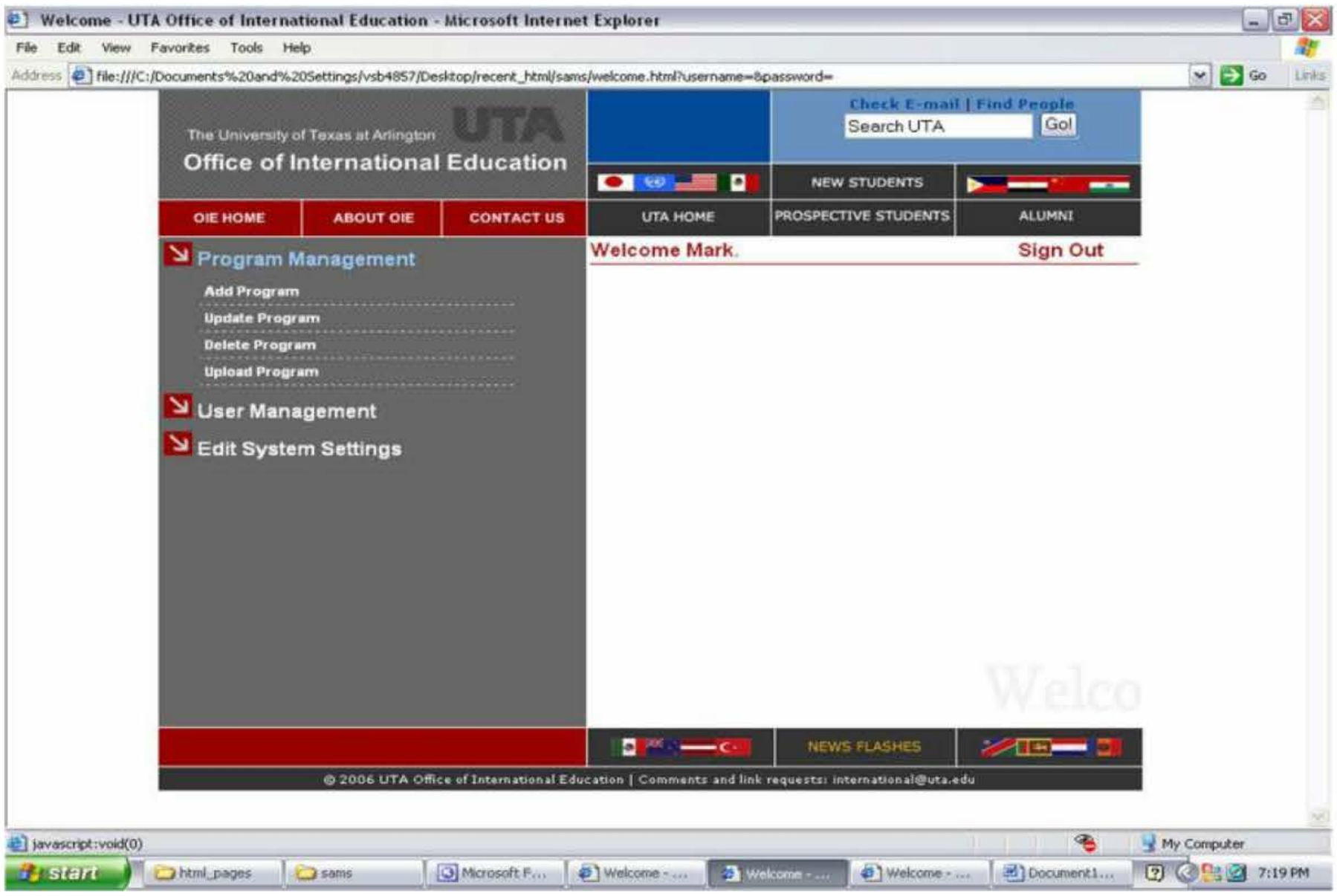
شکل ۸-۵: یک مورد کاربرد گسترده و پیش‌نمونه‌های واسط کاربر

کنشگر: کارمند	سیستم: SAMS
	۰- سیستم صفحه خوشامدگویی را نشان می‌دهد.
۱- TUCBW کارمند روی پیوند Program Management در صفحه خوشامدگویی کلیک می‌کند.	۲- سیستم زیرفهرست Program Management را نمایش می‌دهد.
۳- کارمند روی پیوند Add Program کلیک می‌کند.	۴- سیستم فرم Add Program را نشان می‌دهد.
۵- کارمند فرم را تکمیل و روی دکمه‌ی Submit کلیک می‌کند.	۶- سیستم پیغامی دال بر اضافه شدن برنامه به‌طور موفقیت‌آمیز، نشان می‌دهد.
۷- TUCEW کارمند پیغام اضافه شدن برنامه به‌طور موفقیت‌آمیز را مشاهده می‌کند.	

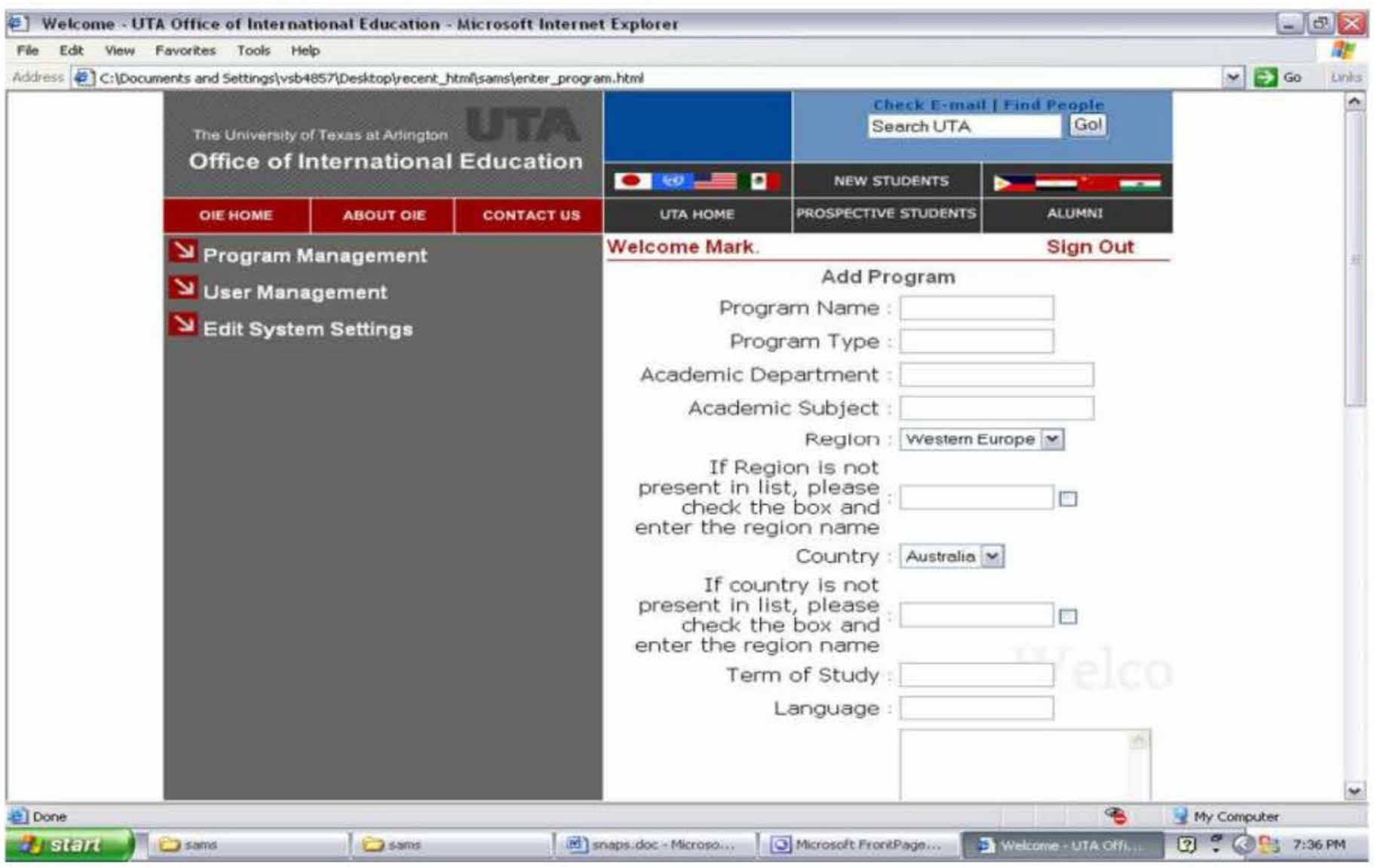
نمایش ۱ را ببینید: زیرفهرست Program Management

نمایش ۲ را ببینید: زیرفهرست Add Program Form

شکل ۸-۶: مورد کاربرد گسترده با ارجاعاتی به نمایش واسط کاربر



شکل ۸-۷: نمایش ۱ از پیش‌نمونه‌ی واسط کاربر



شکل ۸-۸: نمایش ۲ از پیش‌نمونه‌ی واسط کاربر

یکی از چالش‌های مهم پیش روی تحلیل نیازمندی‌های نرم‌افزار و طراحی نرم‌افزار تعیین نیازمندی‌های واسط کاربر است. بسیار دشوار است که بخواهیم با استفاده از زبان طبیعی طرح، ظاهر، رنگ‌آمیزی، و نوع قلم را توصیف نماییم. همان‌طور که در شکل ۸-۵ نشان داده شده است، با استفاده از پیش‌نمونه‌های واسط کاربر در تبیین مورد کاربرد توسعه‌یافته، این مسئله می‌تواند به آسانی حل شود. در واقع، پیش‌نمونه‌های واسط کاربر شکل ۸-۵ نتیجه‌ی اظهارنظر مشتری در رابطه با یک نسخه‌ی قدیمی‌تر است. این نسخه‌ی قدیمی با وب‌گاه‌های موجود مشتری از نظر طرح، ظاهر، رنگ‌آمیزی، و نوع مدل‌سازی قلم سازگاری نداشت. از آن‌جا که این بازخورد (اظهارنظر) در همان اوایلی که مورد کاربرد گسترده در حال ارائه به مشتری بود، دریافت گردید، تیم توسعه از دوباره‌کاری قابل‌ملاحظه‌ای پرهیز نمود. مشتری‌ها و کاربران دوست دارند پیش‌نمونه‌های واسط کاربر را ببینند، زیرا این پیش‌نمونه‌ها به مشتری و کاربر کمک می‌کند بفهمند که پس از پیاده‌سازی سیستم، چه چیزی به دستشان خواهد رسید. این امر ارائه‌ی بازخورد توسط مشتری‌ها و کاربران را آسان می‌کند. مشتری‌ها و کاربران از این‌که بخواهند در رابطه با مورد کاربردهای گسترده که پاسخ سیستم را به صورت متنی تشریح می‌کند بازخورد بدهند محتاطند، زیرا مطمئن نیستند که این توصیفات را به درستی فهمیده باشند.

همان‌گونه که در بخش‌های پیشین گفته شد، از توصیفات مورد کاربردهای گسترده می‌توان نسخه‌ی اولیه‌ای از راهنمای کاربران را تولید کرد. در موقع ایجاد نسخه‌ی اولیه‌ی راهنمای کاربران می‌توان از نمایش پیش‌نمونه‌ی واسط کاربر بازاستفاده کرد. از آنجا که نمایش واسط کاربر جزء ضروری راهنمای کاربران است، ساختن نمایش پیش‌نمونه‌ی واسط کاربر الزاماً تلاش بیشتری را در بر نخواهد داشت. در واقع، تولید نمایش پیش‌نمونه‌ی واسط کاربر در زمان تبیین مورد کاربردهای گسترده می‌تواند تلاش لازم برای توسعه را کاهش دهد چون (۱) بازخورد زودهنگام کاربر جلوی دوباره‌کاری را می‌گیرد، (۲) تلاش لازم برای آماده‌سازی راهنمای کاربر کمتر می‌شود، و (۳) تلاش لازم برای موارد آزمون کاهش می‌یابد. تبیین مورد کاربرد گسترده که همراه با نمایش پیش‌نمونه‌ی واسط کاربر باشد، تولید موارد آزمون را آسان می‌کند چون واسط‌های کاربر گرافیکی مورد انتظار را به تصویر می‌کشد. این‌ها همه بخشی از توصیفات مورد آزمون هستند. برای اطمینان از این‌که سیستم پنجره‌ها، جعبه‌های گفت‌وگو، و نتایج محاسبات را به‌درستی نشان می‌دهد، لازم است که مورد کاربردهایی تولید و اجرا شوند. آزمون در فصل ۲۰ تشریح شده است.

۸-۶ رسیدگی به استثناها[1] را نشان ندهید

مورد کاربرد گسترده مسیر عادی تعامل بین کنشگر و سیستم را تبیین می‌کند. بنابه ملاحظات ذیل، در این تبیین نباید موارد استثنایی کار با این موارد را دخیل نمود.

- موارد استثنا و جریان‌های جایگزین با هم فرق دارند. جریان‌های جایگزین در واقع جریان‌های کاری رایج در کسب‌وکار هستند ولی موارد استثنا موقعیت‌های غیرعادی می‌باشند که با آن‌ها باید در سطح پایین‌تر، یعنی سطح پیاده‌سازی، برخورد کرد، نه در سطح تبیین مورد کاربرد.
- گرچه موارد استثنا موقعیت‌های غیرمعمول را نشان می‌دهند، ولی تعداد آن‌ها برای یک سیستم ممکن است خیلی زیاد باشد. برای مثال، ممکن است شبکه قطع باشد، تاریخ اعتبار حساب گذشته باشد، برق قطع باشد، کامپیوتر ویروسی شده باشد، یا سایر مشکلاتی که می‌تواند رخ دهد. مشخص کردن موارد استثنایی در مورد کاربرد گسترده امر مطلوبی نیست چون در این‌صورت، مشخصات پیچیده به نظر خواهد رسید و درک جریان معمول تعامل خیلی دشوار خواهد بود.

---

[1] Exception Handling

۸-۷ پیش‌شرط و پس‌شرط برای مورد کاربردها

مورد کاربردها فرایندهای کسب‌وکاری هستند. قبل از این‌که یک فرایند کسب‌وکار بتواند اجرا شود، ممکن است لازم باشد شرایط معینی برقرار باشد. برای مثال، در برخی سیستم‌ها قبل از این‌که کاربر بتواند به سیستم وارد شود لازم است حساب کاربر فعال شده باشد. قبل از این‌که دانشجو بتواند درسی را اخذ نماید باید تسویه حساب کرده باشد. شرایطی از این قبیل را با عنوان پیش‌شرط[۱] می‌شناسیم. این‌ها بیان‌گر شرایطی از سیستم هستند که باید برای اجرای موفق یک مورد کاربرد برقرار باشند و چنین شرایطی اگر به‌صراحت بیان نشوند، بدیهی و آشکار نخواهند بود. به‌همین ترتیب، پس‌شرط[۲] برای یک مورد کاربرد بیان‌گر شرایطی از وضعیت سیستم است که بعد از اجرای مورد کاربرد برقرار خواهد بود و چنان‌چه به‌صراحت بیان نشوند، بدیهی و آشکار نخواهند بود.

شکل ۸-۹ پیش‌شرط یک مورد کاربرد را درباره‌ی وضعیت یک شئ مشخص که توسط آن مورد کاربرد پردازش خواهد شد نشان می‌دهد. در این مثال، پیش‌شرط می‌گوید که دانشجو باید در دروسی که قصد ثبت‌نام دارد، حسابش روشن شده باشد.

توجه کنید که بین پیش‌شرط و نمایش اولیه‌ی سیستم که در قدم ۰ مشخص شده است، تفاوت وجود دارد. نمایش اولیه‌ی سیستم مشخص می‌کند که واسط کاربر یا چیزی که کنشگر قبل ازشروع مورد کاربرد باید ببیند چیست؟ این نمایش، شرطی از وضعیت سیستم نیست که واضح نبوده و برای اجرای موفق مورد کاربرد نیاز به تبیین آن باشد. به‌طور مشابه، بین پس‌شرط و نمایش سیستم که در بند TUCEW یا آخرین ردیف مورد کاربرد مشخص شده‌اند، تفاوت وجود دارد. پس‌شرط بیان‌گر وضعیت حاصل از سیستم است که اگر به صراحت مشخص نشود، آشکار و بدیهی نخواهد بود. نمایش سیستم، آنچه که کاربر خواهد دید را نشان می‌دهد. پیش‌شرط‌ها و پس‌شرط‌ها همواره موردنیاز نیستند ولی نمایش اولیه و نمایش نهایی سیستم برای تمام مورد کاربردهای توسعه‌یافته لازمند.

UC2: ثبت‌نام یک درس

پیش‌شرط: دانشجو باید قبل از ثبت‌نام در یک درس، تسویه حساب کرده باشد.	
کنشگر: دانشجو	سیستم: مدیر مطالعات
	۰- سیستم صفحه خوشامدگویی را نشان می‌دهد.
۱- TUCBW دانشجو روی پیوند Enroll در صفحه خوشامدگویی کلیک می‌کند.	۲- سیستم صفحه‌ی Select Semester را نمایش می‌دهد.
۳- دانشجو نیم‌سال را انتخاب و روی دکمه‌ی OK کلیک می‌کند.	۴- سیستم درس‌هایی که دانشجو مجاز به انتخاب می‌باشد را نشان می‌دهد.
۵- دانشجو درس‌ها را انتخاب و روی دکمه‌ی Submit کلیک می‌کند.	۶- در صورت موفقیت‌آمیز بودن ثبت‌نام، سیستم پیغام "Enrollment is successful" و در غیراین‌صورت یک پیغام خطا نشان می‌دهد.
۷- TUCEW دانشجو پیغام "Enrollment is successful" یا پیغام خطا را مشاهده می‌کند.	

شکل ۸-۹: پیش‌شرط مورد کاربرد برای وضعیت یک شئ

[۱] Precondition

[۲] Postcondition

۸-۸ شمول مورد کاربردهای دیگر

مورد کاربردها فرایندهای کسب‌وکاری هستند. این‌که یک فرایند کسب‌وکاری شامل چند فرایند کسب‌وکاری دیگر، به‌عنوان زیرفرایند، باشد امری متداول است. برای مثال، وقتی کاربری جدید اولین بار وارد سیستم می‌شود، اغلب سیستم‌ها از کاربر می‌خواهند که رمز عبور موقتی خود را تغییر دهد. در این حالت، مورد کاربرد ورود به‌طور شرطی شامل مورد کاربرد تجدید رمز عبور (Reset Password) خواهد شد. شکل ۸-۱۰ نشان می‌دهد که چگونه این مورد کاربرد گسترده را باید مشخص کرد. در این شکل، TUCCW معادل جمله‌ی «این مورد کاربرد ادامه می‌یابد با» می‌باشد. (مترجم: TUCCW برگرفته از سرکلمات جمله‌ی انگلیسی "This Use Case Continues With" است).

کنشگر: کاربر	سیستم: WebApp
	۰- WebApp صفحه‌ی خانگی را نشان می‌دهد.
۱- TUCBW کاربر روی پیوند Login در صفحه‌ی خانگی کلیک می‌کند.	۲- WebApp صفحه‌ی ورود را نمایش می‌دهد.
۳- کاربر نام‌کاربری و کلمه‌عبور را وارد و روی دکمه‌ی Login کلیک می‌کند.	۴ الف) اگر این اولین بار ورود کاربر است، TUCCW مورد کاربرد Reset Password ۴ ب) در غیراین‌صورت، WebApp صفحه خوشامدگویی را نشان می‌دهد.
۵- TUCEW کاربر صفحه خوشامدگویی را مشاهده می‌کند.	

شکل ۸-۱۰: مورد کاربرد گسترده‌ای که شامل مورد کاربرد دیگری است

۸-۹ ادامه دادن با مورد کاربردهای دیگر

تکنیک معرفی شده در بخش قبلی را می‌توان برای مشخص کردن جریان‌های جایگزین که ممکن است به دیگر مورد کاربردها انشعاب بزنند به‌کار برد. برای مثال، یک سرمایه‌گذار ممکن است راه‌های مختلفی را برای انتقال وجه به‌کار ببرد. مورد کاربرد انتقال وجه به‌صورت برخط[۱] به سرمایه‌گذار اجازه می‌دهد که وجوه را از یک حساب به حساب دیگر در همان موسسه‌ی کارگزاری منتقل نماید. مورد کاربرد انتقال وجه از طریق چک[۲] به سرمایه‌گذار اجازه می‌دهد که برای انتقال وجه از یک حساب بانکی از چک استفاده نماید. مورد کاربرد انتقال وجه از طریق فرم انتقالی[۳] به سرمایه‌گذار اجازه می‌دهد فرمی را تکمیل، چاپ، امضا، و پست نماید تا وجه از یک موسسه کارگزاری به موسسه‌ی دیگری منتقل شود. تا جایی که به مدل‌سازی تعامل کنشگر-سیستم مربوط می‌شود، هر سه مورد کاربرد انتقال وجه به‌صورت برخط، انتقال وجه از طریق چک، و انتقال وجه از طریق فرم انتقالی موارد خاصی از انتقال وجه تلقی می‌شوند. بنابراین، مورد کاربرد انتقال وجه می‌تواند در قالب یکی از این سه مورد کاربرد محقق شود. این وضعیت را در شکل ۸-۱۱ مشاهده می‌کنید.

تبیین مورد کاربرد گسترده پیش‌درآمدی برای سه روش مختلف انتقال وجه می‌باشد. بعد از این‌که سرمایه‌گذار روی پیوند انتقال وجه کلیک کرد، سیستم روش‌های مختلف انتقال وجه را به وی نشان می‌دهد. به‌جای این‌که مستقیماً به‌سراغ هر کدام

[۱] Transfer Funds Online
[۲] Transfer Funds With a Check
[۳] Transfer Funds With a Transfer Form

UC3: انتقال وجه

کنشگر: سرمایه‌گذار	سیستم: مدیر سرمایه‌گذاری
	۵- سیستم صفحه‌ی خانگی Fund Management را نشان می‌دهد.
۱- TUCBW سرمایه‌گذار روی پیوند Transfer Funds در صفحه‌ی خانگی Fund Management کلیک می‌کند.	۲- سیستم صفحه‌ی Fund Transfer را با گزینه‌های زیر نمایش می‌دهد: ۲ الف) Transfer Funds Online ۲ ب) Transfer Funds with a Check ۲ ج) Transfer Funds with a Transfer From
۳- TUCCW یکی از مورد کاربردهای زیر: ۳ الف) UC4: Transfer Funds Online ۳ ب) UC5: Transfer Funds with a Check ۳ ج) UC6: Transfer Funds with a Transfer From	

شکل ۸-۱۱: مشخص کردن جریان‌های جایگزین به مورد کاربردهای دیگر

از این انتخاب‌ها که سرمایه‌گذار می‌تواند انتخاب کند برویم، مورد کاربرد گسترده شکل ۸-۱۱ در گام ۳ می‌گوید که این مورد کاربرد ادامه می‌یابد با یکی از مورد کاربردهای بعدی و آن‌ها را فهرست می‌کند. به‌منظور نشان دادن این‌که چگونه این مورد کاربرد می‌تواند با یکی از مورد کاربردهای انتقال وجه ادامه یابد، شکل ۸-۱۲ مورد کاربرد گسترده را برای مورد کاربرد انتقال وجه از طریق چک نشان داده است.

UC5: انتقال وجه با چک

کنشگر: سرمایه‌گذار	سیستم: مدیر سرمایه‌گذاری
	۵- سیستم صفحه‌ی Fund Transfer را با گزینه‌های مختلف نشان می‌دهد.
۱- TUCBW سرمایه‌گذار روی گزینه‌ی Transfer Funds with a Check کلیک می‌کند.	۲- سیستم یک فرم Fund Transfer را نمایش می‌دهد که نوع واریز، مبلغ انتقالی، واحد ارزی، شماره مسیریابی بانکی، شماره حساب بانکی، و درصورت لزوم شماره چک را درخواست می‌نماید.
۳- سرمایه‌گذار اطلاعات را وارد و روی دکمه‌ی Continue کلیک می‌کند.	۴- سیستم تەبرگ Fund Transfer را به سرمایه‌گذار نمایش می‌دهد تا آن را چاپ کرده و همراه چک ارسال نماید.
۵- TUCEW سرمایه‌گذار تەبرگ Fund Transfer را چاپ کرده و همراه چک ارسال می‌نماید.	

شکل ۸-۱۲: انتقال وجه از طریق چک، مورد کاربرد دیگری را ادامه می‌دهد.

۸-۱۰ مشکلات رایج

در این بخش با کمک شکل ۸-۱۳ سعی می‌شود مشکلاتی که معمولاً در رابطه با تبیین مورد کاربردهای گسترده مشاهده می‌شوند و روش رفع این مشکلات معرفی شوند. نخست این‌که، نمایش اولیه‌ی سیستم مشخص نشده باشد. عبارت TUCBW مشخص نمی‌کند که دکمه‌ی ثبت‌نام (Register) در کجا یا کدام صفحه قرار دارد. این نکته در یادداشت (الف) از شکل ۸-۱۳ نشان داده شده است. بدون این اطلاعات، کنشگر نمی‌داند دکمه را از کجا پیدا کند و برنامه‌نویس هم نمی‌داند دکمه را کجا قرار دهد. بنابراین، لازم است به‌صراحت معلوم شود که مورد کاربرد از کجا شروع می‌شود.

همان‌گونه که در یادداشت‌های (ب) و (هـ) اشاره شده است، تعامل بین کنشگر و سیستم باید به صورت جریان مداومی از درخواست‌ها و پاسخ‌های تعاملی بین کنشگر و سیستم باشد، نه این‌که به‌صورت نوبه‌ای اتفاق بیافتد. یادداشت (ج) می‌گوید که ورودی کنشگر و کنش کنشگر باید در یک گام انجام شود، نه در دو گام. یادداشت (ب) می‌گوید که برای ورودی کنشگر و کنش کنشگر واکنش یا پاسخ سیستم مشخص نشده است. کنشگر نمی‌داند که آیا این گام با موفقیت اجرا شده و لذا ممکن است تا ابد منتظر پاسخ سیستم باشد. گام‌های ۵ و ۶ پردازش‌های پس‌زمینه را نشان می‌دهد، که در تبیین مورد کاربرد گسترده توصیه نمی‌شود. پردازش پس‌زمینه باید در زمان مدل‌سازی تعامل اشیا توصیف شود (فصل ۹ را ببینید). از سوی دیگر، گام‌ها نمایش‌های مورد انتظار از سیستم را مشخص نمی‌کنند. در نهایت، همان‌گونه که در یادداشت (و) نشان داده شده است، مورد کاربرد گسترده عبارت TUCEW را مشخص نکرده است. عبارت TUCEW باید بگوید که مورد کاربرد چه کاری برای کنشگر انجام می‌دهد. برای رفع این مشکل، باید گام ۴ را با گام ۳ ادغام کرد، گام‌های ۵ و ۶ را حذف نمود، گام ۷ را به گام ۴ تغییر داد، و یک گام TUCEW اضافه کرد به صورتی که پیغام «ثبت‌نام با موفقیت انجام شد» را به دانشجو نشان دهد.



شکل ۸-۱۳: تبیین مورد کاربرد گسترده مشکل‌دار

سایر مشکلات رایج به شرح زیر می‌باشند.

- مشخص کردن رسیدگی به استثناها در مورد کاربردهای گسترده.
- فراموش کردن ورودی کنشگر یا کنش کنشگر، چنان‌چه توسط خانه‌های خالی در ستون راست شکل ۸-۱۳ نشان داده شده است.

- پیچیده کردن غیرضروری مشخصات، همان‌گونه که در شکل ۸-۱۴ نشان داده شده است. این شکل را می‌توان با ترکیب گام‌های ۲ تا ۱۵ در قالب یک گام به دست آورد، مانند «۳- کاربر نام، نام خانوادگی، رایانامه، شماره تلفن، نام کاربری، رمز عبور، و تکرار رمز عبور را وارد فرم ثبت‌نام کرده و روی دکمه ی ارسال کلیک می‌کند.»
- تبیین مبهم ورودی کنشگر و کنش کنشگر.
- عدم به‌کارگیری جملات حال ساده و سوم مشخص.

UC1 : Register a New User

Actor: User	System: NTSS
	0. NTSS displays the homepage.
1. TUCBW user clicks the Register link.	2. NTSS prompts the user to enter first name.
3. User enters first name.	4. NTSS prompts the user to enter last name.
5. User enters last name.	6. NTSS prompts the user to enter email address.
7. User enters email address.	8. NTSS prompts the user to enter phone number.
9. User enters phone number.	10. NTSS prompts the user to enter user name.
11. User enters user name.	12. NTSS prompts the user to enter a password.
13. User enters password.	14. NTSS prompts the user to re-enter a password.
15. User re-enter password.	16. NTSS verifies the information entered and a. displays the "Account is successfully created" message; or b. prompts the user to re-enter the incorrectly entered information.
17. TUCEW user sees the "Account is successfully created." message.	

شکل ۸-۱۴: تعامل کنشگر سیستم که بدون دلیل پیچیده شده است

۸-۱۱ اعمال اصول چابکی

رهنمون ۸-۱

ساده و ساده‌لوح بودن را رعایت کنید.

هر مورد کاربرد گسترده باید تنها شامل تعداد کمی گام باشد. به علاوه، هر گام نیز باید ساده، واضح، و سرراست باشد. چنان‌چه یک مورد کاربرد گسترده بیش از پنج یا شش گام داشته باشد، مانند آن‌چه در شکل ۸-۱۴ نشان داده شده است، دو احتمال وجود دارد. یک احتمال این است که اعمال سطح پائین به‌عنوان گام‌های فرایند در نظر گرفته شده‌اند. احتمال دیگر این است که آن مورد کاربرد در واقع الحاق دو مورد کاربرد باشد. در این حالت، باید مورد کاربردهای گسترده را به دقت بررسی

کرد تا اعمال مناسب جهت بهبود آن را مشخص نمود. بدین معنی که، مورد کاربرد گسترده باید اعمال سطح پایین را مخفی کرده و تنها گام‌های فرایند کسب‌وکار را نشان دهد. برای مثال، گام‌های ۳ الی ۱۵ را باید در قالب یک گام ادغام نمود. اگر مورد کاربرد، دو یا چند فرایند کسب‌وکاری متوالی را ترکیب کرده باشد، این مورد کاربرد باید تجزیه شود. برای مثال، گاهی خرید جنس[۱] و انجام پرداخت[۲] باهم ادغام می‌شوند، درحالی‌که باید دو مورد کاربرد مجزا باشند.

رهنمون ۸-۲

از جریان‌های جایگزین پرهیز کنید.

یک مورد کاربرد گسترده که جریان‌های جایگزین زیادی داشته باشد، پیچیدگی تعامل کنشگر-سیستم و نیز منحنی یادگیری کاربران را افزایش می‌دهد. لذا درک، پیاده‌سازی، آزمون، و نگهداری آن دشوارتر خواهد بود. گاهی ممکن است یک مورد کاربرد یک فرایند کسب‌وکاری نبوده بلکه ترکیب چندین فرایند کسب‌وکاری باشد. جریان‌های جایگزین در واقع بازتاب انتخاب فرایندهای کسب‌وکاری هستند. برای تعیین این موضوع، سعی کنید یک توصیف کارکردی یک خطی از مورد کاربرد تولید کنید. چنان‌چه توصیف خیلی طولانی شد، یا باید فهرستی از کارهایی که می‌کند تهیه شود، این احتمال می‌رود که فرایندهای کسب‌وکاری متعددی به مورد کاربرد واگذار شده باشد. مورد کاربرد را برحسب کاری که انجام می‌دهد، یعنی فرایند کسب‌وکاری که سعی در انجام آن دارد، بشکنید.

رهنمون ۸-۳

پردازش‌های پس‌زمینه و مدیریت استثناها را در مورد کاربرد گسترده نیاورید.

پردازش‌های پس‌زمینه و مدیریت استثناها جزئی از رفتار تعاملی کنشگر-سیستم نیستند. لذا، نباید در مورد کاربرد گسترده ظاهر شوند. پردازش‌های پس‌زمینه‌ای را برای مدل‌سازی تعاملی اشیا، و مدیریت استثناها را برای پیاده‌سازی بگذارید.

رهنمون ۸-۴

به حد کفایت خوب بودن کافی است. به‌سرعت به طرف پیاده‌سازی نرم‌افزار خوب حرکت کنید.

هیچ طراحی بهینه‌ای برای رفتار تعاملی کنشگر- سیستم وجود ندارد. حرف آخر را مشتریان و کاربران می‌زنند، و آن‌ها هم تا وقتی با نرم‌افزار تجربه‌ی کار عملی نداشته باشند نمی‌توانند تصمیمی بگیرند. بنابراین، مهم است که سریعاً به سوی پیاده‌سازی حرکت شود زیرا این‌کار به کاربران و مشتریان امکان می‌دهد که طراحی را با تجربه‌ی واقعی مورد ارزیابی قرار دهند. این همان معنی «به حد کفایت خوب بودن کافی است» را می‌رساند.

خلاصه

این فصل چگونگی استفاده از مورد کاربرد گسترده در مدل‌سازی تعاملی کنشگر سیستم و نیز گام‌هایی برای ساخت مورد کاربردهای گسترده را معرفی می‌کند. همچنین می‌گوید که چگونه از پیش‌نمونه‌های واسط کاربری استفاده می‌شود تا دریافت

[۱] Purchase Item

[۲] Make Payment

بازخورد از کاربران را آسان نماید. برای بیان مفروضات ضمنی که برای اجرای موفقیت‌آمیز مورد کاربردها لازمند، از پیش‌شرط استفاده می‌شود. به‌طور مشابه، از پس‌شرط‌ها برای بیان وضعیت نتیجه استفاده می‌شود. این فصل نشان می‌دهد که چگونه مورد کاربردهای گسترده را برای موقعیت‌های مختلف مانند جریان‌های جایگزین، شمول یک مورد کاربرد در مورد کاربردهای دیگر، و ادامه‌ی یک مورد کاربرد در دیگری مشخص کنیم. مسائل رایج در خصوص تبیین مورد کاربردهای گسترده مورد بحث قرار گرفتند تا به مبتدیان کمک شود که از چنین مسائلی پرهیز شود.

مورد کاربردهای گسترده پردازش‌های پیش‌زمینه‌ای[1] مربوط به فرایندهای کسب‌وکار را، آن‌گونه که توسط کنشگران و سیستم انجام می‌شوند، مشخص می‌نمایند. پردازش‌های پس‌زمینه ای فرایندهای کسب‌وکار را عمداً مطرح نکرده و برای فصل ۹ (مدل‌سازی تعامل شئ) گذاشتیم.

برای مطالعه‌ی بیش‌تر

مدل‌سازی مورد کاربرد در [۲۹، ۳۰، ۴۹، ۹۰، ۱۰۴] معرفی شده است. به‌خصوص، از یک قالب متنی برای توصیف مورد کاربردهای گسترده استفاده گردیده است. اما، در نسخه‌های اولیه از قالب جدولی استفاده می‌شد [۱۰۴].

سوال‌های مروری فصل

۱. مورد کاربرد گسترده چیست و چرا استفاده می‌شود؟
۲. ستون‌های چپ و راست در مورد کاربرد گسترده چه چیزی را مشخص می‌کنند؟
۳. سه سطح در تبیین مورد کاربرد کدامند و چگونه به یکدیگر ارتباط دارند.
۴. عبارت‌های TUCBW و TUCEW در کجای جدول مورد کاربرد گسترده قرار می‌گیرند و چرا؟
۵. آیا بین عبارت TUCBW و TUCEW درج اقلام خالی مجاز می‌باشد؟ چرا؟
۶. آیا پردازش پس‌زمینه باید در مورد کاربرد گسترده مشخص شده باشد؟ چرا؟
۷. جریان‌های پردازشی جایگزین در مورد کاربرد گسترده چگونه مشخص می‌کنند؟ حسن انجام این کار در چیست؟

[1] Foreground

تمرین‌ها

۸-۱ برای مورد کاربردهای واریز وجه، برداشت وجه، بررسی موجودی، و انتقال وجه از یک برنامه دستگاه خودپرداز، مورد کاربردهای گسترده بنویسید.

۸-۲ برای مورد کاربرد ویرایش گزارش از یک واژه‌پرداز نوعی، مورد کاربرد گسترده بنویسید.

۸-۳ برای مورد کاربرد انجام پرداخت در یک کسب‌وکار کوچک، مورد کاربرد گسترده بنویسید. فرض کنید مشتری می‌تواند به سه طریق پرداخت انجام دهد: پرداخت با چک، پرداخت با کارت اعتباری، و پرداخت نقدی.

۸-۴ برای مورد کاربردهای جست‌وجوی خودرو، رزرو خودرو، ویرایش رزرو، و لغو رزرو در یک سیستم برخط اجاره خودرو که در پیوست ت-۱ تشریح شده است، مورد کاربردهای گسترده بنویسید.

۸-۵ در پیوست ت-۲ در مورد کسب‌وکار خدمات نمایشگاه‌های ملی توضیح داده شده است. در تمرین‌های ۴-۶ و ۷-۳ شما نیازمندی‌ها و مورد کاربردهای استنتاج شده برای این کاربرد را تهیه کردید. حالا مورد کاربردهای گسترده را بنویسید.

۸-۶ چند نیازمندی برای یک ویراستار نمودار توالی[۱] تنظیم کنید. از این نیازمندی‌ها، مورد کاربردهایی را استنتاج کرده و برای هر کدام مورد کاربرد گسترده را نیز بنویسید.

---

[۱] Sequence Diagram Editor

۹

مدل‌سازی تعامل شئ

نکات کلیدی

- مدل‌سازی تعامل شئ به تیم توسعه کمک می‌کند فرایندهای کسب‌وکاری موجود را درک نموده و رفتارهای تعامل اشیا را به‌گونه‌ای طراحی کند که باعث بهبود کسب‌وکار شود.
- مدل‌سازی تعامل کنشگر- سیستم و مدل‌سازی تعامل شئ به ترتیب با پردازش پیش‌زمینه و پس‌زمینه‌ی یک مورد کاربرد مرتبط می‌شوند.

مورد کاربرد یک فرایند کسب‌وکار است که قرار است توسط سیستم نرم‌افزاری در دست ساخت خودکارسازی شود. به بیان دقیق‌تر، فرایند کسب‌وکار باید به‌طور مشترک توسط کنشگر و سیستم در دو قسمت انجام شود: قسمت پیش‌زمینه‌ای تعامل کنشگر- سیستم و قسمت پس‌زمینه‌ای پردازش سیستم. در فصل ۸ دیدیم که چگونه با استفاده از مورد کاربردهای گسترده، تعامل کنشگر-سیستم مشخص می‌شود. فصل جاری به مدل‌سازی و تبیین این موضوع اختصاص دارد که چگونه اشیا با یکدیگر تعامل می‌کنند تا فرایندهای کسب‌وکار را به انجام رسانند. در این فصل، موارد زیر را فراخواهید گرفت:

- تمرکز مدل‌سازی تعامل شئ
- نمودار توالی UML
- نمودار توالی تحلیل و نمودار توالی طراحی
- گام‌های مدل‌سازی تعامل شئ

۹-۱ مدل‌سازی تعامل شئ چیست؟

در نگرش شئ گرا، اشیا بلوک‌های سازنده‌ی اصلی هستند. در این نگرش، دنیای واقعی و نیز سیستم مورد نظر، متشکل از اشیایی مرتبط باهم تصور می‌شوند که با هم در تعامل هستند. اشیا در قالب روابط ارث‌بری، تجمیع، و انجمنی با یکدیگر ارتباط دارند. تعامل بین اشیا نیز از طریق درخواست سرویس از دیگر اشیا یا اجازه‌ی انجام اعمال به دیگر اشیا میسر می‌گردد. بدیهی

است که اشیا نباید به‌طور دلخواه با یکدیگر در تعامل باشند، بلکه باید به‌گونه‌ای باهم تعامل کنند که فرایندهای کسب‌وکاری مورد کاربردها را به انجام رسانند. مانند آن‌چه که در توسعه‌ی نرم‌افزار مطرح است، دو مسئله باید در نظر گرفته شود:

۱. **مسئله‌ی تحلیل:** درفرایندهای کسب‌وکاری موجود، و احتمالاً دستی، اشیا چگونه با یکدیگر تعامل می‌کنند تا وظایف کسب‌وکاری را انجام دهند؟
۲. **مسئله‌ی طراحی:** در سیستم پیشنهادی، اشیا چگونه با یکدیگر تعامل کنند تا فرایندهای کسب‌وکار را بهبود بخشند؟

مسئله‌ی تحلیل از این جهت مهم است که ممکن است تیم توسعه با فرایندهای کسب‌وکار جاری آشنا نباشد. در این حالت، ممکن است نیاز باشد که تیم توسعه اطلاعاتی جمع‌آوری نماید و مدل‌هایی را بسازد که به فهم فرایندهای کسب‌وکار موجود کمک نمایند. به‌علاوه، ممکن است تیم توسعه برای شناسایی مشکلات و ضعف‌های فرایندهای کسب‌وکار جاری نیاز به مدل‌های تعامل شئ نیز داشته باشد. بنابراین، ساخت مدل‌های تعامل شئ برای تحلیل مفید خواهد بود.

مسئله‌ی طراحی نیز از این جهت مهم است که فرایندهای کسب‌وکاری موجود سال‌ها پیش طراحی شده‌اند. این فرایندها شاید دستی و یا حتی از رده خارج باشند، زیرا ممکن است فرایند به‌طور قابل ملاحظه‌ای گسترش یافته باشد، یا محیط کسب‌وکاری به شدت تغییر کرده باشد، و احتمال دارد فناوری به‌طرز مشهودی پیشرفت کرده باشد. فرایندهای جاری ممکن است نیاز به طراحی مجدد داشته باشند. بنابراین، لازم است مدل‌های تعامل شئ به‌منظور طراحی ساخته شوند. این ما را به تعریف زیر می‌رساند.

تعریف ۹-۱

مدل‌سازی تعامل شئ (OIM) عبارت است از فرایندی که:

۱. کمک می‌کند بفهمیم در سیستم جاری یا در فرایندهای کسب‌وکار جاری چگونه اشیا با هم تعامل می‌کنند.
۲. کمک می‌کند مشکلات و محدودیت‌های سیستم جاری یا فرایندهای کسب‌وکار جاری را شناسایی کنیم.
۳. طراحی و تعیین می‌کند که در سیستم نرم‌افزاری پیشنهادی چگونه اشیا با هم تعامل داشته باشند تا مورد کاربردها را به انجام برسانند.

در این تعریف، دو هدف اول مسائل تحلیل و هدف آخر مسئله‌ی طراحی را مدّ نظر قرار دارند. به بیان دقیق‌تر، OIM مسائل تحلیل و طراحی ذیل را مدّ نظر قرار می‌دهد:

۱. **مدل‌سازی و تحلیل:** این موضوعی است که دو هدف اول بیان شده در بالا را مدّ نظر قرار می‌دهد.
۲. **طراحی تعامل شئ:** این موضوعی است که آخرین هدف را مدّ نظر قرار می‌دهد، یعنی، طراحی رفتارهای تعاملی اشیا به‌منظور حل مسائل طراحی زیر:

الف) **طراحی رفتار تعامل شئ:** در این‌جا مسئله، طراحی دنباله‌ای از پیغام‌های بین اشیا است یا این‌که اشیا چگونه با هم همکاری کنند تا فرایندهای کسب‌وکاری جاری را بهبود بخشند.

ب) **برقراری واسط اشیا:** مسئله در این‌جا عبارت است از طراحی امضاها[1] و گونه‌های برگشتی برای توابع اشیا.

[1] Signatures

ج) **طراحی بهتر**[۱]: آن‌چه در این‌جا مدّ نظر است، به‌کار بردن الگوهای طراحی[۲] است به‌گونه‌ای که یک طراحی تولید شود که نیازهای خاص را برآورده نموده و ویژگی‌های مطلوب از قبیل جفت‌شدگی کم، چسبندگی زیاد، تخصیص درست مسئولیت‌ها به اشیا، و تطبیق آسان با تغییر در نیازمندی‌ها و محیط عملیاتی را داشته باشد.

براساس اصل جداسازی دغدغه‌ها[۳]، این فصل به‌طور عمد به مدل‌سازی، تحلیل، و طراحی رفتار تعامل اشیا محدود گردیده است. اِعمال الگوهای طراحی به‌منظور تولید طراحی‌های بهتر در فصول بعدی مطرح شده‌اند.

۹-۲ نمودار توالی UML

نمودارهای توالی[۴] UML به‌طور گسترده برای مدل‌سازی تعامل شئ استفاده می‌شوند. نمودار UML دیگری که برای مدل‌سازی تعامل شئ استفاده می‌شود نمودار ارتباط[۵] نام دارد. به دلیل این‌که منظور از این دو نمودار مدل‌سازی تعامل شئ می‌باشد، نمودار توالی و نمودار ارتباط را نمودارهای تعامل[۶] می‌نامند. این بخش به معرفی نمودار توالی می‌پردازد که برای معرفی گام‌های مدل‌سازی تعامل شئ کاربرد دارد. هر نمودار توالی UML، یا نمودار توالی برای اختصار، تعامل اشیا را به‌صورت دنباله‌ای از پیغام‌های مرتب شده بر حسب زمان، که بین اشیا ارسال و دریافت می‌شوند، نمایش می‌دهد. برخلاف نمودار توالی، نمودار ارتباط، برای نمایش این‌که چه پیغام‌هایی بین اشیا رد و بدل می‌شوند مناسب است. نمودار توالی و نمودار ارتباط از نظر معنا معادل هستند. در این کتاب فقط از نمودار توالی استفاده شده است.

۹-۲-۱- انگاره‌ها و نمادگان

شکل ۹-۱ انگاره‌ها و نمادگان متداول مربوط به نمودارهای توالی را همراه با معانی مربوطه نشان می‌دهد. اتصال بین نمادها، یا همان نحو، در ستون آخر شکل توصیف شده است.

۹-۲-۲- نمایش نمونه‌های یک کلاس

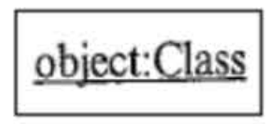
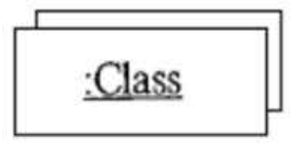
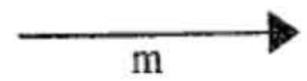
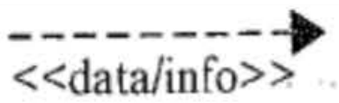
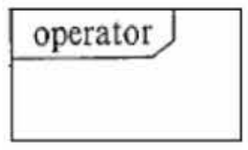
مدل‌سازی تعامل شئ نیاز دارد که به اشیا ارجاع کرده و آن‌ها را اجرا کند. برای مثال، نیاز دارد یک شئ ایجاد کند، شئ را تغییر دهد، و شئ را درون پایگاه‌داده ذخیره نماید. در این مثال، شئ باید در سه جای مختلف نمودار توالی ظاهر شود. بدین معنی که، به یک نمونه‌ی با نام از شئ نیاز است تا بتوان با استفاده از نام به آن دستیابی پیدا کرد. به‌عنوان مثالی دیگر، یک ویرایشگر نمودار کلاس باید بتواند عناصر نمودار را درون یک اجتماع از عناصر نمودار درج نماید. بنابراین، به نمادی برای نمایش یک اجتماع از اشیا نیاز است. شکل ۹-۲ نشان می‌دهد که چگونه نمونه‌های مختلف از یک شئ در UML نمایش داده می‌شوند.

اولین مورد، یک نمونه‌ی بی‌نام از یک کلاس را نشان می‌دهد. از این مورد در جاهایی استفاده می‌شود که نیاز نیست به آن نمونه در جای دیگری از همان نمودار توالی ارجاع شود. برای مثال، چنان‌چه نمونه به‌عنوان پارامتر در فراخوانی یک تابع یا به‌عنوان یک مقدار برگشتی ارسال نشده است، در این صورت نیازی به ارجاع با آن نمونه در هیچ جای دیگر از نمودار نیست.

مورد دوم یک نمونه‌ی با نام از یک کلاس بی‌نام را نشان می‌دهد. از این مورد وقتی استفاده می‌شود که هنوز در مرحله‌ی طراحی راجع به نام کلاس‌ها تصمیم‌گیری نشده باشد، یا این‌که کلاس دقیق فقط در زمان اجرا مشخص می‌شود. در این گونه موارد، نمونه باید نام‌گذاری شده باشد؛ و گرنه دسترسی به آن غیرممکن خواهد بود. مورد سوم، یک نمونه‌ی با نام از یک کلاس با نام را نشان

---

[۱] Design for better
[۲] Design Patterns
[۳] Separation of Concerns
[۴] Sequence Diagram
[۵] Communication Diagram
[۶] Interaction Diagram

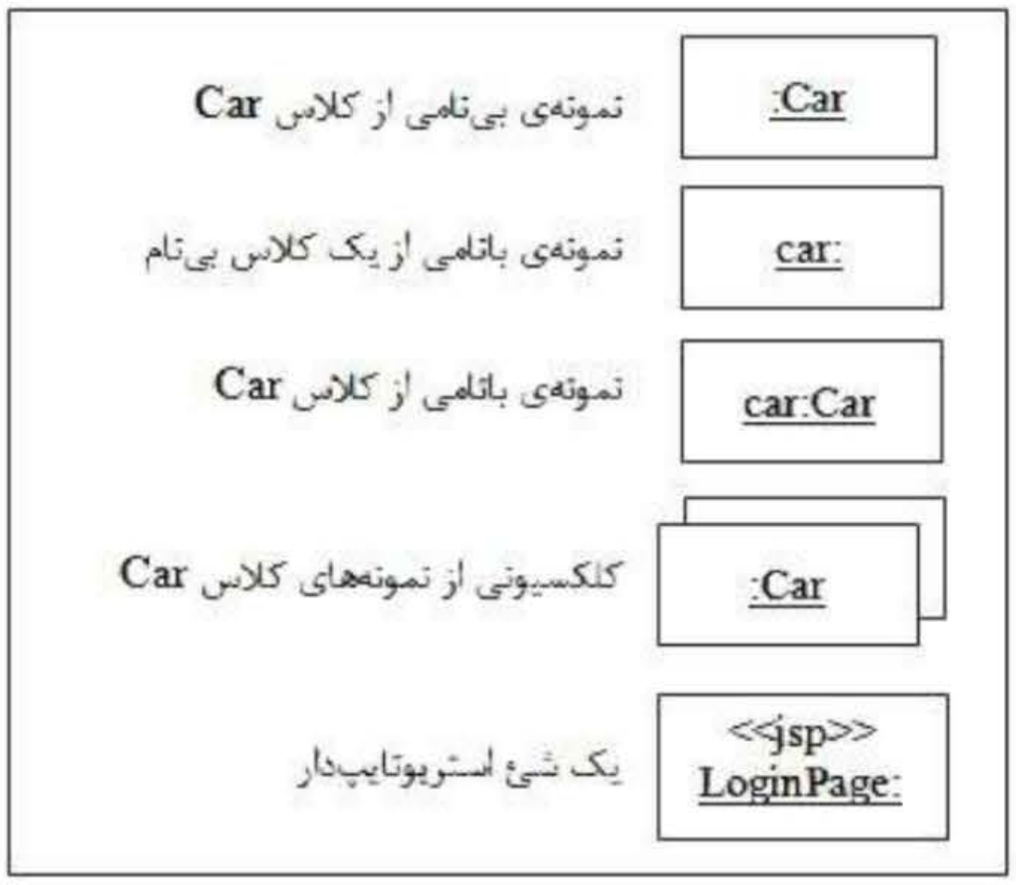
انگاره	نمادگان	معنا	ارتباط
عامل		یک نقش به واسطه‌ی مجموعه‌ای از موجودیت‌ها یا ذینفعان که خارج از سیستم هستند و با آن تعامل دارند، اجرا می‌شود.	از طریق یک پیام استریوتایپ به/ از اجرای متد متصل می‌شود.
شئ		یک شی از یک کلاس مشخص	از طریق یک بخش از خط عمر به اجرای متد متصل می‌شود.
کلکسیون		کلکسیونی از اشیای یک کلاس مشخص. این کلکسیون در بالای خط عمر قرار می‌گیرد. زیرخط، یک شئ یا کلکسیونی از اشیا را نشان می‌دهد. نام شئ قبل از دو نقطه و نام کلاس بعد از دو نقطه قرار می‌گیرد.	
خط عمر		نشان می‌دهد که شئ در سیستم وجود دارد گرچه در حال اجرای یک متد نیست.	یک شئ/ کلکسیون را به اجرای متد متصل می‌کند و اجرای دو متد مجاور را به یکدیگر متصل می‌کند.
اجرای متد		این نماد با شکل مستطیل، نشان می‌دهد که شئ در سیستم وجود دارد و در حال اجرای یک متد است. خط چین، نشان‌دهنده خط عمر است.	به یک شئ/ کلکسیون متصل می‌شود یا از طریق یک بخش از خط عمر به اجرای متد متصل می‌گردد.
نابود کردن شئ		نشان می‌دهد که شئ نابود شده یا برای خروج، متوقف می‌شود.	
انتقال پیام		یک پیام از یک شئ به شئ دیگری ارسال می‌شود. به عبارتی، یک کارگرد از یک شئ به شئ دیگری فراخوانی می‌شود.	اجرای متد دو شئ در حال تعامل را به یک‌دیگر متصل می‌کند.
پیام استریوتایپ		یک استریوتایپ یا پیام تعریف شده توسط کاربر	اجرای متد دو شئ یا یک عامل و اجرای متدی را به یک‌دیگر متصل می‌کند.
جزء ترکیبی		با استفاده از یک عملگر تعاملی مانند loop یا alt، بر روی یک بخش از نمودار توالی، عبارتی را تعریف می‌کند.	

شکل ۹-۱: انگاره‌ها و نمادگان نمودار توالی

می‌دهد. این واقعیت که نمونه با نام است بدین معنی نیست که نمونه باید در جایی دیگر مورد ارجاع قرار گرفته باشد. ولی، اگر در جایی دیگر از نمودار توالی به نمونه ارجاع شده باشد، در این‌صورت نمونه باید دارای نام باشد؛ در غیر این‌صورت چگونه می‌توان به آن نمونه ارجاع داد؟ مورد چهارم، اجتماعی از نمونه‌های یک کلاس را نشان می‌دهد. مثلاً آرایه‌ای از اشیای کتاب، البته این نمایش راجع به نوع اجتماع، که آیا آرایه، فهرست متصل[۱]، یا جدول درهم[۲] است چیزی نمی‌گوید. آخرین مورد، یک نمونه‌ی استریوتایپ‌دار است. استریوتایپ در UML، به کاربران اجازه می‌دهد دسته‌بندهای[۳] مورد نظر خود را تعریف کنند تا مفاهیم و ساختارهای استاندارد UML را گسترش دهند. برای مثال، در آخرین ردیف شکل ۹-۲ شئ استریوتایپ‌دارLogin Page که یک Java Server Page (JSP) می‌باشد، نشان داده شده است. رشته‌ی <<JSP>> می‌گوید که Login Page یک شئ معمولی نیست، بلکه یک صفحه‌ی JSP است. به‌همین روش می‌توان از استریوتایپ استفاده کرد و نشان داد که یک اجتماع مورد نظر، یک جدول درهم است.

---

[۱] Linked List
[۲] Hash Table
[۳] Classifier



شکل ۹-۲: نمونه‌های مختلف یک شئ از یک کلاس

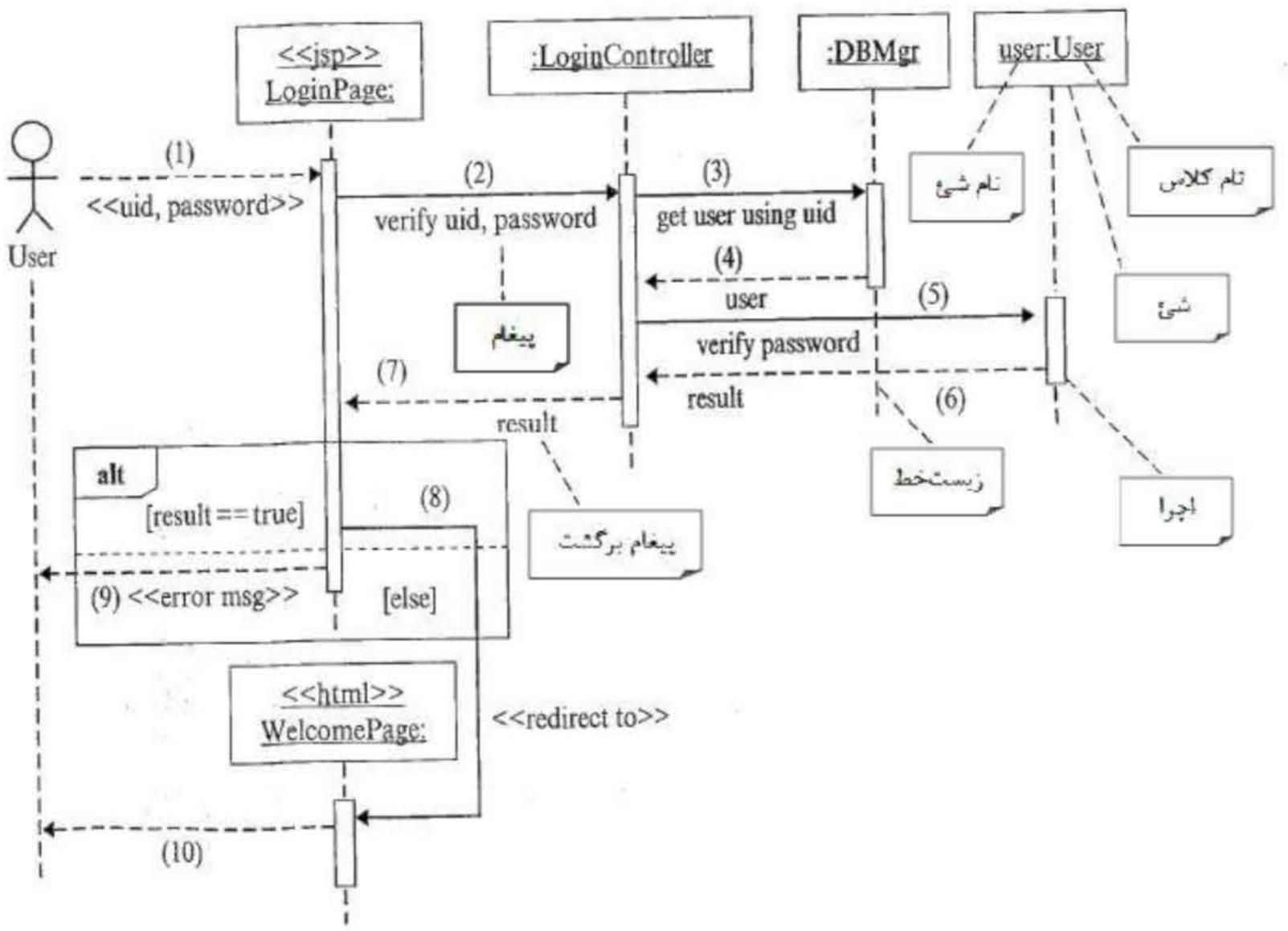
۹-۲-۳- چند نمودار توالی نمونه

شکل ۹-۳ یک نمودار توالی برای یک نمونه ساده از مورد کاربرد Login را نشان می‌دهد. این نمودار در واقع نمایش بصری سناریوی زیر است. شماره‌هایی که روی نمودار درج شده برای این است که تناظر بین سناریو و نمودار توالی را نشان دهد.

۱. کاربر شماره کاربری (uid) و کلمه عبور (password) را به Login Page ارسال می‌کند.
۲. Login Page شماره کاربری و کلمه عبور را از طریق Login Controller وارسی می‌کند.
۳. Login Controller با استفاده از شماره کاربری، کاربر (شئ) را از مدیر پایگاه‌داده(DBMgr) دریافت می‌کند.
۴. DBMgr کاربر (شئ) را به Login Controller برمی‌گرداند.
۵. Login Controller با استفاده از کلمه‌ی عبور، کاربر (شئ) را وارسی می‌کند.
۶. کاربر (شئ)، نتیجه را به Login Controller بر می‌گرداند.
۷. Login Controller نتیجه را به Login Page بر می‌گرداند.
۸. اگر نتیجه مثبت باشد، Login Page کنترل را به Welcome Page می‌سپارد.
۹. اگر نتیجه منفی باشد، Login Page پیغام خطایی به کاربر می‌فرستد.
۱۰. صفحه‌ی خوش‌آمدگویی (Welcome Page) (یا پیغام خطا) به کاربر نشان داده می‌شود.

این مثال نشان می‌دهد که از روی یک نمودار توالی (که به‌دقت ترسیم شده باشد) می‌توان سناریو را تولید نمود. به همین ترتیب، از روی یک سناریوی خوش‌ساخت نیز می‌توان نمودار توالی را تولید نمود. به‌عنوان مثالی دیگر، شکل ۹-۴ یک نمودار توالی برای قسمتی از مورد کاربرد دریافت مدرک را نشان می‌دهد. این نمودار سناریوی زیر را به تصویر می‌کشد.

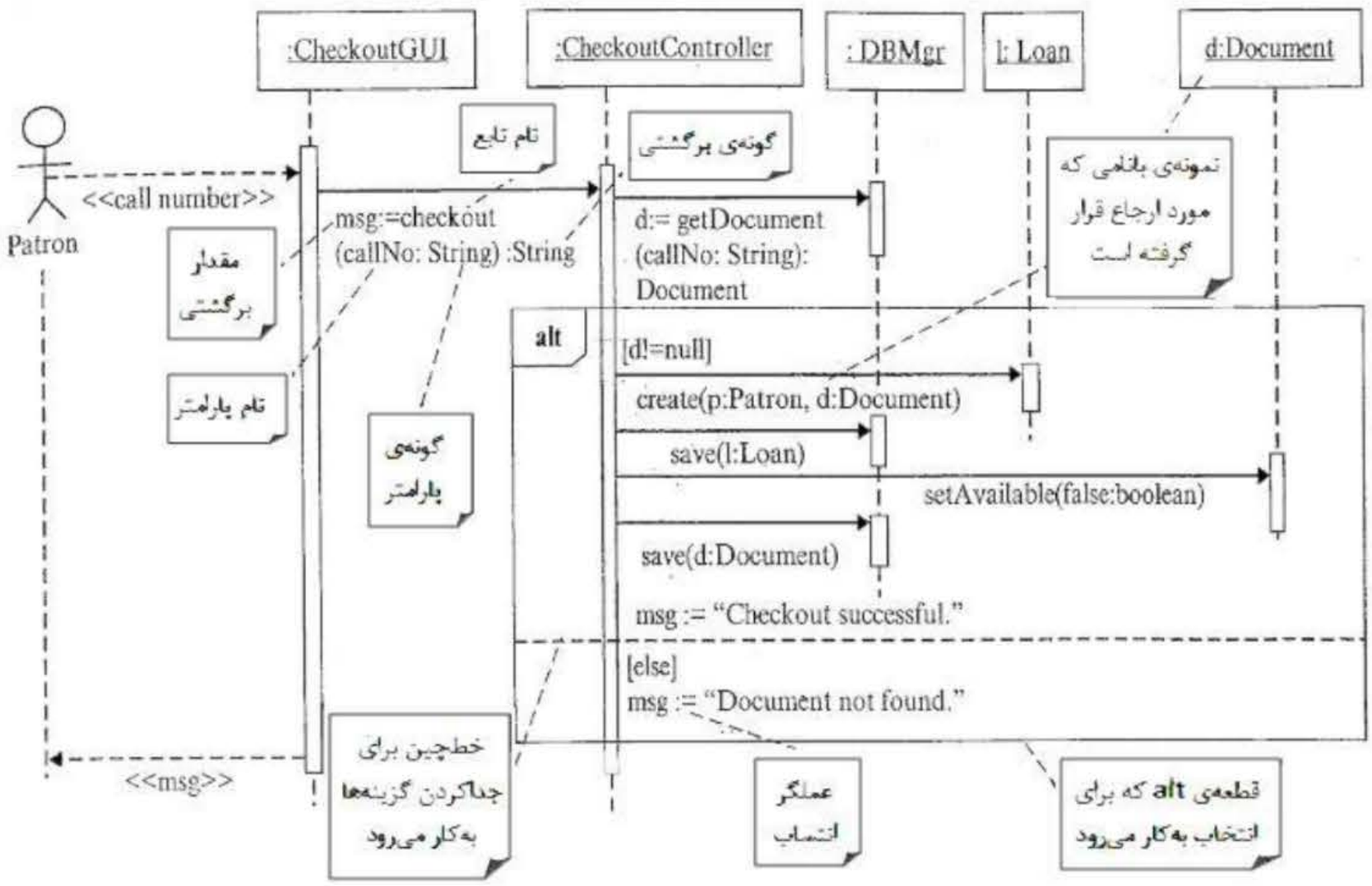
- عضو، شماره‌ی مدرک را در Checkout GUI وارد می‌کند.
- Checkout GUI، متد Checkout(callNo: String) از Checkout Controller را فرامی‌خواند.
- Checkout Controller، متد getDocument(callNo: String) از DBMgr را فرامی‌خواند.
- DBMgr، مدرک d را به Checkout Controller برمی‌گرداند.



شکل ۹-۳: نمودار توالی برای سناریوی Login

- اگر d پوچ (Null) نباشد، آنگاه
  - Checkout Controller، یک شئ Loan به نام l ایجاد می‌کند.
  - Checkout Controller، متد save(l:Loan) از DBMgr را فرامی‌خواند.
- Checkout Controller، متد set Available(false:boolean) از مدرک d را فرامی‌خواند.
- Checkout Controller، متد save(d:Document) از DBMgr را فرامی‌خواند.
- Checkout Controller، پیام msg را معادل "Checkout Successful" قرار می‌دهد.
- در غیراین‌صورت، Checkuot Controller پیام msg را معادل "Document not found" قرار می‌دهد.
- Checkout Controller، پیام msg را به Checkout GUI برمی‌گرداند.
- Checkout GUI، پیام msg را به عضو Patron نشان می‌دهد.
- عضو، پیام msg را مشاهده می‌کند.

توجه کنید که حرف d در چندین موقعیت، از جمله به‌عنوان نمونه‌ی با نام، در آخرین ستون نمودار توالی، استفاده شده است تا بتوان به یک شئ Document ارجاع داشت. در عمل، در حین نوشتن سناریوها، از انتخاب نام متغیر برای نام‌گذاری نمونه‌ها پرهیز می‌شود؛ ولی این متغیرها، در نمودارهای توالی بیشتر به‌کار می‌روند. در این مثال، متغیر برای نمونه‌ها، هم در توصیف سناریو استفاده شد و هم در نمودار توالی، و دلیل آن هم آسان‌تر دیده شدن تناظر بین توصیف سناریو و توالی است.



شکل ۹-۴: نمودار توالی برای مورد کاربرد دریافت مدرک

۹-۲-۴- استفاده از نمودار توالی در تحلیل و طراحی

به تفاوت در سبک تبیین پیغام‌ها در شکل‌های ۹-۳ و ۹-۴ توجه کنید. در شکل ۹-۳ پیام‌هایی که بین اشیا رد و بدل می‌شوند با استفاده از متون انگلیسی به‌صورت غیررسمی بیان شده‌اند. این سبک تبیین غیررسمی تعامل اشیا، می‌تواند در مراحل اولیه‌ی OIM یا در مدل‌سازی یک کاربرد موجود استفاده شود. در حین مرحله‌ی تحلیل یا مرحله‌ی اولیه‌ی طراحی تعامل اشیا، تیم توسعه سعی می‌کند فرایندهای کسب‌وکاری موجود را بفهمد یا پیشنهادهایی جهت بهبود این فرایندها ارائه نماید. در طی این مرحله از OIM، بیان تعامل یا امضای اعمال به صورت رسمی، به چند دلیل میسر نیست. نخست آن‌که، مدل‌سازی تعامل شئ برای نمایش یک سیستم دستی موجود ساخته می‌شود. در این حالت، بیان رسمی پیغام‌ها بی‌معنی است، زیرا هنوز اشیای نرم‌افزاری وجود ندارند. حتی اگر سیستم موجود دستی هم نباشد، در صورتی که تیم توسعه فقط قصد دارد درک اولیه‌ای از سیستم موجود را به‌دست آورد، بیان پیام‌ها به صورت غیررسمی و با هدف صرفه‌جویی در تلاش و زمان،کفایت می‌کند. دلیل دیگر برای تبیین پیغام‌ها به صورت غیررسمی در این مرحله، این است که هنوز ایده‌های طراحی مبهم بوده و لذا تبیین پیام‌ها به صورت رسمی مشکل خواهد بود. در تمام این موارد، تیم توسعه قصد دارد روی یادگیری و کشف ایده‌ها تمرکز کند، نه این‌که با تبیین واسطهای دقیق بین اشیای متعامل، خود را به زحمت اندازد. به همین دلیل، نمودار توالی شکل ۹-۳ را به‌عنوان نمودار توالی تحلیل یا غیررسمی می‌شناسیم.

از سوی دیگر، پیام‌های بین اشیا در شکل ۹-۴ به‌طور رسمی تبیین شده‌اند؛ یعنی، پیام‌ها بر حسب یک گرامر، که شبیه نحو یک زبان برنامه‌سازی است، بیان شده‌اند. گرامری که در شکل ۹-۴ نشان داده شده است شبیه به نحو زبان برنامه‌سازی Pascal [۱۱۷] است. نمودار توالی که به صورت رسمی تعریف شده باشد بیش‌تر برای مواقعی مناسب است که بخواهیم الگوهای طراحی را اعمال کنیم، از قطعات موجود استفاده کنیم، نمودار کلاس طراحی را از آن استنتاج نماییم، و یا اسکلت کد

تولید کنیم. به همین دلیل، نمودار توالی که در آن پیام‌ها به صورت رسمی بیان شده باشند را نمودار توالی طراحی[۱] می‌نامیم.

تفاوت بین تحلیل و طراحی در شکل ۹-۵ نشان داده شده است. نخست، تحلیل مرتبط با مسئله‌ی کاربردی است[۲] در حالی که طراحی مرتبط با جواب نرم‌افزاری است[۳]. این بدان معناست که در حین مرحله‌ی تحلیل، تیم توسعه بیشتر روی فهم دامنه‌ی کاربرد، شناسایی مشکلات کاربرد موجود، و ارائه‌ی راه‌حل‌های احتمالاً مبتکرانه تمرکز می‌کند. در حین مرحله‌ی طراحی، تیم توسعه بیشتر روی توسعه و تبیین یک معماری نرم‌افزاری مناسب و اجزای آن تمرکز دارد به‌نحوی که این معماری، راه‌حل (جواب) پیشنهادی را تا حدی محقق سازد که این راه‌حل بتواند با برنامه‌های کامپیوتری پیاده‌سازی گردد. به‌علاوه، جواب نرم‌افزاری باید برخی اصول طراحی نرم‌افزار، مانند چسبندگی زیاد، جفت‌شدگی کم، جداسازی دغدغه‌ها، و طراحی برای تغییر را نیز مد نظر قرار دهد.

دومین تفاوت بین تحلیل و طراحی در چیزی است که قرار است مدل شود. در حین مرحله‌ی تحلیل، تیم توسعه مدل‌هایی در خصوص دامنه‌ی کاربرد می‌سازد که به فهم کاربرد کمک می‌کند. در حین مرحله‌ی طراحی، مدل‌هایی از سیستم نرم‌افزاری ساخته خواهند شد. برای مثال، در حین مرحله‌ی تحلیل، یک مدل دامنه و احتمالاً چند نمودار توالی برای فرایندهای کسب‌وکاری موجود ساخته می‌شوند تا به توسعه‌دهندگان کمک کنند که کاربرد موجود را درک کنند. در حین مرحله‌ی طراحی، نمودارهای توالی بدین منظور ساخته می‌شوند که مشخص شود اشیا چگونه با هم تعامل می‌کنند تا یک مورد کاربرد انجام شود. از روی نمودار توالی طراحی، نمودار کلاس طراحی استنتاج می‌شود تا کلاس‌ها، رفتار و صفت‌های آن‌ها، و رابطه بین آن‌ها را در یک سیستم نرم‌افزاری نشان دهد.

سومین تفاوت بین تحلیل و طراحی به تفاوت میان ادراک[۴] و نمایش کامپیوتری آن ادراک بر می‌گردد. کلاس‌ها و اشیا در مدل تحلیل همان ادراک مفاهیم دامنه و نمونه‌های آن مفاهیم می‌باشند. در مدل طراحی، کلاس‌ها و اشیا همان کلاس‌ها و اشیا نرم‌افزاری هستند. این‌ها نمایش رقمی یا کامپیوتری برای مفاهیم و نمونه‌های ادراک شده از دامنه هستند.

تفاوت دیگر این است که مدل‌های تحلیل دنیا را همان‌گونه که هست توصیف می‌کنند در حالی که مدل‌های طراحی یک راه‌حل نرم‌افزاری تجویز می‌کنند. این بدین خاطر است که مدل‌های تحلیل دامنه‌ی کاربرد را توصیف کرده و به تیم توسعه در فهم آن کاربرد کمک می‌کنند. بنابراین، مدل‌های تحلیل توصیفی هستند. از سوی دیگر، مدل‌های طراحی یک راه‌حل نرم‌افزاری تجویز می‌کنند زیرا طراحی پیشنهادیه‌هایی را برای بهبود سیستم موجود لحاظ کرده است.

از آن‌جا که مدل‌های تحلیل «دنیا را آن‌گونه که هست توصیف می کنند» نباید هیچ محدودیتی بر روی فضای طراحی تحمیل نمایند. بنابراین، یک مدل تحلیل ایده‌آل باید تمام انتخاب‌های ممکن طراحی را مجاز بشمارد. از طرف دیگر، مدل‌های طراحی نتیجه‌ی تصمیمات طراحی می‌باشند، لذا، ممکن است فضای پیاده‌سازی را محدود نمایند. به‌عنوان مثال، این تصمیم طراحی را در نظر بگیرید که به اشیای کسب‌وکاری اجازه‌ی دسترسی به یک پایگاه‌داده‌ی رابطه‌ای را می‌دهد. ممکن است دلیل اتخاذ این تصمیم، سادگی و نیز بازدهی آن بوده باشد. اما این کار ممکن است برای محصول نرم‌افزاری، در دسترسی به یک پایگاه‌داده‌ی دیگر مشکل ایجاد کند زیرا اشیای کسب‌وکاری و پایگاه‌داده کاملاً به‌هم وابسته‌اند.

۹-۲-۵- استفاده‌ی درست از نمادگان

UML یک زبان است که دارای واژگان، نحو، و معنا می‌باشد. نمودار توالی نیز یک نمودار UML است که واژگان، نحو، و معنای آن زیر مجموعه‌ای از بخش‌های مربوطه در UML است. در موقع ساخت نمودار توالی، مهم است که این عناصر به درستی استفاده شوند، وگرنه ممکن است یک نمودار توالی تولید شود که مبهم، نادرست، یا غیرقابل‌فهم باشد. از آن‌جا که محدودیتی در تعداد روش‌های استفاده‌ی نابجا از تکنیک‌های رسم نمودار وجود ندارد، در ترسیم نمودارهای توالی می‌توان مرتکب خطاهای

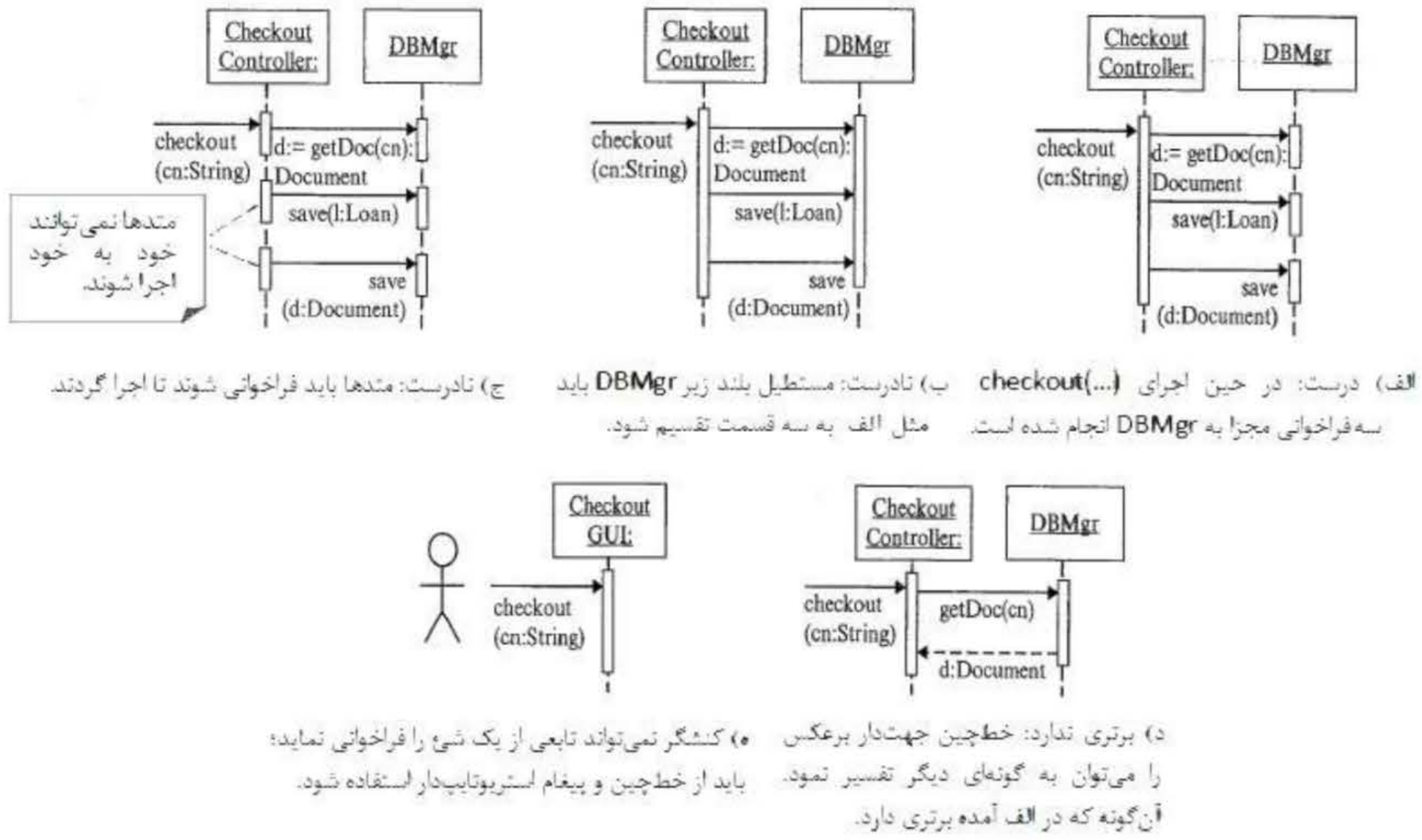
---

[۱] Design Sequence Diagram
[۲] Application Problem–Oriented
[۳] Software Solution-Oriented
[۴] Perception

تحلیل	طراحی
• مسئله کاربردی گرا • یک دامنه کاربردی را مدل می‌کند. • کلاس‌ها و اشیا، مفاهیم و نمونه‌های دامنه هستند. • چیستی جهان را بیان می‌کند. • اجازه چندین انخاب طراحی را می‌دهد.	• راه حل نرم‌افزاری گرا • یک سیستم نرم‌افزاری را مدل می‌کند. • کلاس‌ها و اشیا، کلاس‌ها و اشیای نرم‌افزاری هستند. • یک راه حل نرم‌افزاری تجویز می‌کند. • امکان دارد که انتخاب‌های پیاده‌سازی را کاهش دهد.

شکل ۹-۵: تفاوت میان تحلیل و طراحی

متعددی شد. شکل ۹-۶ یک استفاده‌ی درست و چند استفاده‌ی نادرست از نمادگان مدل‌سازی نمودار توالی را نشان می‌دهد.



الف) درست: در حین اجرای checkout(...) سه فراخوانی مجزا به DBMgr انجام شده است.

ب) نادرست: مستطیل بلند زیر DBMgr باید مثل الف به سه قسمت تقسیم شود.

ج) نادرست: متدها باید فراخوانی شوند تا اجرا گردند.

د) برتری ندارد: خط‌چین جهت‌دار برعکس را می‌توان به گونه‌ای دیگر تفسیر نمود. آن‌گونه که در الف آمده برتری دارد.

ه) کنشگر نمی‌تواند تابعی از یک شئ را فراخوانی نماید؛ باید از خط‌چین و پیغام استریوتایپ‌دار استفاده شود.

شکل ۹-۶: استفاده‌های درست و نادرست از نمادگان

شکل ۹-۶-الف یک نمودار توالی درست را نشان می‌دهد. طبق این نمودار، در حین اجرای عمل خروج مدرک از Checkout Controller، سه فراخوانی مجزا به توابع شئ DBMgr انجام می‌شود. نمودار توالی شکل ۹-۶-ب نادرست است چون سه فراخوانی به متدهای شئ DBMgr فقط به‌وسیله‌ی یک اجرای متد نشان داده شده‌اند. شکل ۹-۶- ج نادرست است، یا حداقل این‌که فهم آن آسان نیست، زیرا دومین و سومین فراخوانی‌ها از کنترلگر به شی DBMgr، هیچ‌گونه فراخوانی ورودی ندارند، در حالی‌که این فراخوانی‌های ورودی برای برانگیختن[۱] اجرای این دو متد «بی‌نام»[۲] ضروری می‌باشند. در شکل ۹-۶-د، پیکان خط‌چین از شئ DBMgr به کنترلگر فقط باید در نمودار تحلیل به‌کار رود. شکل ۹-۶-د یک نمودار توالی طراحی است، بنابراین، پیکان خط‌چین و برچسب "d:Document" باید حذف شوند و فراخوانی از کنترلگر به شئ DBMgr باید برچسب

[۱] Trigger

[۲] Anonymous

"d:= get Document(cn:String):Document" بخورد. در شکل ۹-۶-ه پیکان خط ممتد نباید استفاده شود، زیرا بیان‌گر فراخوان تابع است و نمی‌تواند از کنشگر بیاید. این خط باید با یک پیکان خط‌چین جایگزین شود. این پیکان خط‌چین نیز باید با یک پیغام استریوتایپ مانند "Checkout cn" برچسب بخورد.

۹-۳ گام‌های مدل‌سازی تعامل شئ

گام‌های OIM در شکل ۹-۷ نشان داده شده و در فهرست زیر معرفی شده‌اند. این گام‌ها برای مورد کاربردهایی که به تکرار جاری تخصیص یافته‌اند اجرا می‌شوند.

گام ۱. جمع‌آوری اطلاعات درباره‌ی فرایندهای کسب‌وکار موجود.

در این گام، تیم توسعه درباره‌ی فرایندهای کسب‌وکار موجود مرتبط با مورد کاربردها اطلاعاتی را جمع‌آوری و مطالعه می‌کند.

گام ۲. سناریوهایی را برای گام‌های غیربدیهی از مورد کاربردهای گسترده تبیین کنید (اختیاری).

در این گام، گام‌های غیربدیهی از مورد کاربردهای گسترده شناسایی می‌شوند و سناریوهای این گام‌ها نیز تبیین می‌گردند. سناریو دنباله‌ای از جملات اعلانی است که می‌گوید اشیا چگونه با یکدیگر تعامل می‌کنند تا یک گام غیربدیهی انجام شود. ورودی این گام مورد کاربردهای گسترده برای تکرار فعلی و اطلاعات جمع‌آوری شده در گام آخر است. خروجی این گام، فهرستی از توصیف سناریوهاست.

گام ۳. ساخت جداول سناریو (اختیاری).

در این گام، نمایش جدولی هر سناریو، که به آن جدول سناریو می‌گوییم، تولید می‌شود تا در ساخت نمودار توالی از آن کمک گرفته شود. ورودی این گام، فهرستی از توصیف سناریوهاست. خروجی این گام، مجموعه‌ای از جداول سناریو است.

گام ۴. استنتاج نمودار توالی از جداول سناریو

در این گام، توصیف سناریوها یا جداول سناریو به نمودارهای توالی UML تبدیل می‌شوند. علاوه بر این، گونه‌ها و واسط‌های اشیای مربوطه تعیین می‌شوند. ورودی این مرحله توصیف سناریوها یا جداول سناریو، نمودار کلاس طراحی تولید شده در تکرارهای قبلی، و مدل دامنه می‌باشد. خروجی این گام مجموعه ای از نمودارهای توالی است.

گام ۵. مرور مدل‌های تعامل شئ

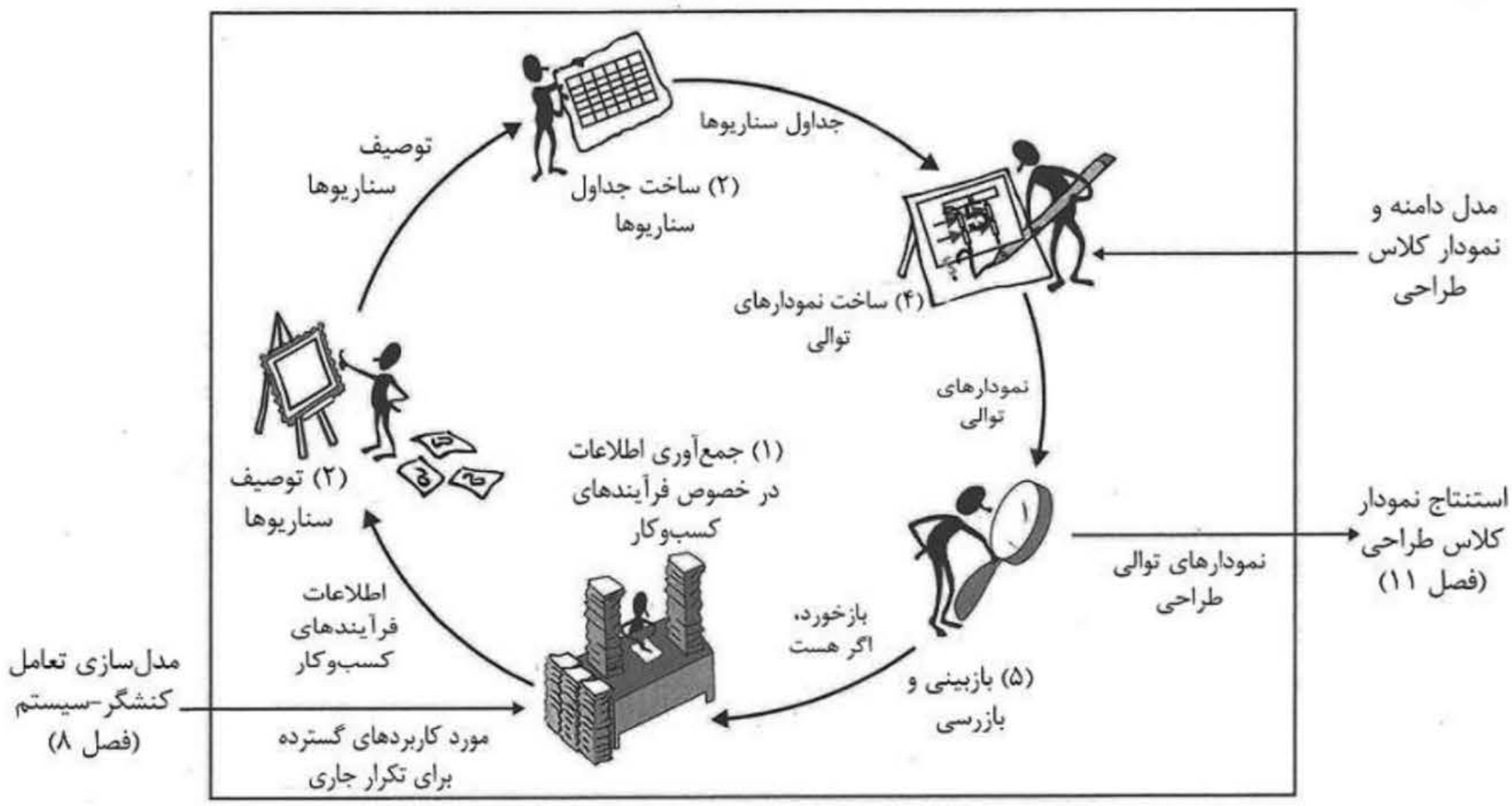
در این گام، مدل‌های تعامل شئ از نظر سازگاری، کامل بودن، و درستی مورد بازبینی و بازنگری قرار می‌گیرند.

نمودارهای توالی بدست آمده برای استنتاج کلاس‌هایی که باید پیاده‌سازی شوند مورد استفاده قرار می‌گیرند. همچنین، صفت‌ها و اعمال کلاس‌ها و روابط وابستگی بین کلاس‌ها نیز استنتاج می‌گردند. نتایج در قالب یک نمودار کلاس UML که به آن نمودار کلاس (طراحی)[۱] گفته می‌شود نشان داده می‌شوند. از DCD به‌عنوان یک نقشه‌ی طراحی برای پیاده‌سازی و آزمون استفاده می‌شود. استنتاج DCD در فصل ۱۱ تشریح شده است. در توسعه‌ی چابک، چنان‌چه تیم دانش و تجربه‌ی کافی داشته باشد، برخی یا تمام گام‌های OIM حذف می‌شوند. ولی، برای مبتدیان و تیم‌هایی که دانش و تجربه‌ی کمتری دارند، توصیه می‌شود که گام‌های پیش‌تر گفته شده را حداقل در مورد چند پروژه‌ی اول، دنبال نمایند. بخش‌های بعدی به توصیف هرکدام از گام‌ها با جزییات بیشتر می‌پردازند.

۹-۳-۱- جمع‌آوری اطلاعات درباره‌ی فرایندهای کسب‌وکار

OIM می‌گوید که تیم توسعه باید دانش کافی درباره‌ی فرایندهای موجود کسب‌وکاری مرتبط با مورد کاربردها داشته باشد. برای اکتساب چنین دانشی، تیم توسعه با مشتریان و کاربران کار می‌کند تا اطلاعاتی درباره‌ی فرایندهای کسب‌وکار جمع‌آوری نماید.

[۱] Design Class Diagram (DCD)



شکل ۹-۷: مروری بر مدل‌سازی تعامل شئ

اطلاعات جمع‌آوری شده در حین مرحله‌ی تحلیل نیازمندی‌ها را می‌توان بازاستفاده نمود و در صورت لزوم اطلاعات اضافی در این گام جمع‌آوری می‌شود. در ادامه یک فهرست جزیی از اقلامی که باید در حین فرایند جمع‌آوری اطلاعات بر روی آن‌ها تمرکز نمود ارائه می‌گردد.

۱. موقعیت کسب‌وکار فعلی چیست و چگونه کار می‌کند؟
۲. کسب‌وکاری که قرار است سیستم کامپیوتری برای آن ساخته شود چیست؟
۳. اهداف کسب‌وکار یا اهداف محصول کدام‌اند؟
۴. برای انجام اهداف از کدام فرایندهای کسب‌وکاری موجود استفاده می‌شود؟ کارکرد هرکدام از این فرایندها چیست؟
۵. ورودی و خروجی هرکدام از فرایندهای کسب‌وکاری موجود کدام‌اند؟
۶. فرایندهای موجود چگونه کار می‌کنند؟
۷. فرایندهای موجود چگونه به هم مرتبط بوده و با هم کار می‌کنند؟
۸. چه بهبودها یا پیشرفت‌هایی را کاربران و مشتریان انتظار دارند؟

تکنیک‌های جمع‌آوری اطلاعات در بخش ۴-۵-۱ معرفی شدند. برای جزییات بیشتر به آن بخش مراجعه کنید. در این‌جا، این تکنیک‌ها به اختصار معرفی می‌شوند.

۱. ارائه‌ی مشتری. ارائه‌ی مشتری روشی مؤثر است که به تیم کمک می‌کند اطلاعات مورد نیاز را جمع‌آوری کند. ارائه باید حداقل اقلامی که در بالا بدان‌ها اشاره شد را پوشش دهد و نباید بیش از دو ساعت بطول انجامد.
۲. بررسی سوابق موجود. بررسی سوابق دانش دامنه‌ای مورد نیاز و نیز اطلاعاتی در خصوص فرایندهای کسب‌وکاری را برای تیم فراهم می‌آورد. با مطالعه‌ی پروژه‌های مشابه، تیم از تجارب آن پروژه‌ها، اعم از کارکرد سیستم‌های مشابه و چگونگی طراحی و پیاده‌سازی این سیستمها، استفاده می‌نماید.

۳. استفاده از پرسشنامه. پرسشنامه‌های تحقیقی برای جمع‌آوری اطلاعات از کاربران مفید هستند. اگر این پرسشنامه‌ها بعد از ارائه مشتری استفاده شوند مؤثرتر خواهند بود، زیرا تیم بهتر می‌داند که چه چیزهایی را باید در پرسشنامه بیاورد. لیست سؤال‌های تحقیق باید خلاصه و برای کاربران به‌سادگی قابل فهم باشد . اگر فرایندهای کسب و کار گروه‌های مختلف کاربران متفاوت باشد، شاید مناسب باشد که سؤال‌های متفاوتی نیز از آن‌ها پرسیده شود.
۴. مصاحبه با کاربران. مصاحبه‌ی تک به تک با کاربران به تیم امکان می‌دهد که مشکلاتی که در تحقیق به آن‌ها برخورده‌اند را شفاف کنند. ایجاد و حفظ یک فضای صمیمی در زمان مصاحبه مهم است زیرا باعث بهبود اثرگذاری ارتباط می‌شود. هر جلسه مصاحبه باید حداکثر یکساعت باشد. مصاحبه‌کنندگان باید به جای حل مشکلات کسب و کار جاری، بر جمع‌آوری اطلاعات تمرکز کنند.
۵. مطالعه‌ی رویه‌ها و فرم‌ها. همه‌ی کسب‌وکارها فرم‌ها و روال‌های عملیاتی و نیز گزارش‌ها و ثبت وقایع را دارند. این‌گونه اقلام کسب‌وکاری منابع باارزشی برای جمع‌آوری اطلاعات محسوب می‌شوند.

۹-۳-۲- شناسایی گام‌های غیربدیهی

مورد کاربرد گسترده در فصل ۸ معرفی گردید. این یک جدول دو ستونی است که می‌گوید چگونه یک کنشگر با سیستم تعامل می‌کند تا یک فرایند کسب‌وکار انجام شود. به بیان دقیق‌تر، ستون چپ مورد کاربرد گسترده درخواست‌های کنشگر را مشخص می‌کند، در حالی که ستون راست پاسخ‌های سیستم را نشان می‌دهد. یکی از انواع درخواست‌های کنشگر از سیستم می‌خواهد که پردازش‌های پس‌زمینه‌ای را انجام دهد. بدین معنی که، اشیا نرم‌افزاری را مجبور می‌کند با هم تعامل و همکاری داشته باشند و درخواست مورد نظر را برآورده نمایند. چنین درخواست کنشگری یک درخواست کنشگر غیربدیهی یا درخواست غیربدیهی است. به‌عنوان مثال، مورد کاربرد گسترده نشان داده شده در شکل ۹-۸ را در نظر بگیرید. درخواست کنشگر مربوط به گام ۳، غیربدیهی است چون سیستم مجبور است مستندات را از پایگاه‌داده بازیابی نماید، اشیای امانت[۱] یا (رکوردهای امانت[۲]) را ایجاد کند، مستندات را بروزرسانی نماید و تمام این‌ها را درون پایگاه‌داده ذخیره نماید. تولید پاسخ سیستم برای گام ۴ غیربدیهی است. بنابراین، گام ۳ یک درخواست کنشگر غیربدیهی، و گام ۴ یک گام غیربدیهی است.

سیستم: LIS	عامل: Patron
۰. سیستم واسط اصلی را نمایش می‌دهد.	
۲. سیستم واسط خروج را نمایش می‌دهد.	۱. TUCBW وقتی عضو روی دکمه Checkout Document از واسط اصلی کلیک می‌کند.
۴. واسط خروج یک پیام خروج همراه با جزئیات نمایش می‌دهد.	۳. عضو، شماره‌ی مستنداتی که می‌خواهد به امانت ببرد را وارد کرده و روی دکمه‌ی Submit کلیک می‌کند.
	۵. TUCEW وقتی عضو پیام خروج را می‌بیند.

شکل ۹-۸: مورد کاربرد گسترده خروج مدرک

گام‌های غیربدیهی سیستم را وادار به اجرای پردازش‌های پس‌زمینه‌ای می‌کنند. نکات زیر برای شناسایی گام‌های غیربدیهی مفید هستند:

۱. اگر پاسخ سیستم تنها نمایش یک فهرست یا یک جعبه‌ی ورودی ساده باشد، در این صورت آن گام بدیهی است. یکی

[۱] Loan Objects
[۲] Loan Records

از ویژگی‌های نمایش این است که برای تمام کنشگرها یا مورد کاربردها یکسان باشد. برای مثال، در شکل ۹-۸ گام ۲، GUI خروج را نشان می‌دهد، که برای همه‌ی اعضا یکسان خواهد بود. بنابراین، گام ۲ بدیهی است.

۲. اگر پاسخ سیستم برای کنشگرهای متفاوت مختلف باشد، دراین‌صورت آن گام غیربدیهی است. گام ۴ در شکل ۹-۸ این شرط را داراست.

۹-۳-۳- نوشتن سناریو برای گام‌های غیربدیهی

سناریوها به‌طور گسترده برای توصیف این‌که چگونه اشیا با هم‌دیگر تعامل می‌کنند تا یک کار انجام شود، استفاده می‌شوند. سپس سناریوها به نمودارهای توالی UML تبدیل می‌شوند. چنان‌چه فرایندهای کسب‌وکاری که قرار است مدل شوند، پیچیده بوده یا شامل تعداد زیادی گام باشند، در این صورت نمودار توالی مربوطه نیز پیچیده و طولانی خواهد شد. در نتیجه، فهم آن دشوار خواهد بود. برای پرهیز از این مشکل، متدولوژی ارائه شده در این کتاب از نمودارهای توالی فقط برای توصیف گام‌های غیربدیهی یک مورد کاربرد گسترده استفاده می‌کند.

سناریو چیست؟

تعامل اشیا شبیه تعامل بین هنرپیشه‌ها در یک نمایش است. این هنرپیشه‌ها بر طبق یک فیلم‌نامه یا سناریو تعامل می‌کنند. این تشابه باعث شد جامعه‌ی مهندسی نرم‌افزار واژه‌ی «سناریو» را برای توصیف چگونگی تعامل اشیا با یکدیگر برای انجام یک مورد کاربرد، از تئاتر قرض بگیرد. گرچه سناریوی نمایش چیز جذابی است، ولی سناریویی که چگونگی تعامل اشیا برای انجام یک مورد کاربرد را نشان می‌دهد چیز خسته‌کننده‌ای است و دلیل آن هم این است که کامپیوتر که زیربناست خیلی کسل‌کننده است. این به ما می‌گوید که زبانی که برای توصیف سناریوهای تعامل شئ به‌کار می‌رود باید به زیرمجموعه‌ی کوچکی از زبان طبیعی محدود شود. جملات زبان به صورت زیر تعریف می‌شوند.

تعریف ۹-۲

هرجمله‌ی تعامل شئ اتمیک، جمله‌ای اعلانی به شکل سوم شخص و در زمان حال ساده است. این جمله شامل یک فاعل، یک مفعول که کار روی آن انجام می‌شود و یک کار یا کنش که آن فاعل انجام می‌دهد، و احتمالاً داده یا اشیایی که مورد نیاز آن کار است، می‌باشد.

مثال ۹-۱

مثال‌هایی از جملات تعامل شئ در زیر آمده‌اند:

- رومئو یک شاخه‌ی گل به ژولیت می‌دهد.

در اینجا، «رومئو» فاعل، «می‌دهد» کنش یا کار فاعل، «ژولیت» مفعولی که بر روی آن کار صورت می‌گیرد، و «یک شاخه‌ی گل» مفعول ثانویه یا مفعولی که مورد نیاز کنش فاعل است، هستند.

- کاربر، شناسه‌ی کاربری و رمز عبور خود را درون واسط کاربری گرافیکی مربوط به ورود به سیستم (Login GUI) وارد می‌کند. در این مورد، «کاربر» فاعل، «وارد می‌کند» کنش فاعل، و «شناسه‌ی کاربری» و «رمز عبور» داده‌هایی هستند که مورد نیاز کنش فاعل می‌باشند. مفعولی که کار روی آن صورت گرفته «Login GUI» است.

- کنترلگر خروج با استفاده از شماره مدرک، مدارک را از مدیر پایگاه‌داده دریافت می‌کند.

در این مورد، فاعل «کنترلگر خروج»، کنش فاعل «دریافت مدرک»، مفعولی که کنش روی آن صورت می‌گیرد «مدیر پایگاه‌داده»، داده یا مفعولی که کنش فاعل به آن نیاز دارد «شماره مدرک» می‌باشد.

توجه کنید که در این مثال‌ها جملات به زمان حال ساده و از زبان سوم شخص نوشته شده‌اند.

جملات مرکب تعامل شئ را می‌توان با استفاده از جملات تعامل شئ اتمیک و با اعمال عملگرهای ترکیب رایج از قبیل شرط[۱]، تکرار[۲] و سایر عملگرهای بامعنی، ساخت. توجه کنید که UML مجموعه‌ای غنی از عملگرهای تعامل مانند انتخاب[۳] و حلقه[۴] در خود دارد، ولی در این کتاب فقط تعداد محدودی از آن‌ها معرفی خواهند شد.

تعریف ۹-۳

جمله‌ی مرکب تعامل شئ به‌طور بازگشتی به صورت زیر تعریف می‌شود.

- هر جمله‌ی اتمیک تعامل شئ یک جمله‌ی مرکب تعامل شئ است.
- اگر S1,S2,…,Sn جملات مرکب تعامل شئ و OP یک عملگر ترکیب nتایی باشد، آنگاه جمله‌ی حاصل از اعمال OP روی S1,S2,…,Sn نیز یک جمله‌ی مرکب تعامل شئ خواهد بود.

در این کتاب جمله، جمله‌ی تعامل شئ، جمله‌ی اتمیک تعامل شئ، و جمله‌ی مرکب تعامل شئ همگی به‌جای هم به‌کار می‌روند، به جز در مواردی که غیر از این مشخص شده باشد.

تعریف ۹-۴

سناریو، دنباله‌ای از جملات تعامل شئ است.

گام‌های نوشتن سناریو برای گام‌های غیربدیهی به شرح زیر می‌باشند.

گام ۲-۱. گام‌های غیربدیهی را از ستون راست مورد کاربرد گسترده شناسایی کنید و آن را با علامت ستاره (*) مشخص کنید. سناریویی که با گام متناظر از ستون سمت چپ شروع شده و با گام علامت خورده در ستون سمت راست خاتمه می‌یابد، باید تبیین شود.

گام ۲-۲. برای هر گام غیربدیهی یک سناریو تبیین کنید. این سناریو باید توصیف سطح بالایی از یک الگوریتم باشد. این سناریو با ورودی کنشگر و کنش کنشگر در گام مربوطه از ستون سمت چپ شروع شده، و وقتی که پاسخ سیستم که در گام غیربدیهی توضیح داده شده تولید شود، خاتمه می‌یابد. این گام یک سناریوی جزیی مورد کاربرد تولید می‌کند. سناریوی کامل مورد کاربرد شامل تمام گام‌هایی است که در مورد کاربردی گسترده مشخص شده است، به‌گونه‌ای که در آن گام‌های غیربدیهی با سناریوهای جزیی مورد کاربرد جایگزین شده باشد.

گام ۲-۳. جملات سناریو را با اعداد دهدهی مثل ۴-۱، ۴-۲، و ۴-۲-۱ شماره‌گذاری کنید تا ترتیب و تودرتویی کاملاً مشخص باشد.

---

[۱] If-then-else
[۲] Repetition
[۳] Alternative (Alt)
[۴] Iteration (Loop)

مثال ۹-۲

برای مورد کاربرد گسترده‌ی شکل ۹-۸ یک سناریوی مورد کاربرد تولید کنید.

جواب: برطبق معیارهایی که در بخش ۹-۳-۲ معرفی شدند، گام ۴ یک گام غیربدیهی است. بنابراین، گام ۴ را با ستاره علامت می‌زنیم. یک سناریو برای این گام در شکل ۹-۹ نشان داده شده است. سناریوی کامل مورد کاربرد شامل تمام گام‌های مورد کاربردی گسترده است که در آن گام‌های ۳ و ۴ توسط گام‌های شکل ۹-۹ جایگزین شده‌اند.

۳. عضو شماره‌ی مستنداتی که می‌خواهد به امانت ببرد را وارد کرده و روی دکمه‌ی Submit کلیک می‌کند.
۱.۴. واسط خروج (Checkout GUI) بر اساس شماره‌ی مستندات، آن‌ها را با کنترلگر خروج بررسی می‌کند.
۲.۴. کنترلگر خروج یک پیغام خالی msg ایجاد می‌کند.
۳.۴. برای هر کدام از شماره مستندات،
۱.۳.۴. کنترلگر خروج با استفاده از شماره‌ی مستند، آن را از مدیر پایگاه‌داده(DBMgr) درخواست می‌کند.
۲.۳.۴. DBMgr سند d را به کنترلگر خروج برمی‌گرداند.
۳.۳.۴. اگر سندی وجود داشته باشد (یعنی d != null)
۱.۳.۳.۴. کنترلگر خروج بررسی می‌کند که آیا سند در دسترس است (برای خروج).
۲.۳.۳.۴. اگر سند برای خروج در دسترس است،
۱.۲.۳.۳.۴. کنترلگر خروج یک شئ Loan به ازای عضو p و سند d ایجاد می‌کند،
۲.۲.۳.۳.۴. کنترلگر خروج سند dرا غیرقابل دسترس می‌کند،
۳.۲.۳.۳.۴. کنترلگر خروج شئ Loan را با DBMgr ذخیره می‌کند،
۴.۲.۳.۳.۴. کنترلگر خروج سند d را با DBMgr ذخیره می‌کند،
۵.۲.۳.۳.۴. کنترلگر خروج شئ "checkout successful" را روی msg می‌نویسد،
۳.۳.۳.۴. درغیر این‌صورت،
۱.۳.۳.۴. کنترلگر خروج "document not available" را روی msg می‌نویسد،
۴.۳.۴. درغیر این‌صورت،
۱.۴.۳.۴. کنترلگر خروج "document not found" را روی msg می‌نویسد،
۴.۴. کنترلگر خروج پیام msg را به checkout GUI برمی‌گرداند،
۵.۴. checkout GUI پیام msg را به عضو نمایش می‌دهد.

شکل ۹-۹: سناریوی گام ۴ از شکل ۹-۸

تیم توسعه باید فرایند کسب‌وکاری را بشناسد و بداند چگونه آن را با استفاده از کامپیوتر پیاده‌سازی کند. این‌ها برای نوشتن یک سناریو لازمند. بدون این‌ها، درک سناریو و نمودار توالی به‌دست آمده مشکل و لذا کیفیت نرم‌افزار نیز پایین خواهد بود. از این نظر، قدرت تیم در مفهوم‌سازی یک الگوریتم پردازشی خوب یک ضرورت محسوب می‌شود. هیچ راه‌حل جادویی وجود ندارد[۱]. توجه کنید که سناریوی ۹-۹ نسبتاً پیچیده شده و باید آن را ساده‌تر کرد. این موضوع بعداً بحث خواهد شد.

[۱] There Is No Silver Bullet!

۹-۳-۴- ساخت جداول سناریو

برای برخی نوآموزان استنتاج نمودارهای توالی از روی سناریوها کار ساده‌ای نیست. زیرا رابطه‌ی بین این دو واضح و روشن نیست. یک راه برای حل این مشکل استفاده از یک نمایش بینابینی است. این نمایش را جدول سناریو می‌نامند. این جدول به‌سادگی از روی سناریو به‌دست آمده و سپس به‌آسانی تبدیل به نمودار توالی می‌شود. جدول سناریو پنج ستون دارد. اولین ستون شماره‌ی جمله‌ی تعامل شئ است. دیگر ستون‌ها به‌ترتیب مربوط به فاعل، کنش فاعل، داده یا شئ موردنیاز کنش فاعل، و مفعولی که کار روی آن صورت می‌گیرد، می‌باشند. به‌عبارت دیگر، جدول سناریو یک نمایش جدولی از سناریو است. این جدول اجزای جملات سناریو را در ستون‌هایی از جدول سناریو تفکیک می‌کند. همان‌گونه که در بخش بعدی نشان داده خواهد شد، این نمایش جدولی، تبدیل سناریو به نمودار توالی را تسهیل می‌نماید. دلیل دیگر برای ساخت جدول سناریو این است که تولید خودکار نمودار توالی را راحت می‌کند. زیرا نگاشت از جدول به نمودار توالی می‌تواند به صورت مکانیکی انجام شود.

تبدیل یک سناریو به جدول سناریو شامل شناسایی و مشخص نمودن فاعل، کنش فاعل، داده یا اشیایی که مورد نیاز کنش فاعل می‌باشند، و مفعول (شئ) که عمل روی آن صورت می‌گیرد، خواهد بود. سپس این اقلام به صورت ردیف به ردیف و ستون به ستون وارد جدول سناریو می شوند. مثال بعدی این را نشان می‌دهد.

مثال ۹-۳

شکل ۹-۱۰ سناریوی مورد کاربردی خروج مدرک Checkout Document را که در آن مواردی برجسته شده‌اند نشان می‌دهد. شکل ۹-۱۱ نیز جدول سناریوی حاصل از این سناریوی علامت‌گذاری شده را نشان می‌دهد.

نتیجه‌ی تبدیل در شکل ۹-۱۱ ارائه شده است و نشان می‌دهد که توصیف سناریوهای متنی و جداول سناریو نمایش‌های معادلی هستند. بنابراین، تیم ممکن است بخواهد هر دوی آن‌ها یا فقط یکی از آن‌ها را بسازد. ذکر این نکته قابل توجه است که در اکثر ردیف‌های شکل ۹-۱۱ ، فاعل یا خود فاعل است یا مفعولی است که در ردیف قبل کاری روی آن صورت گرفته است. این موضوع برای توصیف متنی سناریوها نیز درست است. توسعه‌دهنده باید در موقعی که برای یک گام غیرواقعی سناریو می‌نویسد این را ملاحظه نماید.

۹-۳-۵- چگونه سناریو بنویسیم؟

نوشتن سناریو برای یک گام غیربدیهی مانند نوشتن شبه‌کد[1] یک الگوریتم برای حل یک مسئله است، ولی تفاوت در نحوه‌ی تفکر است. نویسندگان الگوریتم‌های متداول (سنتی) براساس برنامه‌سازی ترتیبی[2] فکر می‌کنند. نوشتن سناریو برای یک سیستم شئ‌گرا نیاز دارد که تیم بر این اساس فکر کنند که چگونه اشیا برای انجام یک کار با هم تعامل می‌کنند. بدین معنی که هر جمله از سناریو متشکل از یک شئ است که از شئ دیگری درخواست می‌کند تا کاری را انجام دهد. در ادامه نیز ممکن است از دیگر اشیا بخواهد کارهایی را انجام دهند. این بحث به ما می گوید که نوشتن سناریو برای گام‌های غیربدیهی نیاز به در نظر گرفتن تصمیم‌های زیر خواهد داشت.

۱. برای انجام درخواست غیربدیهی کنشگر، چه کارهایی باید انجام شود؟ یعنی، وظایفی که لازم است انجام شود تا درخواست کنشگر برآورده شود باید شناسایی شوند.
۲. ترتیب مطلوب برای انجام وظایف چیست؟

---

[1] Pseudo-Code

[2] Sequential Programming

۳.	عضو شماره‌ی مستنداتی که می‌خواهد به امانت ببرد را وارد کرده و روی دکمه‌ی submit کلیک می‌کند.
۱.۴.	واسط خروج (Checkout GUI) بر اساس شماره‌ی مستندات، آن‌ها را با کنترلگر خروج بررسی می‌کند.
۲.۴.	کنترلگر خروج یک پیغام خالی msg ایجاد می‌کند.
۳.۴.	**برای هر** کدام از شماره مستندات،
۱.۳.۴.	کنترلگر خروج با استفاده از شماره‌ی مستند، آن را از مدیر پایگاه‌داده(DBMgr) درخواست می‌کند.
۲.۳.۴.	DBMgr سند d را به کنترلگر خروج برمی‌گرداند.
۳.۳.۴.	**اگر** سندی وجود داشته باشد (یعنی d != null)
۱.۳.۳.۴.	کنترلگر خروج بررسی می‌کند که آیا سند در دسترس است (برای خروج).
۲.۳.۳.۴.	**اگر** سند برای خروج در دسترس است،
۱.۲.۳.۳.۴.	کنترلگر خروج یک شئ Loan به ازای عضو p و سند d ایجاد می‌کند،
۲.۲.۳.۳.۴.	کنترلگر خروج سند d را غیرقابل دسترس می‌کند،
۳.۲.۳.۳.۴.	کنترلگر خروج شئ Loan را با DBMgr ذخیره می‌کند،
۴.۲.۳.۳.۴.	کنترلگر خروج سند d را با DBMgr ذخیره می‌کند،
۵.۲.۳.۳.۴.	کنترلگر خروج شئ "checkout successful" را روی msg می‌نویسد،
۳.۳.۳.۴.	**درغیر این‌صورت،**
۱.۳.۳.۴.	کنترلگر خروج "document not available" را روی msg می‌نویسد،
۴.۳.۴.	**درغیر این‌صورت،**
۱.۴.۳.۴.	کنترلگر خروج "document not found" را روی msg می‌نویسد
۴.۴.	کنترلگر خروج پیام msg را به checkout GUI برمی‌گرداند،
۵.۴.	checkout GUI پیام msg را به عضو نمایش می‌دهد.

راهنما: فاعل کنش فاعل شیئی که کنش روی آن انجام می‌شود دیگر داده‌ها/اشیا **عملگر ترکیبی**

شکل ۹-۱۰: سناریوی علامت‌گذاری شده‌ی یک گام غیربدیهی

۳. برای هر کدام از وظایف، شیئی که باید کار را انجام دهد، یعنی فاعل، را مشخص کنید.
۴. برای هر کدام از وظایف، شیئی که درخواست انجام وظیفه را صادر می‌کند، را شناسایی کنید.

در باقیمانده‌ی این بخش، فعالیت‌های طراحی که این تصمیم‌ها را اتخاذ می‌کنند معرفی خواهند شد. این یک فرایند تکراری است که با گام غیربدیهی شروع شده و آن را به تعدادی وظیفه می‌شکنند. هر کدام از این وظایف شکسته می‌شوند تا جایی که پالایش بیشتری نیاز نباشد. گام بعدی این وظایف را به ترتیب مطلوب می‌چینند تا اجرا شوند.

مثال ۹-۴

تنها گام غیربدیهی در شکل ۹-۸ گام ۴ است. این کار با خروج مدرک شروع می‌شود که نام مورد کاربرد هم هست. این وظیفه را به تعدادی زیروظیفه که از فرایند کسب‌وکاری برنامه‌ی کتابخانه استنتاج شده‌اند، می‌شکنیم. برای مثال، وظیفه‌ی خروج مدرک می‌بایست مدارک را از پایگاه‌داده بازیابی نماید، اشیای Loan را ایجاد کند، مدارک را به‌روزرسانی نماید، و اشیای مدرک و Loan را در پایگاه‌داده ذخیره نماید. این وظیفه همچنین یک پیغام خروج برمی‌گرداند که محتوای این پیغام در مشروح گام ۴ معلوم می‌شود. این زیروظایف به شرح زیر فهرست می‌شوند:

#	فاعل	کنش فاعل	دیگر داده‌ها/اشیا	شیئی که کنش روی آن انجام می‌شود
۳.	عضو	وارد می‌کند	شماره مدرک	واسط گرافیکی خروج (Checkout GUI)
۱.۴.	واسط گرافیکی خروج	خارج می‌کند	شماره مدرک	کنترلگر خروج (checkout controller)
۲.۴.	کنترلگر خروج	ایجاد می‌کند		پیغام (msg)
۳.۴.	برای هر کدام از شماره مدرک‌ها			
۱.۳.۴.	کنترلگر خروج	مدارک را دریافت می‌کند	شماره مدرک	مدیر پایگاه‌داده(DBMgr)
۲.۳.۴.	مدیر پایگاه‌داده	برمی‌گرداند	مدرک d	کنترلگر خروج
۳.۳.۴.	اگر مدرک وجود داشته باشد (d != null)			
۱.۳.۳.۴.	کنترلگر خروج	در دسترس بودن را بررسی می‌کند		مدرک d
۲.۳.۳.۴.	اگر سند در دسترس باشد			
۱.۲.۳.۳.۴.	کنترلگر خروج	ایجاد می‌کند	عضو p ، سند d	شئ Loan
۲.۲.۳.۳.۴.	کنترلگر خروج	در دسترس‌بودن را تنظیم می‌کند	نادرست	مدرک
۳.۲.۳.۳.۴.	کنترلگر خروج	ذخیره می‌کند	شئ Loan	مدیر پایگاه‌داده
۴.۲.۳.۳.۴.	کنترلگر خروج	ذخیره می‌کند	مدرک d	مدیر پایگاه‌داده
۵.۲.۳.۳.۴.	کنترلگر خروج	اضافه می‌کند	"checkout successful"	پیغام
۳.۳.۳.۴.	درغیر این‌صورت			
۱.۳.۳.۴.	کنترلگر خروج	اضافه می‌کند	"document not available"	پیغام
۴.۳.۴.	درغیر این‌صورت			
۱.۴.۳.۴.	کنترلگر خروج	اضافه می‌کند	"document not found"	پیغام
۴.۴.	کنترلگر خروج	برمی‌گرداند	پیغام	واسط گرافیکی خروج
۵.۴.	واسط گرافیکی خروج	نمایش می‌دهد	پیغام	عضو

شکل ۹-۱۱: جدول سناریو برای مورد کاربردی خروج مدرک

خروج مدرک

دریافت مدارک از پایگاه‌داده

ایجاد اشیای Loan

تعیین وضعیت غیرقابل دسترس برای مدارک

ذخیره‌ی مدارک و اشیای Loan درون پایگاه‌داده

بازگرداندن پیام خروج

یک تحلیل دقیق نشان می‌دهد که زیروظیفه‌ی «ذخیره‌ی مدارک و اشیای Loan درون پایگاه‌داده» را می‌توان به دو زیروظیفه به شکلی تجزیه کرد که هر کدام فقط یک کار را انجام دهند. این یکی از کاربردهای اصول طراحی جداسازی دغدغه‌ها و نیز چسبندگی بالا می‌باشد. نتیجه به صورت زیر درخواهد آمد:

خروج مدرک

دریافت مدارک از پایگاه‌داده

ایجاد اشیای Loan

تعیین وضعیت غیرقابل دسترس برای مدارک

ذخیره‌ی اشیای Loan درون پایگاه‌داده

ذخیره‌ی مدارک درون پایگاه‌داده

بازگرداندن پیام خروج

فعالیت بعدی، شناسایی شیئی است که هر کدام از این وظایف را اجرا خواهد کرد. برای انجام این کار، از کاربرگ[۱] نشان داده شده در شکل ۹-۱۲ استفاده می‌شود. وظیفه و زیروظایف آن در سومین ستون درج می‌شوند که در آخرین ردیف آن، گام غیربدیهی نشان داده خواهد شد. هر کدام از وظایف را بررسی کرده و مشخص کنید کدام شئ باید آن وظیفه را اجرا کند. نتیجه را در آخرین ستون یا همان ستون «شیئی که کنش روی آن انجام می‌شود»[۲] وارد کنید. در ضمن ورودی لازم را نیز مشخص کنید. برای تعیین این‌که کدام شئ وظیفه را انجام می‌دهد، در محل‌های زیر به دنبال آن بگردید.

#	فاعل	کنش فاعل	دیگر داده‌ها/اشیا	شیئی که کنش روی آن انجام می‌شود
		خروج مدرک		
		دریافت مدارک از پایگاه داده		
		ایجاد اشیای Loan		
		تعیین وضعیت غیرقابل دسترس برای مدارک		
		ذخیره اشیای Loan درون پایگاه داده		
		ذخیره مدارک درون پایگاه داده		
		بازگرداندن پیام خروج		
		نمایش پیام خروج		

شکل ۹-۱۲: کاربرگ برای توسعه سناریو

۱. نخست، در دومین و آخرین ستون از کاربرگ جاری به‌دنبال شئ بگردید. این‌ها اشیایی هستند که در سناریوی فعلی نقش دارند. اگر شئ موردنظر یافت شد، نیازی به معرفی اشیا دیگر در سناریوی فعلی نیست. این طراحی و پیاده‌سازی را ساده می‌کند.
۲. دوم، در نمودار کلاس طراحی به‌دنبال شئ بگردید. نمودار کلاس طراحی، که در فصل ۱۱ معرفی خواهد شد، کلاس‌هایی از اشیایی را تعریف می‌کند که در سناریوهای سایر مورد کاربردها نقش دارند. بنابراین، این‌ها را بعداً باید در نظر بگیریم. اگر شئ موردنظر یافت شد، نیازی به معرفی دیگر اشیا در نمودار کلاس طراحی نمی‌باشد. این تعداد کلاس‌هایی که باید پیاده‌سازی شوند را کاهش می‌دهد.
۳. سوم، در مدل دامنه به دنبال شئ بگردید. مدل دامنه‌ی اشیای کسب‌وکار، اولویت‌ها و روابط بین آن‌ها را مشخص می‌کند. این مدل منبعی با ارزش برای شناسایی شیئی است که وظیفه را به انجام می‌رساند.
۴. سرانجام، در جایی دیگر به‌دنبال شئ بگردید. اگر شئ در محل‌های فوق پیدا نشد، در محل‌های دیگر مانند کد یا مستندات کسب‌وکاری مرتبط به‌دنبال آن بگردید.

---

[۱] Worksheet

[۲] Object Acted Upon

مثال ۹-۵

اولین وظیفه در شکل ۹-۱۲ خروج مدرک است. کدام شئ باید این‌کار را انجام دهد؟ یک نامزد احتمالی CheckoutGUI است زیرا می‌داند چه موقع عضو روی دکمه‌ی Submit کلیک می‌کند. ولی، واگذاری این وظیفه به CheckoutGUI ممکن است بار شئ CheckoutGUI را با مسئولیت‌هایی که ربطی به یک شئ GUI ندارند سنگین کند. یک طراحی بهتر این است که این وظیفه به کنترلگر خروج (Checkout Controller) سپرده شود. این طراحی به CheckoutGUI اجازه می‌دهد که روی مسئولیتش تمرکز کند- یعنی، ارائه‌ی اطلاعات به کاربر. برای خروج مدرک، کنترلگر نیاز به دانستن شماره مدرک دارد. بنابراین، ورودی مورد نیاز شماره‌ی مدرک است.

دومین وظیفه در شکل ۹-۱۲ دریافت مدارک از پایگاه‌داده است. این وظیفه، اتصال به پایگاه‌داده، پرس‌وجو از پایگاه‌داده، و پردازش نتایج پرس‌وجو را نیاز دارد. اگر این وظیفه به کنترلگر خروج واگذار شود، بار کنترلگر با عملیات مرتبط با پایگاه‌داده سنگین می‌شود. بنابراین، تصمیم این است که وظیفه به مدیر پایگاه‌داده واگذار شود. ورودی موردنیاز شماره‌ی مدارک است. به دلیل مشابه، دو وظیفه‌ی دیگر مرتبط با پایگاه‌داده، یعنی ذخیره‌ی مدارک و اشیای Loan نیز به مدیر پایگاه داده واگذار می‌شوند. ورودی لازم برای انجام این وظایف، به ترتیب مدارک و اشیای Loan هستند. اشیا Loan با فراخوانی سازنده‌ی کلاس Loan ایجاد می‌شوند. پارامترهای ورودی، عضو و مدارکی که باید امانت گرفته شوند می‌باشند. براساس روش برنامه‌سازی شئ‌گرا، وظیفه‌ی «تعیین وضعیت غیرقابل‌دسترسی برای مدارک» باید توسط خود مدارک انجام پذیرد. نتایج این تصمیمات طراحی در شکل ۹-۱۳ نشان داده شده‌اند.

حالا مشخص کنید که کدام شئ باید درخواست انجام هریک از وظایف را صادر نماید. به‌عنوان یک تجربه، شئ صادرکننده‌ی درخواست یا فاعلی است که در ردیف قبلی ظاهر شده است یا مفعولی است که در ردیف قبلی آمده است. جزییات این بحث در ذیل آمده است.

۱. اگر وظیفه‌ی ردیف جاری زیروظیفه‌ای از وظیفه‌ی ردیف قبلی است، شئ درخواست‌کننده مفعول ردیف قبلی است.
۲. اگر وظیفه‌ی ردیف جاری زیروظیفه‌ای از وظیفه‌ی ردیف قبلی نیست، شئ درخواست‌کننده فاعل یکی از ردیف‌های قبلی است. در بیشتر موارد، شئ درخواست‌کننده فاعل ردیف قبلی است.
۳. برای اولین وظیفه، شئ درخواست‌کننده شئ GUIی است که درخواست کنشگر غیربدیهی را دریافت می‌کند. نام این شئ GUI معمولاً براساس نام مورد کاربرد انتخاب می‌شود. به‌عنوان مثال، Login JSP برای مورد کاربرد Login، و CheckoutGUI برای مورد کاربرد Checkout Document انتخاب می‌شود.

مثال ۹-۶

اولین وظیفه در شکل ۹-۱۲ خروج مدارک است. کدام شئ باید درخواست انجام این وظیفه را صادر نماید؟ جواب CheckoutGUI است زیرا همان‌طور که در گام ۳ از شکل ۹-۸ مشاهده می‌شود، اوست که درخواست کنشگر برای خروج مدارک را دریافت می‌کند. مورد بعدی این‌که، کدام شئ باید درخواست دریافت مدارک را صادر کند؟ از آن‌جا که دریافت مدارک زیروظیفه‌ی خروج مستندات است، کنترلگر خروج باید این درخواست را صادر نماید. زیرا این مفعول ردیف قبلی است. کنترلگر همچنین درخواست پنج وظیفه‌ی بعدی را نیز صادر می‌نماید. سرانجام، کدام شئ باید پیغام خروج را صادر کند؟ از آن‌جا که وظیفه‌ی مورد نظر زیروظیفه‌ای از وظیفه‌ی ردیف قبلی نیست، باید فاعل یکی از ردیف‌های قبلی باشد. در

#	فاعل	کنش فاعل	دیگر داده‌ها/اشیا	شیئی که کنش روی آن انجام می‌شود
		خروج مدرک	شماره مدارک	کنترلگر خروج (checkout controller)
		دریافت مدارک از پایگاه داده	شماره مدارک	مدیر پایگاه‌داده (DBMgr)
		ایجاد اشیای Loan	عضو، مدارک	امانت (loan)
		تعیین وضعیت غیرقابل دسترس برای مدارک		مدرک (document)
		ذخیره اشیای Loan درون پایگاه داده	اشیای Loan	مدیر پایگاه‌داده (DBMgr)
		ذخیره مدارک درون پایگاه داده	مدارک	مدیر پایگاه‌داده (DBMgr)
		بازگرداندن پیام خروج		واسط گرافیکی خروج (Checkout GUI)
		نمایش پیام خروج		

شکل ۹-۱۳: واگذاری وظایف به اشیا

این مورد، این Checkout GUI از ردیف قبلی است. نتایج این تصمیمات طراحی در شکل ۹-۱۴ نشان داده شده است.

برای تکمیل این سناریو، دو فعالیت دیگر نیز انجام می‌شود. اول، برای هر وظیفه که یک نتیجه برمی‌گرداند، ردیفی برای بازگرداندن نتیجه از شیئی که کنش روی آن صورت می‌گیرد به شئ متقاضی درج کنید. دو وظیفه از این دست در شکل ۹-۱۴ مشاهده می‌شود- خروج مدرک و دریافت مدارک. وظیفه‌ی دریافت مدارک، همان‌طور که نام این وظیفه پیشنهاد می‌دهد، باید مدارکی که درخواست شده‌اند را برگرداند. بنابراین، بعد از ردیف دریافت مدارک، ردیفی که نشان می‌دهد DBMgr مدارک را به کنترلگر خروج برمی‌گرداند، اضافه می‌شود. وظیفه‌ی خروج مدارک باید یک پیغام خروج برگرداند زیرا CheckoutGUI پیغام را به عضو نمایش می‌دهد. بنابراین، ردیفی که لازم است بیانگر این باشد که کنترلگر خروج یک پیغام خروج را به CheckoutGUI برمی‌گرداند. این ردیف از قبل وجود دارد. بنابراین، هیچ ردیف جدیدی اضافه نخواهد شد. فعالیت دوم، در صورت نیاز دستورات شرطی و حلقه را در محل‌های مناسب درج می‌کند. سرانجام، شماره‌ی دستورات در اولین ستون وارد می‌شوند. نتایج این فعالیت‌ها در شکل ۹-۱۵ نشان داده شده‌اند.

#	فاعل	کنش فاعل	دیگر داده‌ها/اشیا	شیئی که کنش روی آن انجام می‌شود
	واسط گرافیکی خروج	خروج مدرک	شماره مدارک	کنترلگر خروج (checkout controller)
	کنترلگر خروج	دریافت مدارک از پایگاه داده	شماره مدارک	مدیر پایگاه‌داده (DBMgr)
	کنترلگر خروج	ایجاد اشیای Loan	عضو، مدارک	امانت (loan)
	کنترلگر خروج	تعیین وضعیت غیرقابل دسترس برای مدارک		مدرک (document)
	کنترلگر خروج	ذخیره اشیای Loan درون پایگاه داده	اشیای Loan	مدیر پایگاه‌داده (DBMgr)
	کنترلگر خروج	ذخیره مدارک درون پایگاه داده	مدارک	مدیر پایگاه‌داده (DBMgr)
	کنترلگر خروج	بازگرداندن پیام خروج		واسط گرافیکی خروج (Checkout GUI)
	واسط گرافیکی خروج	نمایش پیام خروج		

شکل ۹-۱۴: تکمیل اشیای درخواست‌کننده

۹-۳-۶- استنتاج نمودارهای توالی از جداول سناریو

نوشتن سناریوها با استفاده از زیرمجموعه‌ای از زبان طبیعی نسبتاً آسان است. اما، قالب متنی محدودیت‌هایی دارد. نخست این‌که، به سادگی نمی‌توان دید که کدام شئ چه پیغامی را به کدام اشیا دیگر می‌فرستد و در برگشت، اشیای فرستنده چه چیزی را از اشیا گیرنده بازخواهند گرفت؟ دوم این‌که، به‌سادگی نمی‌توان دید که شئ گیرنده وقتی یک پیغام را دریافت می‌کند چه کارهایی

#	فاعل	کنش فاعل	دیگر داده‌ها/اشیا	شیئی که کنش روی آن انجام می‌شود
.۳	واسط گرافیکی خروج	خروج مدرک	شماره مدارک	کنترلگر خروج
.۱.۴	کنترلگر خروج	دریافت مدارک	شماره مدارک	مدیر پایگاه‌داده
.۲.۴	مدیر پایگاه‌داده	بازگرداندن مدارک		کنترلگر خروج
.۳.۴	برای هر مدرک مثل d			
.۱.۳.۴	اگر d وجود دارد			
.۱.۱.۳.۴	کنترلگر خروج	ایجاد شئ Loan	عضو d و	Loan
۲.۱.۳.۴	کنترلگر خروج	تعیین وضعیت دسترس برای مدارک	false	d
۳.۱.۳.۴	کنترلگر خروج	ذخیره	شئ Loan	مدیر پایگاه‌داده
۴.۱.۳.۴	کنترلگر خروج	ذخیره	d	مدیر پایگاه‌داده
۵.۱.۳.۴	کنترلگر خروج	نوشتن	d با موفقیت خارج شد	پیام خروج
۲.۳.۴	درغیر این‌صورت			
۱.۲.۳.۴	کنترلگر خروج	نوشتن	d غیرقابل دسترس است	پیام خروج
.۴.۴	کنترلگر خروج	بازگرداندن پیام خروج		واسط گرافیکی خروج
.۵.۴	واسط گرافیکی خروج	نمای پیام خروج		

شکل ۹-۱۵: تکمیل توصیف سناریو

انجام می‌دهد. تولید طراحی نرم‌افزار از روی توصیف متنی و غیررسمی سناریوها کار ساده‌ای نیست. سرانجام، سبک غیررسمی توصیف سناریو برای وارسی طراحی و تولید اسکلت کد مناسب نیست. از آن‌جا که جدول سناریو تنها یک نمایش جدولی از یک سناریو است، همین مسائل در مورد جدول سناریو نیز صادق است. برای غلبه بر این مشکلات، برای توسعه‌دهنده مفیدتر آن است که سناریوهای تعامل اشیا را با نمودارهای توالی UML به تصویر بکشد. این‌کار شامل گام‌های زیر می‌شود:

گام ۱-۴- تبدیل جداول سناریو به نمودارهای توالی. در این گام، از روی هر جدول سناریو یک یا چند نمودار توالی غیررسمی تولید می‌شود. برای انجام این وظیفه، قوانین تبدیل نشان داده شده در شکل ۹-۱۶ قابل استفاده خواهد بود.

گام ۲-۴- تصمیم‌گیری در مورد نام و گونه‌ی نمونه‌ها. در این گام، نام و گونه‌ی نمونه‌ی اشیا که پیغام‌ها را ارسال و دریافت می‌کنند، تعریف می‌شوند.

گام ۳-۴- تصمیم‌گیری در مورد واسط اشیا. در این گام، نام توابع، پارامترها، و گونه‌های برگشتی آن‌ها مشخص می‌شوند.

در تبدیل جداول سناریو به نمودارهای توالی از قانون‌های تبدیل ذکر شده در شکل ۹-۱۶ استفاده خواهد شد. به‌ویژه، هر ردیف از جدول سناریو به یک پیغام که بین دو شئ یا بین یک کنشگر و یک شئ ردوبدل می‌شود، ترجمه خواهد شد. چهار حالت وجود خواهد داشت، که همان‌طور که در شکل ۹-۱۶ نشان‌داده شده است به‌طرز متفاوتی مورد پردازش قرار خواهند گرفت.

حالت ۱) فاعل، یک کنشگر می‌باشد. در این حالت، شیئی که کنش روی آن انجام می‌شود فقط می‌تواند یک شئ باشد. این حالت یک کنشگر را نشان می‌دهد که یک درخواست به سیستم نرم افزاری می‌فرستد. درخواست به‌نحوی از طریق صفحه‌کلید، ماوس، صفحه‌ی لمسی[۱] ، یا سایر دستگاه‌های سخت‌افزاری و ورودی وارد می‌شود. این به‌صورت یک پیغام استریوتایپ از کنشگر به سیستم نرم‌افزاری درخواهد آمد. همان‌طور که در مورد ۱ از شکل ۹-۱۶ مشاهده می‌شود، پیغام استریوتایپ یک پیکان خط‌چین است که برچسب آن درون یک جفت گیومه (یا پرانتزهای شکسته) محصور شده‌است.

[۱] Touch Screen

حالت ۲) فاعل یک شئ است و شیئی که کنش روی آن انجام می‌شود یک کنشگر است. این حالت بیان‌گر وقتی است که سیستم پاسخ سیستم را به کنشگر ارسال می‌نماید. بیشتر اوقات این‌کار با نمایش یک فهرست، یک جعبه‌ی گفت‌وگو، یا اطلاعاتی روی صفحه‌نمایش انجام می‌گیرد. همان‌طور که در مورد ۲ از شکل ۹-۱۶ نشان داده شده است، این به‌صورت یک پیغام استریوتایپ از سیستم به کنشگر درخواهد آمد.

حالت ۳) فاعل یک شئ بوده و شیئی که کنش روی آن انجام می‌شود نیز یک شئ است. این حالت یک فراخوانی تابع از یک شئ به شئ دیگر را نشان می‌دهد و عموماً با عنوان تبادل پیغام[1] از آن یاد می‌شود. فراخوانی می‌تواند شامل یک نتیجه‌ی برگشتی از شئ فراخوانده شده به شیئی که فراخوانی را صادر کرده است باشد. این حالت به صورت یک پیکان ممتد دارای برچسب متنی از صدازننده به مخاطب نشان داده می‌شود. نتیجه‌ی برگشتی، درصورت وجود، به‌صورت یک پیکان خط‌چین که برچسب آن نتیجه‌ی برگشتی است و از مخاطب به صدازننده برمی‌گردد، مدل‌سازی می‌شود. مورد ۳ از شکل ۹-۱۶ نشان می‌دهد که چگونه باید این حالت را تبدیل به نمودار کنیم. پیکان خط‌چین برای تمایز بین نتیجه‌ی بازگشتی از فراخوانی تابع، که با پیکان ممتد نشان داده می‌شود، استفاده می‌گردد. توجه کنید که استفاده از پیکان خط‌چین در این حالت و در حالت‌های ۱ و ۲ باعث ابهام نخواهد شد چون در این حالت دو شئ نرم‌افزاری درگیر هستند در حالی که در حالت‌های ۱ و ۲ یک شئ و یک کنشگر درگیر هستند.

حالت ۴) فاعل و شیئی که کنش روی آن صورت می‌گیرد هر دو یک شئ یکسان هستند. این حالت بیانگر یک فراخوانی از یک تابع از یک شئ به تابع دیگری از همان شئ است. همان‌طور که در آخرین مورد از شکل ۹-۱۶ نشان داده شده است، این به‌صورت یک پیکان ممتد از شئ به خودش مدل‌سازی می‌شود.

مثال ۹-۷

یک نمودار توالی از روی جدول سناریوی شکل ۹-۱۷ بسازید.

جواب: جدول سناریو به نمودار توالی نشان داده شده در شکل ۹-۳ تبدیل می‌شود.

در ادامه یک مثال پیچیده‌تر از تبدیل جدول سناریو به نمودار توالی را می‌بینید.

مثال ۹-۸

از جدول سناریوی شکل ۹-۱۱ یک نمودار توالی بسازید.

جواب: شکل ۹-۱۸ یک نمودار توالی غیررسمی که از جدول سناریوی خروج مدرک در شکل ۹-۱۱ به‌دست آمده است را نشان می‌دهد.

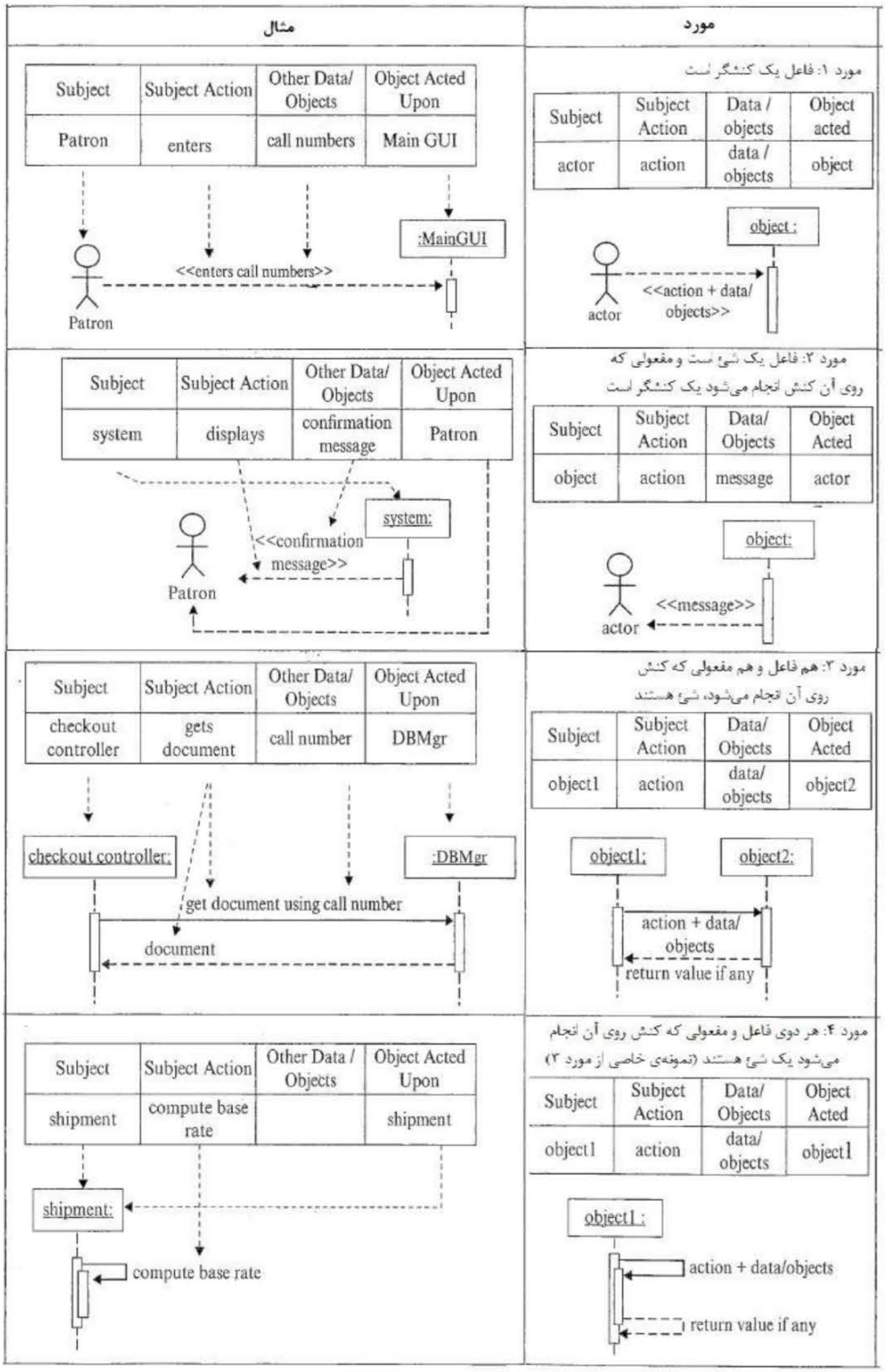
تصمیم‌گیری در مورد نام و گونه‌ی نمونه‌ها

به‌خاطر آورید که وقتی جدول سناریو به نمودار توالی تبدیل می‌شود، نام و گونه‌ی نمونه‌ها مشخص نمی‌شوند. این گام، نام و گونه‌ی نمونه‌ها را با اعمال قاعده‌های ذیل مشخص می‌کند.

۱. اگر نمونه‌ای به‌عنوان پارامتر یا مقدار برگشتی در نمودار توالی به‌کار رود، به نمونه نامی اختصاص بدهید که این نام بتواند برای رجوع به آن نمونه به‌کار برده شود.

---

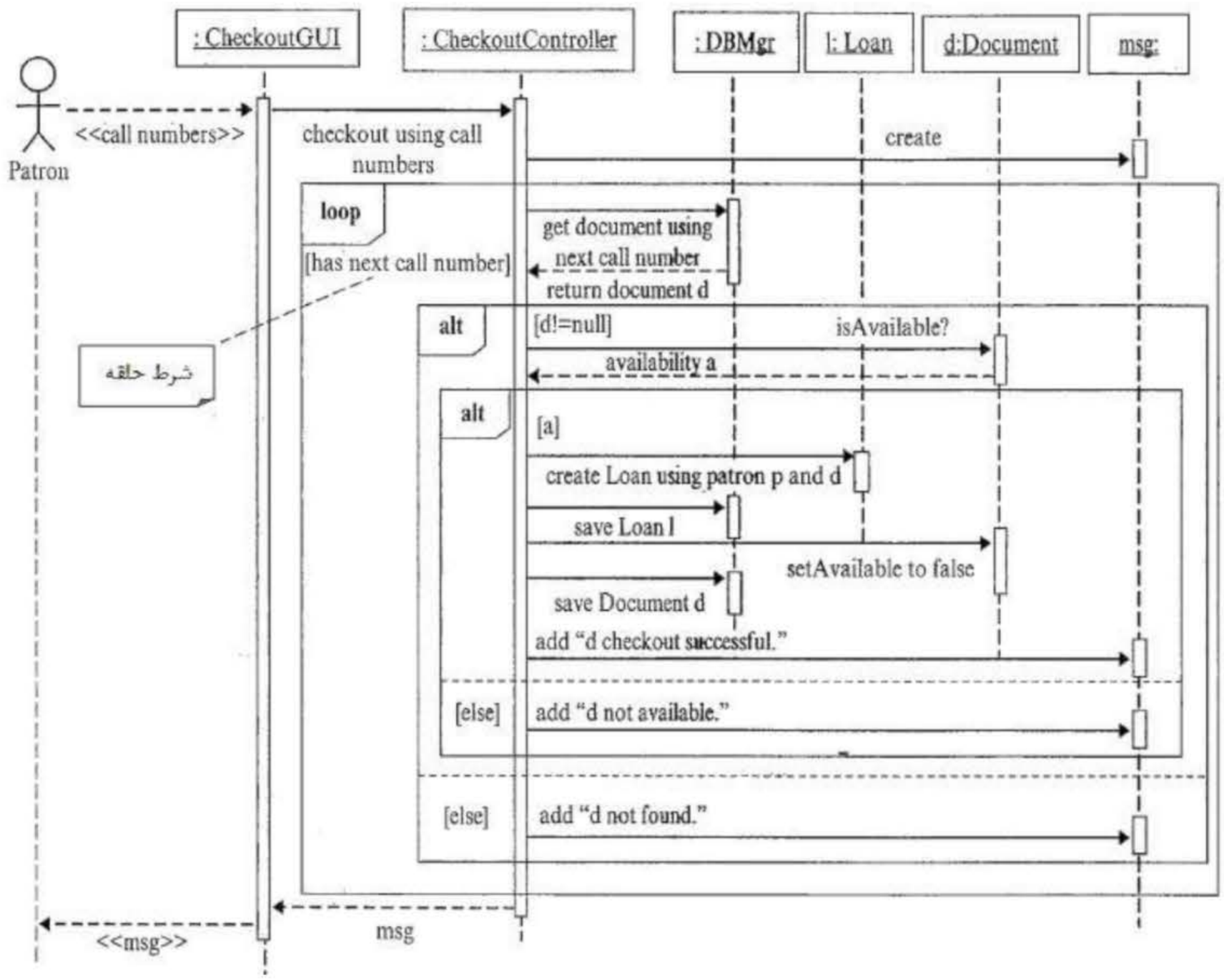
[1] Message Passing



شکل ۹-۱۶: از جدول سناریو به نمودار توالی

#	فاعل	کنش فاعل	داده‌ها/اشیای مورد نیاز	شیئی که کنش روی آن انجام می‌شود
۱.	کاربر	ارسال می‌کند	نام کاربری، رمز عبور	صفحه ورود (LoginPage)
۲.	صفحه ورود	وارسی می‌کند	نام کاربری، رمز عبور	کنترلگر ورود (LoginController)
۳.	کنترلگر ورود	کاربر را دریافت می‌کند	نام کاربری	مدیر پایگاه داده (DBMgr)
۴.	مدیر پایگاه داده	بازمی‌گرداند	نام کاربری	کنترلگر ورود
۵.	کنترلگر ورود	وارسی می‌کند	رمز عبور	کاربر (User)
۶.	کاربر	بازمی‌گرداند	نتیجه وارسی	کنترلگر ورود
۷.	کنترلگر ورود	بازمی‌گرداند	نتیجه وارسی	صفحه ورود
۸.	اگر نتیجه وارسی درست است، صفحه ورود	هدایت می‌کند		صفحه خوش‌آمدگویی (WelcomePage)
۹.	اگر نتیجه وارسی نادرست است، صفحه ورود	نمایش می‌دهد	پیام خطا	کاربر
۱۰.	کاربر	می‌بیند	پیام خطا/ صفحه ورود	

شکل ۹-۱۷: جدول سناریوی Login همراه با شماره ردیف



شکل ۹-۱۸: یک نمودار توالی غیررسمی دارای حلقه

۲. اگر نمونه، نمونه‌ای از یک کلاس نباشد -اگر یک صفحه‌ی JSP باشد- به آن نمونه نامی بدهید و آن را یک نمونه‌ی استریوتایپ‌دار قرار دهید. به نمونه، نام کلاس ندهید زیرا این باعث ابهام می‌شود.

قوانین زیر برای تعیین گونه‌ی نمونه‌ها به‌کار می‌روند.

۱. وقتی در مورد کلاس یک نمونه تصمیم‌گیری می‌کنید، اول در نمودار توالی جاری، بعد در نمودار کلاس طراحی، و سرانجام

در مدل دامنه جست‌وجو کنید، سپس یک کلاس جدید معرفی کرده و به آن یک نام کلاس بامعنی بدهید.

۲. وقتی روی کلاس مربوط به عناصر یک اجتماع تصمیم‌گیری می‌کنید، دو مورد مختلف در نظر گرفته می‌شوند:

الف) اگر عناصر اجتماع تنها نمونه‌هایی از یک کلاس باشند، آن کلاس عناصر اجتماع خواهد بود.

ب) اگر عناصر اجتماع نمونه‌هایی از زیرکلاس‌های یک ابرکلاس باشند، ابرکلاس به‌عنوان کلاس عناصر اجتماع تعیین می‌شود.

تصمیم‌گیری در مورد واسط اشیا

این بخش در مورد چگونگی تبدیل پیام‌هایی که به صورت غیررسمی تبیین شده‌اند به پیام‌هایی که به صورت رسمی تبیین شده‌اند می‌باشد. در واقع بحث تبدیل نمودارهای توالی غیررسمی یا تحلیل، به نمودارهای توالی طراحی می‌باشد. لازم به ذکر است که در اکثر حالات تنها لازم است پیغام‌هایی که بین دو شئ نرم‌افزاری ردوبدل شده است در نظر گرفته شود، یعنی لازم نیست پیغام‌هایی که بین یک شئ و یک کنشگر ردوبدل می‌شوند درنظر گرفته شوند. استثنا در این‌جا وقتی است که کنشگر یک جزء یا زیرسیستم نرم‌افزاری است که نیاز به یک واسط دارد که به‌صورت رسمی تعریف شده باشد. دو حالت باید در نظر گرفته شوند. حالت اول وقتی است که کنشگر نرم‌افزار قبلاً وجود دارد. در این حالت باید واسط خارجی تعریف شده باشد و تیم توسعه موظف است از آن واسط استفاده کند. حالت دیگر این است که سیستم نرم‌افزاری دردست ساخت بتواند واسط کنشگر خود را به‌صورت یک واسط برنامه‌نویسی کاربردی[۱] برای مشتریان بالقوه تعریف نماید. در این حالت، تیم توسعه باید در تعریف واسط کنشگر آزاد باشد.

پیغامی که به‌صورت غیررسمی بیان شده باشد به‌صورت زیر به یک پیغام رسمی ترجمه می‌شود. کنش فاعل به فراخوانی تابع، و داده‌ها یا اشیایی که موردنیاز کنش فاعل هستند به پارامتر تبدیل می‌شوند. اگر تابع نتیجه‌ای هم برمی‌گرداند، آن نتیجه درون یک متغیر که به همین منظور معرفی شده است ذخیره می‌شود. اگر بیش از یک قلم اطلاعات برگردانده می‌شود، در این‌صورت فراخوانی‌های اضافی به توابع دریافت[۲] متناسب معرفی می‌شود تا نتایج را دریافت کند. در بیشتر مواقع، تبدیل‌ها ساده بوده و توسط یک عضو تیم قابل انجام است.

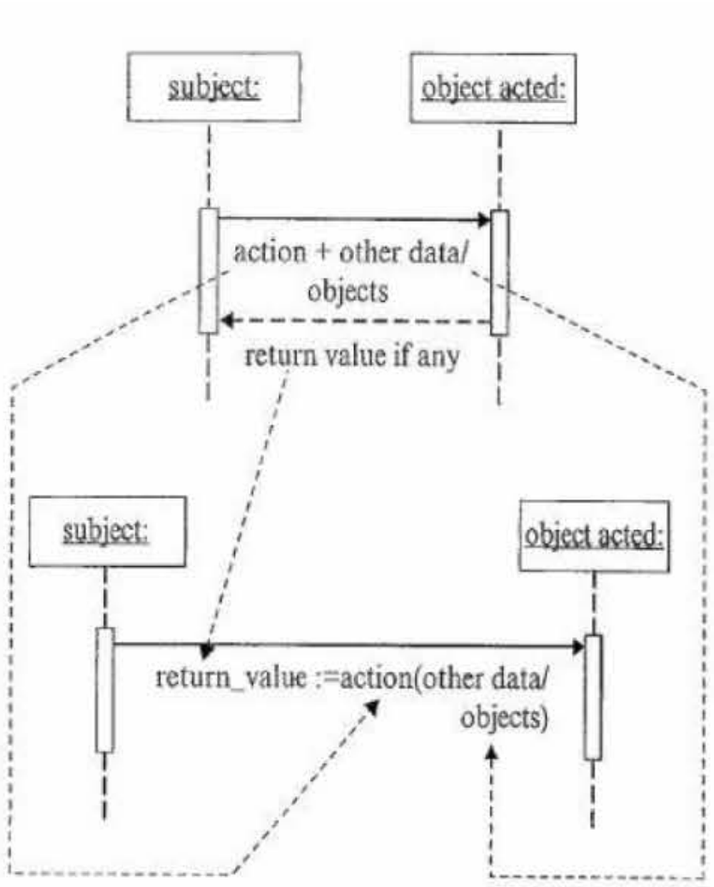
شکل ۹-۱۹ قوانین و مثال‌های تبدیل را نشان می‌دهد. شکل ۹-۱۹-الف قانون کلی و شکل ۹-۱۹-ب یک مثال را نشان می‌دهد. شکل ۹-۱۹-ج موردی را نشان می‌دهد که نتیجه‌ی برگشتی ندارد. شکل ۹-۱۹-د موقعیتی را به تصویر می‌کشد که چنان‌چه فراخوانی تابع جست‌وجو مقدار درست برگرداند، نتایج جست‌وجو در یک گام جداگانه بازیابی می‌شوند. شکل ۹-۱۹-ه گاهی مواقع به‌عنوان راه‌حل جایگزینی برای شکل ۹-۱۹-د درنظر گرفته می‌شود. بدین معنی که، به جای این‌که SearchJSP متد getPrograms() از searchController را فراخوانی نماید، یک فراخوانی از searchController به searchJSP به‌کار می‌رود. ولی از این راه‌حل باید پرهیز نمود. نخست، این راه‌حل یک جفت‌شدگی حلقوی بین searchController و searchJSP ایجاد می‌نماید. این جفت‌شدگی محکم بازاستفاده از نرم‌افزار را سخت‌تر می‌کند. هرگونه بازاستفاده از searchController مجبور است searchJSP را هم دخیل کند و برعکس، بدون توجه به این‌که دیگری مورد نیاز باشد یا خیر. دوم، این راه‌حل رابطه‌ی مشتری-سرور را تغییر می‌دهد- یعنی، از این به بعد نه searchController دیگر سرور است و نه searchJSP دیگر مشتری است.

تصمیم‌گیری درباره‌ی گونه‌ی پارامترها و گونه‌ی برگشتی‌ها یک مسئله‌ی وابسته به کاربرد و نیز یک ملاحظه‌ی طراحی نرم‌افزار است. بدین‌معنی که، دانش دامنه مورد نیاز است و این دانش می‌تواند در مدل دامنه یافت شود. این‌که می گوییم یک

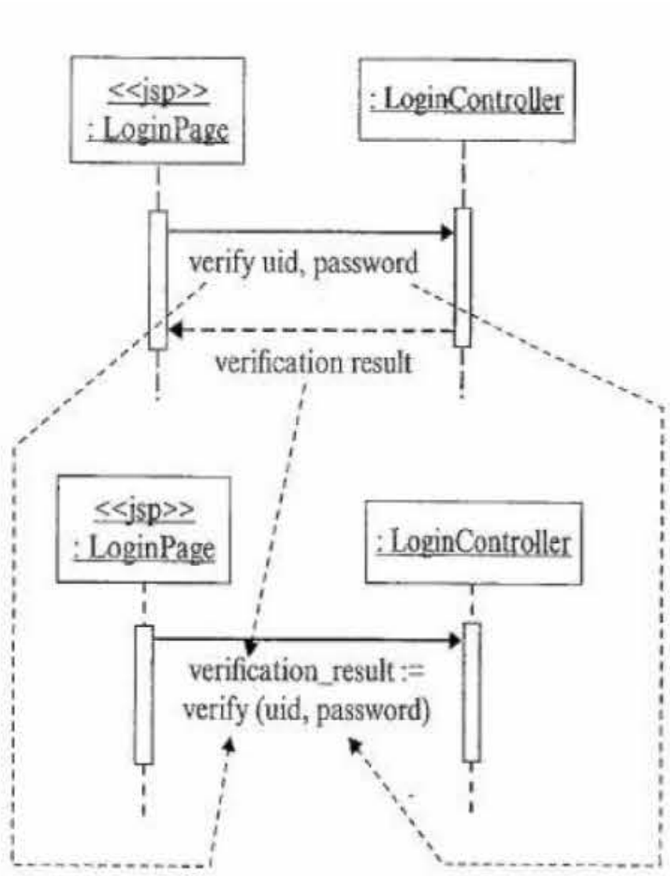
---

[۱] Application Programming Interface (API)

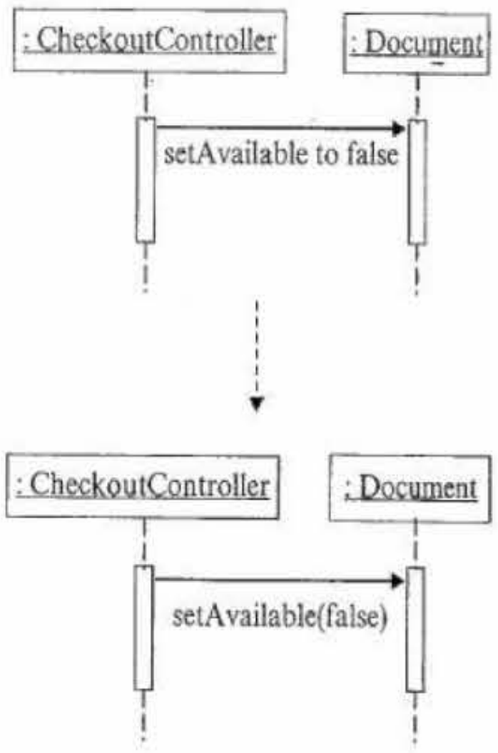
[۲] Get Functions



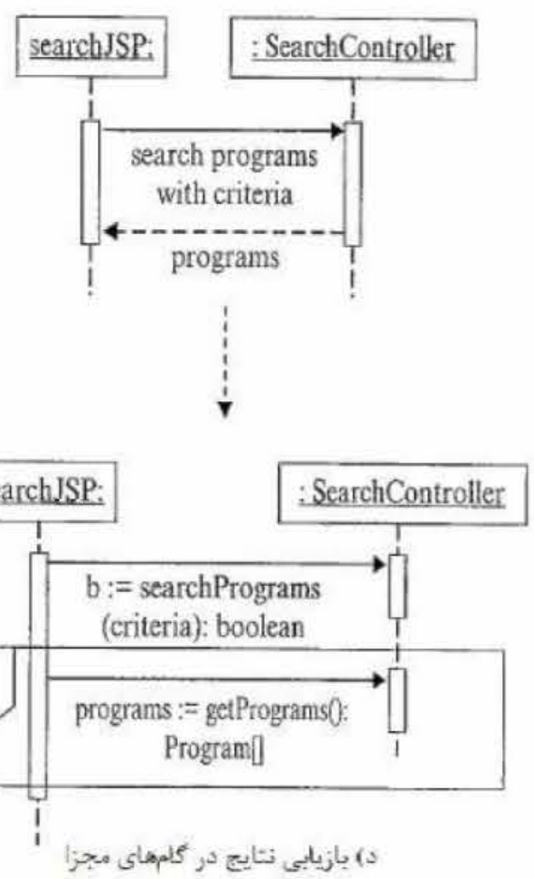
الف) مثالی از قاعده‌ی تبدیل عمومی



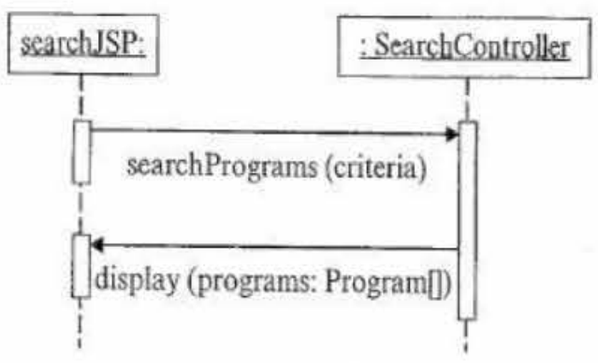
ب) یک مثال



ج) یک مثال دیگر



د) بازیابی نتایج در گام‌های مجزا



هـ) جواب نادرستی برای مورد (د)

شکل ۹-۱۹: تصمیم‌گیری درباره‌ی نام، پارامترها، و متغیرهای برگشتی توابع

مسئله‌ی طراحی است بدین خاطر است که روش‌های مختلفی برای تبادل داده میان دو شئ وجود دارند. از این لحاظ، مطلوب این است که به جفت‌شدگی پایینی دست یابیم، همان‌گونه که در رهنمون‌های زیر توصیف شده است.

رهنمون ۹-۱

هر کجا می‌توانید و مناسب است از جفت‌شدگی داده‌ای (Data Coupling) استفاده کنید.

منظور از جفت‌شدگی داده‌ای این است که یک مقدار داده به‌عنوان پارامتر یا مقدار برگشتی از یک تابع به تابع دیگر منتقل می‌شود. مقدار داده فقط وقتی در یک محاسبه استفاده می‌شود که نتیجه‌ای تولید شود، نه این‌که در انتخاب یک مسیر یا جریان کنترلی در یک برنامه استفاده شود.

رهنمون ۹-۲

جفت‌شدگی شئ (Object Coupling) را بر جفت‌شدگی تمبر (Stamp Coupling) ترجیح دهید. مطمئن شوید که توابعی که بروزرسانی انجام می‌دهند، بررسی‌های لازم برای یکپارچگی را انجام می‌دهند.

جفت‌شدگی تمبر بدین معنی است که یک داده‌ساختار به‌عنوان پارامتر یا مقدار برگشتی از یک تابع به دیگری منتقل شود. به‌طرز مشابه، جفت‌شدگی شئ بدین معنی است که یک شئ بین دو تابع منتقل شود. اغلب اوقات این داده‌ساختار معانی و مفروضاتی دارد. این نکات ممکن است بدیهی هم نباشد. بنابراین، ممکن است از داده‌ساختار به طرز نادرستی استفاده شود. جفت‌شدگی شئ ترجیح داده می‌شود، چون توابع دستیابی[۱]، مانند getX() و setX() می‌توانند بررسی‌های یکپارچه‌سازی را پیاده‌سازی کنند تا تضمین شود که داده‌ساختار به‌درستی به‌کار رفته است.

رهنمون ۹-۳

چنان‌چه چند صفت از یک شئ به‌عنوان پارامتر به یک فراخوانی تابع انتقال یابد، از جفت‌شدگی شئ استفاده کنید.

رهنمون ۹-۴

از جفت‌شدگی کنترلی (Control Coupling) و جفت‌شدگی عمومی (Common Coupling) پرهیز کنید و هرگز جفت‌شدگی خارجی (External Coupling) را استفاده نکنید.

جفت‌شدگی کنترلی بدین معنی است که دو تابع با یک متغیر کنترلی که مسیر برنامه یا جریان کنترلی در زمان اجرا را تعیین می‌کند به هم جفت شده‌اند. برای مثال، پارامتری که به‌عنوان متغیر Case در دستور Switch به‌کار می‌رود یک متغیر کنترلی است. مقدار این پارامتر، دستور Caseای که باید اجرا شود را تعیین می‌کند. جفت‌شدگی کنترلی بدین معنی است که رفتار یک شئ توسط شئ دیگری کنترل می‌شود. پیش بینی، آزمایش، و اشکال‌زدایی از رفتار زمان اجرا بسیار دشوار است. بنابراین، باید از جفت‌شدگی کنترلی پرهیز نمود.

منظور از جفت‌شدگی عمومی این است که توابع به واسطه‌ی یک ناحیه‌ی داده‌ی سراسری یا یک متغیر سراسری با هم در ارتباطند. به‌روزرسانی همزمان چنین متغیرهایی ممکن است رفتار غیر قابل انتظاری را تولید نماید. در برنامه‌سازی شئ‌گرا،

---

[۱] Accessor Functions

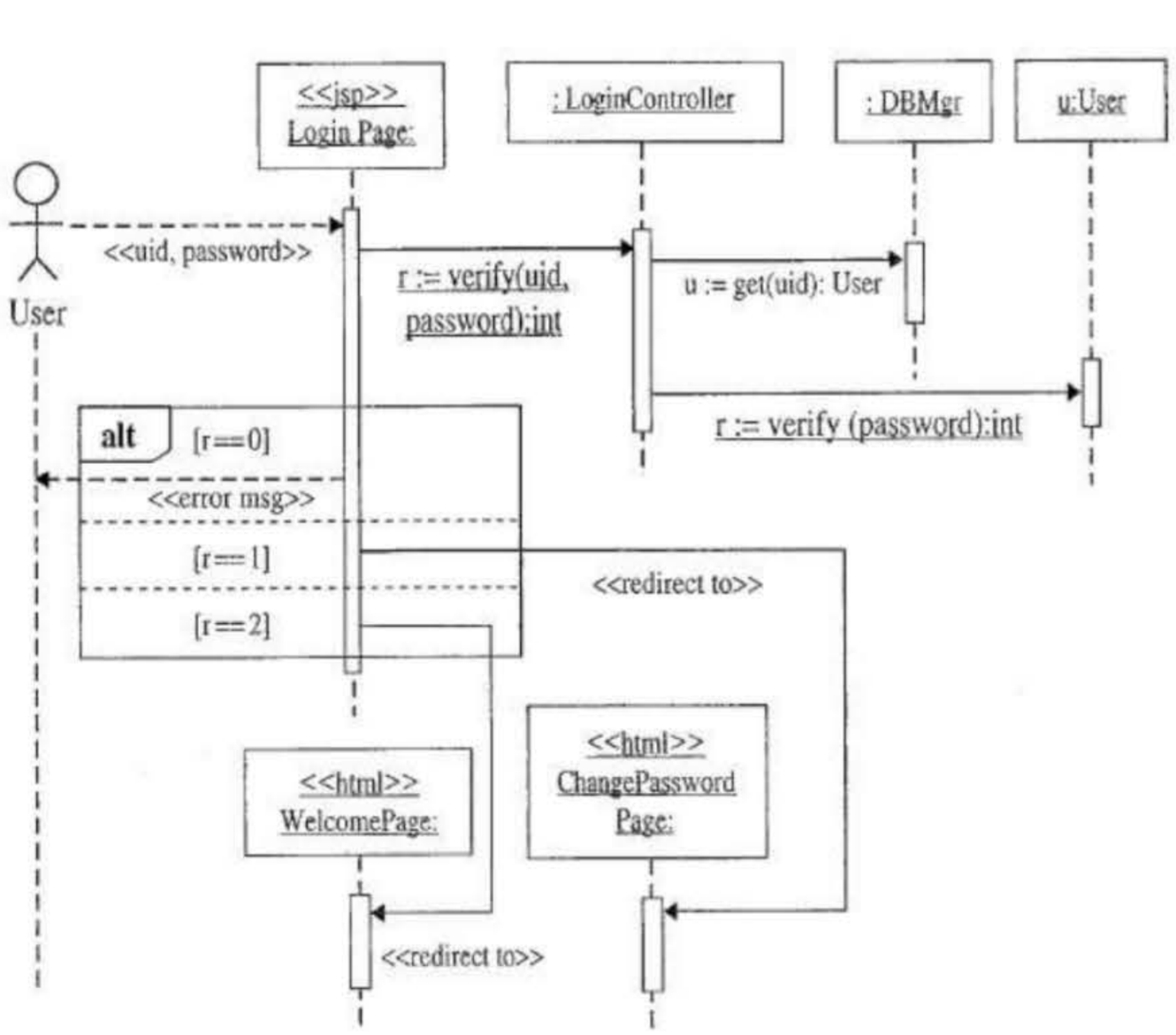
استفاده از یک داده‌ی عضو ایستای عمومی[۱] باعث جفت‌شدگی عمومی می‌شود زیرا آن داده عضو می‌تواند توسط تمام اشیا مورد دستیابی و به‌روزرسانی قرار گیرد. جفت‌شدگی خارجی[۲] بدین معنی است که توابع از طریق فضای حافظه‌ای یک دستگاه جانبی (خارجی) با یکدیگر ارتباط برقرار می‌نمایند. از این نوع جفت‌شدگی نیز باید پرهیز نمود زیرا فضای حافظه‌ای خارجی می‌تواند توسط برنامه‌های خارجی تغییر کند که منجر به رفتار غیرمنتظره خواهد شد.

مثال ۹-۹

در مورد طراحی شکل ۹-۲۰ بحث کرده و بهبودهای لازم را اعمال کنید.

جواب: طراحی شکل ۹-۲۰ جفت‌شدگی کنترلی را براثر استفاده از متغیر کنترلی r به نمایش می‌گذارد. مقدار r توسط شئ User محاسبه می‌شود و بر روی رفتار صفحه‌ی JSP اثر می‌گذارد. شکل ۹-۲۱ نمودار توالی بهبودیافته را نشان می‌دهد که در آن متغیر کنترلی سه مقداری r با دو متغیر منطقی (بولی) جایگزین شده است. بدیهی است که معنی متغیرهای منطقی و نمودار توالی واضح‌تر است.

فرض کنید تابع وارسی از کلاس User به‌گونه‌ای تغییر کرده باشد که به‌جای سه مقدار، چهار مقدار برگرداند، بدین‌صورت که منظور از مقدار چهارم این باشد که مشخص کند حساب کاربر مجدداً فعال شده است. فرض کنید به هر دلیلی معنی مقادیر بازگشتی از نو تعیین شده باشند؛ مثلاً ۰ به معنی ورود معتبر، ۱ به معنی تغییر کلمه‌عبور، ۲ به معنی ورود نامعتبر، و ۳ به معنی فعال شدن مجدد حساب کاربر باشد. در این حالت، سیستمی که برطبق طراحی شکل ۹-۲۰ پیاده‌سازی شده باشد دیگر کار نمی‌کند. ولی، سیستمی که برطبق طراحی شکل ۹-۲۱ پیاده‌سازی شده باشد هم‌چنان قادر به کار خواهد بود. به جز این‌که مورد اخیراً اضافه شده تا وقتی طراحی و صفحه‌ی JSP به‌تناسب تغییر پیدا نکند، در جریان نخواهد بود.

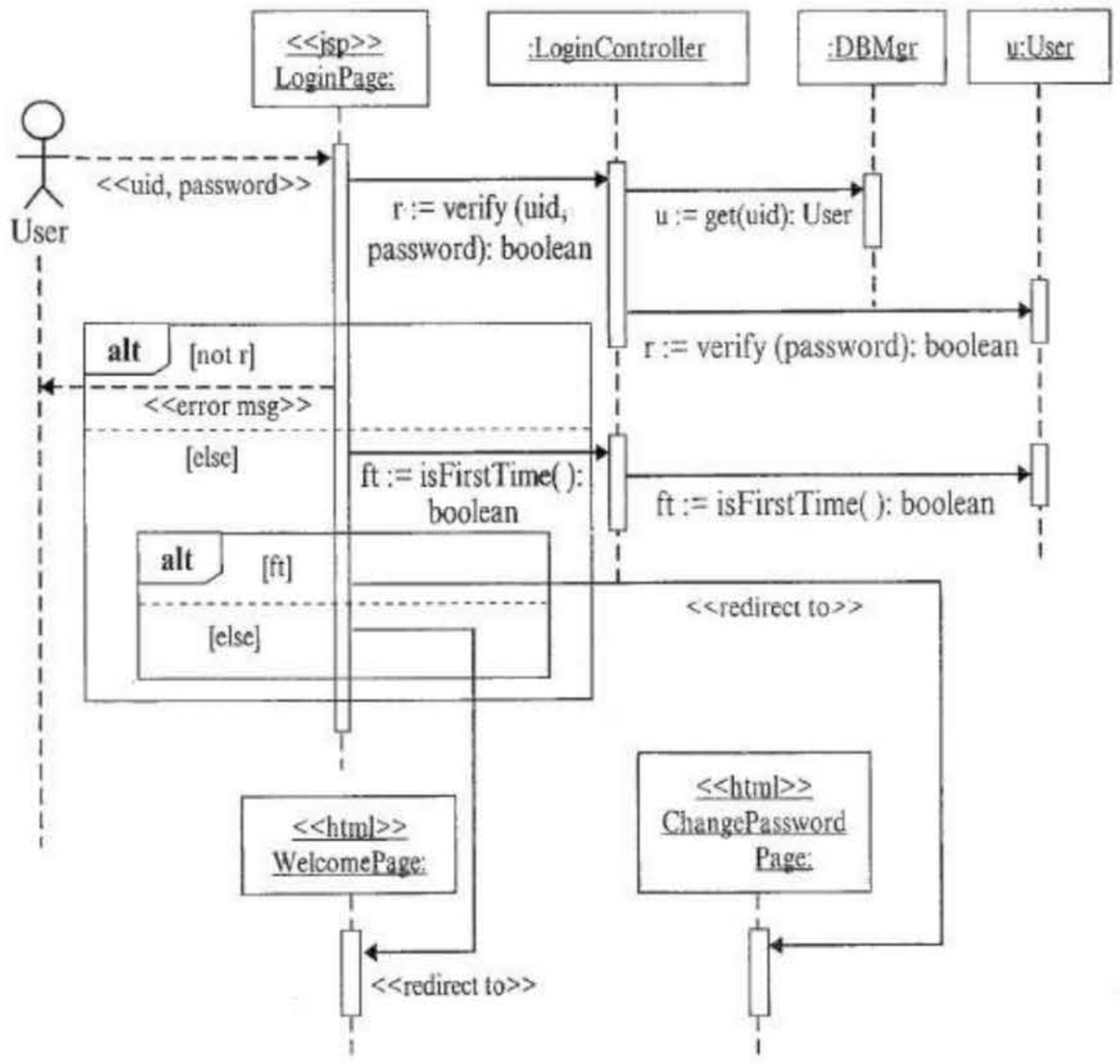


شکل ۹-۲۰: اشیایی که از یک متغیر کنترلی به‌عنوان واسط استفاده می‌کنند.

---

[۱] Public Static Data Member

[۲] External Coupling



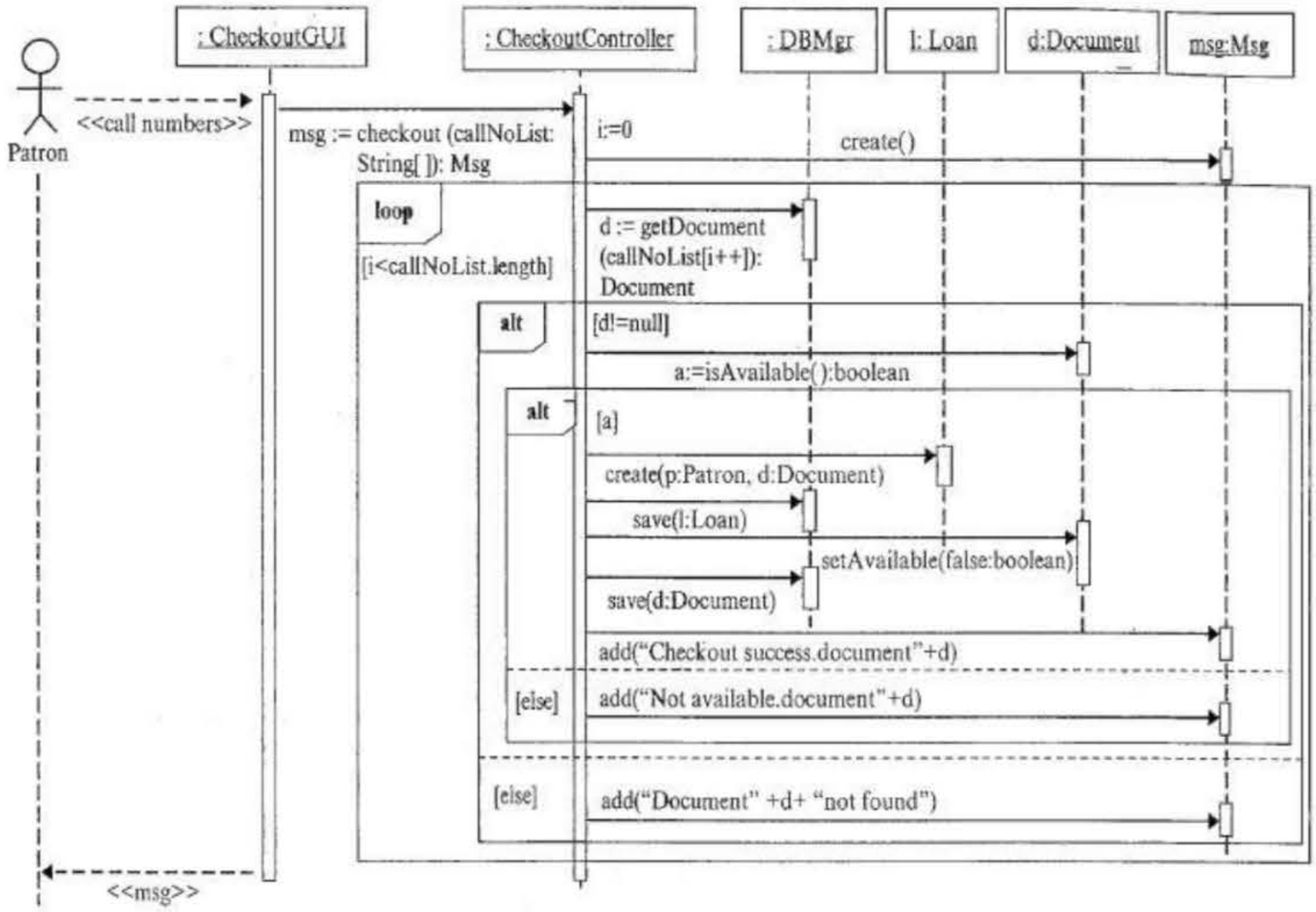
شکل ۹-۲۱: بهبودیافته: جفت‌شدگی کنترلی حذف شده است.

مثال ۹-۱۰

نمودار توالی غیررسمی شکل ۹-۱۸ را به یک نمودار توالی که پیام‌ها در آن به‌طور رسمی مشخص شده‌باشد تبدیل کنید.
جواب: شکل ۹-۲۲ نمودار توالی که در اثر تبدیل به‌طور رسمی مشخص شده است را نشان می‌دهد.

۹-۳-۷- فهرست بررسی جهت بازبینی مدل‌سازی تعامل شئ

۱. مطمئن شوید که سناریوها و نمودارهای توالی به‌درستی و به‌اندازه‌ی کافی فرایندهای کسب‌وکاری بهبودیافته را توصیف می‌نمایند.
۲. مطمئن شوید که تمام پیغام‌های یک نمودار توالی طراحی به‌طور رسمی مشخص شده‌اند و از خطوط پیکان خط‌چین برای برگرداندن نتیجه به یک شئ استفاده نشده است.
۳. مطمئن شوید که تمام پارامترهای لازم و مقادیر بازگشتی وجود دارند و پارامترها قبل از این‌که در یک فراخوانی تابع مورد استفاده قرار گیرند، تعریف شده‌اند.
۴. مطمئن شوید که هیچ شیئی بدون این‌که در نهایت در پایگاه‌داده ذخیره شود از پایگاه‌داده بازیابی و به‌روزرسانی نمی‌شود.
۵. مطمئن شوید که نام توابع به درستی کارکرد آن‌ها را می‌رساند.
۶. مطمئن شوید که هر نمودار توالی تمام شرایط زیر را برآورده می‌سازد.



شکل ۹-۲۲: نمودار توالی خروج مدرک که در آن پیام‌ها به‌طور رسمی مشخص شده‌اند.

الف) نمودار توالی صحبت می‌کند؛ یعنی، نمودار توالی از نظر نحوی درست است. برای مثال، شکل‌های ۹-۶ (ب) الی (ه) از نظر نحوی درست نیستند.

ب) نمودار توالی ایده‌های درستی را بیان می‌کند؛ بدین معنی که نمودار توالی به‌درستی الگوریتم یا سناریو را توصیف می‌کند.

.

۹-۴ اعمال اصول چابکی

مدل‌سازی ابزاری مفید برای بهبود درک و ارتباط میان اعضای تیم است. مدل‌های ایجاد شده بخش مهمی از مستندات سیستم‌اند که خود برای نگهداری سیستم بسیار باارزش است. ولی، مدل‌سازی زیاده از حدمی‌تواند بیش‌تر مضر باشد تا مفید. مدل‌سازی باید منظور موردنظر را برآورده کند؛ تولید نمودار صرفاً به‌منظور کسب برخی گواهینامه‌ها، اتلاف زمان و تلاش است. در این بخش، با در نظرگرفتن اصول مدل‌سازی چابک، رهنمون‌هایی در مورد مدل‌سازی تعامل شئ ارائه می‌گردد.

رهنمون ۹-۵

با کاربران و مشتریان به‌طور تنگاتنگ کار کنید تا فرایندهای کسب‌وکاری ایشان و به‌خصوص این‌که چگونه کسب‌وکار تراکنش‌ها را پردازش می‌کند را بفهمید.

توسعه‌ی چابک بر مشارکت فعال کاربر در حین کل فرایند توسعه تاکید دارد. به‌ویژه، با مشارکت فعال کاربر، اطلاعات در خصوص فرایندهای کسب‌وکار موجود را می‌توان آسان‌تر و دقیق‌تر کسب نمود.این اطلاعات برای ساخت مدل‌های تعامل شئ، که به تیم توسعه کمک می‌کنند فرایندهای جاری را درک کنند، مفید است. همچنین کاربران به تیم توسعه کمک می‌کنند تا فرایندهای کسب‌وکار جاری را تحلیل و مشکلات را شناسایی نموده و راه‌حل‌های قابل اتکا تولید نمایند. فایده‌ی دیگر مشارکت فعال کاربران این است که تایید کنند آیا راه‌حل کامپیوتری ارائه شده مشکلات کسب‌وکاری‌شان را حل می‌کند یا خیر.باید توجه داشت که گرچه فهم نمودارهای توالی UML برای تیم توسعه نسبتاً آسان است ولی این‌کار برای اغلب کاربران ساده نیست. بنابراین، برای ارتباط با کاربران باید به‌جای نمودارهای توالی از سناریو بهره جست.

رهنمون ۹-۶

اعضای تیم با همدیگر سناریو را نوشته و جداول سناریو را می‌سازند. این کارها را به صورت انفرادی انجام ندهید.

داشتن درک مشترک از فرایندهای کسب‌وکار در میان اعضای تیم، برای موفقیت پروژه‌ی نرم‌افزاری بحرانی است. کار گروهی برای تولید سناریوها یک روش مؤثر برای ایجاد چنین درک مشترکی است. در حین فرایند، اعضای تیم دیدگاه‌ها و ایده‌های طراحی خود را مطرح و بر روی مسائل بحث می‌کنند. سناریوها و جداول سناریو نتایج نهایی خواهند بود. یک تفاوت عمده در این است که فرایند منجر به نتیجه می‌شود ولی نتیجه، معادل فرایندی که باعث حصول درک مشترک شده، و بحرانی‌ترین بخش مدل‌سازی می‌باشد، نخواهد بود. یک راه‌حل دیگر برای تهیه‌ی سناریوها با همدیگر این است که اجازه دهیم هر یک از اعضای تیم سناریوها را به‌طور جداگانه بنویسند. این روش هم باعث تولید سناریوها می‌شود، ولی فرایند فراموش شده است. این مسئله اغلب در پروژه‌های تیمی دانشجویی اتفاق می‌افتد که در آن اعضای تیم نمی‌توانند یا نمی‌خواهند با همدیگر کار کنند، لذا کار را بین خودشان تقسیم می‌کنند. چنین پروژه‌هایی اغلب به شکست می‌انجامد چون سناریوهایی که توسط اعضای مختلف تهیه شده است با یکدیگر سازگار نیستند. برای تصحیح چنین مشکلاتی وقت و تلاش بسیار زیادی لازم است. تیم در مراحل طراحی، پیاده‌سازی، یکپارچه‌سازی، و آزمون با مسائل و مشکلات عدیده‌ای مواجه می‌شود زیرا درک مشترکی وجود نداشته و ناسازگاری‌ها فراوان است.

رهنمون ۹-۷

چنان‌چه اعضای تیم سناریویی را به‌خوبی می‌فهمند نیازی به ساخت سناریو، جدول سناریو، یا نمودار توالی نیست.

اصل «مدل‌سازی به حد کافی»[1] می‌گوید که مدل‌سازی فقط وقتی انجام می‌شود که لازم باشد. بنابراین، اگر تیم درک مناسبی از سناریو دارد، از مدل‌سازی تعامل شئ برای آن سناریو می‌توان صرف‌نظر نمود مگر این‌که مستندسازی اجباری باشد. این بدین خاطر است که هدف از مدل‌سازی کمک به فهم و ارتباط می‌باشد. چنان‌چه سناریو به قدر کافی ساده است یا تیم تجربه‌ی قبلی در آن رابطه دارد، می‌توان از مدل‌سازی صرف‌نظر کرد.

به‌عنوان مثال، مورد کاربرد گسترده Update Configuration که در شکل ۹-۲۳ نشان داده شده است را در نظر بگیرید. دو گام غیربدیهی در آن وجود دارد. سناریوی گام ۲ وقتی شروع می‌شود که Update Config GUI متد getConfig():Config از Update Config Controller را فرامی‌خواند، که آن هم متد getConfig():Config از مدیر پایگاه‌داده را فرامی‌خواند. سناریو وقتی خاتمه می‌یابد که Update Config GUI جعبه‌ی گفت‌وگوی Update Configuration را نمایش می‌دهد. این سناریو

[1] Barely Enough Modeling

ساده و سرراست است. اگر متدهای getConfig():Config از کنترلگر و مدیر پایگاه‌داده قبلاً تعریف شده باشند، در این‌صورت ساخت این سناریو ضرورتی ندارد. به همین ترتیب، ساخت سناریو برای گام ۴ نیز لازم نیست.

کنشگر: کاربر (User)	سیستم: LIS
	۰. LIS فهرست اصلی را نشان می‌دهد.
۱. TUCBW وقتی که کاربر روی دکمه‌ی Update Configuration از فهرست اصلی کلیک می‌کند.	۲. LIS جعبه‌ی گفتگوی Update Configuration را نشان می‌دهد.
۳. کاربر اطلاعات پیکربندی را تغییر داده و روی دکمه‌ی Save کلیک می‌کند.	۴. LIS جعبه‌ی پیام Update Configuration را نشان می‌دهد.
۵. TUCEW وقتی کاربر روی دکمه‌ی OK از جعبه‌ی پیام Update Configuration کلیک می‌کند.	

گام‌های غیر بدیهی

شکل ۹-۲۳: مورد کاربرد گسترده با دو گام غیربدیهی

رهنمون ۹-۸

تبدیل سناریو یا جدول سناریو به نمودار توالی باید جداگانه انجام شود و به‌عنوان وظیفه‌ای تک نفره در نظر گرفته شود.

وقتی سناریو یا جدول سناریو تولید شد، ترسیم نمودار توالی ساده‌ترین کار است. در واقع، نمودار توالی می‌تواند توسط یک کامپیوتر تولید شود. بنابراین، ترسیم نمودارها باید توسط یکی از اعضای تیم و به‌طور جداگانه انجام شود.

رهنمون ۹-۹

ساده و ساده‌لوح بودن را حفظ کنید. جزییات برنامه‌سازی را از سناریوها و نمودارهای توالی حذف کنید.

سناریوها شامل جملاتی هستند که اعمالی که اشیا انجام می‌دهند را پالایش می‌کنند. به‌عنوان مثال، گام‌های ۳-۳-۴ الی ۱-۴-۳-۴ در شکل ۹-۹ را در نظر بگیرید که توصیف دقیقی از این‌که کنترلگر خروج چگونه مدرک برگشتی توسط شئ DBMgr را پردازش می‌کند، ارائه می‌نماید. اگر این گام‌ها برای اعضای تیم بدیهی هستند، می‌توان آن‌ها را با گام زیر جایگزین نمود:

۴-۳-۳-کنترلگر خروج، مدرک را پردازش نموده و msg به‌روز شده را برمی‌گرداند.

به این ترتیب، جزییات از طراحی حذف شده و به پیاده‌سازی داده می‌شود. شکل ۹-۲۴ توصیف سناریوی ساده شده را نشان می‌دهد.

۳. عضو شماره‌ی مستنداتی که می‌خواهد به امانت ببرد را وارد کرده و روی دکمه‌ی submit کلیک می‌کند.
۱.۴. واسط خروج (Checkout GUI) بر اساس شماره‌ی مستندات، آن‌ها را با کنترلگر خروج بررسی می‌کند.
۲.۴. کنترلگر خروج یک پیغام خالی msg ایجاد می‌کند.
۳.۴. برای هر کدام از شماره مستندات،
۱.۳.۴. کنترلگر خروج با استفاده از شماره‌ی مستند، آن را از مدیر پایگاه‌داده(DBMgr) درخواست می‌کند.
۲.۳.۴. DBMgr سند d را به کنترلگر خروج برمی‌گرداند.
۳.۳.۴. کنترلگر خروج سند را پردازش کرده و پیام msg را به‌روزرسانی می‌کند.
۴.۴. کنترلگر خروج، پیام msg را به واسط خروج برمی‌گرداند.
۴.۵. واسط خروج پیام msg را به عضو نشان می‌دهد.

شکل ۹-۲۴: توصیف یک سناریوی ساده شده

رهنمون ۹-۱۰

با استفاده از پیش‌نمونه‌سازی تردیدها را برطرف کنید.

مدل‌سازی تعامل شئ، مدل‌سازی رفتاری است. نوشتن سناریو مانند نوشتن الگوریتمی است که پردازش پس‌زمینه را انجام می‌دهد. تفاوت در اینجاست که مدل‌سازی تعامل شئ همان برنامه‌نویسی برحسب تبادل پیام بین اشیا می‌باشد. یکی از چالش‌ها در طراحی الگوریتم این است که بعضی مواقع فرد نمی‌داند آیا الگوریتم مسئله‌ی مورد نظر را حل می‌کند و کارایی خوبی دارد یا خیر. در این گونه موارد، می‌توان یک پیش‌نمونه‌ی اجرایی ساخت و اطمینان حاصل نمود که الگوریتم اهداف طراحی‌اش را برآورده می‌نماید.

۹-۵ پشتیبانی ابزاری برای مدل‌سازی تعامل شئ

ابزارهایی برای ترسیم و مدیریت نمودارهای توالی وجود دارند، مانند IBM Rational Modeller، Microsoft Visio، ArgoUML، Net Beans UML Plugin. این ابزارها قابلیت‌های ویرایش و سایر امکانات مربوطه را در اختیار می‌گذارند، همچنین از مهندسی معکوس و تولید کد نیز پشتیبانی می‌کنند. یک ابزار OIM نیز در دانشگاه Texas at Arlington در دست تهیه است. این ابزار شناسایی دستی و خودکار فاعل، کنش فاعل، شئ مورد نظر و پارامترها را پشتیبانی می‌کند. این ابزار نمودار توالی را به‌طور خودکار تولید می‌نماید.

خلاصه

این فصل متدولوژی OIM و نمودار توالی UML را معرفی می‌نماید. سناریو و جدول سناریو تعریف می‌شوند تا ساخت نمودار توالی را تسهیل کنند. برای هر گام غیربدیهی از یک مورد کاربرد گسترده یک سناریو تولید می‌شود. سپس سناریو به جدول

سناریو تبدیل می‌شود، که آن‌هم به یک نمودار توالی غیررسمی تبدیل می‌گردد. در یک نمودار توالی غیررسمی، پیام‌هایی که به‌طور غیررسمی مشخص شده‌اند به پیام‌هایی که به‌طور رسمی مشخص شده باشند ترجمه شده و در نتیجه یک نمودار توالی طراحی حاصل می‌شود.

OIM یک گام مهم در تحلیل و طراحی شئ‌گرا است. این کار با تحلیل و طراحی یکی از سه جنبه‌ی رفتاری یک سیستم شئ‌گرا -یعنی، مدل‌سازی تعامل شئ، مدل‌سازی حالت، و مدل‌سازی فعالیت- سروکار دارد. این فصل اصول طراحی شئ‌گرای زیادی را در نظر نمی‌گیرد. این کار با منظور بوده است تا به دانشجویان اجازه دهد در هر دو زمان روی یک جنبه تمرکز نمایند. فصل بعدی می‌گوید که چگونه الگوهای طراحی واگذاری مسئولیت را در حین OIM به کار بندید. الگوهای طراحی دربرگیرنده‌ی اصول طراحی شئ‌گرا هستند. بنابراین، وقتی الگوهای طراحی به‌درستی به‌کار روند، اصول طراحی نیز اعمال شده‌اند.

برای مطالعه‌ی بیش‌تر

مفهوم سناریو در دهه‌ی ۱۹۷۰ میلادی معرفی شد. روش Object Modeling Technique (OMT) [۱۳۱] نیز برای توصیف تعامل میان اشیا از سناریو استفاده می‌کند. در UML1.0 نمودار ارتباط با عنوان نمودار همکاری نامیده می‌شد. کتاب نوشته شده توسط Booch, Rumbaugh, and Jacobson [۳۶] مقدمه‌ی خوبی بر نمودار توالی و نمودار ارتباط UML 2.0 ارائه می‌دهد. تبیین جامع‌تر را می‌توان در آدرس www.omg.org یافت. در OMT، هر سناریو به یک نمودار ردیابی رویداد ترجمه می‌شود که نسخه‌ی قدیمی‌تری از نمودار توالی است. OMT همچنین نمودار جریان رویداد را معرفی می‌کند که نسخه‌ی قدیمی‌تری از نمودار ارتباط است. Larman [۱۰۴] از یک نمودار توالی، که به آن نمودار توالی سیستم[۱] می‌گوید، برای برجسته کردن تعامل بین کنشگر و سیستم استفاده می‌کند. هر ورودی کنشگر به‌عنوان یک رویداد سیستم در نظر گرفته می‌شود. پردازش رویدادهای سیستم توسط نمودارهای توالی یا نمودارهای ارتباط اضافی مدل‌سازی می‌شود.

سوال‌های مروری فصل

۱. مدل‌سازی تعامل شئ چیست؟
۲. فایده‌ی مدل‌سازی تعامل شئ چیست؟
۳. نمودار توالی چیست؟
۴. رابطه‌ی بین مدل‌سازی تعامل شئ و نمودارهای توالی چیست؟
۵. نمودارهای توالی تحلیل و طراحی چیست؟ استفاده‌ی این نمودارها چیست؟
۶. رابطه‌ی بین مورد کاربرد گسترده و نمودار توالی طراحی در خصوص یک مورد کاربرد داده شده چیست؟
۷. سناریو و جدول سناریو چیست؟
۸. رابطه‌ی بین سناریو، جدول سناریو، نمودار توالی تحلیل، و نمودار توالی طراحی چیست؟
۹. گام‌های مدل‌سازی تعامل شئ کدامند؟
۱۰. اصول چابکی چگونه در حین مدل‌سازی تعامل شئ به کار می‌روند؟

[۱] System Sequence Diagram

تمرین‌ها

۹-۱ سناریویی برای مورد کاربرد «سفارش غذا»[1] در یک رستوران که به صورت دستی عمل می‌کند بنویسید. برای سادگی، فرض کنید که فقط یک غذا سفارش داده می‌شود.

۹-۲ فاعل‌ها، کنش فاعل‌ها، اشیایی که روی آن‌ها کنش انجام می‌شود، و داده‌ها و اشیایی که در سناریوی «سفارش غذا» موردنیاز کنش فاعل هستند را مشخص کنید.

۹-۳ سناریوی «سفارش غذا» را با استفاده از یک جدول سناریو بیان کنید.

۹-۴ یک نمودار توالی غیررسمی برای سناریوی «سفارش غذا» بسازید.

۹-۵ سناریوها، جداول سناریو، و نمودارهای توالی طراحی را برای مورد کاربردهای گسترده «واریز وجه»، «برداشت وجه»، «بررسی موجودی»، و «انتقال وجه» که برای کاربرد ATM در تمرین ۸-۱ تولید کردید بسازید.

۹-۶ برای کاربرد اجاره‌ی برخط خودرو که در پیوست ت توصیف شده است، سناریوها، جداول سناریو، و نمودارهای توالی طراحی را برای گام‌های غیر بدیهی از مورد کاربردهای گسترده «جست‌وجوی خودرو»، «رزرو خودرو»، «ویرایش رزرواسیون»، و «لغو رزرواسیون» که در تمرین ۸-۴ تولید کردید بسازید.

۹-۷ سناریوها، جداول سناریو، و نمودارهای توالی طراحی را برای گام‌های غیر بدیهی مورد کاربردهای گسترده‌ای که برای مسئله‌ی ویراستار نمودار حالت در تمرین ۸-۶ تولید کردید، بسازید.

۹-۸ فرض کنید که قرار است با استفاده از متدولوژی ارائه شده در این فصل یک ابزار نرم‌افزاری برای OIM بسازید. ابزار باید گام‌های متدولوژی بحث شده را پشتیبانی نموده و هر کدام از دست‌ساخته‌ها طراحی را که امکان دارد به‌طور خودکار تولید نماید. از جمله، تولید خودکار نمودارهای توالی از روی جداول سناریو باید انجام شود- یعنی، نیازی نیست کاربر نمودارهای توالی را رسم کند، گرچه کاربر ممکن است نیاز داشته باشد نمودارهای تولید شده را ویرایش نماید. برخی وظایف تحلیل و طراحی به شرح ذیل می‌باشند:

الف) فهرستی از نیازمندی‌های این ابزار نرم‌افزاری را تولید کنید.

ب) از روی نیازمندی‌ها مورد کاربردها را استنتاج کنید.

ج) ماتریس ردیابی‌پذیری نیازمندی- مورد کاربرد را بسازید.

د) برای این کاربرد یک مدل دامنه بسازید.

ه) مورد کاربردهای سطح بالا و گسترده را مشخص نمایید.

و) برای برخی از مورد کاربردهای غیر بدیهی OIM را انجام دهید.

کدام یک از گام‌های OMI می‌تواند کلاً یا در سطح بالا خودکار شود و چگونه؟

۹-۹ مورد کاربرد گسترده شکل ۹-۸ را تغییر دهید به‌گونه‌ای که به عضو اجازه دهد قبل یا بعد از ورود مورد کاربرد را شروع کند. در صورتی که عضو وارد نشده باشد، مورد کاربرد عضو را به مورد کاربرد ورود هدایت می‌کند. بعد از این‌که عضو وارد شد، مورد کاربرد فرایند امانت را از سر می‌گیرد. گام‌هایی که در این فصل تشریح شد را اجرا کنید و نمودار توالی طراحی برای مورد کاربرد گسترده تغییر یافته بسازید.

---

[1] Order a Dish

۱۰ اعمال الگوهای واگذاری مسئولیت

نکات کلیدی

- الگوهای طراحی تجرید راه حل‌های اثبات شده‌ی طراحی هستند برای مسائل طراحی که عموماً به آنها برخورد می‌کنیم.
- الگوهای کنترلگر[۱]، خبره[۲]، و خالق[۳] برای تمامی سیستم‌های شئ‌گرا قابل استفاده‌اند.

فصل ۹ یک متدولوژی پایه برای مدل‌سازی تعامل شئ ارائه نمود. به این متدولوژی پایه می‌گوییم چون اصول مختلف طراحی نرم‌افزار را دخیل نمی‌کند. این کار را به‌طور عمدی انجام دادیم تا فهم و پیروی از گام‌ها آسان باشد. این فصل نشان می‌دهد که در حین OIM چگونه باید الگوهای مشخصی از طراحی نرم‌افزار را اعمال نمود. همچنین اصولی از طراحی نرم‌افزار که مرتبط باشند مطرح می‌شوند تا به فهم این‌که یک طراحی خوب چیست، کمک کنند. به‌طور خاص، در این فصل مسائلی که در برخی از نمودارهای توالی طراحی معمول مشاهده می‌شوند، مورد بحث قرار می‌گیرند. هر کدام از مسائل از نقطه نظر نقض اصول طراحی نرم‌افزار تشریح می‌گردد.

برای حل این مسائل، الگوهای طراحی مختلفی معرفی شده‌اند و چگونگی اعمال این الگوها برای بهبود طراحی نیز نشان داده شده است. در طول این فصل، شما پاسخ سؤالات زیر را خواهید یافت:

- الگوی طراحی چیست؟
- الگوهای وابسته به موقعیت چیستند؟
- الگوهای عمومی نرم‌افزاری واگذاری مسئولیت[۴] کدامند؟
- فواید استفاده از الگوهای طراحی کدامند؟
- چگونه می‌توان با اِعمال سه تا از الگوهای GRASP یک طراحی را بهبود داده و چگونه می‌توان این بهبود را درک کرد؟

---

[۱] Controller
[۲] Expert
[۳] Creator
[۴] General Responsibility Assignment Software Patterns (GRASP)

۱۰-۱ الگوهای طراحی چیستند؟

بشر از زمان‌های قدیم از الگوهای طراحی استفاده می‌کرده است. برای مثال، کشاورزان از الگوی ابرها استفاده می‌کرده‌اند تا وضعیت آب و هوا را پیش‌بینی کنند و سهام‌داران بازار بورس از الگوهای نموداری استفاده می‌کنند تا تغییرات قیمت‌ها و سهام را پیش‌بینی نمایند. الگوهای طراحی در معماری برای اولین بار توسط کریستوفر الکساندر مورد مطالعه قرار گرفت. وی کشف نمود که ساختمان‌هایی که می‌توانند در مقابل بلایای طبیعی دوام بیاورند یک چیز مشترک دارند. تمام این ساختمان‌ها مجموعه‌ای از ایده‌های طراحی را به نمایش می‌گذارند که او آنها را به شکل الگوهای طراحی درآورد.

تعریف ۱۰-۱

الگوهای طراحی، تجریدهایی از راه حل های ثابت شده ی طراحی، برای مسایل معمول مطرح در طراحی هستند.

نخست این‌که برخلاف یک طراحی واقعی، الگو یک تجرید از طراحی است. این باعث می‌شود که یک الگو بتواند تعداد زیادی مسائل طراحی مشابه را حل نماید. هر راه‌حل نمونه‌ای از آن الگو خواهد بود. همچنین، هر الگو یک راه حل طراحی نیز هست؛ بدین معنی که یک مسئله‌ی طراحی یا یک کلاس از مسائل طراحی مشابه را حل می‌کند. مسئله‌ی طراحی برای الگو منحصر به فرد است. هر الگو یک راه حل طراحی اثبات شده است. یعنی، مؤثر بودن آن فقط یک ادعا نیست، بلکه در کاربردهای عملی نشان داده شده است. درنهایت، هر الگو یک مسئله‌ی طراحی که معمولاً رخ می‌دهد را حل می‌کند؛ و بنابراین، آن الگو را می‌توان بارها و بارها به کار برد تا بسیاری از مسائل طراحی مشابه را در طیف گسترده‌ای از کاربردها حل نمود.

جامعه‌ی مهندسی نرم‌افزار این ایده را گرفت و تحقیق و توسعه در زمینه‌ی الگوهای طراحی نرم‌افزاری در دهه‌ی ۱۹۸۰ میلادی آغاز شد. تا امروز، شناخته شده‌ترین و پرنفوذترین مجموعه‌ی الگوهای طراحی نرم‌افزاری همچنان ۲۳ الگویی است که به الگوهای چهار تفنگدار [۱] معروف هستند [۶۷]. به این الگوها، الگوهای چهار تفنگدار گویند چون کتاب دارای چهار مؤلف می‌باشد. هر الگو دارای نام بوده و حاوی تجریدی از مسئله‌ی طراحی و راه حل طراحی می‌باشد. برای مثال، الگوی Singleton مسئله‌ی طراحی زیر را حل می‌کند: «چگونه می‌توان کلاسی طراحی کرد که حداکثر یک نمونه‌ی سراسری قابل دستیابی دارد؟»

این مسئله‌ی طراحی در بسیاری از کاربردهای عملی متداول است. برای مثال، هر سیستم به یک شئ پیکربندی سیستم نیاز دارد و اجزای متعدد به اطلاعات پیکربندی احتیاج دارند. بنابراین، لازم است که این شئ به‌طور سراسری قابل دستیابی باشد. به‌عنوان کاربردهای دیگر از Singleton می‌توان به پرونده‌ی ثبت سیستم [۲] و کاتالوگ یک کتابخانه اشاره نمود. شکل ۱۰-۱-الف یک نمودار کلاس UML را نشان می‌دهد که الگو را توصیف می‌کند و شکل ۱۰-۱-ب یک پیاده‌سازی در جاوا را نشان می‌دهد. کلاس Subject کلاس مربوط به کاربرد است که باید حداکثر یک نمونه‌ی سراسری قابل دستیابی باشد، مثل کلاس پیکربندی سیستم، ثبت سیستم، کاتالوگ کتابخانه. این کلاس یک نمونه‌ی ایستای خصوصی از گونه‌ی خودش دارد. این نمونه در ابتدا نول [۳] است. کلاس Subject یک سازنده‌ی خصوصی نیز دارد. با این کار مطمئن خواهیم شد که دیگر اشیا قادر نیستند هیچ نمونه‌ای از کلاس Subject ایجاد کنند. تابع getInstance() نیز تضمین می‌کند که حداکثر یک نمونه از کلاس ایجاد شده است. این تابع به دیگر اشیا امکان می‌دهد که با فراخوانی‌های ایستا به تنها نمونه‌ی کلاس دسترسی پیدا کنند. همان‌گونه که در شکل نشان داده شده است، الگوها اغلب با استفاده از نمودار کلاس و گاهی نیز با استفاده از نمودارهای توالی توصیف می‌شوند. نمودار کلاس، مشارکت کنندگان در الگو، نقش‌ها و مسئولیت‌های ایشان، و چگونگی ارتباط‌شان با یکدیگر را نشان می‌دهد. نمودار توالی نشان می‌دهد که مشارکت کنندگان چگونه با یکدیگر تعامل می‌کنند تا مسئله‌ی طراحی را حل کنند.

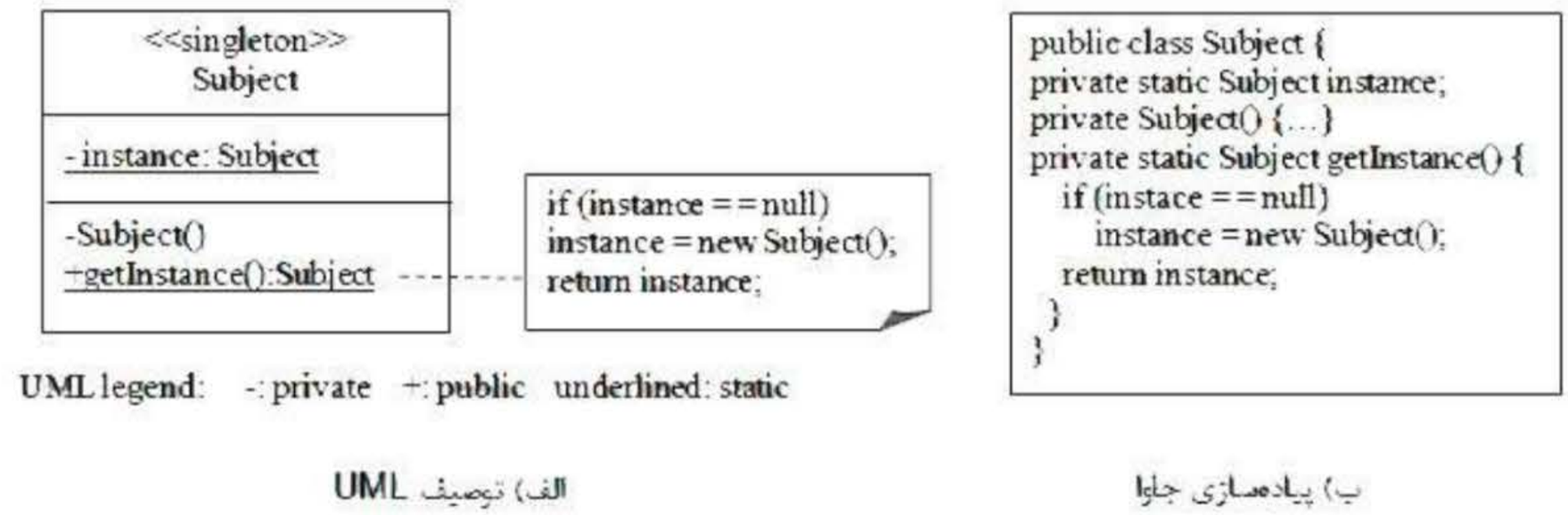
---

[۱] Gang of Four (GoF)

[۲] System Log File

[۳] null

همان‌طور که در شکل ۱۰-۱-الف نشان داده شده است، از جعبه‌های توضیح UML[1] نیز معمولاً استفاده می‌شود تا اطلاعات اضافی فراهم گردد.



الف) توصیف UML

ب) پیاده‌سازی جاوا

شکل ۱۰-۱: الگوی تک

۱۰-۲ چرا الگوهای طراحی؟

الگوها راه‌حل‌های ثابت‌شده‌ی طراحی برای مسائل طراحی متداول که معمولاً با آنها مواجه می‌شویم، هستند. به عبارت دیگر، الگوها عناصر نرم‌افزاری بازاستفاده‌پذیر هستند. الگوها را می‌توان برای مسائل طراحی بزرگ و پیچیده ترکیب نمود. علاوه بر این، الگوها فواید دیگری نیز دارند. نخست، الگوها باعث می‌شوند ارتباط بین اعضای تیم بهبود یابد زیرا نام الگوها به‌طور مؤثری مسائل طراحی و راه حل‌های آنها را منعکس می‌کند. بهبود در ارتباط منجر به بهبود در کار تیمی شده و اخلاق تیمی را بهبود می‌دهد. خیلی از الگوها اصول طراحی نرم‌افزاری را پیاده‌سازی می‌کنند. بنابراین، الگوها ساختار و رفتار سیستم‌های نرم‌افزاری را بهبود می‌بخشند. اجزایی که با الگوها طراحی و پیاده‌سازی شوند، فهم، آزمون، و نگهداری‌شان ساده است. الگوها طراحی‌های بازاستفاده‌پذیر را می‌سازند. بازاستفاده‌ی موفق اجزای نرم‌افزاری که به خوبی طراحی و آزمایش شده باشند بهره‌وری و کیفیت نرم‌افزار را بهبود می‌بخشند. تجربه‌های زیادی نشان می‌دهند که استفاده از الگوهای طراحی توانایی تیم توسعه در برخورد با مسائل طراحی پیچیده را معنادارانه بهبود می‌دهد. الگوها به توسعه دهندگان کم‌تجربه قدرت می‌دهند چون ایشان می‌توانند برای تولید نرم‌افزارهای باکیفیت از الگوها استفاده نمایند. به‌طور خلاصه، الگوها بهره‌وری و کیفیت نرم‌افزار را بهبود بخشیده و هزینه‌ی نرم‌افزار و زمان تا تحویل محصول به بازار را کاهش می‌دهند.

۱۰-۳ الگوهای خاص موقعیت و واگذاری- مسئولیت

الگوهای GoF الگوهای خاص- موقعیت هستند، بدین معنی که هر کدام از این الگوها دسته‌ی خاصی از مسائل طراحی را حل می‌کند. برای مثال، کاربردهای زیادی از الگوی تک وجود دارند. به علاوه، دو یا چند الگو را می‌توان به روش‌های نامحدودی ترکیب کرد تا مسائل طراحی بی‌شماری را حل نمود. الگوهای GoF به سه طبقه تقسیم می‌شوند: الگوهای ایجادی، الگوهای ساختاری، و الگوهای رفتاری. الگوهای ایجادی برای ساخت اشیا پیچیده یا خاص استفاده می‌شوند، مثلاً چگونه می‌توان یک ساختار کلاس پیچیده که شامل کلاس‌های مختلف و روابط پیچیده بین کلاسها می‌باشد را ساخت یا مقدار اولیه داد. الگوهای

[1] UML notes

ساختاری برای حل مسائل طراحی ساختاری استفاده می‌شوند. به‌عنوان مثال، چگونه می‌توان یک ساختار کلاسی پیچیده مانند دیاگرام بلوکی را نمایش داد. الگوهای رفتاری، برای حل مسائل طراحی الگوریتمی یا رفتاری به کار می‌روند. نمونه‌هایی از این مسائل عبارتند از: چگونه اجرای اعمال را زمان‌بندی نماییم، چگونه رویدادها را پردازش کنیم، چگونه الگوریتم‌ها یا عناصر یک ساختار کلاس پیچیده را تحلیل کنیم.

یک مجموعه‌ی دیگر از الگوهای طراحی نرم‌افزاری شناخته شده الگوهای عمومی نرم‌افزاری واگذاری مسئولیت می‌باشند، که توسط Graig Larman منتشر شده‌اند [۱۰۴]. برخلاف الگوهای GoF، الگوهای GRASP الگوهای عمومی واگذاری مسئولیت می‌باشند. منظور از کلمه‌ی «عمومی» این است که این الگوها را می‌توان تقریباً برای طراحی هر سیستم نرم‌افزاری به کار برد. اصطلاح «واگذاری مسئولیت» نیز نشان می‌دهد که این الگوها یک مسئله‌ی مهم طراحی شیءگرا را پوشش می‌دهند. یعنی این‌که، هر مسئولیت را باید به کدام شیء واگذار نمود به‌گونه‌ای که طراحی حاصل ویژگی‌هایی که در اصول طراحی نرم‌افزار تبلیغ می‌شوند را به نمایش گذارد. این فصل به معرفی و کاربرد سه تا از الگوهای GRASP اختصاص دارد. این الگوها عبارتند از: کنترلگر، خبره، و ایجادکننده.

۱۰-۴ تبیین الگو

تبیین الگو، یک توصیف ساخت‌یافته از هر الگو است که فهم و کاربرد الگو را تسهیل می‌نماید. برای مثال، شکل ۱۰-۲ تبیین الگوی تک که پیش‌تر بحث شد را نشان می‌دهد. تبیین الگو، جنبه‌های مهم یا مفید یک الگو را توصیف می‌کند:

۱. نام و گونه[۱]. این دو نام و گونه‌ی الگو را مشخص می‌کنند. گونه‌ی الگو، خانواده‌ی الگو را تعیین می‌کند. برای مثال، الگوهای GoF، GRASP، یا دیگر گونه‌های الگو. خود الگوهای GoF نیز به سه گونه‌ی ایجادی، ساختاری، و رفتاری تقسیم می‌شوند.
۲. مشخصات[۲]. این بخش مسئله‌ی طراحی و راه حل طراحی مربوط به الگو را توصیف می‌کند.
۳. طراحی[۳]. این بخش طراحی ساختاری و طراحی رفتاری الگو را مشخص می‌کند. برای توصیف طراحی ساختاری از نمودار کلاس UML استفاده می‌شود. طراحی رفتاری نیز توسط نمودار توالی یا متن توصیف می‌شود. علاوه بر این، برای هر کلاس، نقش و مسئولیت‌های آن کلاس در حل مسئله‌ی طراحی نیز توصیف می‌شود.
۴. فواید و عواقب[۴]. این دو مزایای کاربرد الگو و مشکلات احتمالی آن را توصیف می‌کنند.
۵. رهنمون‌ها[۵]. برخی مواقع اطلاعات مفیدی برای اعمال الگو ارائه می‌شود.
۶. الگوهای مرتبط[۶]. الگوهایی که به طرق مختلف به این الگو مربوط می‌شوند در این بخش مشخص می‌شوند.
۷. استفاده‌ها[۷]. کاربردهای عمومی یا خاص از الگو می‌تواند در این بخش معین شود.

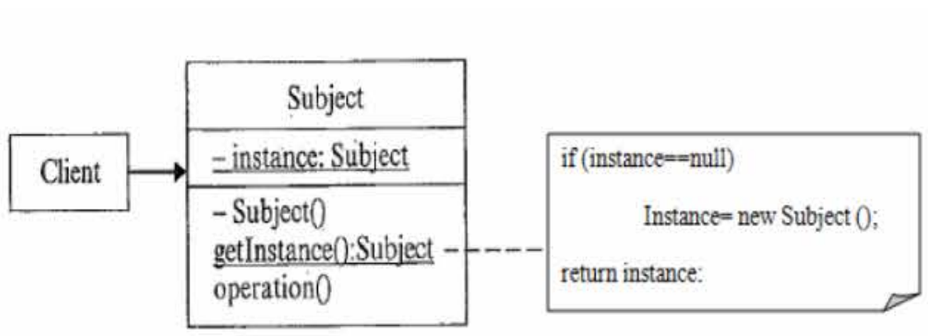
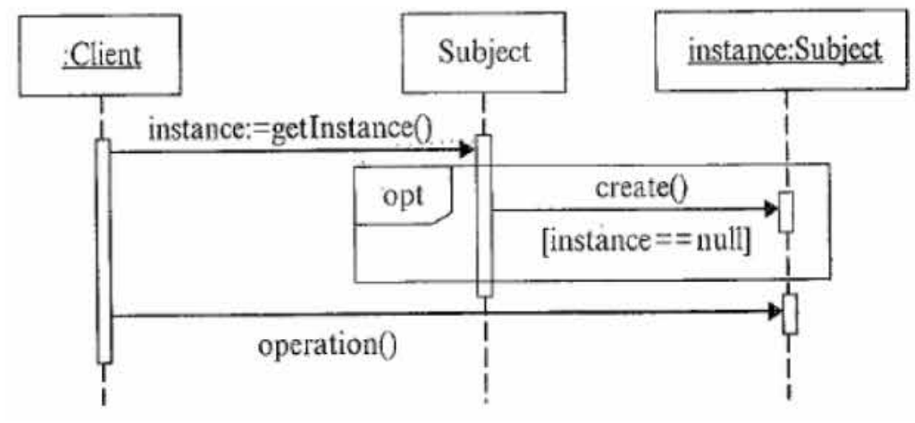
۱۰-۵ الگوی کنترلگر

الگوی کنترلگر تقریباً برای هر مورد کاربردی به کار می‌رود. مسئله‌ای که این الگو مدّ نظر قرار می‌دهد این است که سیستم‌های نرم‌افزاری را چگونه طراحی کنیم که به واسطهای کاربری و اشیای کسب و کاری خود اجازه دهند بدون وابستگی و بدون تاثیرگذاری بر روی یکدیگر تغییر کنند. مسئله‌ی طراحی دیگری که این الگو مدّ نظر قرار می‌دهد این است که چگونه واسطهای

---

[۱] Name and Type
[۲] Specification
[۳] Design
[۴] Benefits and Liabilities
[۵] Guidelines
[۶] Related Patterns
[۷] Uses

کاربری مختلف مانند واسط کاربری رومیزی[۱]، واسط کاربری مبتنی بر وب، یا دیگر واسط‌ها را پشتیبانی نمایند. الگوی طراحی مورد خاصی از الگوی مشهور مدل- نما- کنترلگر است. به عبارت دیگر، الگوی کنترلگر کاربردی از الگوی MVC برای مدیریت درخواست‌های کنشگران یک مورد کاربرد است.

نام	Singleton (تک)
گونه	GOF / Creational
تبیین	
تبیین مسئله	چگونه می‌توان کلاسی طراحی نمود که تنها تعداد محدودی نمونه‌ی سراسری قابل دستیابی داشته باشد؟
راه حل	سازنده‌ی کلاس را خصوصی کنید و یک متد ایستای عمومی تعریف کنید تا ایجاد نمونه‌ها را کنترل نماید.
طراحی	
ساختاری	
رفتاری	
نقش‌ها و مسئولیتها	• Subject یک متد ایستای عمومی بنام ()getInstance در اختیار مشتریان قرار می‌دهد تا نمونه‌ی کلاس را بدست آورند. • Client متد ()getInstance از کلاس Subject را فرامی‌خواند تا نمونه را بدست آورد و نیز متد Operation U خود را فرامی‌خواند.
فواید	• تعداد نمونه‌ها را محدود می‌کند. • دسترسی سراسری به نمونه‌ها را پشتیبانی می‌کند.
عواقب	• بروزرسانی هم‌زمان نمونه‌ی مشترک ممکن است باعث اثرات ناخواسته گردد.
رهنمون‌ها	
الگوهای مرتبط	• الگوی Singleton تعداد نمونه‌های یک کلاس را محدود می‌کند. الگوی Flyweight وقوع متد از یک شئ را پشتیبانی می‌کند. الگوی Prototype تعداد کلاس‌ها را کاهش می‌دهد. • الگوی Visitor اغلب یک الگوی تک است.
استفاده‌ها	الگوی Singleton در کاربردهای متعدد استفاده می‌شود. برنامه‌های جاوای Calendar، Desktop، Runtime، و Toolkit هم از الگوی تک (Singleton) استفاده کرده‌اند.

شکل ۱۰-۲: تبیین الگوی تک

[۱] Desktop

۱۰-۵-۱- مثال انگیزشی

شکل ۱۰-۳ یک نمودار توالی طراحی معمول برای مورد کاربرد خروج مدرک را نشان می‌دهد که در آن، مدرکی که دارای شماره است به امانت برده می‌شود. طراحی شکل ۱۰-۳ دارای مشکلات زیر می‌باشد:

۱. **جفت‌شدگی محکم بین اشیای نمایش و کسب‌وکار.** در شکل ۱۰-۳ شئ Checkout GUI معمولاً به‌عنوان نمایش تلقی می‌شود چون مسئول نمایش اطلاعات به کنشگر می‌باشد. اشیای سند و امانت نیز اشیای کسب‌وکاری نامیده می‌شوند زیرا اشیا از دامنه‌ی کسب‌وکار، که در اینجا دامنه‌ی کاربرد کتابخانه است، می‌باشند.

شکل نشان می‌دهد که نمایش مستقیماً به اشیای کسب‌وکار دسترسی دارد. بدین معنی که، شئ Checkout GUI شئ Loan را ایجاد و توابع ()isAvailable و setAvailable(fake) از شئ Document را فراخوانی می‌کند. فراخوانی این توابع باعث جفت‌شدگی محکم بین اشیای نمایش و کسب‌وکاری می‌شود. به دلایل زیر، این جفت‌شدگی پسندیده نیست.

جفت‌شدگی محکم بدین معنی است که تغییر اشیای کسب‌وکار ممکن است نیاز به تغییر نمایش داشته باشد و برعکس. واضح است که تغییر در واسط‌های اشیای کسب‌وکار، تغییر در نمایش را لازم می‌کند. اما، تغییر در کارکرد نیز ممکن است نیاز به تغییر در نمایش داشته باشد. برای مثال، در نظر بگیرید که تابعی از یک شئ کسب‌وکار که قبلاً از دو عدد صحیح فقط یکی را برمی‌گردانده است طوری تغییر کند که بیش از یک مقدار را برگرداند. اگر عدد صحیح برگشتی از تابع توسط نمایش به‌عنوان متغیر کنترلی استفاده می‌شده تا تعیین کند کدام مسیر را دنبال کند یا چه چیزی را به کاربر نمایش دهد، در این صورت باید نمایش را نیز تغییر داد تا مقادیر اضافی برگشتی توسط تابع را نیز در نظر بگیرد. مورد دیگر وقتی است که تعداد برگشتی‌ها توسط تابع تغییر نکرده ولی تاثیر مقادیر برگشتی تغییر کرده باشد. این‌گونه تغییرات همراه دو عمل اتفاق می‌افتند. برای مثال، یک نفر فکر می‌کند که جابه‌جایی مقادیر منطقی‌تر به نظر می‌رسد. در این حالت نیز تغییر در نمایش ضروری خواهد بود.

جفت‌شدگی محکم بین نمایش و اشیای کسب‌وکار پشتیبانی از چند نمایش مختلف را سخت و پرهزینه می‌کند. مثال‌هایی از نمایش‌های مختلف عبارتند از: واسط کاربری رومیزی، واسط کاربری مبتنی بر وب، واسط کاربری PDA، و واسط کاربری تلفن همراه. دلیل این امر این است که در هر کدام از این نمایش‌ها منطق کسب‌وکار باید تکرار شود. یعنی، مجبور خواهیم بود هر پیغام یا هر فراخوانی تابع در شکل ۱۰-۳ را برای هر کدام از نمایش‌ها تکرار نماییم.

در نتیجه، تغییر در منطق کسب‌وکار دشوار خواهد بود، چون هر کدام از نمایش‌ها نیز باید تغییر کنند. اگر همه‌ی نمایش‌ها به طرز مناسب و سازگار تغییر نکنند، سیستم نرم‌افزاری به صورت ناسازگار رفتار کرده یا نتایج ناسازگاری را تولید خواهد نمود.

برای مثال ساده‌ای که در شکل ۱۰-۳ نشان داده شد، به نظر نمی‌رسد که چنین مشکلی پیش بیاید. ولی اگر تلاش لازم برای انجام تغییرات و دردسر حفظ سازگاری را در تعداد مورد کاربردهایی که در یک کاربرد دنیای واقعی خواهیم دید ضرب کنید، متوجه اندازه‌ی مشکل خواهید شد. این تعداد برای یک کاربرد خیلی کوچک به بیش از ۲۰ مورد کاربرد و برای یک سیستم بزرگ به صدها مورد کاربرد خواهد رسید.

جفت‌شدگی محکم مشکلاتی نیز برای تکامل و نگهداری نرم‌افزار ایجاد خواهد نمود. یکی از انواع تکامل، بهبود[1] است. برای مثال، سیستمی را درنظر بگیرید که ابتدا به‌عنوان یک کاربرد رومیزی طراحی شده بوده است. پس از مدتی، کسب‌وکار مربوطه تصمیم می‌گیرد که آن را تبدیل به یک کاربرد برخط کند. در این حالت یک واسط کاربری مبتنی بر

[1] Enhancement

وب اضافه خواهد شد.جفت‌شدگی محکم بین نمایش و اشیای کسب‌وکار، اضافه‌کردن واسط مبتنی بر وب را سخت و پرهزینه خواهد نمود، زیرا همه چیز را باید در پیاده‌سازی واسط جدید تکرار نمود. این یکی از دلایلی است که شرکت‌ها سرانجام تصمیم می‌گیرند سیستم نرم‌افزاری‌شان را بازمهندسی[۱] نمایند تا بهبودشان در آینده آسان‌تر شود.

۲. **نمایش مجبور است به درخواست‌های کنشگر نیز رسیدگی کند.** طراحی شکل ۱۰-۳ نشان می‌دهد که نمایش مجبور است درخواست‌های کنشگر که نیاز به پردازش‌های پیش‌زمینه‌ای دارند را رسیدگی کند. برای مثال، نمایش شکل ۱۰-۳ درخواست خروج مدرک که توسط عضو ارسال می‌شود را پردازش می‌نماید.مدیریت چنین درخواست‌های کنشگری نباید مسئولیت نمایش باشد، زیرا وظیفه‌ی نمایش نشان‌دادن واسط کاربری و واکنش‌های سیستم به کاربر است.

در بخش ۶-۵-۳ از فصل ۶، اصل جداسازی دغدغه‌ها مورد بحث قرار گرفت. این اصل ابتدا توسط دایکسترا[۲] به‌عنوان یک اصل حل مسئله مطرح شد، که می‌گوید به‌جای حمله به تمام جنبه‌ها با هم، روی یک جنبه از مسئله تمرکز شود. در رابطه با طراحی نرم‌افزار، این اصل پیشنهاد می‌کند که هر جزء باید روی یک جنبه از موضوع مورد نظر[۳] تمرکز نماید. طبق این اصل، مسئولیت رسیدگی به درخواست‌های کنشگران که نیاز به پردازش پیش‌زمینه‌ای دارند به‌جای واگذاری به نمایش، باید به یکی از اشیای کسب‌وکار واگذار شود.

۳. **تعداد زیادی مسئولیت به نمایش واگذار شده است.** این بدان معنی است که مسئولیت‌های نمایش زیاد شده است. یعنی، علاوه بر مسئولیت نمایش واکنش‌های سیستم، مسئولیت پردازش منطق کسب‌وکار نیز بر عهده‌ی نمایش گذاشته شده است. این یک «شئ نادان» نیست. زیرا می‌داند چگونه هر دو کار را انجام دهد. این کار همچنین اصل چسبندگی زیاد[۴] را نیز نقض می‌کند زیرا این دو مجموعه مسئولیت چسبنده نیستند و باید به دو یا چند شئ مجزا واگذار گردند.

۱۰-۵-۲- کنترلگر چیست؟

همان گونه که در بالا بحث شد، نمودار توالی طراحی شکل ۱۰-۳ سه مشکل دارد: (۱) نمایش، جفت‌شدگی محکمی به اشیای کسب‌وکار دارد، (۲) نمایش مجبور است درخواست‌های کنشگر که نیاز به پردازش‌های پیش زمینه‌ای دارند را رسیدگی کند، و (۳) مسئولیت‌های زیادی به نمایش واگذار شده است.

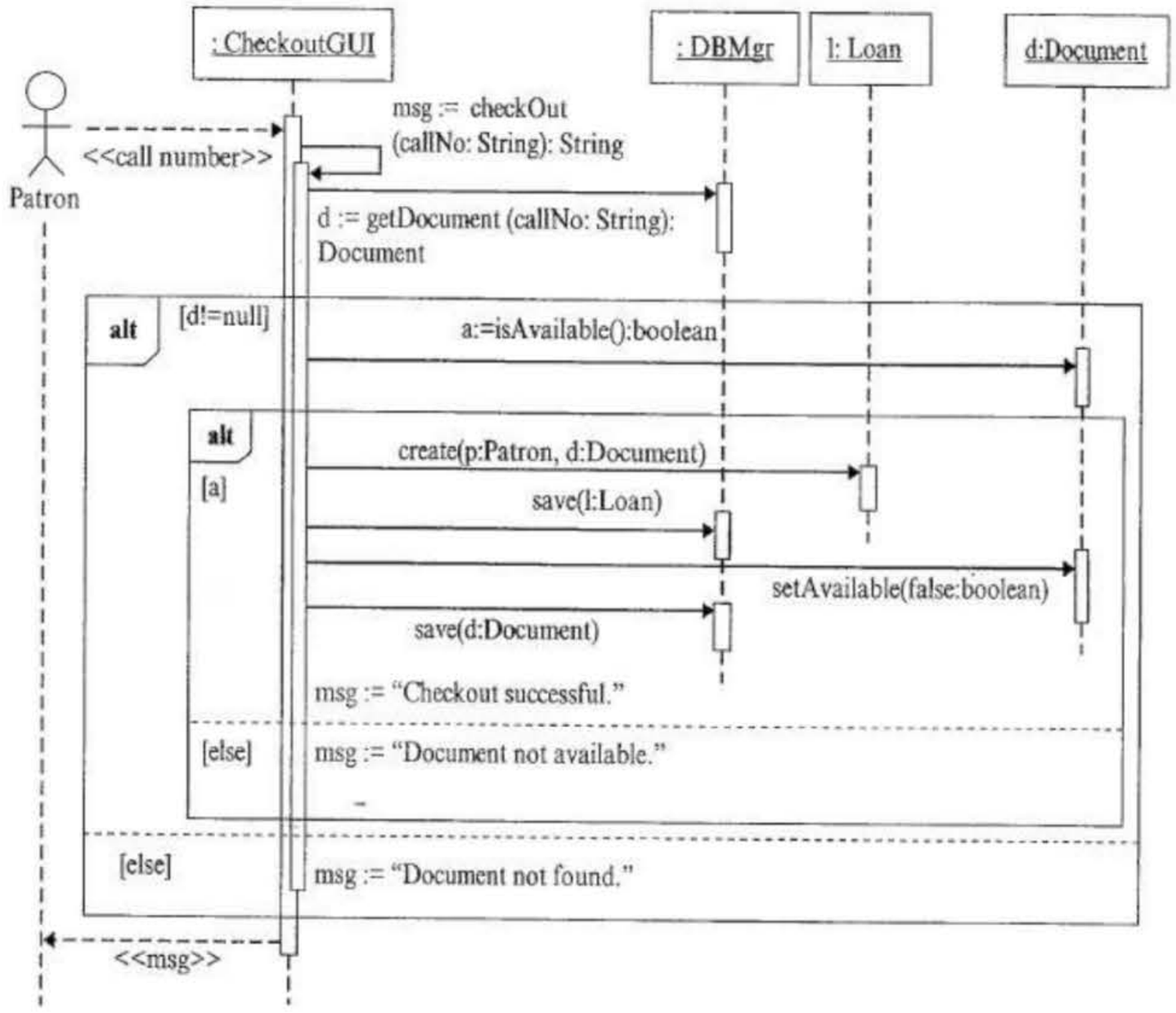
یک راه حل برای این مسائل می‌تواند این باشد که (۱) اشیای نمایش و کسب‌وکار را از هم تفکیک کنیم، (۲) مسئولیت رسیدگی به درخواست کنشگر را از نمایش گرفته و به اشیای دیگر واگذار کنیم.

سؤال این است که مسئولیت رسیدگی به درخواست کنشگر را به چه شییی واگذار کنیم؟ همان‌گونه که در شکل ۱۰-۴ نشان داده شده است، پاسخ کنترلگر است.شکل نشان می‌دهد که یک کنترلگر معرفی شده و مابین نمایش و اشیای کسب‌وکارقرار گرفته است.نمایش، درخواست کنشگر را گرفته و به کنترلگر تحویل می‌دهد. کنترلگر با اشیای کسب‌وکار تعامل می‌کند تا به این درخواست پاسخ دهد. کنترلگر نتیجه را به نمایش تحویل می‌دهد، که او هم نتیجه را به کنشگر نمایش می‌دهد. شکل ۱۰-۵ جزییات بیش‌تری از الگوی کنترلگر را نشان می‌دهد.

الگوی کنترلگر مسئله‌ای که در بخش ۱۰-۵-۱ بحث شد را به زیبایی حل می‌کند. اکنون نمایش از اشیای کسب‌وکاری تفکیک شده است. در نتیجه، تغییر در یکی تاثیری روی دیگری نمی‌گذارد. پشتیبانی از گونه‌های متعدد نمایش آسان است. برای افزودن یک گونه‌ی جدید نمایش، فقط کافی است که آن نمایش جدید را طراحی و پیاده‌سازی نموده وازآن بخواهیم

---

[۱] Reengineering
[۲] Dijkstra
[۳] Subject Matter
[۴] High Cohesion



شکل ۱۰-۳: یک نمودار توالی متداول برای خروج مدرک

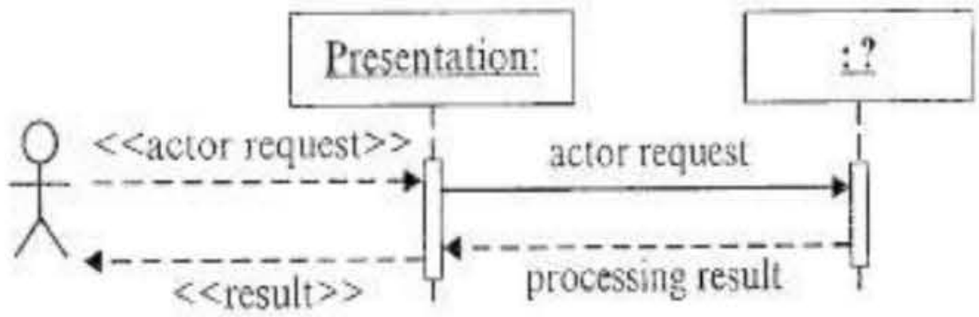
درخواست‌های کنشگر را به کنترلگر ارسال نماید. این‌کار به‌خوبی نگهداری تکامل یا بهبود نرم‌افزار را تسهیل می‌کند. مسئولیت رسیدگی به درخواست کنشگر از نمایش برداشته و به عهده‌ی کنترلگر گذاشته شده است.

۱۰-۵-۳- اعمال الگوی کنترلگر

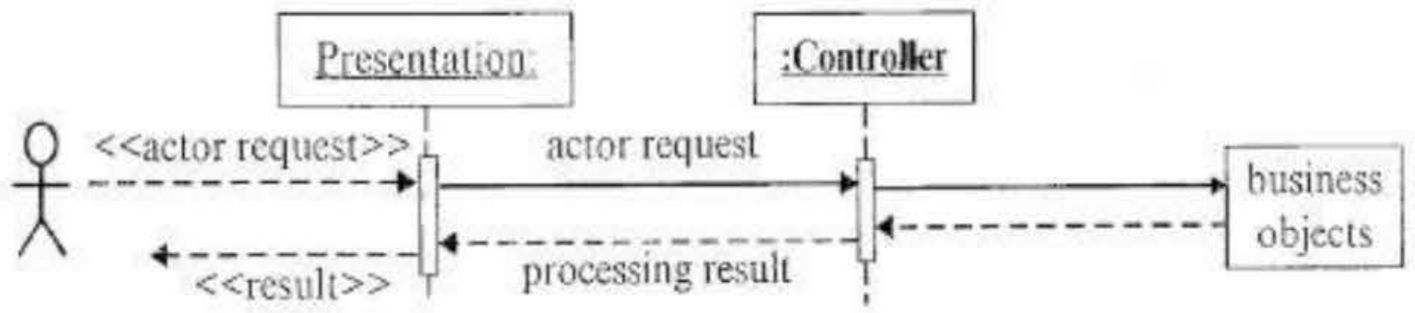
برای اعمال الگوی کنترلگر روی طراحی شکل ۱۰-۳، می‌توان به‌آسانی یک کنترلگر میان نمایش و اشیای کسب‌وکار معرفی نمود. به‌علاوه، منطق کسب‌وکار از نمایش برداشته و به کنترلگر واگذار می‌شود. شکل ۱۰-۶ نتیجه را نشان می‌دهد. توجه کنید که حالا شئ Checkout GUI تنها مسئول نمایش اطلاعات به عضو(کاربر) است.مسئولیت پردازش درخواست خروج مدرک نیز به شئ Checkout Controller واگذار گردیده است، که با DBMgr و سایر اشیای کسب‌وکار تعامل می‌کند تا مسئولیت را برآورده نماید.

آموزنده خواهد بود که طراحی شکل ۱۰-۶ را با توجه به اصول طراحی معرفی شده در بخش ۶-۵ از فصل ۶ به اختصار مورد بحث قرار دهیم. هدف از این بحث آن است که نشان دهیم چگونه می‌توان یک طراحی را با استفاده از اصول طراحی نرم‌افزار بهبود بخشید. به‌علاوه؛ این بحث به دانشجویان کمک خواهد نمود تا اصول طراحی را درک کنند. صفت‌های ضروری یک معماری نرم‌افزار در زیر آمده است.

۱. **طراحی برای تغییر.** تغییرات در منطق کسب‌وکار یا اشیای کسب‌وکار اثر کمی بر روی نمایش خواهد داشت زیرا واسط و رفتار تعاملی Checkout Controller تغییر نکرده است.



مسئله: مسئولیت رسیدگی به درخواست کنشگر به چه کسی باید واگذار شود؟



جواب: مسئولیت را به یک کنترلر واگذار کنید
که با اشیای کسب‌وکار همکاری نموده و این مسئولیت را برآورده می‌نماید.

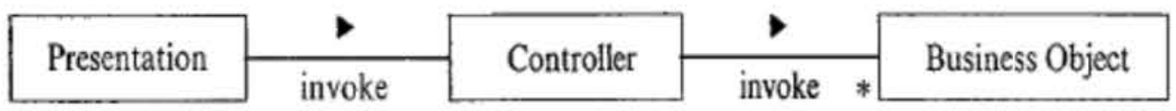
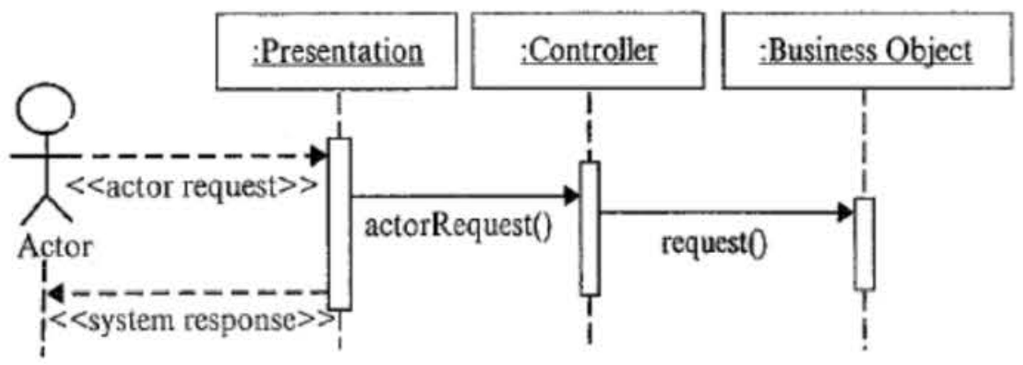
شکل ۱۰-۴: نمایش الگوی کنترلگر

تغییر در برخی نیازمندی‌های خاص را می‌توان به‌سادگی انجام داد. فرض کنید که سیستم اطلاعات کتابخانه با نیازمندی‌های اضافه‌تری گسترش پیدا کند تا چندین واسط کاربری براساس فناوری‌های مختلفی مثل واسط وب، واسط تلفن همراه، و واسط صوتی را پشتیبانی کنند. با طراحی قبلی که در شکل ۱۰-۳ ارائه شد، ما مجبوریم واسط‌های کاربری جدید را طراحی و پیاده‌سازی نماییم. به علاوه، منطق کسب‌وکار هم باید در هرکدام از این نمایش‌ها تکرار شود. تغییر در منطق کسب‌وکار یا اشیای کسب‌وکار نیازمند انجام تغییرات مشابه در هر کدام از این تکرارها می‌باشد. چنان‌چه تعداد قابل توجهی مورد کاربرد داشته باشیم، تلاش لازم برای انجام این تغییرات و حفظ سازگاری غیرقابل تصور خواهد بود.

طراحی شکل ۱۰-۶ تنها نیاز دارد که واسط‌های جدید را اضافه کنیم و بگذاریم هر کدام از آن‌ها توابع متناسب از Checkout Controller را فراخوانی نمایند. تغییر در منطق کسب‌وکار یا اشیای کسب‌وکار تاثیر اندکی بر روی واسط‌های کاربری خواهد داشت. وقتی تعداد مورد کاربردها زیاد باشد این روش جذاب خواهد بود.

۲. **جداسازی دغدغه‌ها.** جداسازی دغدغه‌ها توسط طراحی به‌خوبی پشتیبانی می‌شود. شئ Checkout GUI حالا تنها با جنبه‌ی نمایش سروکار دارد در حالی که Checkout Controller مسئول پذیرش مورد کاربردCheckout Document است. در طراحی قبلی، هر دو دغدغه به Checkout GUI واگذار شده بود.

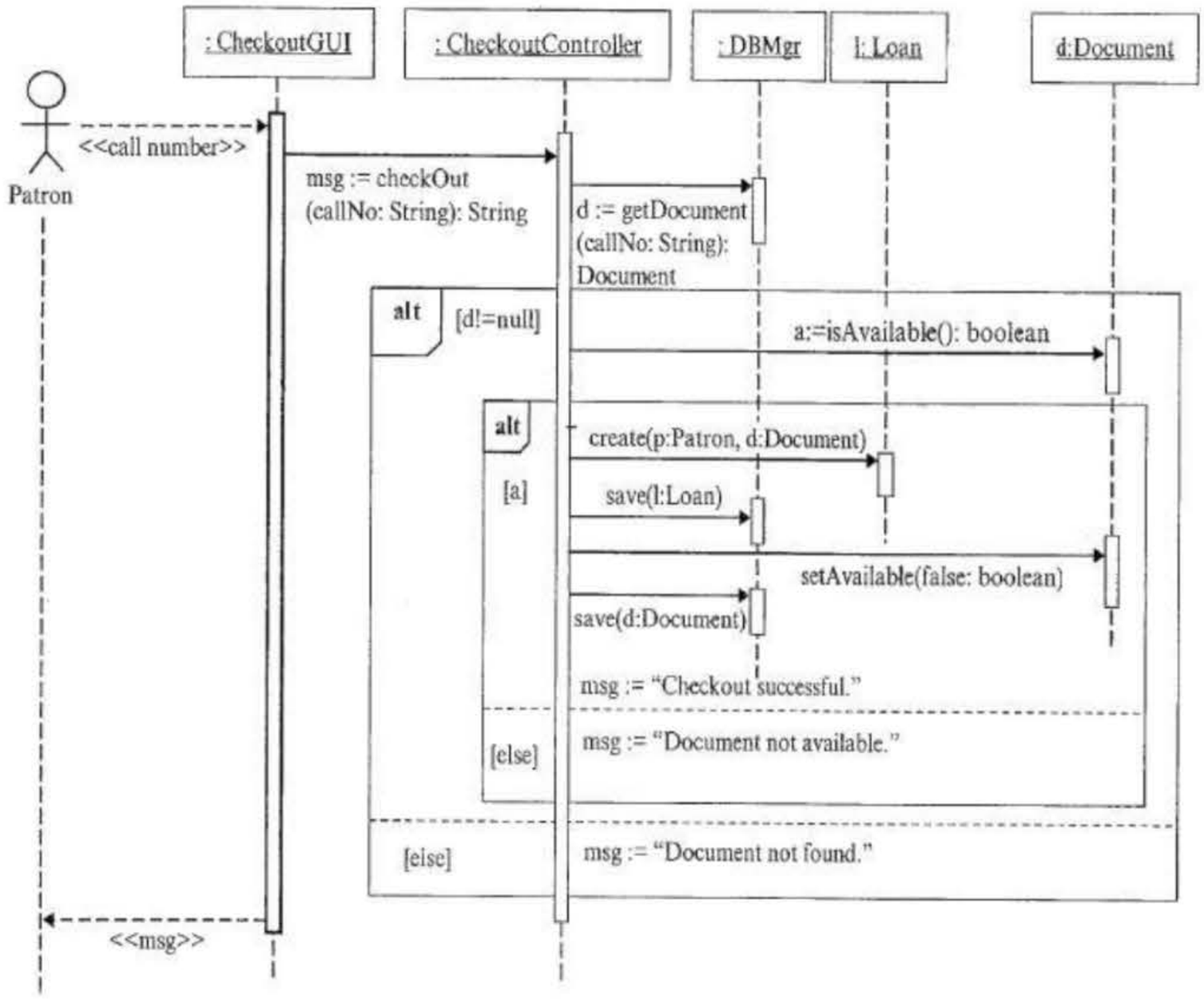
۳. **چسبندگی زیاد.** پیش‌تر مسئولیت‌های نمایش اطلاعات به عضو (کاربر) و پردازش مورد کاربرد Checkout Document به Checkout GUI واگذار شده بود. اما این دو مجموعه کارکردی به یک تابع هسته‌ای منفرد تعلق ندارند. بنابراین، چسبندگی طراحی قبلی مطمئناً چسبندگی تابعی نیست. این واقعیت که Checkout GUI ابتدا درخواست کنشگر را دریافت نموده و سپس آن را پردازش می‌نماید نشان می‌دهد که Checkout GUI در طراحی قبلی چسبندگی رویه‌ای را به نمایش می‌گذارد. در طراحی جدید، به دلایل روشن، هر دوی Checkout GUI و Checkout Controller چسبندگی

نام	کنترلگر(Controller)
گونه	GRASP
تبیین	
مسئله	چه کسی باید مسئول رسیدگی به درخواست کنشگر باشد؟
راه‌حل	مسئولیت رسیدگی به درخواست را به یک کلاس مختص به‌نام کنترلگر واگذار کنید.
طراحی	
ساختاری	
رفتاری	
نقش‌ها و مسئولیت‌ها	• اشیای کسب‌وکار مسئول منطق کسب‌وکار در کاربرد هستند. • کنترلگر کلاسی است مخصوص رسیدگی به درخواست‌های کنشگر که درخواست‌ها را از نمایش گرفته و با دیگر اشیا کار می‌کند تا درخواست را پاسخ دهد. یک کنترلگر مورد کاربرد برای رسیدگی به تمام درخواست‌های کنشگران در خصوص یک مورد کاربرد مشخص تخصیص می‌یابد. این کنترلگر وضعیت مورد کاربرد را ردگیری می‌نماید. • نمایش کلاسی است که مسئول تعامل با کنشگر سیستم می‌باشد. این کلاس درخواست‌های کنشگر را به کنترلگر فرستاده و پاسخ‌ها را نیز از کنترلگر به کنشگر تحویل می‌نماید.
فواید	• نمایش را از اشیای کسب‌وکار مجزا می‌کند. • اثر تغییر نمایش روی اشیای کسب‌وکار یا برعکس را کاهش می‌دهد. • چندنمایشی را پشتیبانی می‌کند. • وضعیت مورد کاربردها را ردگیری می‌کند.
عواقب	ممکن است به کنترلگر مسئولیت‌های زیادی واگذار گردد که منجر به ایجاد به‌اصطلاح کنترلگر متورم (Bloated Controller) می‌شود. کنترلگر متورم پیچیده بوده و درک، پیاده‌سازی، آزمون، و نگهداشت آن دشوار خواهد بود.
رهنمون‌ها	• هرجا که امکان داشت کنترلگرهای مورد کاربرد را به‌کار ببرید. • از به‌کار گرفتن یک کنترلگر برای بیش از یک مورد کاربرد پرهیز نمایید. • کنترلگر باید با اشیای کسب‌وکار همکاری نموده و مسئولیت‌ها را به آن‌ها واگذار نماید.
الگوهای مرتبط	• کنترلگر یک حالت خاص از الگوی مدل-نما-کنترلگر(MVC) است. • کنترلگر می‌تواند از State استفاده کرده و وضعیت یک مورد کاربرد را ردگیری نماید.
استفاده‌ها	در طراحی تمام سیستم‌های تعاملی برای جداسازی نمایش از اشیای کسب‌وکار

شکل ۱۰-۵: توصیف کنترلگر بر اساس قالب الگو

تابعی را به نمایش می‌گذارند.

۴. **طراحی «اشیای نادان».** طراحی جدید، اصل «ساده و احمقانه بگیر» و به خصوص طراحی «اشیای نادان» را به نمایش می‌گذارد. پیش‌تر، Checkout GUI می‌دانست که چگونه اطلاعات را به عضو (کاربر) نمایش دهد و نیز چگونه منطق کسب‌وکار را پردازش نماید. در طراحی بهبودیافته، هر کدام از Checkout GUI و Checkout Controller تنها یک چیز را می‌دانند، که به‌ترتیب عبارت است از نمایش اطلاعات و پردازش مورد کاربردDocumentCheckout.



شکل ۱۰-۶: اعمال الگوی کنترلگر

۱۰-۵-۴- گونه‌های کنترلگر

دو گونه کنترلگر وجود دارند: ۱) کنترلگر مورد کاربرد[۱]، ۲) کنترلگرنما[۲].

کنترلگر مورد کاربرد مسئول رسیدگی به تمام درخواست‌های کنشگران است که مربوط به یک مورد کاربرد می‌شوند. منظور از کنترلگر شکل ۱۰-۶ این بوده که یک کنترلگر مورد کاربرد برای مورد کاربرد Checkout باشد. این را می‌توان از نام کنترلگر که «Checkout Controller» است مشاهده نمود، که نشان می‌دهد این کنترلگر مربوط به مورد کاربرد Checkout است.

کنترلگرهای مورد کاربرد متداول‌ترین و توصیه‌شده‌ترین کنترل‌ها هستند زیرا چسبندگی زیاد، جداسازی دغدغه‌ها، طراحی برای تغییر، و نهان‌سازی اطلاعات را عملی می‌کنند. این بدین خاطر است که بین مورد کاربردها و کنترلگرهای مورد کاربرد تناظر یک به یک وجود دارد. بدین معنی که، برای هر مورد کاربرد سیستم یک کنترلگر مورد کاربرد طراحی و پیاده‌سازی می‌شود. بنابراین، هر کنترلگر مورد کاربرد سعی می‌کند چسبندگی تابعی، جداسازی دغدغه‌ها، و طراحی برای تغییر را محقق سازد.اگر نیازمندی‌های جدید اضافه شده و مورد کاربردهای جدیدی معرفی شوند، تنها نیاز است کنترلگرهای مورد کاربرد متناسب را نیز اضافه نمود. تغییر در نیازمندی‌های فعلی محدود به تغییر در کنترلگرهای مورد کاربرد و نیز اشیای کسب‌وکاری مرتبط خواهد شد. نهان‌سازی اطلاعات مورد پشتیبانی قرار می‌گیرد زیرا تغییر در اشیای کسب‌وکار از نمایش جدا شده است، بدین صورت که واسط‌های کنترلگرها پایدار نگه داشته شده‌اند.بنابراین، سیستم‌هایی که با استفاده از کنترلگرهای مورد کاربرد طراحی و پیاده‌سازی شده‌اند از نظر درک، پیاده‌سازی، آزمایش، و نگهداری ساده‌تر هستند. رسم است که درهنگام نام‌گذاری، نام کنترلگرهای مورد کاربرد را متناسب با خود مورد کاربرد انتخاب کرد، مثلاً، Login Controller برای مورد کاربرد Login،

---

[۱] Use- Case Controller

[۲] Façade Controller

Checkout Controller برای مورد کاربرد Checkout ، و Payment Controller برای مورد کاربرد MakePayment.

۱۰-۵-۵- ردگیری وضعیت مورد کاربرد

طبق تعریف، مورد کاربردها فرایندهای کسب‌وکاری هستند. برخی از فرایندهای کسب‌وکار رفتار وابسته به حالت از خود نشان می‌دهند. برای سیستم این مهم است که وضعیت مورد کاربرد را ردیابی کند به‌طوری که سیستم وضعیت مورد کاربرد و درخواست‌های کنشگر که خارج از ترتیب هستند را تشخیص دهد. کنترلگر مورد کاربرد بهترین مکان برای پیاده‌سازی چنین رفتار وابسته به حالات از هر مورد کاربرد است.

مورد کاربرد Reserve Flight از یک سیستم رزرو پرواز مبتنی بر وب را در نظر بگیرید. این مورد کاربرد وقتی شروع می‌شود که مشتری روی پیوند Flight Reservation کلیک می‌کند. سیستم صفحه‌ی Flight Reservation را نشان می‌دهد. مشتری Round Trip یا One Way را انتخاب نموده و فرودگاه مبدأ، فرودگاه مقصد، و تاریخ سفر را وارد می‌کند. سیستم مسیرهای موجود را نشان می‌دهد. مشتری مسیر مطلوب را انتخاب نموده و روی دکمه‌ی Continue کلیک می‌کند. سیستم از مشتری می‌خواهد که اطلاعات مسافر را وارد نماید. مشتری اطلاعات مسافر را وارد نموده و روی Continue کلیک می‌کند. سیستم از مشتری می‌خواهد تا یکی از دو گزینه‌ی Purchase یا Hold را انتخاب نماید. اگر مشتری Hold را انتخاب کند، سیستم برای ۲۴ ساعت این رزرواسیون را برای مشتری نگه می‌دارد. اگر مشتری Purchase را انتخاب نماید، سیستم از مشتری می‌خواهد که گزینه‌ی پرداخت را انتخاب نماید. مشتری گزینه‌ی پرداخت را انتخاب می‌کند. سیستم از مشتری می‌خواهد تا اطلاعات مربوط به گزینه‌ی پرداخت انتخاب شده را وارد نماید. مشتری اطلاعات مورد نیاز را وارد می‌نماید. سیستم از مشتری می‌خواهد تا گزینه‌های بلیط را انتخاب نماید. مشتری گزینه‌های بلیط را انتخاب می‌کند. سیستم پیغام موفق بودن رزرو پرواز را نمایش می‌دهد. وقتی مشتری پیام موفقیت رزرو پرواز را مشاهده نمود، مورد کاربرد پایان می‌پذیرد.

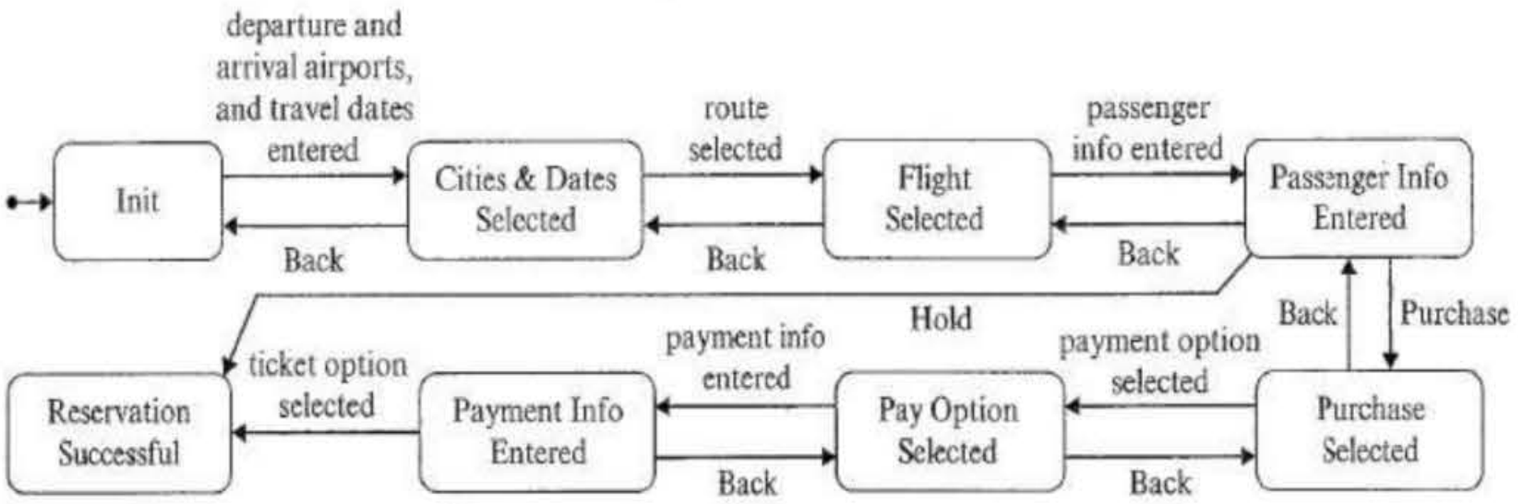
کنترلگر مورد کاربرد مسئول رسیدگی به تمام درخواست‌های کنشگر درباره‌ی این مورد کاربرد است. این بدان معنی است که هرکدام از این درخواست‌های کنشگر به روشی به یک کنترلگر مورد کاربرد تحویل داده می‌شود تا پردازش شود. پردازش تمام این درخواست‌ها فرایند کسب‌وکار فرایند خرید پرواز را تشکیل می‌دهد. در سناریوی فوق، هفت درخواست کنشگر مرتبط با مورد کاربرد وجود دارند: ۱) مشتری Round trip یا One Way را انتخاب نموده و شهر مبدأ، مقصد، و تاریخ سفر را وارد می‌کند، ۲) مشتری مسیر را انتخاب نموده و روی دکمه‌ی Continue کلیک می‌کند، ۳) مشتری اطلاعات مسافر را وارد نموده و روی دکمه‌ی Continue کلیک می‌کند، ۴) مشتری Purchase یا Hold را انتخاب می‌کند، ۵) مشتری گزینه‌ی پرداخت را انتخاب می‌کند،۶) مشتری اطلاعات پرداخت را وارد می‌کند، و ۷) مشتری گزینه‌های بلیط را انتخاب می‌کند.

همان‌طور که هر کدام از درخواست‌های کنشگر با موفقیت پردازش می‌شود، وضعیت مورد کاربرد عوض می‌شود. شکل ۱۰-۷ نمودار انتقال وضعیت را برای مورد کاربرد Reserve Flight نشان می‌دهد. با این مدل وضعیت، سیستم می‌تواند کاربر پسندتر و محکم‌تر ساخته شود، زیرا سیستم می‌داند که کاربر در گام بعدی چه‌کار خواهد کرد و وضعیت بعدی چه خواهد بود. بنابراین، با این کار، درخواست‌های خارج از کلاس حذف شده و سیستم محکم‌تر[1] خواهد شد.

کنترلگرنما مسئول رسیدگی به تمام درخواست‌های کنشگر در کل سیستم می‌باشد. کنترلگرنما وقتی به کار می‌رود که تنها تعداد اندکی مورد کاربرد داشته باشیم و احتمال گسترش سیستم کم باشد. اگر سیستم گسترش پیدا کند، در این صورت مورد کاربردهای بیشتری به آن افزوده خواهد شد. هر چه مورد کاربردهای بیشتری اضافه شوند، ویژگی‌های چسبندگی، جفت‌شدگی، و جداسازی دغدغه‌های مربوط به کنترلگرنما نیز بدتر می‌شوند. سرانجام، کنترلگرنما با مسئولیت‌های اضافه‌ای که برای رسیدگی به مورد کاربردها و درخواست‌های زیاد کنشگران برعهده گرفته است، سربارگذاری خواهد شد.درک، پیاده‌سازی، آزمون، و نگهداری

[1] Robust

سیستم دشوارتر خواهد بود. از آن‌جا که در بسیاری از حالات گسترش سیستم اجتناب‌ناپذیر است، اصل طراحی برای تغییر پیشنهاد می‌کند که کنترلگرهای مورد کاربرد برای اغلب کاربردها استفاده گردد.



شکل ۱۰-۷: نمودار انتقال وضعیت برای مورد کاربرد Reserve Flight

۱۰-۵-۶- کنترلگر متورم

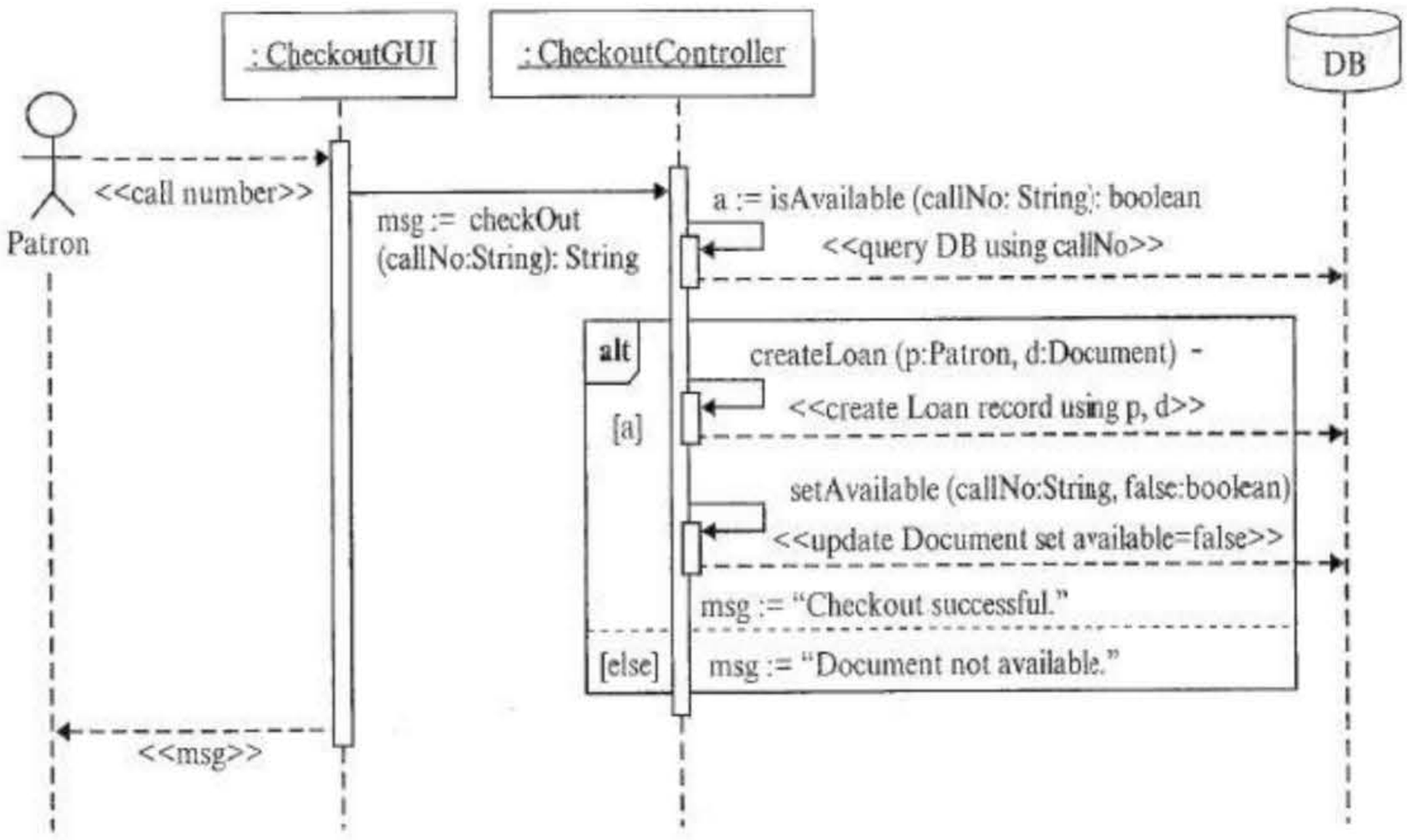
محاسن یک کنترلگر، به‌خصوص کنترلگر مورد کاربرد، عبارتند از: چسبندگی زیاد، جداسازی دغدغه‌ها، طراحی برای تغییر، و نهان‌سازی اطلاعات. ولی چنان‌چه کنترلگر به‌درستی استفاده نشود، ممکن است با مسئولیت‌های اضافی انباشته شده و منظور اصلی محقق نشود. شکل ۱۰-۸ کنترلگری را نشان می‌دهد که همه چیز را خودش انجام می‌دهد. کنترلگر با پرس‌وجو از پایگاه‌داده موجود بودن سندی که قرار است امانت گرفته شود را بررسی می‌کند. اگر سند موجود باشد، یک رکورد امانت (Loan) در پایگاه‌داده ایجاد شده، صفت موجودبودن سند را در رکورد سند برابر نادرست، و مقدار پیام msg برابر "Checkout Successful" یا "Document not Available" قرار داده می‌شود.

همان‌طور که پیش‌تر گفته شد، به کنترلگر مسئولیت‌های زیادی واگذار شده است. این کار چندین اصل طراحی نرم‌افزار از قبیل طراحی برای تغییر، جداسازی دغدغه‌ها، چسبندگی زیاد،جفت‌شدگی کم، نهان‌کردن اطلاعات، و ساده واحمقانه بگیر را نقض می‌کند. در بخش بعدی این موضوع به تفصیل بحث خواهد شد.

برای اعمال درست الگوی کنترلگر، مسئولیت‌های پردازش مورد کاربرد Checkout باید بین کنترلگر و اشیای کسب‌وکار توزیع شوند. در این مثال، مسئولیت‌های دستیابی به پایگاه‌داده باید به عهده‌ی یک شئ دسترسی به پایگاه‌داده واگذار شود. مسئولیت‌های پرس‌وجو و به‌روزرسانی اطلاعات وضعیت مربوط به یک شئ کسب‌وکار باید به شئ کسب‌وکار واگذار شود. ایجاد یک شئ کسب‌وکار باید با فراخوانی تابع سازنده‌ی آن شئ کسب‌وکار انجام گردد، نه با ایجاد یک رکورد در پایگاه‌داده. نتیجه‌ی این تغییرات یک نمودار توالی طراحی است که در شکل ۱۰-۶ نشان داده شده است.

۱۰-۵-۷- مقایسه‌ی طراحی‌های مختلف

با نگاهی به شکل ۱۰-۸ و شکل ۱۰-۶ ممکن است این‌طور به نظر برسد که اولی ساده‌تر از دومی است ولی اگر اصل ساده و احمقانه بگیر را در نظر بگیریم آیا نباید شکل پیشین (اولی) را به‌عنوان طراحی برتر انتخاب نمود؟ برای این‌که بفهمیم طراحی بهتر کدام است، یک تحقیق عمقی مورد نیاز می‌باشد. بدین معنی که، دو طراحی را باید برحسب اصول طراحی نرم‌افزار ارزیابی نمود. بخش‌های زیر این تحلیل را نشان می‌دهند.



شکل ۱۰-۸: یک کنترلگر متورم که همه کار را خودش انجام می‌دهد.

طراحی برای تغییر

جفت‌شدگی محکم بین کنترلگر و پایگاه‌داده پشتیبانی از پایگاه‌داده‌ی متفاوت را دشوار می‌کند. طراحی شکل ۱۰-۶ کنترلگر و پایگاه‌داده را از هم مجزا می‌کند. از آنجا که پایگاه‌داده پشت DBMgr مخفی است، تغییر در پایگاه‌داده اثر ناچیز یا اندکی روی کنترلگر و اشیای کسب‌وکار دارد. همان‌گونه که در بخش ۶-۵-۲ مطرح شد و در فصل ۱۶ نیز بحث خواهد شد، استفاده از الگوی پل به DBmgr اجازه می‌دهد با پایگاه‌داده‌های مختلفی کار کند. بنابراین، طراحی برای تغییر به خوبی توسط طراحی شکل ۱۰-۶ پشتیبانی می‌شود ولی توسط طراحی شکل ۱۰-۸ پشتیبانی نمی‌شود.

جداسازی دغدغه‌ها

کنترلگر شکل ۱۰-۸ تمام مسئولیت‌های لازم برای پردازش مورد کاربرد Checkout را دارد. بنابراین، جداسازی دغدغه‌ها محقق نمی‌شود. از سوی دیگر، در شکل ۱۰-۶ فقط مسئولیت انجام رویه‌ی امانت به عهده‌ی کنترلگر واگذار شده است. مسئولیت‌های دست‌یابی به پایگاه‌داده به DBMgr و مسئولیت نگهداری وضعیت اشیای کسب‌وکار نیز به اشیای کسب‌وکار واگذار گردیده است. بنابراین، طراحی شکل ۱۰-۶ به خوبی موضوع جداسازی دغدغه‌ها را پشتیبانی می‌نماید، اما طراحی شکل ۱۰-۸ این پشتیبانی را ارائه نمی‌کند.

چسبندگی زیاد

در شکل ۱۰-۸ مسئولیت‌های متعددی بر عهده‌ی کنترلگر گذاشته شده است. در شکل ۱۰-۶ این مسئولیت‌ها از نظر کارکردی تقسیم شده و اشیای مختلف توزیع شده‌اند. بنابراین، طراحی شکل ۱۰-۶ نسبت به طراحی شکل ۱۰-۸ چسبندگی بیشتری را نشان می‌دهد.

جفت‌شدگی کم

جفت‌شدگی زیاد بین کنترلگر و پایگاه‌داده در شکل ۱۰-۸ بدین معنی است که اگر پایگاه‌داده عوض شود یا شمای[1] پایگاه‌داده تغییرکند، تغییر مهمی اتفاق خواهد افتاد. بدین سبب،جفت‌شدگی کم محقق نخواهد شد. از طرف دیگر، DBMgr در شکل ۱۰-۶ کنترلگر را از پایگاه‌داده مجزا مینماید. بنابراین، تاثیر تغییر کم خواهد بود.

نهان‌سازی اطلاعات

در شکل ۱۰-۶ DBmgr پایگاه‌داده را از کنترلگر مخفی می نماید. در شکل ۱۰-۸ جزییات پیاده‌سازی پایگاه‌داده از کنترلگر مستقل نشده است. بنابراین، نهان کردن اطلاعات نقض می‌شود.

ساده و احمقانه بگیر

کنترلگر شکل ۱۰-۸ گام های فرایند کسب‌وکار مورد کاربرد checkout را می داند. ولی در ضمن سازماندهی پایگاه‌داده را نیز می‌شناسد و می‌داند چگونه با پایگاه‌داده پرس‌وجو نموده و آن را به‌روزرسانی نماید. همچنین وضعیت و رفتار اشیای کسب‌وکار را نیز می‌شناسد (به گونه‌ای که می‌تواند به طرز مناسبی اشیای کسب‌وکار را به‌روزرسانی کند). کنترلگر «نادان» نیست و ساده هم نیست زیرا مجبور است تمام این مسئولیت‌ها را پیاده‌سازی نماید. از دیگر سو، طراحی شکل ۱۰-۶ منجر به تولید «اشیای نادان» می‌شود که هر کدام از آن‌ها فقط یک مجموعه‌ی چسبنده از وظایف را برعهده دارد.

۱۰-۵-۸- چه موقع باید الگوی کنترلگر را به‌کار برد؟

الگوی کنترلگر را می‌توان در مراحل مختلفی از گام‌های مدل‌سازی تعامل شئ به‌کار برد:

۱. در موقع نوشتن سناریوی مورد کاربرد. بهترین زمان برای اعمال الگوی کنترلگر وقتی است که سناریوی مورد کاربرد نوشته می‌شود. طراح باید در خاطر داشته باشد که کنترلگر مورد کاربرد باید نمایش را از اشیای کسب‌وکار مجزا نماید. بنابراین، وقتی دارد سناریوی مورد کاربرد را می نویسد الگوی کنترلگر را به‌کار می‌برد. حسن این کار آن است که هیچ دوباره‌کاری نیاز نیست. حسن دیگر این است که می‌توان اکثر گام‌های باقی‌مانده را خودکار یا نیمه‌خودکار انجام داد.
۲. با تغییر یک سناریوی موجود. اغلب اوقات یک سناریوی مورد کاربرد مناسب قبلاً نوشته شده است. مثلاً در حین مرحله‌ی تحلیل این‌کار انجام شده است تا فرایند کسب‌وکاری مورد کاربرد را توصیف نماید. در این حالت، می‌توان به‌گونه‌ای سناریو را تغییر داد که یک کنترلگر مورد کاربرد را نیز شامل شود. علاوه بر محاسن فوق، این روش امکان بازاستفاده از سناریوهای موجود را نیز فراهم می‌آورد.
۳. در موقع تشکیل جدول سناریو. الگوی کنترلگر را می‌توان در زمانی که جدول سناریو در حال ساخته شدن است اعمال نمود. بدین معنی که کنترلگر مورد کاربرد در خطوط متناسب از جدول سناریو معرفی شود تا نمایش را از اشیای کسب‌وکار جدا نماید. گرچه این گزینه به‌اندازه‌ی قبلی جذاب نیست، ولی با این حال امکان خودکار یا نیمه خودکار نمودن تولید نمودار توالی طراحی را از روی جدول سناریو فراهم می‌آورد.
۴. در موقع ساخت نمودار توالی طراحی. انتخاب دیگر این است که الگوی کنترلگر را در موقع ساختن نمودار توالی طراحی به‌کار ببریم. این انتخاب معمولاً توسط طراحان با تجربه، که می‌توانند نمودار توالی طراحی را بدون نیاز به توصیف سناریوی مورد کاربرد تولید کنند، به‌کار برده می‌شود.

---

[1] Schema

۵. با تغییر یک نمودار توالی طراحی. بعضی مواقع، ممکن است یک نمودار توالی طراحی ساخته شده باشد، اما کنترلگر مورد کاربرد به‌کار نرفته باشد. در این حالت، نمودار توالی طراحی را می‌توان به‌گونه‌ای تغییر داد که یک کنترلگر مورد کاربرد در آن قرار گیرد و نمایش را از اشیای کسب‌وکار مجزا کند.

۱۰-۵-۹- رهنمون‌هایی برای استفاده از کنترلگر

رهنمون ۱۰-۱

وقتی فقط تعداد کمی درخواست کنشگر وجود دارد و سیستم در آینده به واسطه‌ی درخواست‌های بیش‌تر کنشگر گسترش نخواهد یافت، از یک کنترلگرنما استفاده کنید.

اگر تعداد زیادی درخواست کنشگر دارید یا احتمال گسترش سیستم با مورد کاربردهای بیش‌تری هم وجود دارد، باید کنترلگرهای مورد کاربرد را به‌کار برد.

رهنمون ۱۰-۲

از سربارگذاری (Overloading) کنترلگر پرهیز کنید.

اگر مراقب نباشید، یک کنترلگر ممکن است با تعداد بسیار زیادی مسئولیت غیرمرتبط سربار شود. این اتفاق می‌تواند در طراحی اولیه بیافتد که به کنترلگر امکان پردازش درخواست‌های کنشگر را می‌دهد که مربوط به چندین مورد کاربرد خواهند بود. وقتی طراحی را بازنگری کنید، تنها تعداد کمی درخواست کنشگر شناسایی می‌شوند. هر چه طراحی و پیاده‌سازی پیش می‌رود، درخواست‌های کنشگر بیش‌تری شناسایی می‌شوند. در نتیجه، مسئولیت‌های کنترلگر افزایش می‌یابند. از آن‌جا که مسئولیت‌ها از مورد کاربردهای مختلف می‌آیند، کنترلگر با مسئولیت‌هایی سربار می‌شود که باید به کنترلگرهای مورد کاربرد مجزایی واگذار شده باشند. در این حالت، کنترلگر سربارگذاری شده را باید به چندین کنترلگر مورد کاربرد شکست، که هر کدام مسئول رسیدگی به درخواست‌های کنشگر مربوط به یک مورد کاربرد خواهد بود.

چنان‌چه نقش و مسئولیت‌های کنترلگر درست فهمیده نشود، ممکن است تعداد زیادی مسئولیت‌های غیرمرتبط بر عهده‌ی او گذاشته شود. همان‌طور که پیش‌تر بحث شد، کنترلگر نمایش را از اشیای کسب‌وکار جدا می‌کند. مسئولیت کنترلگر این است که بر اشیای کسب‌وکار نظارت کرده و یا آن‌ها را هماهنگ کند تا به اتفاق یکدیگر گام‌های یک مورد کاربرد را انجام دهند. به عبارت دیگر، کنترلگر مسئول این است که ببیند چه موقع چه کاری باید انجام شود، اشیای کسب‌وکار متناسب را فرا بخواند تا آن گام را انجام دهند، و همچنین فعالیت‌های اشیای کسب‌وکار را هماهنگ کند. اگر نقش و مسئولیت‌های یک کنترلگر درست فهمیده نشود، در این‌صورت ممکن است مسئولیت‌های غیرمرتبط به او واگذار شود. چنان‌چه تعداد زیادی از این قبیل مسئولیت‌ها به کنترلگر واگذار شود، در این‌صورت چسبندگی کارکردی کنترلگر کاهش یافته و کنترلگر با مسئولیت‌های غیرمرتبط سربار می‌شود. این ما را به رهنمون بعدی می‌رساند.

رهنمون ۱۰-۳

برای پردازش یک مورد کاربرد، مسئولیت‌ها را به‌درستی به کنترلگر، اشیای کسب‌وکار، و دیگر اشیای مدیریت منابع واگذار نمایید.

طبق تعریف، مورد کاربرد یک فرایند کسب‌وکاری است که از تعدادی گام تشکیل می‌شود. برخی از این گام‌ها، رویه‌ها یا الگوریتم‌های کسب‌وکار را اجرا کرده و با اشیای دسترسی به پایگاه‌داده و/یا شبکه و نیز اشیای کسب‌وکار تعامل می‌کنند. بنابراین مسئولیت‌های پردازش یک مورد کاربرد باید به‌درستی واگذار شده باشند. به‌طور کلی، مسئولیت‌های انجام الگوریتم یا رویه‌های کسب‌وکار باید برعهده‌ی کنترلگر گذاشته شوند، که ممکن است برخی از این مسئولیت‌ها را به دیگر اشیا منتقل نمایند. مسئولیت‌های بازیابی اشیا از پایگاه‌داده و ذخیره‌ی اشیا در پایگاه‌داده باید بر عهده‌ی یک شئ پایگاه‌داده مانند مدیر پایگاه‌داده گذاشته شود. به همین ترتیب، مسئولیت دسترسی به شبکه نیز باید بر عهده‌ی اشیای دسترسی به شبکه یا مدیر شبکه گذاشته شود. مسئولیت بازیابی و به‌روزرسانی وضعیت اشیای کسب‌وکار باید برعهده‌ی خود اشیای کسب‌وکار گذاشته شود. به‌طور خاص، الگوی خبره که در ادامه معرفی خواهد شد، راه حل مؤثری برای واگذاری مسئولیت‌ها به اشیای کسب‌وکار ارائه می‌نماید.

۱۰-۶ الگوی خبره[1]

بخش ۱۰-۵-۹ رهنمون‌هایی جهت استفاده از کنترلگرها را معرفی نمود. چنان‌چه این رهنمون‌ها دنبال نشوند، ممکن است مسئولیت‌های غیرمرتبطی به یک کنترلگر واگذار شود. این می‌تواند منجر به ایجاد کنترلگرهای متورم گردد، یعنی مسئولیت‌های زیادی به یک کنترلگر واگذار شود. طراحی شکل ۱۰-۸ این مسائل را نمایش می‌دهد.

۱۰-۶-۱- خبره‌ی اطلاعات[2]

راه حل دو مشکلی که در بالا بدان اشاره گردید برداشتن مسئولیت‌های اضافی از کنترلگر و واگذاری آن‌ها به دیگر اشیا می‌باشد. سؤال این است که برای پاسخگویی به یک درخواست، مسئولیت را باید به کدام شئ واگذار کنیم؟ این موضوع در شکل ۱۰-۹-الف نشان داده شده است.

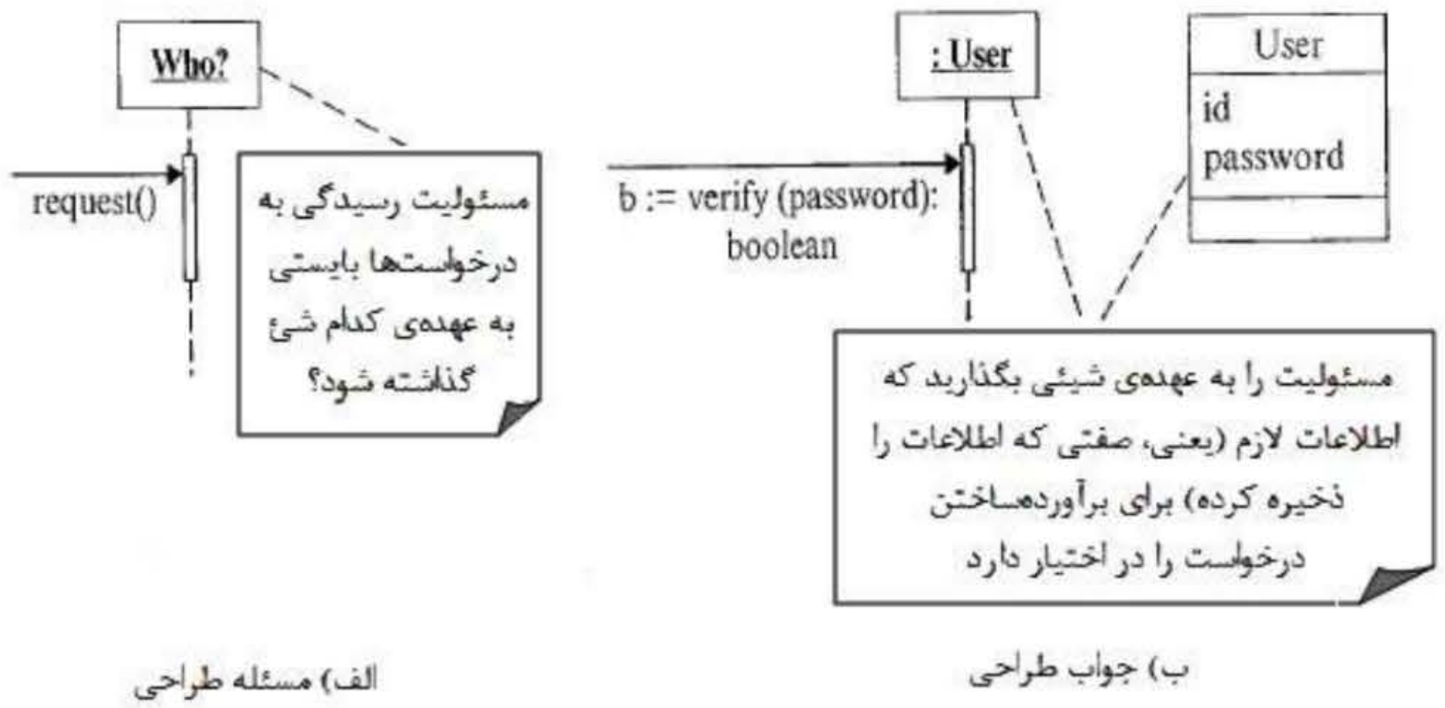
به‌وضوح، شییی که مسئول رسیدگی به یک درخواست است باید اطلاعات لازم برای ارضای آن درخواست را در اختیار داشته باشد. همان‌گونه که در شکل ۱۰-۹-ب نشان داده شده است، درخواستی که وارسی می‌کند آیا کلمه‌ی عبور معتبر است یا خیر، باید به شئ کاربر واگذار شود. این بدین خاطر است که شئ User کلمه‌ی عبور را درون یکی از صفت‌هایش در اختیار دارد؛ بنابراین، می‌تواند با استفاده از آن صفت، اعتبار کلمه‌ی عبور را بررسی نماید. اگر مسئولیت به شئ دیگری، غیر از شئ User واگذار شود، در این‌صورت آن شئ مجبور خواهد بود برای به‌دست آوردن کلمه‌ی عبور از شئ User سؤال کند و اعتبار کلمه‌ی عبور را بررسی کند. در هر حالت،جفت‌شدگی غیرضروری ایجاد شده و امنیت کاربر ممکن است به مخاطره بیفتد.

بحث بالا ما را به الگوی خبره‌ی اطلاعات، یا به سادگی الگوی خبره، می‌رساند. مسئله‌ی طراحی این است که مسئولیت رسیدگی به یک درخواست را به چه کسی باید واگذار نمود؟ راه حل این است که مسئولیت (رسیدگی به درخواست) را به شییی که اطلاعات لازم برای ارضای درخواست را دارد واگذار نماییم.

---

[1] The Expert Pattern

[2] Information Export



الف) مسئله طراحی

ب) جواب طراحی

شکل ۱۰-۹: نمایش الگوی خبره

۱۰-۶-۲- اعمال الگوی خبره

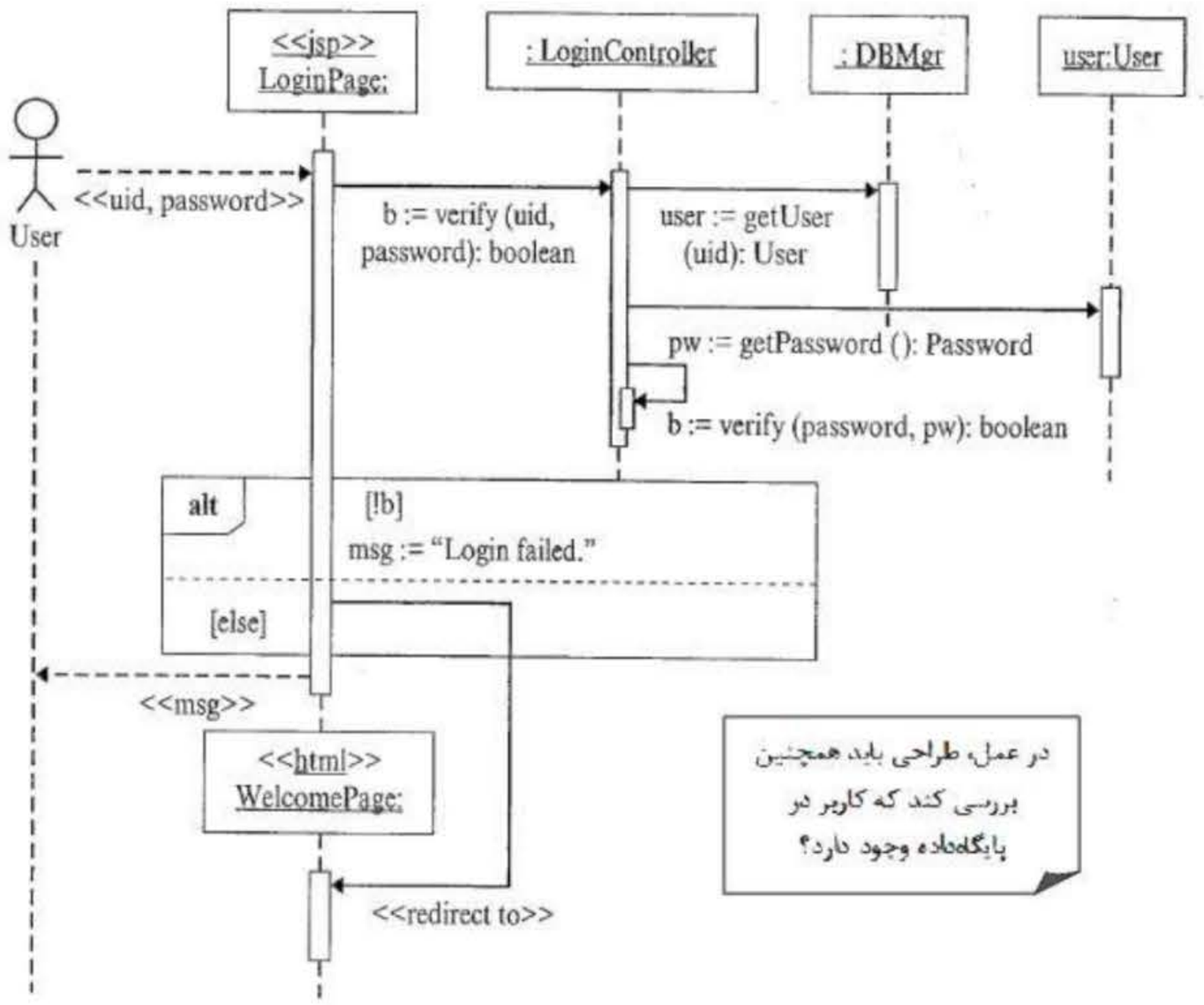
نمودار توالی Login در شکل ۱۰-۱۰ را در نظر بگیرید. برخلاف کنترلگر شکل ۱۰-۶ ، کنترلگر ۱۰-۱۰ همه چیز را خودش انجام نمی‌دهد، بلکه از DBMgr درخواست می‌کند که شئ User را بازیابی نموده و کلمه‌ی عبور را از این شئ بگیرد. سپس این کنترلگر وارسی می‌کند که آیا کلمه‌ی عبور وارد شده با کلمه‌ی عبور ذخیره شده تطابق دارد یا خیر.

طراحی شکل ۱۰-۱۰ غیر معمول نیست، ولی مشکلاتی دارد. نخست، مسئولیت وارسی این‌که آیا کلمه‌ی وارد شده با کلمه‌ی ذخیره شده تطابق دارد یا خیر، باید مسئولیت شئ User باشد، نه مسئولیت کنترلگر Login. بدین معنی که، مسئولیت به شئ درستی واگذار نشده است. خواننده ممکن است تعجب کند که چرا این اتفاق افتاده است. آیا نباید Login Controller مسئول پردازش مورد کاربرد Login و لذا وارسی کلمه عبور باشد؟ دلیل این است که تنها شئ User است که اطلاعات کلمه‌ی عبور را برای وارسی کلمه عبور وارد شده در خود دارد. کنترلگر این اطلاعات را ندارد. اگر این مسئولیت به کنترلگر واگذار شود، آنگاه کنترلگر مجبور خواهد بود از شئ User بخواهد کلمه‌ی عبور را به او بدهد تا بتواند اعتبار کلمه عبور را وارسی کند. اما این کار اصل «ساده واحمقانه بگیر» را ارضا نمی‌کند، زیرا این‌که به شئ User اجازه دهیم کلمه عبور را وارسی کند راه حل ساده‌تری خواهد بود.

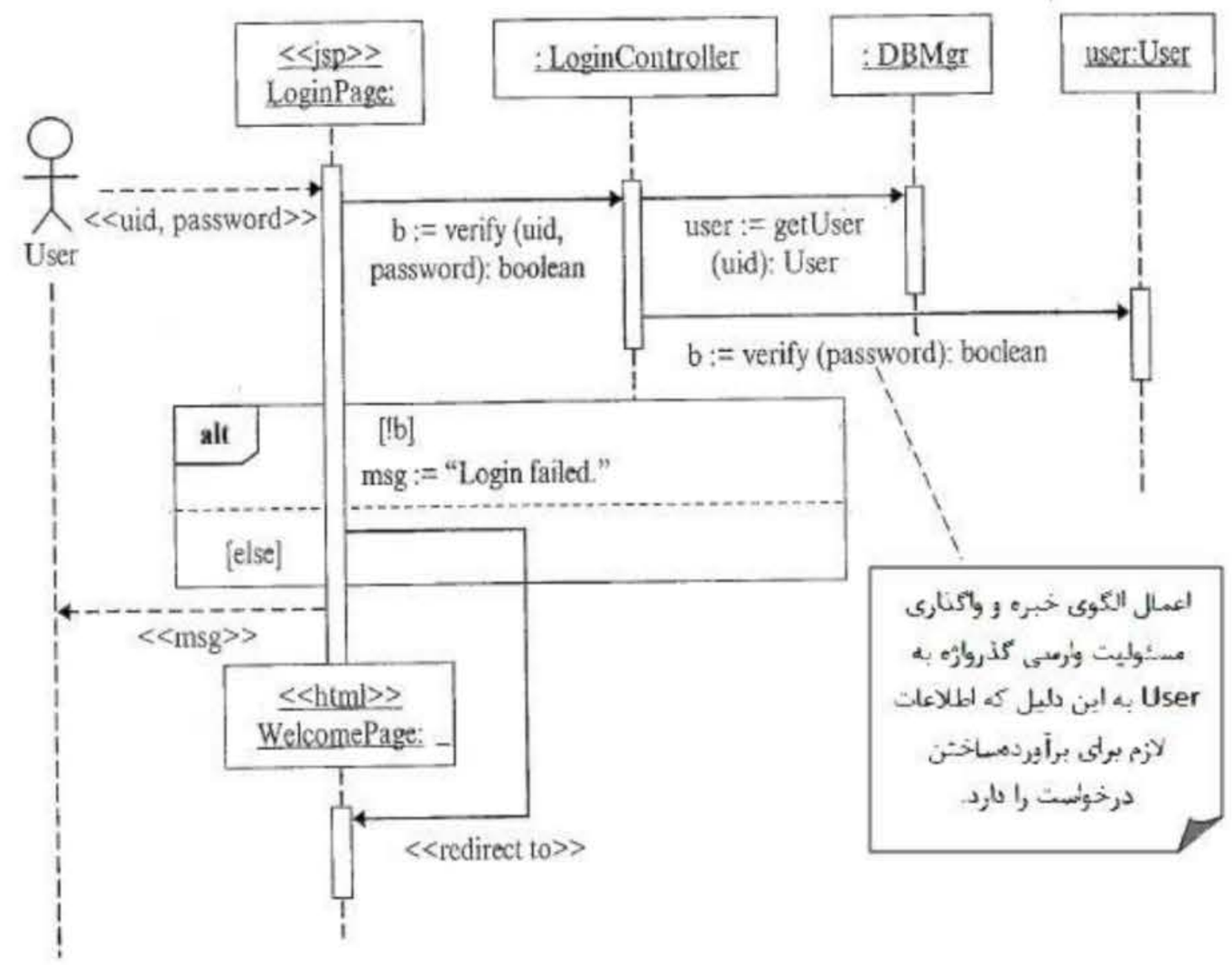
دوم، امنیت اقتضا می‌کند که کلمه‌های عبور باید رمزگذاری شوند. در این حالت، Login Controller شکل ۱۰-۱۰ موظف است رمزگذاری و رمزبرداری کلمه عبور را بداند. مسئله‌ی امنیتی دیگر در مورد طراحی شکل ۱۰-۱۰ این است که تابع getPassword() : Password متعلق به User به یک شئ اجازه می‌دهد کلمه‌ی عبور یک کاربر را بازیابی کند. اگر کلمه‌ی عبور رمزگذاری نشده باشد، در این‌صورت یک آسیب‌پذیری امنیتی وجود خواهد داشت. به‌کاربردن الگوی خبره این مشکلات را حل می‌کند. نتیجه در شکل ۱۰-۱۱ نشان داده شده است، که در آن تنها یک فراخوانی به تابع verify(password) : boolean از شئ User لازم است.

۱۰-۶-۳- الگوی خبره شامل بیش از یک شئ

اغلب، لازم است بیش از یک خبره باهم تعامل کنند تا درخواستی را ارضا نمایند. برای مثال، نمودار توالی Checkout در شکل ۱۰-۶ را درنظر بگیرید. مشخص است که وقتی مدرکی از کتابخانه خارج می‌شود، سیستم تاریخ انقضای امانت (تاریخ



شکل ۱۰-۱۰: یک نمودار توالی Login



شکل ۱۰-۱۱: اعمال الگوی خبره

برگشت) را محاسبه می‌کند. در این نمودار توالی، این‌کار درون سازنده‌ی کلاس Loan انجام شده است، که در نمادگان UML به صورت create(p:patron , d:Document) نوشته شده است. ولی تاریخ برگشت بستگی به عضو دارد، زیرا استاد، کارمند، یا دانشجو هر کدام مدت امانت متفاوتی دارند. تاریخ برگشت همچنین به نوع مدرک هم بستگی دارد؛ مثلاً دانشجو می‌تواند یک کتاب را برای یک ماه و یک مجله را برای دو هفته به امانت ببرد. کتابی که رزرو شده باشد فقط می‌تواند برای مدت زمان مشخصی، که توسط استادی که کتاب را برای درس خاصی رزرو کرده مشخص شده است، به امانت برده شود.

بحث بالا نشان می‌دهد که درخواست محاسبه‌ی تاریخ برگشت باید حداقل سه شئ را درگیر نماید: عضو، مدرک، و تقویم. از آنجا که شئ Loan به تمام این اشیایی که اطلاعات مورد نیاز را در اختیار دارند دسترسی دارد، شایسته است این مسئولیت را به شئ Loan واگذار نماییم.

۱۰-۶-۴- چه موقع الگوی خبره به‌کار می‌رود؟

موقعیت‌های مختلفی وجود دارد که الگوی خبره می‌تواند استفاده شود. این موقعیت‌ها مشابه کاربرد الگوی کنترلگر می‌باشند که در بخش ۱۰-۵-۸ مطرح شد. اما، به جای معرفی یک کنترلگر مورد کاربرد، طراح باید خبره‌ی اطلاعات را شناسایی نموده و درخواست را به او واگذار نماید.

۱۰-۶-۵- رهنمون‌هایی برای استفاده از خبره

رهنمون ۱۰-۴

به دنبال خبره‌ی اطلاعات ابتدا در نمودار توالی جاری، سپس در نمودار کلاس طراحی، و سرانجام در مدل دامنه بگردید.

رهنمون ۱۰-۵

اطلاعات لازم برای برآورده نمودن یک درخواست را می‌توان میان دو یا چند شئ توزیع نمود. در این حالت، باید مسئولیت را به شییی واگذار نمود که به اشیا دسترسی داشته و لذا طراحی به‌دست آمده کمترین جفت‌شدگی ممکن را داشته باشد.

مثال مطرح شده در بخش ۱۰-۶-۳ را در نظر بگیرید. اطلاعات لازم برای برآورده ساختن درخواست محاسبه‌ی تاریخ برگشت در بین چند شئ توزیع شده است. کنترلگر خروج و اشیای Loan به اشیای مربوطه دسترسی دارند. اگر مسئولیت به واگذار شود، در این‌صورت کنترلگر مجبور است با شئ Patron، شئ Document، شئ Calendar ، و شئ Loan تعامل داشته باشد (تا تاریخ برگشت را تنظیم نماید). یعنی، چهار فراخوانی تابع یا چهار وابستگی مورد نیاز است. اگر مسئولیت به شئ Loan واگذار گردد، در این‌صورت تنها سه فراخوانی تابع یا سه وابستگی مورد نیاز است. بنابراین، بهتر است مسئولیت به شئ Loan واگذار گردد.

۱۰-۷ الگوی ایجادکننده

در مثالی در بخش ۱۰-۶-۳ بحث شد، شئ Checkout Controller شئ Loan را ایجاد می‌نماید. ممکن است تعجب کنید که چرا ما به Checkout Controller اجازه دادیم شئ Loan را ایجاد کند و چه کسی باید Checkout Controller، Checkout

GUI، و اشیای دیگر را ایجاد کند؟ الگوی ایجادکننده به این پرسش‌ها پاسخ می‌دهد.

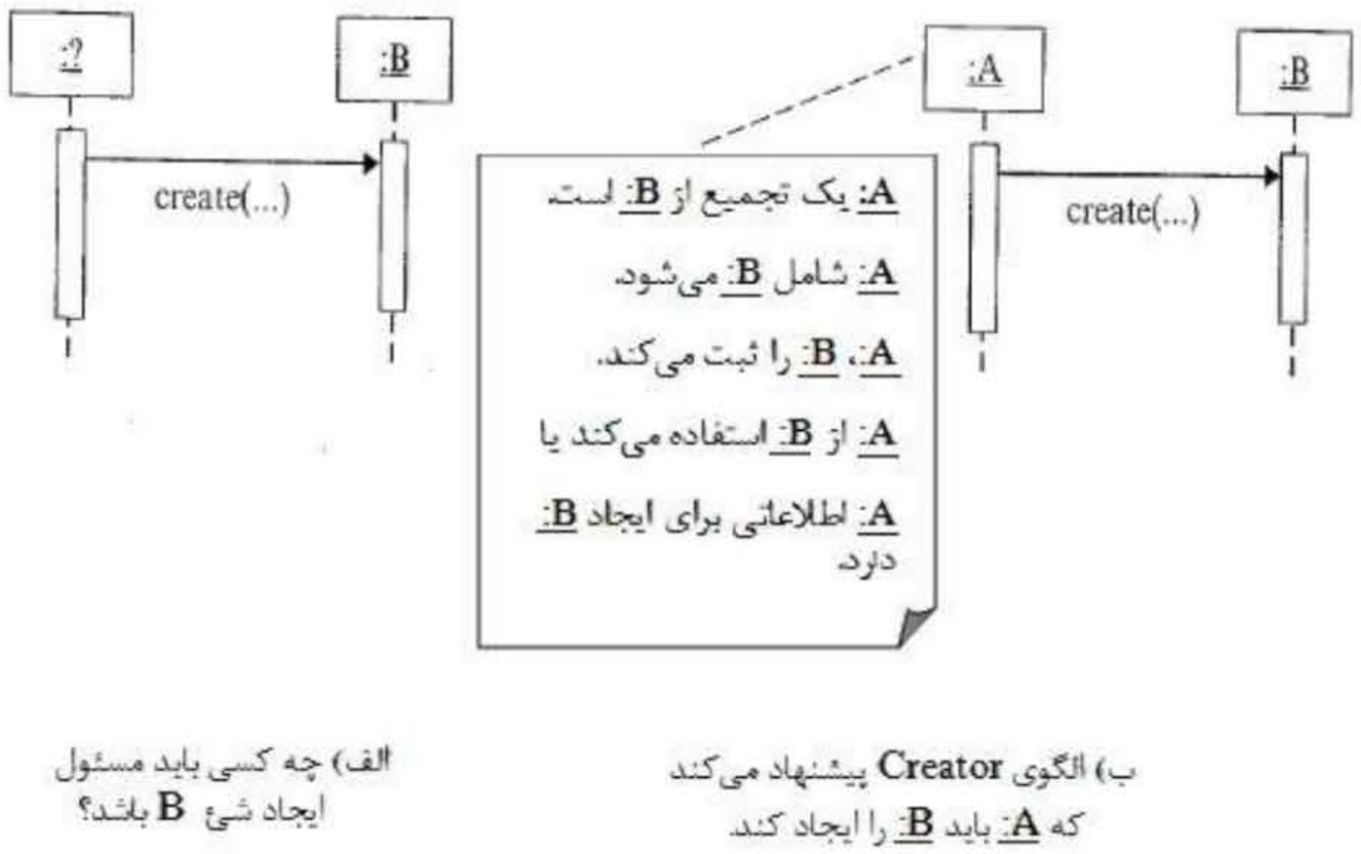
۱۰-۷-۱- ایجادکننده چیست؟

ایجاد شئ یک فعالیت متداول در یک سیستم شئ‌گرا است. بنابراین، این‌که مسئولیت ایجاد یک شئ باید برعهده‌ی چه کسی گذاشته شود نیاز به توجه و راهنمایی خاص دارد. الگوی ایجادکننده به این منظور کمک می‌کند و در شکل ۱۰-۱۲ نشان داده شده است. بدین شکل که، الگوی ایجادکننده پیشنهاد می‌کند که مسئولیت ایجاد یک شئ از کلاس B را باید به شیئی از کلاس A واگذار نمود اگر شرایط زیر برقرار باشند.

۱. کلاس *A* اجتماعی از کلاس*B* است. از آن‌جا که اشیای کلاس A متشکل از اشیای کلاس B هستند، ساده و سرراست این است که مسئولیت ایجاد را به کلاس A بسپاریم. به دلیل این‌که وابستگی کلاس A به کلاس B از قبل وجود دارد، فراخوانی سازنده‌ی کلاس B از قبل وجود دارد، فراخوانی سازنده‌ی کلاس B وابستگی اضافی تولید نمی‌کند. بازاستفاده از کلاس A ساده‌تر از این است که بگذاریم کلاس دیگری اشیای کلاس B را ایجاد کند چون این‌کار لازم دارد که کلاس، بازاستفاده شود.
۲. شیئی از کلاس *A* شامل اشیایی از کلاس *B* است. این واقعیت که اشیای کلاس A شامل اشیایی از کلاس B هستند می‌گوید که ممکن است اولی به کرات نیاز به استفاده یا به‌روزرسانی دومی پیدا کند. اگر این‌طور باشد، این‌که به اشیای کلاس Aاجازه داده شود اشیای کلاس B را ایجاد کنند می‌تواند منجر به طراحی ساده‌تر و قابل فهم‌تری بشود. اما، استثناهایی هم وجود دارد. برای مثال، موارد متعددی وجود دارند که یک کلاس ظرف[۱] فقط برای ذخیره‌ی عناصری که توسط دیگر کلاس‌ها ایجاد شده‌اند، استفاده می‌شود.
۳. یک شئ از کلاس *A* اشیایی از کلاس *B* را رکورد می‌کند. موارد زیادی وجود دارند که یک شئ از کلاس A اشیایی از کلاس B را نگه می دارد. برای مثال، پرونده‌ی پزشکی بیمار نتایج آزمایش‌های وی را رکورد می‌کند؛ ثبت ورود و خروج پرونده‌ی پزشکی فعالیت‌های ورود و خروج را رکورد می‌کند؛ و تاریخچه‌ی خرید جزییات هر قلم خرید را رکورد می‌کند. در این مثال‌ها و موارد مشابه، ممکن است ساده‌تر و راحت‌تر این باشد که پارامترهای لازم را بهترتیب به پرونده‌ی پزشکی، ثبت، و تاریخچه‌ی خرید ارسال نموده و بگذاریم آن‌ها عناصر را ایجاد کنند.
۴. یک شئ از کلاس *A* به‌شدت از اشیای کلاس *B* استفاده می‌نماید. موارد زیادی وجود دارد که یک شئ از کلاس A بهشدت از اشیای کلاس B استفاده می‌کند. این موارد شامل کلاس‌های اجتماع[۲]، ظرف، یا رکوردر می‌شود که عناصر خود را به تناوب بازیابی یا به‌روزرسانی می‌کنند. به عنوان مثال می‌توان به جمع‌دار اموال[۳] یک انبار اشاره نمود که باید مرتب رکوردهای انبار را بررسی و به‌روزرسانی نماید. این‌که به جمع‌دار اموال انبار اجازه داده شود رکوردهای انبار را ایجاد نماید طراحی را ساده‌تر نموده و فهم و بازاستفاده از آن آسان خواهد شد.
۵. یک شئ از کلاس *A* اطلاعات لازم برای ایجاد اشیای کلاس *B* را دارد. در خیلی از موارد پارامترهای ورودی باید به سازنده‌ی یک کلاس ارسال شوند. در چنین حالاتی، ممکن است راحت‌تر باشد که به شیئی که پارامترهای ورودی را دارد اجازه دهیم سازنده را فراخوانی نماید، در صورتی که این کار چسبندگی ایجادکننده را خراب نکند. در شکل ۱۰-۶ سازنده‌ی کلاس Loan نیاز به یک شئ عضو و یک شئ مدرک دارد. کنترلگر خروج[۴] این‌ها را در اختیار دارد، لذا ایجاد شئ Loan مسئولیت این کنترلگر خواهد بود. بنابراین، این کنترلگر برای ایجاد شئ Loan نیز انتخاب خواهد شد.

---

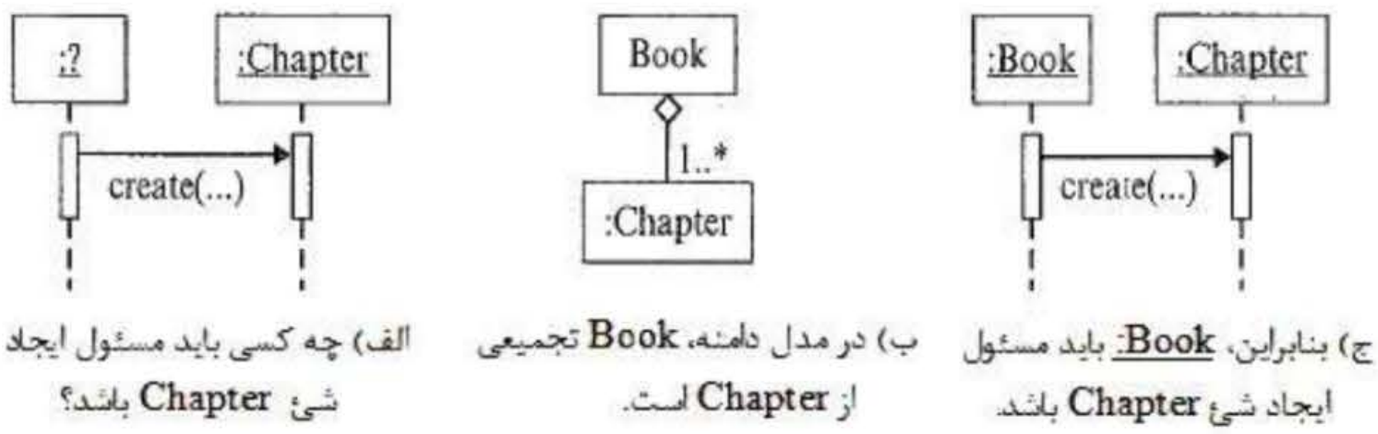
[۱] Container
[۲] Aggregate
[۳] Bookkeeper
[۴] Checkout Controller



شکل ۱۰-۱۲: نمایش الگوی ایجادکننده

۱۰-۷-۲- اعمال الگوی ایجادکننده

در این بخش، با تعدادی مثال نشان داده می‌شود که الگوی ایجادکننده چگونه باید استفاده شود. در شکل ۱۰-۱۳-الف می‌خواهیم ببینیم چه کسی باید مسئول ایجاد یک شئ فصل[1] باشد. با بررسی مدل دامنه، همان‌طور که در شکل ۱۰-۱۳-ب نشان داده شده است، می‌فهمیم که هر شئ کتاب[2] اجتماعی از یک یا چند شئ Chapter است. بنابراین، در شکل ۱۰-۱۳-ج مسئولیت ایجاد شئ Chapter به شئ Book واگذار شده است؛ یعنی Book سازنده‌ی کلاس Chapter را صدا زده است.



شکل ۱۰-۱۳: چه کسی باید یک شئ Chapter ایجاد کند؟

در شکل ۱۰-۱۴-الف، چه کسی باید ایجادکننده‌ی شئ موقعیت[3] باشد؟ در بررسی‌ها می‌بینیم که اصطلاح موقعیت بیان‌گر تعداد سهام‌هایی از بازار بورس که یک نفر خریداری می‌نماید و نیز میانگین هزینه این سهام‌ها می‌باشد. وقتی شما سهامی را می‌خرید یا می‌فروشید، تعداد سهام و میانگین هزینه تغییر می‌کند. همان‌طور که شکل ۱۰-۱۴-ب نشان می‌دهد، Portfolio دربردارنده‌ی Positionها می‌باشد. یعنی، وقتی شما سهامی را خرید یا فروش می‌کنید، موقعیت‌ها از طریق یک

[1] Chapter
[2] Book
[3] Position

شئ Portfolio به‌روزرسانی می‌شوند. بنابراین، اشیای Portfolio باید مسئول ایجاد اشیای Position در Portfolio باشند. این در شکل ۱۰-۱۴-ج نشان داده شده است.



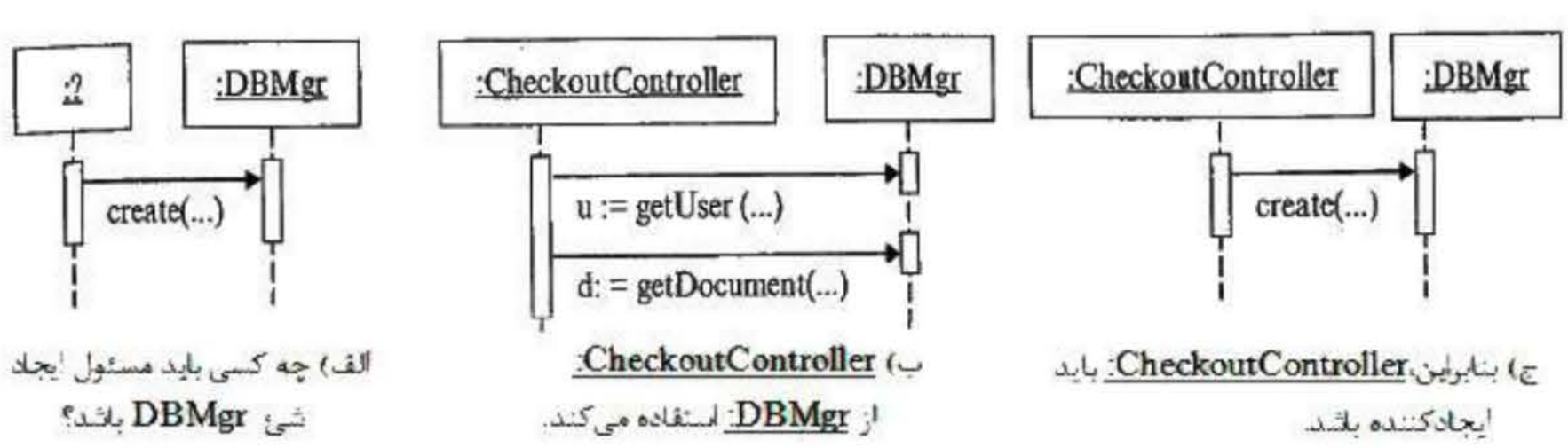
الف) چه کسی باید مسئول ایجاد شئ Position باشد؟

ب) در مدل دامنه، Portfolio نگهدارنده‌ی Positionها است.

ج) بنابراین، Portfolio: باید مسئول ایجاد شئ Position باشد.

شکل ۱۰-۱۴: چه کسی باید مسئول ایجاد شئ Position باشد؟

سومین مثال نشان می‌دهد که Checkout Controller از شئ DBMgr استفاده کرده و اشیای کسب‌وکار را از پایگاه‌داده بازیابی می‌کند؛ بنابراین، شئ Checkout Controller باید مسئول ایجاد شئ DBMgr باشد. این در شکل ۱۰-۱۵ نشان داده شده است. درنهایت، برای ایجاد یک شئ Loan ، باید یک شئ User و یک شئ Document داشته باشیم، زیرا Loan یک کلاس انجمنی برای انجمن قرض (User, Document) است (شکل ۱۰-۱۶-ب) و Checkout Controller این اشیا را دارد همان‌گونه که در شکل ۱۰-۱۶-ب نشان داده شده است. بنابراین، Checkout Controller باید ایجادکننده‌ی شئ Loan باشد.

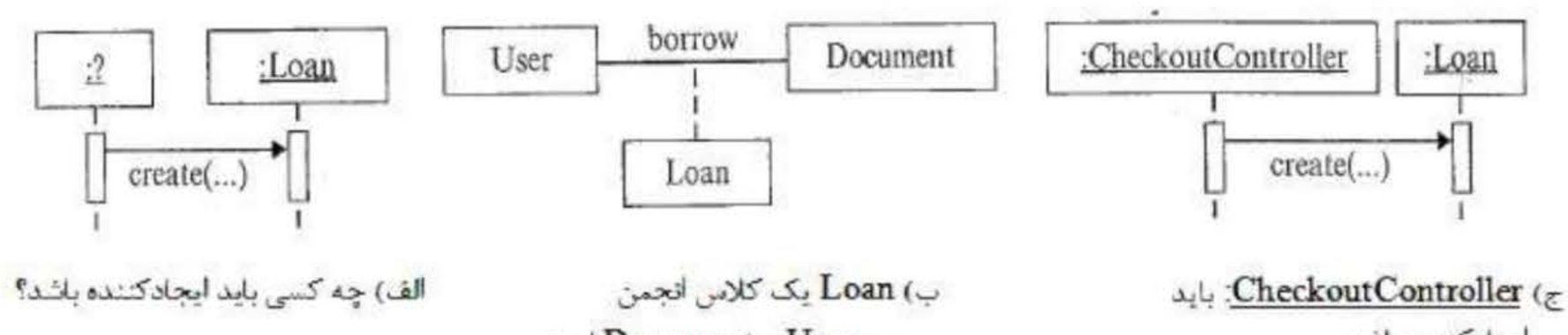


الف) چه کسی باید مسئول ایجاد شئ DBMgr باشد؟

ب) CheckoutController: از DBMgr استفاده می‌کند.

ج) بنابراین، CheckoutController: باید ایجادکننده باشد.

شکل ۱۰-۱۵: چه کسی باید شئ DBMgr را ایجاد کند؟



الف) چه کسی باید ایجادکننده باشد؟

ب) Loan یک کلاس انجمن بین User و Document است.

ج) CheckoutController: باید ایجادکننده باشد.

شکل ۱۰-۱۶: چه کسی باید شئ Loan را ایجاد کند؟

۱۰-۷-۳- فواید الگوی ایجادکننده

الگوی ایجادکننده باعث جفت‌شدگی کم و بازاستفاده‌پذیری بهتر نرم‌افزار می‌شود. وابستگی بین ایجادکننده و شییی که باید ایجاد شود از قبل وجود دارد. بنابراین، الگوی ایجادکننده وابستگی اضافه‌تری تولید نمی‌کند. این بدان معنی است که جفت‌شدگی پایینی خواهیم داشت. این‌کار همچنین بازاستفاده از ایجادکننده را تسهیل می‌کند زیرا ایجادکننده اشیای وابسته‌اش را ایجاد می‌کند؛ به عبارت دیگر، برای ایجاد اشیای وابسته، نیازی به بازاستفاده از هیچ چیز دیگری نیست.

۱۰-۷-۴- چه موقع الگوی ایجادکننده به‌کار می‌رود؟

موقعیت‌های استفاده از الگوی ایجادکننده نیز همانند الگوی خبره می‌باشد. یعنی بسته به زمینه‌ی توسعه وقتی طراح در حال نوشتن یا تغییر سناریوی مورد کاربرد است، یا در حین ساخت نمودار توالی این الگو را می‌توان به کار برد.

خلاصه

این فصل الگوی کنترلگر، خبره، و ایجادکننده را معرفی می‌کند. این الگوها یک مسئله‌ی متداول طراحی را مد نظر قرار می‌دهند؛ و آن این‌که چگونه می‌توان مسئولیت‌ها را به اشیا واگذار نمود؟ به‌طور خاص، این الگوها به سؤالات زیر پاسخ می‌دهند:

۱. مسئولیت رسیدگی به درخواست کنشگر را به چه کسی باید سپرد؟ پاسخ الگوی کنترلگر است.
۲. مسئولیت رسیدگی به درخواستی از لایه‌ی اشیای کسب‌وکار را باید به چه کسی سپرد؟ پاسخ الگوی خبره است.
۳. مسئولیت ایجاد یک شئ را به چه کسی باید داد؟ پاسخ الگوی ایجادکننده است.

منظور از الگوی کنترلگر جداسازی نمایش از منطق کسب‌وکار است، که این کار با واگذاری مسئولیت پردازش درخواست کاربر به یک کنترلگر به‌جای نمایش، شدنی است. الگوی خبره پیشنهاد می‌کند مسئولیت رسیدگی به یک درخواست مشخص در لایه‌ی اشیای کسب‌وکار باید به خبره‌ی اطلاعات واگذار شود؛ که این خبره، اطلاعات لازم برای برآورده کردن درخواست را دارد. الگوی ایجادکننده می‌گوید که مسئولیت ایجاد یک شئ خاص باید به شییی داده شود که یا اجتماعی از آن شئ خاص است؛ یا به آن وابستگی دارد؛ یا اطلاعات لازم برای ایجاد آن شئ را دارد.

این فصل مسائلی را که در طراحی‌های معمول که از الگوهای کنترلگر، خبره، و ایجادکننده استفاده نکرده‌اند، رخ می‌دهد مورد بحث قرار می‌دهد. سپس نشان می‌دهد که چگونه باید از الگوهای نام‌برده استفاده نمود و طراحی را بهبود بخشید. این مباحث به دانشجو کمک می‌کند تا نه تنها چیستی الگوهای کنترلگر، خبره، و ایجادکننده را درک کرده و چگونگی اعمال آن‌ها را فراگیرد، بلکه از آن مهم‌تر، بفهمد که چه موقع و چرا از این الگوها استفاده کند. دانستن این پاسخ‌ها دانشجو را قادر می‌سازد که این سه الگوی GRASP را در دنیای عملی توسعه‌ی نرم‌افزار به‌درستی اعمال کند.

برای مطالعه‌ی بیش‌تر

کتاب Larman [۱۰۴] سه الگوی کنترلگر، خبره، و ایجادکننده و نیز سایر الگوهای GRASP را معرفی می‌نماید. خلاصه‌ای از نه الگوی GRASP در شکل ۱۰-۱۷ ارائه شده است. کتاب مذکور همچنین ۲۳ الگوی خاص موقعیت که به الگوهای چهار تفنگدار [۶۷] معروفند را نیز معرفی می‌نماید. این الگوها در فصول بعدی این کتاب توضیح داده خواهند شد.

#	نام	مسئله	راه حل
۱	Controller	چه کسی مسئول کنترل یک رویداد سیستمی است؟	مسئولیت را به یک کنترلگر واگذار می‌کند.
۲	Creator	چه کسی باید درخواست ایجاد یک شئ را بدهد؟	مسئولیت به کلاس B واگذار می‌شود، اگر: • B یک تجمیع از A باشد. • B شامل اشیای A باشد. • B، A را ثبت کند. • B به تکرار از A استفاده کند. • B اطلاعاتی برای تعریف A دارد.
۳	High Cohesion	چگونه اشیا نگهداری شوند تا درک و مدیریت آسانی داشته باشند؟	اصول طراحی نرم‌افزار چسبده که در فصل طراحی معماری شرح داده شد را اعمال می‌کند.
۴	Indirection	چگونه یک مسئولیت واگذار شود تا از پیوستگی مستقیم اجتناب شود؟	مسئولیت را به یک شئ میانی واگذار می‌کند تا تجزیه بین اشیاء را تعدیل کند.
۵	Expert	چه کسی مسئول کنترل یک درخواست وظیفه‌مند است؟	مسئولیت را به یک شئی واگذار می‌کند که اطلاعاتی (به عنوان مثال صفات) برای تکمیل درخواست دارد.
۶	Low Coupling	چگونه جفت‌شدگی کم، تاثیر کم تغییر و قابلیت استفاده مجدد بالا تضمین می‌شود؟	اصول طراحی نرم‌افزار جفت‌شدگی کم که در فصل طراحی معماری شرح داده شد را اعمال می‌کند.
۷	Polymorphism	چگونه تنوع رفتارها بدون دستورات شرطی کنترل شود؟	در یک کلاس پدر، یک واسط برای رفتارهای متنوع تعریف می‌کند و اجازه پیاده‌سازی تنوع رفتار را برای زیرکلاس‌ها فراهم می‌کند.
۸	Protected Variation	چگونه اشیا طراحی شود تا از تاثیر نامطلوب به سایر اشیاء جلوگیری شود؟	یک واسط پایدار برای پوشاندن تنوع‌های درونی یا ناپایداری‌ها طراحی می‌کند. بنابراین تنوع یا ناپایداری بر سایر اشیا اثر نخواهد داشت.
۹	Pure Fabrication	چه کسی مسئول کنترل مسئولیت‌هایی است که به دلیل تجاوز از چسبندگی بالا و جفت‌شدگی کم، قابل انتساب به اشیای مفاهیم دامنه نیستند؟	این مسئولیت‌ها را به زیرمجموعه مسئولیت‌های چسبنده تقسیم می‌کند و هر زیرمجموعه را به یک کلاس مصنوعی که pure fabrication نامیده می‌شود، واگذار می‌کند.

شکل ۱۰-۱۷: خلاصه ای از الگوهای GRASP

کتاب Mckean , wirfs-Bock [۱۶۳] جدیدترین روش‌ها در طراحی مسئولیت‌رانده[1] را معرفی می‌کند. منظور از طراحی مسئولیت‌رانده عبارت است از طراحی اشیا با نقش‌ها و مسئولیت‌های مشخص و به‌عنوان اعضایی از یک جامعه از اشیا.

سوال‌های مروری فصل

۱. الگوهای طراحی نرم‌افزار چیستند؟
۲. کنترلگر، کنترلگر مورد کاربرد، خبره، و ایجادکننده چیست؟
۳. چه موقع و چگونه باید از الگوهای کنترلگر، خبره، و ایجادکننده استفاده نمود؟
۴. مزایای الگوهای کنترلگر، خبره، و ایجادکننده، چیست؟
۵. اگر جایی که باید از این الگوها استفاده شود، این‌کار صورت نگیرد چه مشکلات احتمالی پیش خواهد آمد؟
۶. آیا یک نمودار توالی می‌تواند بیش از یک کنترلگر مورد کاربرد داشته باشد؟ توضیح دهید چرا؟
۷. کنترلگر متورم چیست؟ و چگونه باید آن را درمان کرد؟

[1] Responsibility-Driven Design (RDD)

تمرین‌ها

۱۰-۱ جدولی بسازید که ستون‌های آن الگوهای کنترلگر، خبره، و ایجادکننده و ردیف‌های آن اصول طراحی که در فصل ۶ معرفی گردیدند، باشد. خانه‌های جدول را به‌گونه‌ای علامت بزنید که نشان دهد کدام الگو از کدام اصل طراحی پشتیبانی می‌کند. برای نتیجه‌ای که به‌دست می‌آید توضیحاتی ارائه دهید.

۱۰-۲ کاربرد اجاره‌ی برخط خودرو[۱] که در پیوست ت معرفی شده است را ملاحظه کنید. برخی از مورد کاربردهای این کاربرد عبارتند از جست‌وجوی خودرو[۲] ، رزرو خودرو[۳]، ویرایش رزرواسیون[۴]، و لغو رزرواسیون[۵]. کارهای زیر را انجام دهید و درعین‌حال هر جا که مناسب است از الگوهای کنترلگر، خبره، و ایجادکننده نیز استفاده کنید.

الف) مورد کاربردهای گسترده را تبیین کرده و گام‌های غیربدیهی آن‌ها را شناسایی کنید (در صورتی که مشخص نشده‌اند).

ب) سناریوها، جدولهای سناریو، و نمودارهای توالی را برای گام‌های غیربدیهی تولید کنید.

۱۰-۳ بهبودهایی که الگوهای کنترلگر، خبره، و ایجادکننده در طراحی رفتاری سیستم اجاره‌ی برخط خودرو معرفی کرده‌اند را شناسایی و توضیح دهید.
راهنمایی: طراحی خود را با آن‌که از الگوها استفاده نکرده‌است مقایسه کرده و بهبودها را شناسایی کنید.

۱۰-۴ مشابه تمرین ۱۰-۲ را در خصوص مورد کاربرد ویرایش نمودار کلاس[۶] از ویراستار نمودار کلاس UML[۷] انجام دهید.

۱۰-۵ بهبودهایی که الگوهای کنترلگر، خبره، و ایجادکننده در طراحی رفتاری ویراستار نمودار کلاس UML معرفی کرده‌اند را شناسایی و توضیح دهید.
راهنمایی: طراحی خود را با آن‌که از الگوها استفاده نکرده‌است مقایسه کرده و بهبودها را شناسایی کنید.

۱۰-۶ کدام‌یك از کلاس‌ها در طراحی ویراستار نمودار کلاس UML می‌تواند منفرد باشد؟ دلیل بیاورید.

۱۰-۷ متنی بنویسید و در آن در مورد شباهت‌ها و تفاوت‌های الگوهای کنترلگر، خبره، و ایجادکننده بحث کنید.

۱۰-۸ برای کاربرد الگوهای کنترلگر، خبره، و ایجادکننده سه موقعیت واقعی، نه الکی یا فرضی، را تشریح کنید.

۱۰-۹ در تمرین ۸-۶ شما مورد کاربردهای گسترده برای یك ویراستار نمودار حالت را تولید کردید. در این تمرین، باید گام‌های غیربدیهی را شناسایی کنید. برای هر گام غیربدیهی، یك سناریو بنویسید، یك جدول سناریو بسازید، و سپس جدول سناریو را به نمودار توالی طراحی تبدیل کنید. در حین این فرایند، هر جا که مناسب است، الگوهای کنترلگر، خبره، ایجادکننده، و منفرد را اعمال نمایید. ترجیحاً این الگوها باید در موقعی که سناریوها را توصیف می‌کنید، به‌کاربرده شوند.

---

[۱] Online Car Rental
[۲] Search for Cars
[۳] Reserve Car
[۴] Edit Reservation
[۵] Cancel Reservation
[۶] Edit Class Diagram
[۷] UML Class Diagram

۱۱ استنتاج نمودار کلاس طراحی

نکات کلیدی

- نمودار کلاس طراحی یك نمودار کلاس UML است که از مدل‌های رفتاری و مدل دامنه به‌دست می‌آید. این نمودار برای توسعه‌ی آزمون‌رانده ،آزمون یکپارچه‌سازی، و نگهداری حکم یك اوزالید طراحی را دارد.
- نمودارهای بسته[۱] برای سازمان‌دهی و مدیریت کلاس‌های یك DCD بزرگ مفید می‌باشند.

در فصول قبلی، فعالیت‌های مدل‌سازی، تحلیل، و طراحی از متدولوژی معرفی گردیدند، این فعالیت‌ها، از نظر ترتیب اجرایی و وابستگی یک فعالیت به دست‌ساخته‌های نرم‌افزاری که در فعالیت دیگر تولید می‌شوند، به‌هم مرتبط می‌باشند. برای ایجاد انگیزه در مورد مطالبی که جلوتر می‌آید، در زیر گام‌هایی که تاکنون معرفی شده‌اند به اختصار مرور می‌شوند.

۱. مهندسی سیستم (فصل ۳). در این گام تبیین نیازمندی‌های سیستم برای کل سیستم و نیز معماری کلی سیستم تولید می‌شوند. معماری کلی سیستم، زیرسیستم‌های سخت‌افزاری، نرم‌افزاری، و انسانی و روابط بین این زیرسیستم‌ها را نشان می‌دهد. نیازمندهای سیستم به زیرسیستم‌ها تخصیص داده می‌شوند.
۲. تحلیل نیازمندی‌های سیستم (فصل ۴). در این گام نیازمندی‌های سیستم که به زیرسیستم‌های نرم‌افزاری تخصیص یافته‌اند، تحلیل و پالایش شده و نیازمندی‌های نرم‌افزاری اضافه شناسایی می‌شوند.
۳. مدل‌سازی دامنه (فصل ۵). در این گام، یك مدل دامنه ساخته یا گسترش می‌یابد که به تیم توسعه کمك می‌کند نیازمندی‌ها و دامنه‌ی کاربرد را درك کنند. اغلب مدل‌سازی دامنه در حین تحلیل نیازمندی‌ها و تکرارهای بعدی انجام می‌شود. مدل دامنه با استفاده از یك نمودار کلاس UML به تصویر کشیده می‌شود.
۴. طراحی معماری (فصل ۶). این گام طراحی معماری را برای سیستم نرم‌افزاری تولید می‌کند، که برای هدایت طراحی بعدی، پیاده‌سازی، و آزمون زیرسیستم‌ها و اجزا مورد استفاده قرار می‌گیرد. دست‌ساخته‌هایی که در طی فعالیت‌های بعدی تولید می‌شوند بر حسب هر زیرسیستم سازمان‌دهی می‌شوند.

---

[۱] Package Diagrams

۵. استنتاج مورد کاربردها از روی نیازمندی‌ها (فصل ۷). در این گام، مورد کاربردها از روی نیازمندی‌های نرم‌افزار استنتاج شده و به تکرارها تخصیص داده می‌شوند. نمودارهای مورد کاربرد UML برای به تصویر کشیدن مورد کاربردها، زیرسیستم‌ها، و رابطه‌ی بین کنشگر-مورد کاربرد تولید می‌شوند. با این گام مرحله‌ی برنامه‌ریزی پایان می‌یابد.
۶. مدل‌سازی تعامل کنشگر-سیستم (فصل ۸). در طول این گام، مورد کاربردهای گسترده برای تکرار جاری تولید می‌شوند. این مورد کاربردها مشخص می‌کنند که چگونه کنشگران با سیستم تعامل می‌کنند تا مورد کاربردها انجام شوند. مدل دامنه اطلاعات لازم برای تبیین ورودی کنشگر و پاسخ‌های سیستم را فراهم می‌سازد.
۷. مدل‌سازی تعامل شئ (فصول ۹ و ۱۰). در این گام، گام‌های غیربدیهی از تعامل کنشگر-سیستم شناسایی شده و نمودارهای توالی تولید می‌شوند که نشان می‌دهند چگونه اشیای نرم‌افزاری با یکدیگر همکاری و تعامل می‌نمایند تا این گام‌ها انجام شوند. الگوهای طراحی نیز برای تولید یك طراحی بهتر مورد استفاده قرار می‌گیرند.

نمودارهای UML تولید شده توسط گام‌های فوق، جنبه‌های مختلف کاربرد و سیستم در دست ساخت را توصیف می‌کنند. ولی، مجموعه‌ی این نمودارها یك ساختار کلاس و یك نقشه‌ی راه که بتواند راهنمایی برای توسعه‌ی آزمون-رانده و آزمون یکپارچه‌سازی باشد را فراهم نمی‌سازند. برای مثال، چه در پیاده‌سازی معمولی و چه در TDD نیاز است کلاس‌هایی که باید پیاده‌سازی شوند و نیز رفتار و روابط این کلاس‌ها را بشناسیم. چنین اطلاعاتی در نمودارها وجود دارد، ولی متأسفانه در نمودارهای مختلف پراکنده است. برای مثال، یك پروژه‌ی سیستم اطلاعات کتابخانه را در نظر بگیرید. کلاس Loan هم در مدل دامنه و هم در چندین نمودار توالی، از جمله، در نمودار توالی «خروج مدرك»، به چشم می‌خورد. به همین ترتیب، مدیر پایگاه‌داده در تمام نمودارهای توالی که به پایگاه‌داده دسترسی دارند ظاهر شده است. بنابراین، اعضای تیم باید تمام نمودارهای UML را بازرسی کرده و کلاس‌ها، توابع و ویژگی‌های آن‌ها، و روابط بین کلاس‌ها را پیدا کنند. این‌کار فقط برای یك پروژه‌ی خیلی کوچك میسر است. برای اکثر پروژه‌های دنیای واقعی این‌کار وحشتناك است. اگر کلاس‌ها یا اعمال نادیده گرفته شوند، مشکل در حین آزمون یکپارچه‌سازی خودش را نشان خواهد داد. این امر، پیشرفت کل پروژه را بطور معناداری کند خواهد کرد.

این بحث نشان می دهد که تیم نیاز به یك نمودار دارد که به‌عنوان مشخصات طراحی کلاس‌ها و ساختار کلاسی عمل کند. به این نمودار، نمودار کلاس طراحی گویند. DCD یك نمودار کلاس UML است که کلاس‌ها، اعمال ویژگی‌های این کلاس‌ها و نیز رابطه‌ی بین کلاس‌ها را نشان می‌دهد. همه‌ی این‌ها از نمودارهایی که در تکرار فعلی تولید شده‌اند استنتاج شده و به DCD اضافه می‌گردند. برای کل سیستم تنها یك DCD وجود دارد، نه این که برای هر تکرار یك DCD داشته باشیم. تمرکز این فصل بر روی چگونگی به‌دست آوردن DCD از روی مجموعه نمودارها می‌باشد. به‌ویژه، شما موارد زیر را خواهید آموخت:

- نمودار کلاس طراحی چیست و چگونه می‌توان آن را از یك مدل دامنه به‌دست آورد؟
- استفاده‌ی نمودار کلاس طراحی چیست؟
- چگونه می‌توان نمودار کلاس طراحی را از نمودارهای تحلیل و طراحی استنتاج نمود؟
- چگونه می‌توان نمودارهای کلاس طراحی بزرگ را با استفاده از نمودارهای بسته‌ی UML سازمان‌دهی نمود؟

۱۱-۱ نمودار کلاس طراحی چیست؟

تعریف ۱۱-۱

نمودار کلاس طراحی یك نمودار کلاس UML است که از روی مدل‌های رفتاری و مدل دامنه به‌دست آمده است. نمودار کلاس طراحی یك نقشه‌ی طراحی است که فعالیت‌های بعدی پیاده‌سازی ، آزمون، و یکپارچه‌سازی را تسهیل می‌کند.

DCD از روی مدل‌های رفتاری و مدل دامنه به‌دست می‌آید. مدل‌های رفتاری شامل نمودارهای توالی (فصل ۹)، نمودارهای حالت (فصل ۱۳)، و نمودارهای فعالیت (فصل ۱۴) می‌باشند. این نمودارها به‌منظور کمك به توسعه، ارتباط، و اعتبارسنجی ایده‌های طراحی ساخته می‌شوند، که در غیر این‌صورت در بین اعضای تیم پخش هستند. فرایندهای بازبینی و بازرسی طراحی به ما اطمینان می دهند که نمودارها نیازمندی‌ها و قیود را ارضا می‌کنند. اگر تیم، DCD را از روی مدل‌های رفتاری استنتاج کند، در این صورت DCD باید شامل تمام کلاس‌ها، اعمال، و روابطی که برای ارضای نیازمندی‌ها مورد نیازند، باشد. تیم فقط تمام کلاس‌هایی را که در DCD هستند پیاده‌سازی نموده و تحویل می‌دهد. بدین معنی که، تمام کلاس‌هایی که برای ارضای نیازمندی‌ها لازم‌اند، پیاده‌سازی می‌شوند و نه بیشتر.

به دلیل کپسوله‌سازی و نهان نمودن اطلاعات، کلاس‌های برنامه‌های شئ گرا متدهای مرسوم get و set را پیاده‌سازی می کنند که به نام getter و setter شناخته می‌شوند. این متدها فقط مقدار ویژگی‌های مربوطه را بر می‌گردانند یا تنظیم می‌کنند. از آنجایی که متدهای زیادی از این نوع خواهیم داشت، اگر تمام آن‌ها روی DCD نشان داده شوند، DCD خیلی شلوغ شده و سایر رفتارهای مهم کلاس‌ها مغفول می‌مانند. بنابراین، برای سادگی، روی DCD نباید متدهای مرسوم get و set را نشان داد. البته لازم به ذکر است که این قرارداد را نباید روی تمام کلاس‌های DCD اعمال نمود. برای مثال، کلاس مدیر پایگاه‌داده مسئول ذخیره و بازیابی اشیا به/از پایگاه‌داده می باشد. اگر اعمال بازیابی اشیا را با نام‌هایی مثل getX(...):X و getY(...):Y نام‌گذاری کنیم که X و Y نام کلاس‌ها باشند، در این‌صورت این اعمال نباید تحت تاثیر مشخصات واسط کلاس مدیر پایگاه قرار گیرند. این بدین خاطر است که این‌ها getter و setter های معمولی نیستند.

۱۱-۲ فواید نمودار کلاس طراحی

فواید DCD شامل موارد زیر می باشد:

۱. مهم‌ترین دست‌ساخته‌های طراحی را در یك نمودار گرد هم می‌آورد. همان‌گونه که در ابتدای این فصل بحث شد، کلاس‌ها و اطلاعاتی در خصوص کلاس‌هایی که باید پیاده‌سازی گردند در مجموعه‌ای از نمودارهای تحلیل و طراحی پخش شده‌اند. DCD این دست‌ساخته‌های طراحی را درون یك نقشه‌ی طراحی مجتمع می‌سازد. این یکپارچه‌سازی باعث تسهیل پیاده‌سازی، آزمون یکپارچه‌سازی، و نگهداری می‌گردد.
۲. اندازه‌گیری و اطمینان از کیفیت را تسهیل می‌کند. DCD مبنایی برای تعریف و محاسبه‌ی متریك‌های کیفیت طراحی، که برای ارزیابی یك طراحی و شناسایی مشکلات احتمالی در طراحی مفید هستند فراهم می‌سازد. برای مثال، اگر یك کلاس تعداد زیادی عمل داشته باشد، در این صورت احتمال دارد که مسئولیت‌های زیادی نیز به کلاس واگذار گردد. کلاسی که یال‌های زیادی از آن خارج شوند بدین معنی است که به کلاس‌های متعدد دیگری وابسته است؛ و بنابراین، بازاستفاده از آن کلاس دشوار خواهد بود.

۳. برای تخمین تلاش، مفید است. بر اساس اعمال هر کلاس، می‌توان یك تخمین غیر دقیق از تلاش لازم برای پیاده‌سازی وآزمون هر کدام از کلاس‌ها داشت. این تخمین‌ها برای واگذاری کلاس‌ها به اعضای تیم جهت پیاده‌سازی وآزمون، که بعداً تشریح می‌شود، مفید می‌باشند.
۴. واگذاری کلاس‌ها به اعضای تیم جهت پیاده‌سازی و آزمون را تسهیل می‌نماید. از روی وابستگی بین کلاس‌ها می‌توان یك سفارش پیاده‌سازی و آزمون کلاس تولید نمود. این کار برای زمان‌بندی پیاده‌سازی کلاس‌ها و تخصیص کار به اعضای تیم مفید است. در موقع تخصیص باید توانایی‌های پیاده‌سازی و آزمون اعضای تیم را مدّنظر قرار داد، تا اطمینان حاصل شود که پیاده‌سازی و آزمون کلاس‌ها طبق زمان‌بندی به انجام خواهد رسید. بدین ترتیب، می‌توان از زمان و تلاش لازم برای ساخت بدنه‌ی آزمون‌ها صرف‌نظر نمود یا آن‌را بسیار کاهش داد.

۱۱-۳ گام‌های استنتاج یك نمودار کلاس طراحی

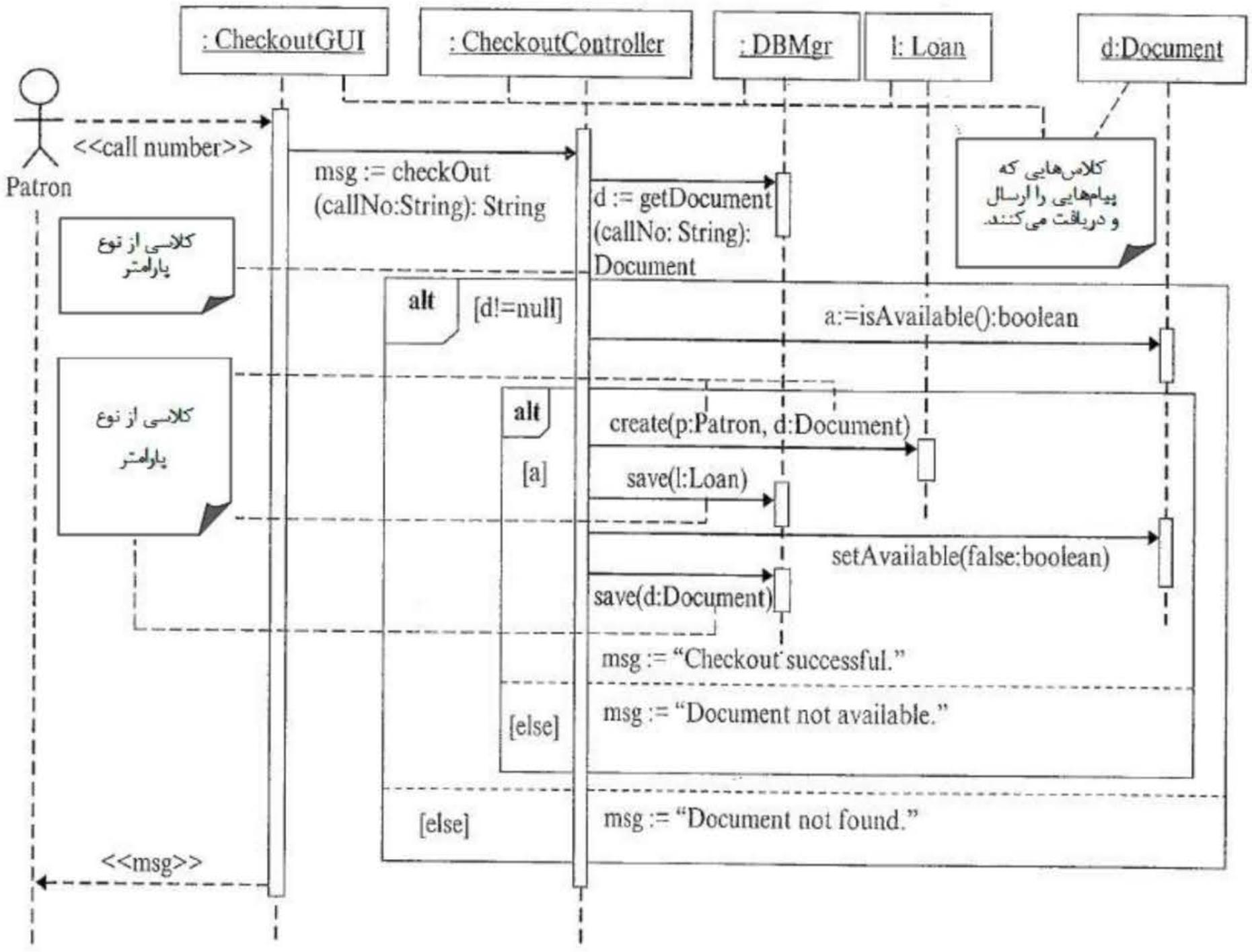
گام‌های استنتاج DCD از روی نمودار توالی طراحی عبارتند از: (۱) شناسایی کلاس‌ها، (۲) شناسایی متدها، (۳) شناسایی ویژگی‌ها، (۴) شناسایی رابطه‌ها، و (۵) مرور DCD. این گام‌ها در چند بخش بعدی تشریح خواهند شد. نمودار توالی نشان داده شده در شکل ۱۰-۶ به‌عنوان مثال جاری استفاده خواهد شد.

۱۱-۳-۱- شناسایی کلاس‌ها

کلاس‌هایی که باید درون DCD قرار گیرند، از روی نمودارهای توالی طراحی که در تکرار فعلی تولید شده‌اند شناسایی می‌شوند. این نمودارها نشان می‌دهند که چگونه اشیای نرم‌افزاری با یکدیگر تعامل و همکاری دارند تا پردازش‌های پیش زمینه‌ای مورد کاربردها را اجرا نمایند. مورد کاربردها از روی نیازمندی‌ها استنتاج می‌شوند. بنابراین، کلاس‌های اشیایی که در نمودارهای توالی طراحی نقشی دارند، باید در DCD درج شوند به‌گونه‌ای که با پیاده‌سازی و استقرار آن‌ها توانایی‌های مشخص شده در نیازمندی‌ها تحویل مشتری گردد. کلاس‌های اشیا را از چهار محل زیر در نمودار توالی طراحی می‌توان شناسایی نمود.

۱. **شناسایی کلاس‌هایی از اشیا که پیام ارسال و دریافت می‌کنند.**
تعامل شئ توسط یك نمودار توالی به صورت تبادل پیام بین اشیا مدل‌سازی می‌شود. اشیایی که پیام‌هایی را ارسال یا دریافت می‌کنند توسط مستطیل‌هایی که نام اشیا در آن‌ها زیرخط‌دار شده و گونه‌ی آن‌ها هم با دونقطه (کولون) جدا شده است نشان داده می‌شوند. این مستطیل‌ها در بالای هر نمودار توالی طراحی قرار می‌گیرند. بنابراین، کلاس‌های اشیایی که پیام‌ها را ارسال یا دریافت می‌کنند به وسیله‌ی این مستطیل‌ها در نمودارهای توالی طراحی شناسایی می‌شوند. شکل ۱۱-۱ نشان می‌دهد که پنج تا از چنین کلاس‌هایی شناسایی شده‌اند: Checkout Gui، Checkout Controller، DBMgr، Loan، Document.

۲. **شناسایی کلاس‌هایی از اشیا که به‌عنوان پارامتر ارسال می‌شوند.**
ارسال یك شئ به‌عنوان پارامتر به یك فراخوان تابع امری متداول در برنامه‌سازی شئ‌گراست. این امر در مورد نمودارهای توالی طراحی نیز درست است، زیرا این نمودارها در واقع «برنامه‌های در مقیاس بزرگ» هستند. به‌طور مشخص، در یك نمودار توالی طراحی، پیام‌هایی که بین اشیا رد و بدل می‌شوند مشابه فراخوانی تابع از یك شئ به شئ دیگر هستند. از آن‌جا که کلاس‌هایی که به‌عنوان پارامتر ارسال می‌شوند باید پیاده‌سازی و تحویل شوند، این کلاس‌ها باید در DCD ظاهر شوند. این کلاس‌ها را می‌توان از پارامترهایی که در پیام‌های ارسال شده یا دریافت شده توسط اشیا در نمودارهای توالی طراحی، ارسال می‌شوند شناسایی نمود. در شکل ۱۱-۱ دو تا از این کلاس‌ها شناسایی می‌شوند؛ یعنی Document و Loan، گرچه این کلاس‌ها همچنین به‌عنوان کلاس‌هایی که پیام‌ها را دریافت می‌کنند، شناسایی می‌شوند.



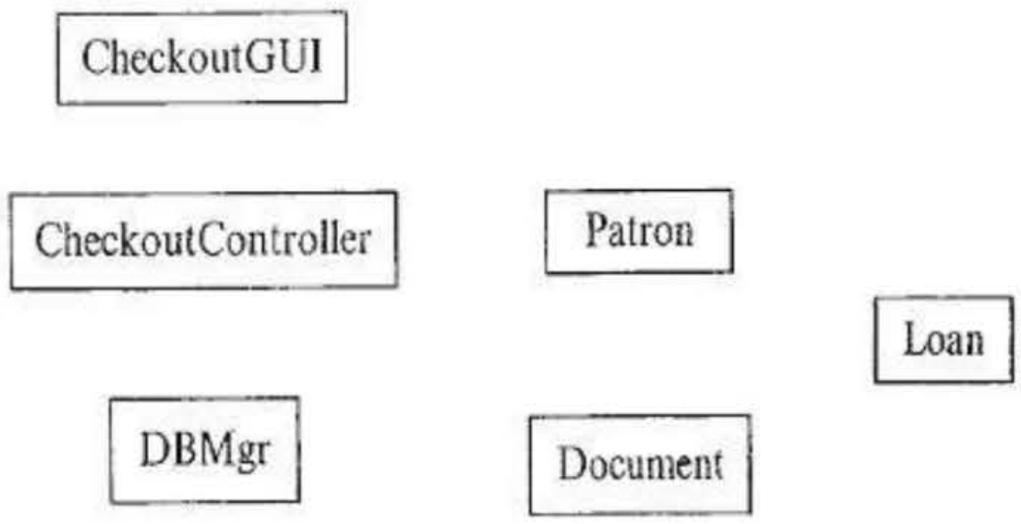
شکل ۱۱-۱: شناسایی کلاس‌ها از روی نمودارهای توالی

۳. **شناسایی کلاس‌هایی که به‌عنوان گونه‌ی برگشتی استفاده می‌شوند.**
بازگرداندن یك شئ از یك فراخوانی تابع یك روش معمول در برنامه‌سازی شئ‌گرا می‌باشد. این موضوع برای نمودارهای توالی طراحی نیز صادق است. کلاس‌هایی که به‌عنوان گونه‌های برگشتی هستند را می‌توان از گونه‌های برگشتی پیام‌های بین اشیا در نمودارهای توالی طراحی شناسایی نمود. در شکل ۱۱-۱ ، Document تنها کلاسی است که به‌عنوان یك گونه‌ی برگشتی استفاده شده است. همان‌گونه که در شکل ۱۱-۱ نشان داده شده است، اعمال سه قانون فوق شش کلاس را شناسایی می‌کند: Checkout GUI، Checkout Controller، DBMgr، Loan، Document، Patron.

این کلاس‌ها در DCD وارد می‌شوند، چنان‌چه در شکل ۱۱-۲ نشان داده شده است.

۴. **شناسایی کلاس‌ها از روی مدل دامنه.**
کلاس‌های مدل دامنه که والدِ کلاس‌های شناسایی شده باشند را می‌توان به DCD افزود، اگر این‌کار فهم، پیاده‌سازی، و آزمون را تسهیل می‌کند.

در حین شناسایی، باید موارد زیر را در نظر داشت: (۱) قوانین فوق را باید در مورد تمام نمودارهای توالی طراحی که در تکرار فعلی تولید شده‌اند اعمال نمود، نه فقط در مورد یکی از نمودارهای توالی تولید شده؛ (۲) یك کلاس ممکن است توسط



شکل ۱۱-۲: DCD که کلاس‌هایی را شناسایی کرده است.

بیش از یك قانون شناسایی شود، بنابراین، وقتی یك کلاس شناسایی شد، دیگر شناسایی نخواهد شد؛ (۳) از آن‌جا که DCD به‌عنوان یك نقشه‌ی طراحی برای سیستم است، باید تنها یك DCD برای تمام مدل‌های رفتاری و تمام تکرارها ایجاد نمود، نه این‌که یك DCD برای هر نمودار توالی یا هر تکرار.

۱۱-۳-۲- شناسایی متدها

در یك نمودار توالی طراحی، پیغامی به صورت x:=m(...):X که از شئ A به شئ دیگر B ارسال می‌شود نشان می‌دهد که A تابع m(...):X از B را فرا می‌خواند و نتیجه را در متغیر x از گونه‌ی X ذخیره می‌کند. بنابراین، متدهای یك کلاس از تمامی چنین پیغام‌ها که در همه‌ی نمودارهای توالی طراحی تولید شده وجود دارند، به دست می‌آیند. به‌طور مشخص‌تر، متدهای کلاس B از پیکان‌های افقی که به زیست‌خط یك شئ از B در هر کدام از نمودارهای توالی طراحی رفته‌اند، شناسایی می‌شوند. از برچسب چنین پیکان‌هایی، متدهای B شناسایی می‌شوند. برای مثال، در شکل ۱۱-۳ ، سه تا از چنین پیکان‌هایی به شئ DBMgr وارد می‌شوند. برچسب این پیکان‌ها به ترتیب چنین است:

“d:=getDocument(callNo:String):Dcoument”

“save(l:Loan)”

“save(d:Docament)”

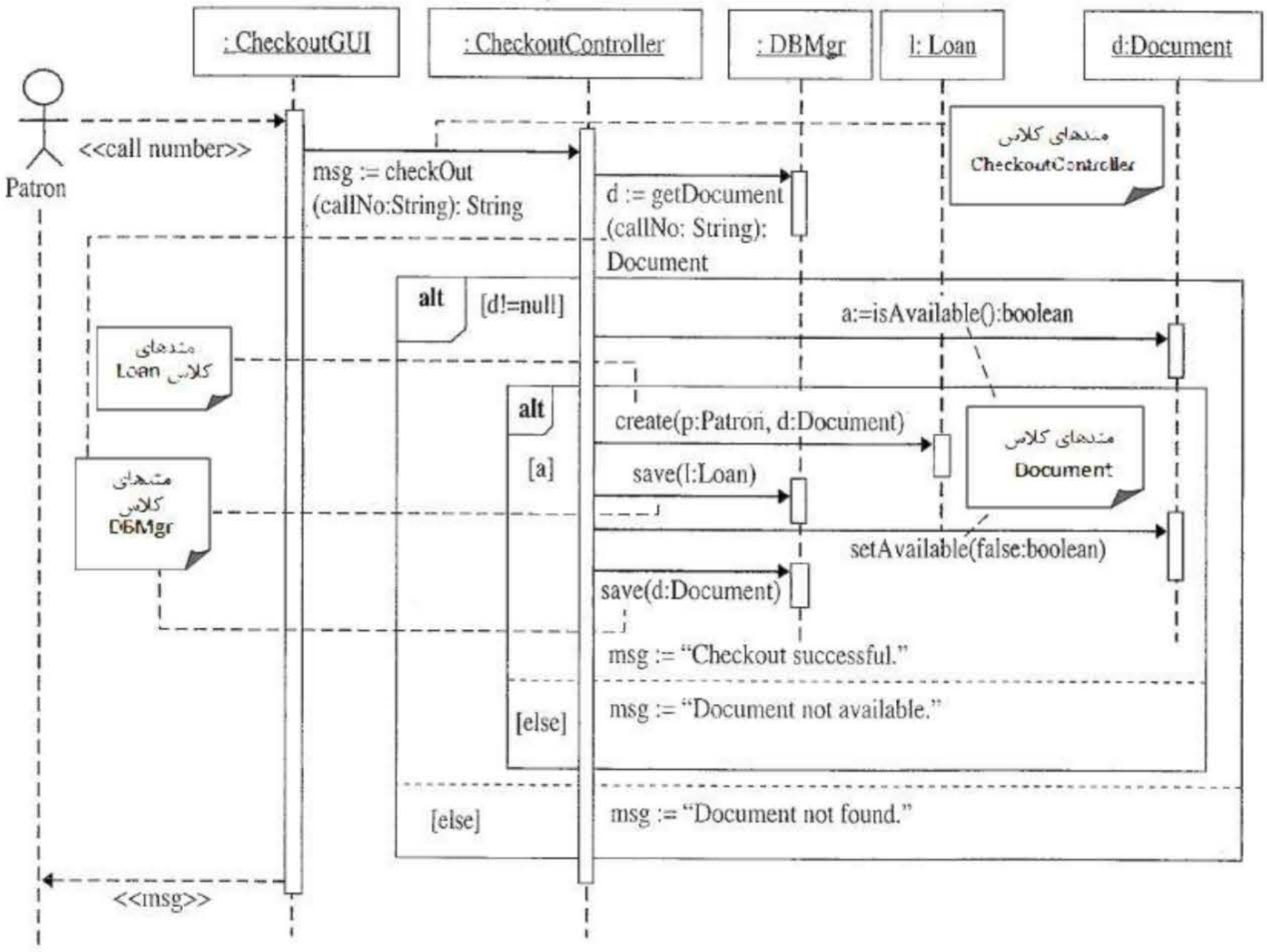
بنابراین، برای کلاس DBMgr سه متد شناسایی می‌شوند ، که عبارتند از

getDocument(callNo:String):Dcoument

save(l:Loan)

save(d:Docament)

شکل ۱۱-۳ نشان می‌دهد که چگونه این قاعده‌ها برای سایر کلاس‌ها اعمال شده‌اند، و نتیجه در شکل ۱۱-۴ خلاصه شده است. متدهای شناسایی شده درون DCD درج شده‌اند همان‌گونه که در شکل ۱۱-۵ نشان داده شده است، که علامت مثبت (“+”) قبل از یك متد نمادگان UMLای است که می‌گوید قابلیت دید متد عمومی است.



شکل ۱۱-۳: شناسایی متدهای یك کلاس

کلاس	متدها
Checkout GUI	
Checkout Controller	Checkout(callNo: String): String
DBMgr	getDocument(callNo: String): Document save(l: Loan) save(d: Document)
Loan	Create (p: Patron, d: Document)
Document	isAvailable(): Boolean setAvailable(b: Boolean)

شکل ۱۱-۴: کلاس‌ها و متدهایی که برای آن‌ها شناسایی شده است

۱۱-۳-۳- شناسایی صفت‌ها

در نظر داشتن این نکته مهم است که صفت‌ها شئ نیستند. صفت‌ها وجود مستقلی در کاربرد یا دامنه‌ی کاربرد ندارند. صفت‌ها گونه‌های اسکالر[1] هستند، نه گونه‌های کلاس شئ . صفت‌ها از روی نمودارهای توالی طراحی با استفاده از قوانین زیر شناسایی

[1] Scalar

می‌شوند:

۱. شناسایی صفت‌ها از روی متدهایی که اشیا را بازیابی می‌نمایند. هر پیغام به شکل getX(a:T):X که در آن X یک کلاس و T یک گونه‌ی اسکالر است، پیشنهاد می‌کند که a یک صفت از X بوده و گونه‌ی آن نیز T باشد. ولی، این‌که آیا a یک صفت از X هست یا خیر را باید بر اساس مدل دامنه یا دانش دامنه بررسی نمود. برای مثال، در شکل ۱۱-۶ پیغام D:getDocument(callNo:String):Document از CheckoutController به شئ DBMgr پیشنهاد می‌کند که CallNo یک صفت از Document باشد. در مثالی دیگر، در نمودار توالی شکل ۱۱-۱ ، getUser(id:String):User و getDocument(callNum:String):Document می‌گوید که id یک صفت از User و callNum یک صفت از Document باشد. گونه‌ی هر دو صفت نیز String خواهد بود. این دو تصمیم با در نظر گرفتن این واقعیت که در دامنه‌ی کاربرد کتابخانه، id و callNum در واقع به ترتیب صفت‌های User و Document هستند تایید می‌شود.

۲. شناسایی صفت‌ها از متدهای معمول *get* و *set* در برنامه‌سازی شئ گرا، از متدهای معمول *get* و *set* برای بازاریابی و بروزرسانی صفت‌های یک شئ استفاده می‌شود. این واقعیت می‌گوید که صفت‌های یک کلاس را می‌توان از پیغام‌هایی چون t:=getX():T و setX(x:T) که به کلاسی مثل A ارسال شده‌اند، شناسایی نمود؛ که در آن X و x اسم بوده یا قرار است اسم باشند و T یک گونه‌ی اسکالر است. این پیغام‌ها می‌گوید که اسم X یا x یک صفت از A بوده و گونه‌ی این صفت نیز T است. برای مثال، پیغام t:=getTitle():String که به شئ Document ارسال شده پیشنهاد می‌دهد که title یک صفت از Document و با گونه‌ی String باشد.

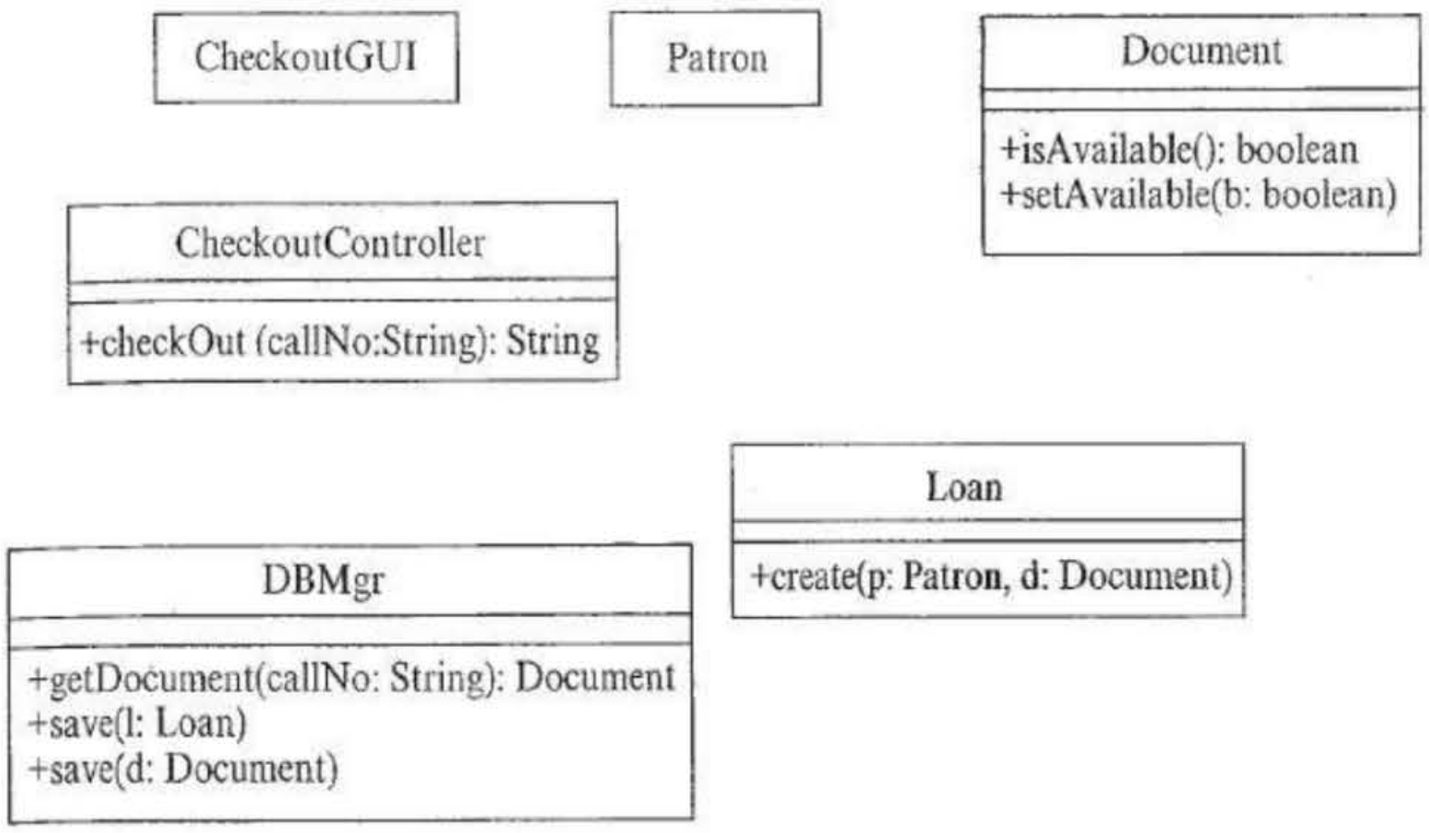
۳. شناسایی صفت‌ها از متدهای *isX() Boolean* و *setX(b:Boolean)* که در آن‌ها *X* یک صفت است. از فصل مدل‌سازی دامنه به خاطر آورید که یک صفت می‌تواند یک صفت یا مقدار یک صفت باشد. متدهایی که به‌صورت isX( ):Boolean یا setX(b:Boolean) نام‌گذاری شده‌اند که در آن X یک صفت است، اغلب پیشنهاد می‌کنند که X یک صفت بولی (Boolean) از شیئی باشد که پیام را دریافت می‌نماید. برای مثال، در شکل ۱۱-۶ پیغام‌های a:=isAvailable( ):Boolean و setIsAvailable(false) به شئ Document پیشنهاد می‌کند که Avaliable یک صفت بولی از Document باشد.

۴. شناسایی صفت‌ها از متدهایی که یک مقدار از گونه‌ی اسکالر را محاسبه می‌کنند. هر پیغام به شکل computeX(…) یا x:=computeX(…):T که در آن T یک گونه‌ی اسکالر و computeX(…) محاسبه‌ای است که مقداری از گونه‌ی اسکالر تولید می‌نماید، ممکن است پیشنهاد دهد که X یک صفت از شئ دریافت‌کننده‌ی پیغام باشد. گونه‌ی این صفت نیز T خواهد بود. در شکل ۱۱-۷(الف) شئ Shipment درخواست computeCharge(…):double را برای محاسبه‌ی هزینه‌ی حمل دریافت می‌کند. اگر شئ Shipment مقدار هزینه‌ی محاسبه شده را ذخیره نماید، در این‌صورت charge یک صفت از Shipment و گونه‌ی آن double خواهد بود.

۵. شناسایی صفت‌ها از پارمترهای متد سازنده. بیشتر اوقات، پارامترهای با گونه‌ی اسکالر که به یک سازنده ارسال می‌شوند برای مقداردهی اولیه‌ی اشیایی که بهتازگی از یک کلاس ایجاد شده‌اند استفاده می‌شوند.

برای مثال ، پیغام create(callNo:String, title:Sting, author:String, year:String,…) که به شئ Document ارسال می‌شود را در نظر بگیرید که در آن نام create توسط UML انتخاب شده تا سازنده‌ی کلاس را مشخص نماید. در اکثر موارد، پارامترهای این پیغام توسط کلاس Document استفاده می‌شود تا صفت‌های شیئی که ایجاد شده است را مقدار اولیه دهند. بنابراین، از این پارامترها می‌شود فهمید که callNo، title، author، year و سایر موارد، صفت‌های Document هستند. گونه‌ی این صفت‌ها String است. مثال دیگری از این قانون در شکل ۱۱-۷(ب) نشان داده شده است. در این‌جا نیز صفت‌ها و گونه‌های شناسایی شده را باید با مدل دامنه و دانش دامنه مورد تایید قرار داد.

۶. شناسایی صفت‌ها از روی مدل دامنه. برای هر کلاس در DCD، کلاس مربوط را در مدل دامنه بررسی نموده و صفت‌هایی که در اعمال آن کلاس DCD، مورد نیاز هستند را شناسایی نمایید.



شکل ۱۱-۵: DCD که متدهای آن پر شده‌اند

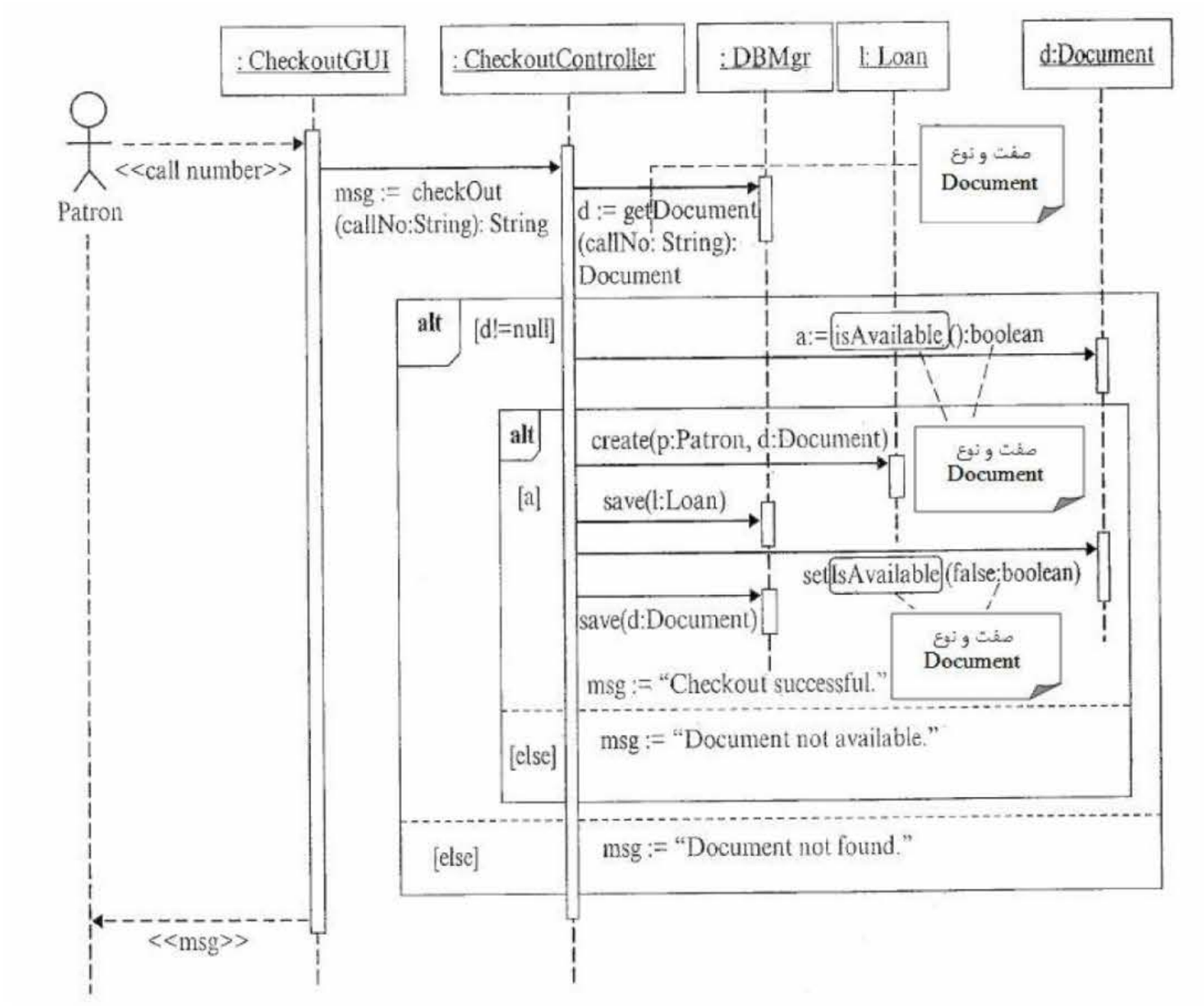
صفت‌های شناسایی شده درون DCD نشان داده شده در شکل ۱۱-۸ نشان داده شده‌اند، که در این شکل علامت منفی ("-") نمادگان UML برای نشان دادن قلمرو خصوصی است. توجه کنید که در شکل، صفت dueDate از کلاس Loan از مدل دامنه شناسایی شده است. این صفت ممکن است توسط سازنده‌ی شئ Loan مورد نیاز باشد تا تاریخ برگشت امانت را تعیین نماید به‌گونه‌ای که بتوان به‌طور خودکار یک پیغام یادآوری به عضو مربوطه ارسال نمود.

۱۱-۳-۴- روابط بین کلاس‌ها

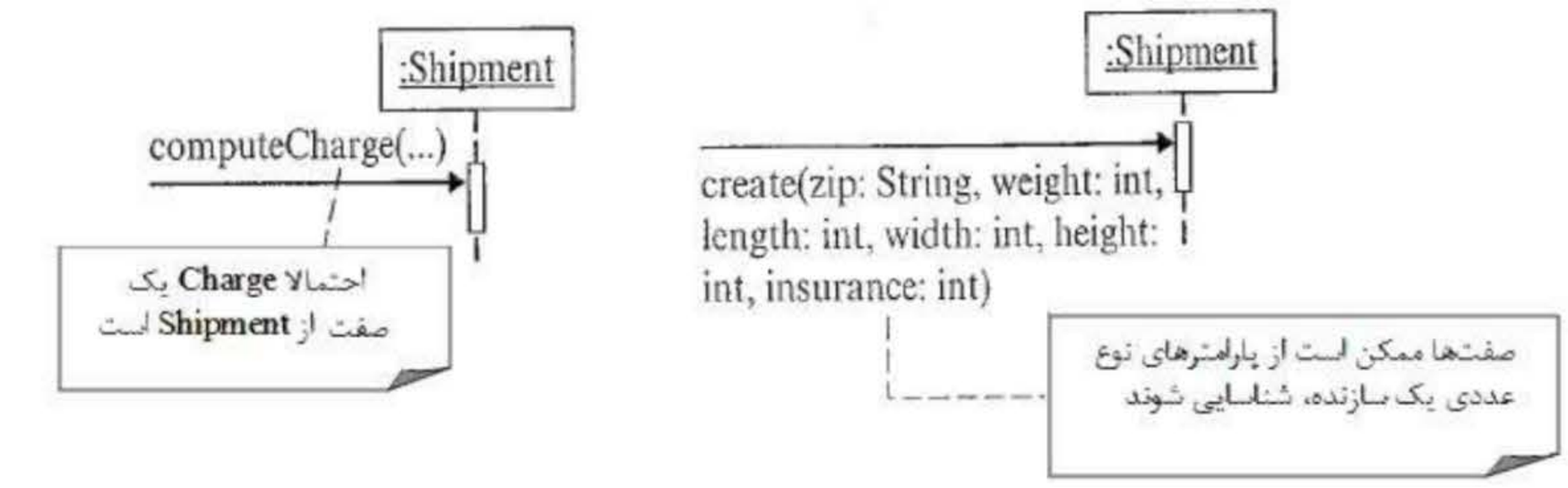
قبل از این‌که به سراغ شناسایی روابط بین کلاس‌ها برویم، باید در خصوص خود روابط و تاثیراتشان بحث کرد. رابطه‌های ارث‌بری، تجمیع[۱]، و انجمن در فصل ۵ با موضوع مدل‌سازی دامنه معرفی گردیدند. این روابط، روابط مفهوم‌سازی می‌باشند که در دامنه‌ی کاربرد، وجود دارند یا در طراحی معرفی می‌شوند. برای مثال، بسیاری از الگوهای طراحی از این رابطه‌ها برای دستیابی به اهداف طراحی خود استفاده می‌کنند. علاوه بر این‌ها، روابط استریوتایپ‌دار تعریف شده در UML نیز هستند، مانند نمونه‌سازی[۲]، فراخوانی[۳]، و استفاده[۴]. این روابط، روابط مرتبط با طراحی هستند، بدین معنی که نتیجه‌ی تصمیمات طراحی می‌باشند. رابطه‌ی نمونه سازی مشخص می‌کند که یک شئ از یک کلاس، شیئی از کلاس دیگر را ایجاد می‌کند. به این رابطه، رابطه‌ی ایجاد[۵] نیز می‌گویند. رابطه‌ی فراخوانی بیان‌گر این واقعیت است که یک شئ از یک کلاس متدی از یک شئ از کلاس دیگری را فرامی‌خواند. رابطه‌ی استفاده می‌گوید که شیئی از یک کلاس به شیئی از کلاس دیگر وابسته است؛ به عبارت دیگر، شئ اول شئ دوم را به‌عنوان مقدار برگشتی از یک فراخوانی تابع دریافت می‌کند، یا شئ اول پیغامی دریافت می‌کند که شئ دوم در قالب پارامتر به آن ارسال شده است. لازم به ذکر است که اگر شیئی از کلاس A شیئی از کلاس B را ایجاد کند، در

---

[۱] Aggregation
[۲] Instantiation
[۳] Call
[۴] Use
[۵] Create



شکل ۱۱-۶: شناسایی صفت‌ها با استفاده از قوانین ۱ تا ۳

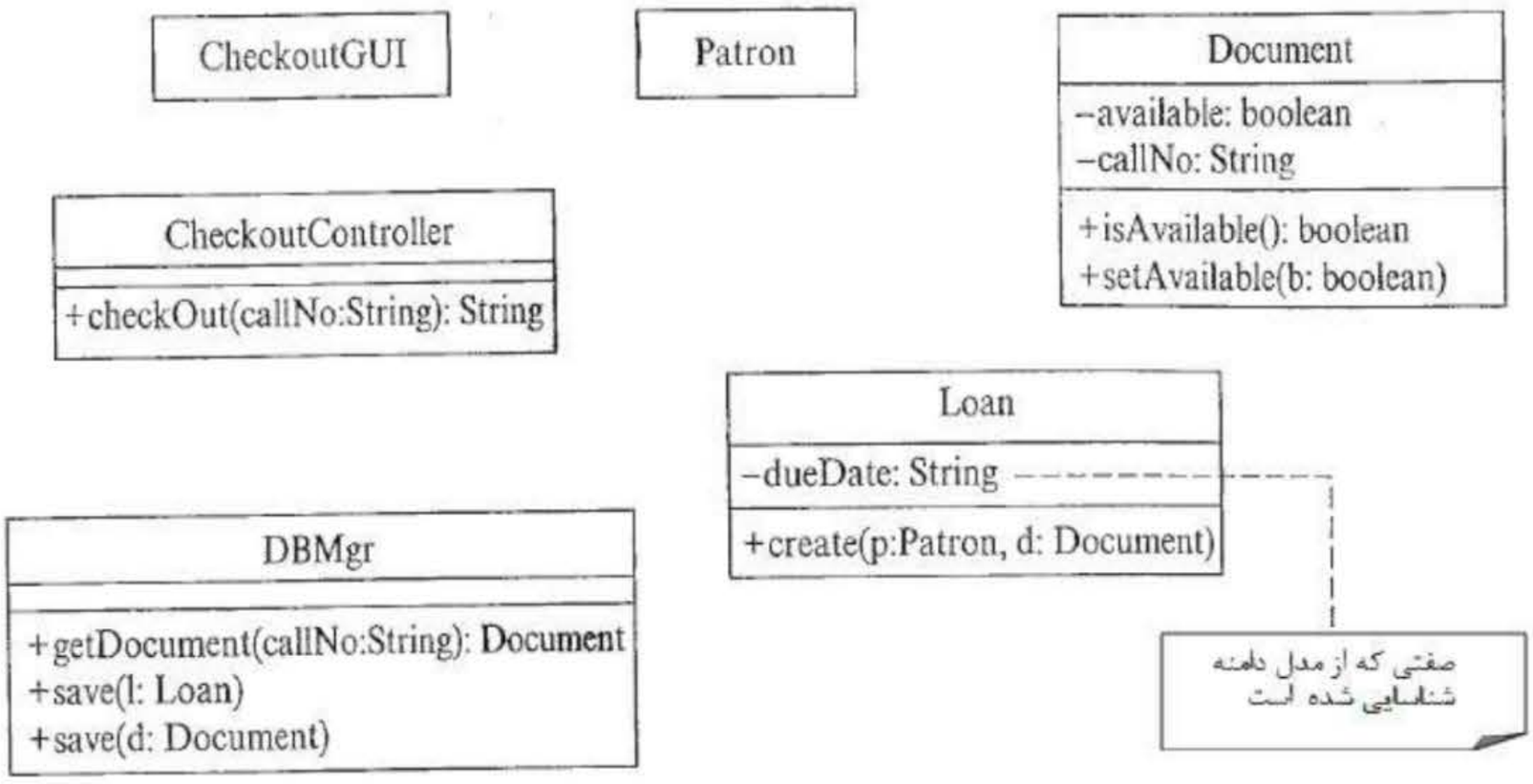


الف) شناسایی صفت x از متدهای computeX(...)

ب) شناسایی صفت‌ها از پارامترهای نوع عددی یک سازنده

شکل ۱۱-۷: شناسایی صفت‌ها با استفاده از قوانین ۴ تا ۵

این‌صورت اول باید سازنده دومی را فراخواند؛ لذا رابطه‌ی ایجاد منجر به رابطه‌ی فراخوانی میگردد. اگر شییی از کلاس A تابعی از یک شئ از کلاس B را فراخواند، در اینصورت اولی به دومی وابسته است. این بدان معنی است که رابطه‌ی فراخوانی منجر به رابطه‌ی استفاده می‌گردد. بدین معنی که، هرگاه رابطه‌ی ایجاد بین دو کلاس وجود داشته باشد، نیازی به شناسایی روابط فراخوانی و استفاده بین دو کلاس نیست. به طرز مشابه، اگر رابطه‌ی فراخوانی بین دو کلاس وجود داشته باشد، نیاز به افزودن



شکل ۱۱-۸: یک DCD که خصیصه‌ها در آن شناسایی شده‌اند

رابطه‌ی استفاده نیست.

۱۱-۳-۵- شناسایی روابط

روابط بین کلاس‌ها حاکی از وابستگی بین کلاس‌هاست. روابط بین کلاس‌ها را با اعمال قوانین زیر از روی نمودارهای توالی طراحی تولید شده در تکرار فعلی و نیز مدل دامنه به‌دست می‌آوریم.

۱. شناسایی روابط ایجاد از روی فراخوانی سازنده‌ها. اگر شییی از کلاس A سازنده‌ای از کلاس B را فراخواند، در این‌صورت A ایجادکننده‌ی B است. در شکل ۱۱-۹ یک رابطه از این دست شناسایی می‌شود: checkoutController ایجادکننده‌ی Loan است.
۲. شناسایی روابط استفاده از کلاس‌هایی که به‌عنوان پارامتر یا گونه‌ی برگشتی ایفای نقش کرده‌اند. منظور این است که کلاس A از کلاس B استفاده می‌کند اگر یکی از شرایط زیر برقرار باشد:

الف) شییی از کلاس A، در یک فراخوانی تابع، شییی از کلاسB را به‌عنوان پارامتر ارسال می‌کند. بدین معنی که A از B در یک فراخوانی تابع استفاده می‌کند. در شکل ۱۱-۹ شئ checkoutController متدهای save(l:Loan) و save(d:Document) از شئ DBMgr را فرامی‌خواند. بنابراین، checkoutController به ترتیب Loan و Document را استفاده می‌کند.

ب) شییی از کلاس A شییی از کلاس B را به‌عنوان مقدار برگشتی دریافت می‌کند. بدین معنی که A شئ B را دریافت می‌کند چون قرار است از آن استفاده کند. در شکل ۱۱-۹ ، یک رابطه از این دست وجود دارد، یعنی با توجه به وجود فراخوانی d:=getDocument(callNo:String):Document از شئ CheckoutController به شئ DBMgr، CheckoutController از Document استفاده می‌کند.

ج) کلاس B گونه‌ی یکی پارامتر در یکی از توابع کلاس A است. بدین معنی که، شئ B به یک تابع از کلاس A ارسال شده است زیرا تابع نیاز دارد که از شئ B استفاده نماید. در شکل ۱۱-۹ دو رابطه از این دست وجود دارد، بدین‌صورت که DBMgr از Loan و Document استفاده می‌کند زیرا save(l:Loan) و save(d:Document) توابعی از DBMgr هستند.

د) کلاس B گونه‌ی برگشتی یکی از توابع کلاس A است. در این حالت، A از B به‌عنوان گونه‌ی برگشتی استفاده می‌کند. حداقل، A باید یک نمونه از B را ایجاد کرده یا به‌دست آورد و آن‌را برگرداند. در شکل ۱۱-۹ یک رابطه از این دست وجود دارد. یعنی، DBMgr از Document استفاده می‌کند چون Document گونه‌ی برگشتی از تابع getDocument(…):Document از DBMgr است.

۳. شناسایی روابط انجمنی از روی فراخوانی سازنده‌هایی که دو شئ را به‌عنوان پارامتر ارسال می‌کنند. اگر یک شئ از کلاس B و یک شئ از کلاس C به‌عنوان پارامتر در فراخوانی سازنده‌ای از کلاس A استفاده شوند، در این‌صورت احتمالاً یک رابطه‌ی انجمنی بین کلاس B و کلاس C وجود دارد که در آن رابطه، A به‌عنوان کلاس انجمنی نقش بازی می‌کند. این تصمیم‌ها باید با توجه به مدل دامنه یا دانش دامنه تایید شوند. در شکل ۱۱-۹ ، یک شئ Pattron و یک شئ Document در فراخوانی سازنده‌ی Loan استفاده شده‌اند. بنابراین، یک رابطه‌ی انجمنی بین Patron و Document وجود دارد که در آن Loan به‌عنوان کلاس انجمنی قرار دارد. شئ Loan ، اطلاعات مرتبط با خروج مدرک، مانند این‌که کدام عضو کدام مدرک را برده است و تاریخ برگشت کدام است، را ذخیره می‌کند. در واقع، با توجه به دانش دامنه‌ی کتابخانه می‌بینیم که این تصمیم‌ها درستند.

۴. شناسایی روابط از پیغام‌های *get* و *set* کردن اشیا. اگر پیغامB:(...)getB یا setB(b:B) پیغامی به یک شئ از کلاس A باشد و A وB کلاس‌های تعریف شده توسط کاربر باشند، در این‌صورت:

الف) A تجمعی از B است اگر یک شئ از B جزیی از یک شئ از A باشد. برای مثال، پیغامی چون e:=getEngine():Engine به یک شئ از کلاس Car چنین رابطه‌ی انجمنی را مشخص می‌کند. این دست رابطه‌ها نیز باید با استفاده از مدل دامنه یا دانش دامنه بررسی شوند. بعضی مواقع، این‌ها را باید با تصمیم‌های طراحی تایید کرد. برای مثال، الگوهای طراحی اغلب از روابط تجمیع ارث‌بری استفاده می‌کنند تا تنوع رفتاری را فراهم آورند. این روابط ممکن است در دامنه‌ی کاربرد وجود نداشته باشند؛ و بنابراین، نمی‌توان این‌ها را با داشتن دامنه تایید کرد.

ب) A از B استفاده می‌کند، همان‌گونه که در بند ۲ فوق توضیح داده شد.

ج) A با B در انجمن است اگر A و B تجمیع یکدیگر نباشند یا از همدیگر استفاده نکنند. روابطی که بدین شکل شناسایی می‌شوند باید با توجه مدل دامنه، دانش دامنه، یا تصمیم‌های طراحی بررسی شوند. مثالی از یک رابطه‌ی انجمنی از این دست ثبت‌نام نمودن Student در یک Course است. این رابطه‌ی انجمنی را می‌توان از پیغام c:=getCourse():Course که به شئ Enrollment ارسال می‌شود شناسایی نمود.

۵. شناسایی روابط فراخوانی. اگرشیئ از کلاس A متدی از کلاس B را فراخوانی نماید، در این‌صورت A ، B را فراخوانی می‌کند. در شکل ۱۱-۹ چند رابطه از این دست وجود دارند، برای مثال، CheckoutGUI ، Checkoutcontroller را فرا می‌خواند.

۶. شناسایی روابط دربرگیری از روی پیغام‌های *add*، *get*، یا *remove* شئ. یک رابطه‌ی دربرگیری بین کلاس A و کلاس B ، بدین معنی که A دربرگیرنده‌ی B است، وقتی شناسایی می‌شود که شیئی از کلاس A پیغامی به شکل add(b:B) ، get(…):B ، getB(…):B ، یا remove(b:B) دریافت کند که در آن A وB کلاس‌های تعریف شده توسط کاربر هستند. برای مثال، اگر

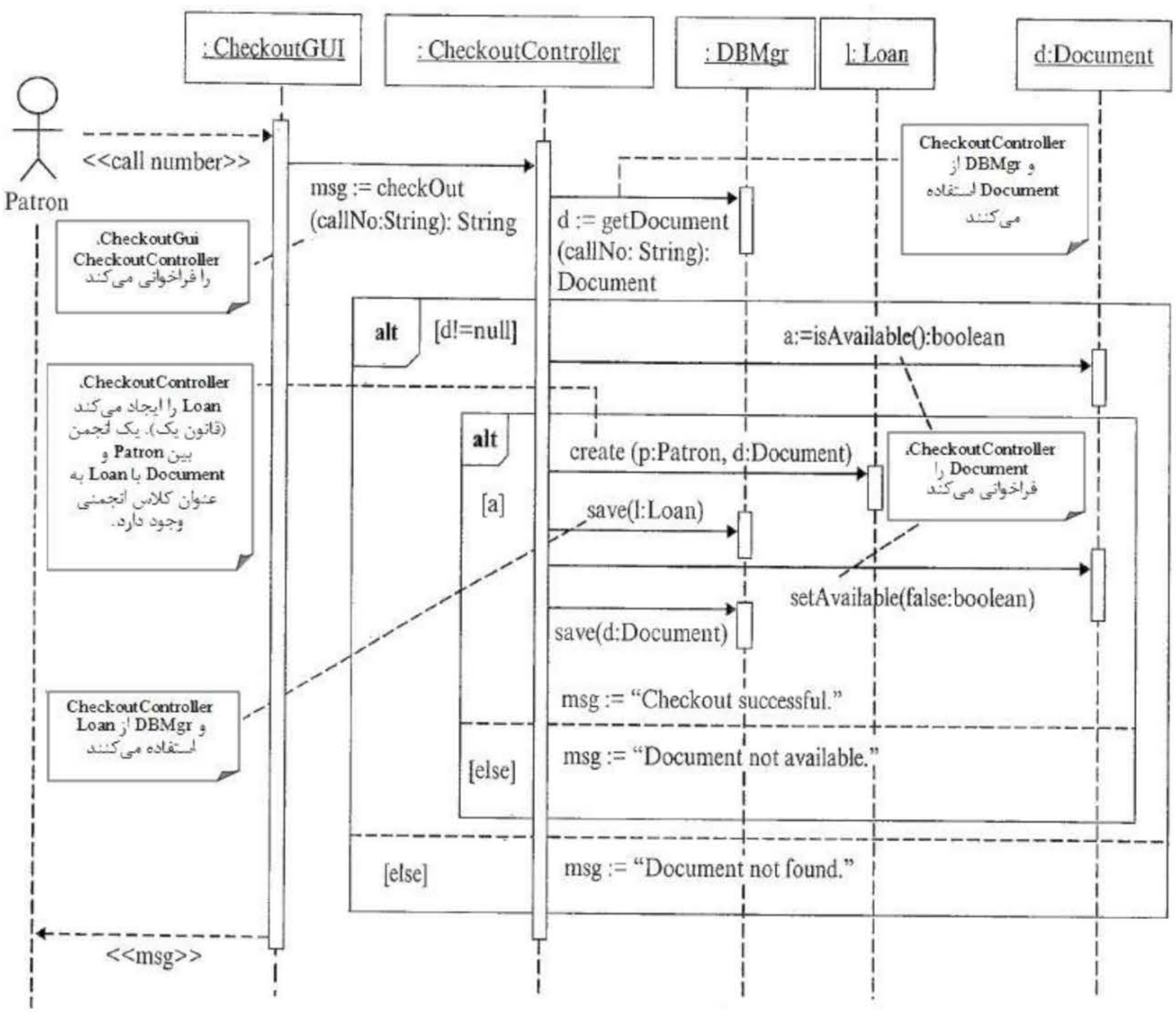
add(v:VendItem)

get(v:VendItem):VendItem

move(v:VendItem)

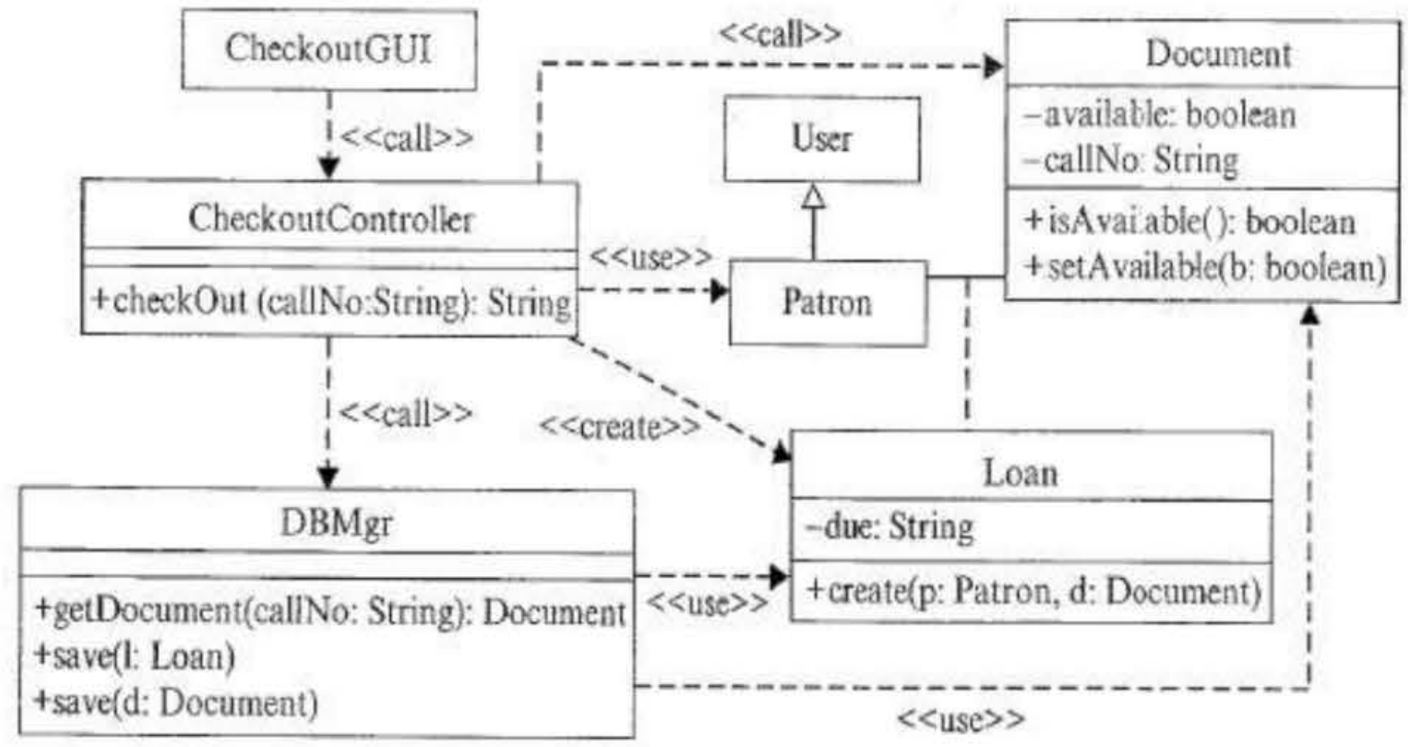
پیغام‌هایی به یک شئ از کلاس Dispenser از یک ماشین فروش[1] باشد، آن‌گاه رابطه‌ی Dispenser که دربرگیرنده‌ی VendItem است، شناسایی می‌شود.

۷. شناسایی روابط اضافی از روی مدل دامنه. روابط ارث‌بری، تجمیع، و انجمنی، و احتمالاً کلاس‌های اضافی، از مدل دامنه شناسایی می‌شوند. برای مثال، ممکن است روی مدل دامنه، User به‌عنوان کلاس والد Patron شناسایی شود. این‌کار افزودن زیرکلاس‌های دیگر به User مانند Staff و System Administrator را در آینده آسان می‌کند.



شکل ۱۱-۹: شناسایی روابط

[1] Vending Machine



شکل ۱۱-۱۰: DCD حاوی روابط و وابستگی call

۱۱-۳-۶- فهرست بررسی برای بازبینی نمودار کلاس طراحی

برای اطمینان از کیفیت، باید با استفاده از فهرست بررسی بازبینی زیر، DCD توسط اعضای تیم بازبینی گردد.

۱. مطمئن شوید که کلاس‌ها، ویژگی‌ها، اعمال، گونه‌ی پارامترها، گونه‌های بازگشتی، و روابط در DCD به درستی بر اساس گام‌هایی که در این فصل ارائه شدند، استنتاج گردیده‌اند.
۲. آیا DCD شامل کلاس‌ها، اعمال، یا روابط غیرضروری می‌شود؟
۳. آیا نام‌گذاری کلاس‌ها، ویژگی‌ها، اعمال و پارامترها به دقت کارکرد مورد نظر را ارائه نموده و فهم آن آسان است؟
۴. آیا DCD به وضوح الگوهای طراحی استفاده شده را نشان می‌دهد؟ (این به برنامه‌نویس در مرحله‌ی پیاده‌سازی کمک می‌کند).
۵. متریک‌هایی چون Fan–in ، Fan–out ، اندازه‌ی کلاس[۱]، عمق درخت ارث‌بری[۲]، و پیوستگی بین کلاس‌ها را محاسبه نموده و مشکلات احتمالی را شناسایی نمایید. برای جزییات بیشتر فصل ۹ با عنوان اطمینان از کیفیت نرم‌افزار را ببینید.

۱۱-۴ سازمان‌دهی کلاس‌ها با نمودار بسته

DCD ممکن است کلاس‌های متعددی داشته باشد و لذا فهم آن مشکل باشد. در این حالت، نمودار بسته‌ی UML[۳] که در فصل ۶ معرفی شد، برای سازمان‌دهی کلاس‌ها درون قطعات منطقی به نام بسته[۴] مفید است. بسته‌ها را می توان به راه‌های مختلف سازمان‌دهی نمود. دو سازمان‌دهی متداول و ترکیب آن‌ها عبارتند از:

۱. سازمان‌دهی زیرسیستم کارکردی
۲. سازمان‌دهی سبک معماری
۳. سازمان‌دهی ترکیبی

---

[۱] Class Size
[۲] Depth in Inheritance Tree
[۳] UML Package Diagram
[۴] Package

در سازمان‌دهی زیرسیستم کارکردی، کلاس‌ها بر حسب زیرسیستم کارکردی سیستم نرم‌افزار تجزیه می‌شوند. نتیجه‌ی این‌کار بسته‌هایی هستند که به زیرسیستم‌های کارکردی سیستم نرم‌افزاری مربوط می‌شوند. برای مثال، زیرسیستم‌های کارکردی یک سیستم اطلاعات کتابخانه شامل زیرسیستم امانت، زیرسیستم فهرست‌نویسی، زیرسیستم خرید، زیرسیستم امانت بین کتابخانه‌ای، و زیرسیستم راهنمایی کاربر می‌شود. علاوه بر این‌ها، سیستم دارای یک زیرسیستم ذخیره‌ی دائمی یا پایگاه‌داده نیز می‌شود. با استفاده از این روش، شش بسته‌ی متناظر تعریف می‌شوند. علاوه بر این بسته‌ها، یک بسته‌ی دیگر نیز خواهیم داشت که شامل تمامی کلاس‌های مربوط به سیستم اطلاعات کتابخانه اعم از Main GUI ، Login and logout GUI ، و Configuration می‌شود. شکل ۱۱-۱۱ زیرسیستم‌ها و بسته‌ها را نشان می‌دهد. یک حسن سازمان‌دهی زیرسیستم کارکردی این است که بازاستفاده زیرسیستم‌ها و بسته‌ها نسبتاً ساده است زیرا هر کدام از بقیه تقریباً مستقل است. یکی از معایب این روش نیز این است که طبقه‌های مختلفی از کلاس‌ها درون یک بسته قرار گرفته اند. برای مثال، هربسته شامل کلاس‌های GUI، کلاس‌های اشیای کسب‌وکار، کلاس‌های کنترلگر و کلاس‌های مشابه می‌شود.

سازمان‌دهی سبک معماری، کلاس‌ها را برحسب سبک معماری سیستم گروه‌بندی می‌کند. بار دیگر سیستم اطلاعات کتابخانه را در نظر بگیرید. سخت نیست که متوجه شویم این یک سیستم تعاملی است. طبق بحث‌های مربوطه در فصل ۶ ، سیستم باید از یک معماری Nتایی[1] استفاده کند. بنابراین، کلاس‌های سیستم درون بسته‌هایی سازمان‌دهی می‌شوند که متناظر با لایه‌های معماری Nتایی باشد. شکل ۱۱-۱۲ بسته‌هایی که احتمالاً با استفاده از این روش ایجاد می‌شوند را نشان می‌دهد. یکی از محاسن این روش این است که هر بسته یک گونه از کلاس‌ها را محفظه‌سازی می‌کند، مثلاً بسته‌ی GUI تنها شامل کلاس‌های مربوط به نمایش است، و بسته‌ی کنترلگر تنها شامل کنترلگرهاست، و بسته‌ی کسب‌وکار تنها شامل کلاس‌های اشیای کسب‌وکاری است، و الی آخر. در این صورت بازاستفاده از هر گونه از این اشیا ساده است. برای مثال، کلاس‌های اشیای کسب‌وکاری کتابخانه از بسته‌ی اشیای کسب‌وکار، بازاستفاده شده است. ولی بازاستفاده از یک زیرسیستم، مثلاً زیرسیستم امانت، در این روش دشوار است.

روش ترکیبی، هر دو سازمان‌دهی سبک معماری و سازمان‌دهی زیرسیستم کارکردی را ترکیب می‌کند. در اینجا یکی از دو روش زیر می‌تواند به کار گرفته شود:

۱. سازمان‌دهی زیرسیستم کارکردی سبک معماری. در این روش، کلاس‌های درون هر کدام از بسته‌های معماری بر طبق زیرسیستم‌های کارکردی سیستم سازمان‌دهی می‌شوند. برای مثال، کلاس‌های بسته‌ی GUI در شکل ۱۱-۱۲ بر طبق زیرسیستم‌های سیستم اطلاعات کتابخانه سازمان‌دهی می‌شوند. نتیجه‌ی این کار زیربسته‌های gui.interlibraryloan، gui.userhelp، gui.purchaising، gui.cataloging، gui.circulation می‌باشد.
۲. سازمان‌دهی سبک معماری زیرسیستم کارکردی. در این روش، کلاس‌های درون هر بسته‌ی کارکردی بر حسب معماری سیستم سازمان‌دهی می‌شوند. برای مثال، به‌کار بردن این روش در مورد بسته‌ی امانت در شکل ۱۱-۱۱ ممکن است منجر به تعریف زیربسته‌های circulation.gui، circulation.controller، cilculation.busibessobjects، circulation.database و circulation.network بشود.

روش ترکیبی محاسن هر دو روش دیگر را دارد. برای مثال، در شکل ۱۱-۱۱ هر بسته شامل بیش از یک گونه از کلاس‌های شئ می‌شود. بنابراین، جداسازی دغدغه‌ها در نظر گرفته نشده است، روش ترکیبی این مشکل را حل می‌کند.

---

[1] N-tier Architecture

زیرسیستم	توصیف کارکردی	بسته	کلاس‌های ممکن	روابط
زیر سیستم امانت Circulation subsystem	کارکردهایی برای جستجوی مدارک، خروج مدارک، بازگشت مدارک، رزرو مدارک، به‌خاطرآوری مدارک، و سایر فعالیت‌های مرتبط فراهم می‌آورد.	Circulation	شامل تمام کلاس‌های مرتبط با امر امانت می‌شود، مانند Document , Loan , Patron , Checkout GUI , Checklout Controllers, Return GUI, Return Controller	وارد کردن بسته‌های پایگاه‌داده و امانت
زیرسیستم فهرست نویسی Cataloguing Subsystem	کارکردهایی برای دسته‌بندی موضوعی سندها، از جمله کتاب و گاهنامه فراهم می‌آورد و به هر سند شماره واحدی اختصاص می‌دهد.	Cataloguing	شامل تمام کلاس‌های مرتبط با امر فهرست‌نویسی مانند Catalog، Subject Area، Description، GUI و کلاس‌های کنترلگر برای مورد کاربردهای مرتبط با فهرست‌نویسی می‌شود.	وارد کردن بسته‌های پایگاه‌داده
زیرسیستم خرید Purchasing Subsystem	کارکردهایی مانند بررسی و پردازش درخواست‌های خرید اعضای هیئت علمی و دانشجویان، سفارش انتشارات از ناشر، پیگیری وضعیت سفارش و پردازش عملیات حمل از ناشرها را فراهم می‌آورد.	Purchasing	شامل همه کلاس‌های وابسته به خرید مانند Publisher، Order، Purchase، Request، Shipment، GUI و کلاس‌های کنترلگر برای موارد کاربرد مربوط با خرید می‌شود.	وارد کردن بسته‌های پایگاه‌داده
زیرسیستم کمک کاربری User Assistance Subsystem	کارکردهایی برای آموزش کاربر، کمک به کاربر، پرسش و پاسخ و دیگر فعالیت‌های کمکی فراهم می‌آورد.	User assistance	شامل همه کلاس‌های کمک کاربر مانند Tutorial، Help Topic، Question، Answer، GUI و کلاس‌های کنترلگر برای مورد استفاده کمک- کاربر می‌شود.	وارد کردن بسته‌های پایگاه‌داده و فهرست‌نویسی
زیرسیستم امانت بین کتابخانه‌ای Interlibrary Loan Subsystem	کارکردهایی برای پردازش امانت بین کتابخانه‌ای را فراهم می‌آورد.	Interlibrary loan	شامل همه کلاس‌های مرتبط با امانت بین کتابخانه‌ای مانند Collaborator، Checkout To، Loan From، GUI و کلاس‌های کنترلگر برای موارد کاربرد امانت بین کتابخانه‌ای می‌شود.	وارد کردن بسته پایگاه‌داده
زیرسیستم پایگاه‌داده Database Subsystem	کارکردهایی را برای ذخیره و بازیابی اشیاء توسط یک سیستم مدیریت پایگاه‌داده فراهم می‌آورد.	Database	شامل همه کلاس‌های مرتبط با پایگاه‌داده مانند DB Manager، DB Connection و کلاس‌های بیشمار دیگری که پیاده‌سازی و نگهداری زیرسیستم پایگاه‌داده را ممکن می‌سازند، می‌شود.	وارد کردن بسته‌های امانت، فهرست‌نویسی، خرید، کمک کاربری و امانت بین کتابخانه‌ای می‌شود.
سیستم اطلاعات کتابخانه Library Information System	کارکردهایی برای درخواست زیرسیستم‌های دیگر، ورود و خروج از سیستم و پیکربندی را فراهم می‌آورد.	Library system	شامل واسط (GUI) اصلی برای هدایت به زیرسیستم‌های کارکردی، Login GUI، Logout GUI و Configuration می‌شود.	وارد کردن بسته‌های امانت، فهرست‌نویسی، خرید، کمک کاربری و امانت بین کتابخانه‌ای می‌شود.

شکل ۱۱-۱۱: بسته‌ها و زیرسیستم‌های کارکردی

۱۱-۵ اعمال اصول چابکی

رهنمون‌های زیر باید در موقع استنتاج DCD در نظر گرفته شوند.

لایه‌ای از nلایه	توصیف	بسته	کلاس‌های موجود در بسته	ارتباطات
نمایش (Presentation)	کارکردهای مربوط به واسط گرافیکی کاربر را برای LIS فراهم می‌سازد.	GUI	کلاس‌های متنوع پنجره و گفتگو نیز کلاس‌های مختلف نمایش اطلاعات از قبیل جداول و نمودارها	واردکردن بسته‌ی کنترلگر
کنترلگر (Controller)	کارکردهایی را برای کنترل رویدادهای سیستم فراهم می‌آورد.	Controller	کلاس‌های کنترلگر مورد کاربرد از قبیل Checkout Controller، Return Controller و غیره.	وارد کردن بسته های اشیای کسب و کار، پایگاه‌داده و شبکه
اشیای کسب وکار (Business object)	کارکردهای اشیای کسب و کار درخواست شده توسط کنترگر را فراهم می‌آورد.	Business object	Document، Loan، Patron، Publisher، Purchas Order، Periodical، Book، Catalog و غیره	
پایگاه‌داده (Database)	کارکردهایی را برای ذخیره و بازیابی اشیاء فراهم می‌آورد.	Database	Database manager، کلاس‌های الگوی طراحی که برای پیاده‌سازی کاکردهای ذخیره و بازیابی اشیاء استفاده شده است.	وارد کردن بسته‌های شبکه و اشیای کسب و کار
شبکه (Network)	کارکردهای ارتباط شبکه از قبیل ارسال و دریافت TCP/IP را فراهم می‌آورد.	Network	کلاس‌هایی برای ارتباطات شبکه	

شکل ۱۱-۱۲: بسته‌بندی کلاس‌ها بر طبق یک سبک معماری

رهنمون ۱۱-۱

به نرم‌افزار درحال‌کار بیش از مستندات جامع بها دهید.

DCD از روی نمودارهای طراحی به‌دست می‌آید، که آن‌ها هم از روی نیازمندی‌ها به‌دست آمده‌اند و انعکاس‌دهنده‌ی ایده‌های طراحی تیم هستند. برای مسائل بدرفتار، نمی‌توان پیاده‌سازی را از تبیین مسئله جدا نمود. تبیین و ایده‌های طراحی باید توسط نرم‌افزار درحال‌کار آزمایش شوند، بنابراین، تیم نباید مقدار زیادی تلاش صرف ساختن یک DCD کامل برای دامنه‌ی کاربرد بنماید. DCD تنها باید کلاس‌ها، ویژگی‌ها، متدها و روابط را برای مورد کاربردهای تخصیص‌یافته به تکرار فعلی اضافه نماید، نه برای تکرارهای بعدی.

رهنمون ۱۱-۲

به اندازه‌ی کافی خوب، کافی است.

DCD که برحسب گام‌های ذکر شده در این فصل به‌دست آمده است شامل اکثر کلاس‌ها، خصیصه‌ها، و روابط لازم برای تکرار فعلی خواهد بود. در بیشتر موارد، این برای توسعه‌ی آزمون‌رانده‌ی تکرار فعلی کفایت می‌کند (یعنی به اندازه‌ی کافی خوب هست). بنابراین، تیم نباید کلاس‌ها و روابط زیاد دیگری را هم اضافه نماید.

۱۱-۶ پشتیبانی ابزاری از نمودار کلاس طراحی

Netbeans UML plugin ، ArgoUML ، Microsoft Visio ، IBM Rational Modeler و سایر ابزارهای مشابه، از ایجاد، ترسیم، ویرایش نمودارهای کلاس و سایر قابلیت‌های مرتبط پشتیبانی می‌کنند. اکثر ابزارها همچنین از مهندسی معکوس هم

پشتیبانی می‌کنند که نمودار کلاس را از روی کد تولید می کند. ولی، هنوز در ادبیات موضوع، پشتیبانی ابزاری از تولید DCD از روی نمودار توالی و مدل دامنه گزارش نشده است. دانشگاه تگزاس در آرلینگتون[۱] در حال توسعه‌ی چنین ابزاری است .

خلاصه

این فصل به معرفی گام‌هایی برای استنتاج DCD از مدل‌های رفتاری و مدل دامنه می‌پردازد. DCD به‌عنوان اوزالید طراحی عمل می‌کند که دست‌ساخته‌های طراحی تولید شده در گام‌های قبلی را یکپارچه می‌سازد. این فصل همچنین نشان می‌دهد که چگونه از نمودارهای بسته برای سازمان‌دهی کلاس‌های DCD استفاده می‌شود. به‌ویژه، سازمان‌دهی زیر سیستم کارکردی، سازمان‌دهی سبک معماری، و ترکیب این دو روش با یک سیستم نمونه‌ی اطلاعات کتابخانه مورد بحث قرار گرفته‌اند.

برای مطالعه‌ی بیش‌تر

مرجع [۳۶] ارائه‌ی خوبی از نمودارهای کلاس UML می‌دهد. وبگاه OMG تبیین جامعی از نمودار کلاس ارائه می‌دهد.

سوال‌های مروری فصل

۱. نمودار کلاس طراحی (DCD) چیست؟
۲. چرا ما به DCD نیاز داریم؟
۳. قوانین مربوط به شناسایی کلاس‌ها، متدها، صفت‌ها، و روابط به ترتیب کدامند؟
۴. سازمان‌دهی زیرسیستم کارکردی و سازمان‌دهی سبک معماری در مورد کلاس‌ها چیست؟ سازمان‌دهی ترکیبی و ترکیب‌های آن چیست؟

تمرین‌ها

۱-۱۱ از روی نمودار توالی شکل ۹-۲۱ یک DCD به‌دست آورید.

۲-۱۱ از روی نمودار توالی که برای مورد کاربرد سفارش غذا[۲] در تمرین‌های فصل ۹ تولید کردید، یک DCD به‌دست آورید. اگر نمودار توالی گفته شده را تولید نکرده‌اید، تمرین‌های مربوطه از فصل ۹ را اول انجام دهید، سپس DCD را تولید کنید.

۳-۱۱ از روی نمودارهای توالی طراحی که برای ویرایشگری نمودار حالت در تمرین ۹-۷ تولید کردید. یک DCD به دست آورید.

۴-۱۱ از روی نمودارهای توالی طراحی که برای کاربرد ATM در تمرین ۹-۵ تولید کردید، یک DCD به دست آورید.

۵-۱۱ از روی نمودارهای توالی طراحی که برای کاربرد اجاره‌ی خودرو در تمرین ۹-۶ تولید کردید، یک DCD به دست آورید.

---

[۱] The University of Texas at Arlington

[۲] Order a Dish

۱۲

طراحی واسط کاربر

نکات کلیدی

- طراحی واسط کاربر با طراحی ظاهر و حس[۱] واسط‌های کاربر سروکار دارد.
- در حین طراحی واسط کاربر باید اصول طراحی نرم‌افزار، شامل طراحی برای تغییر، جداسازی دغدغه‌ها، نهان‌کاری اطلاعات، چسبندگی زیاد، پیوستگی کم، و ساده و احمقانه بگیر اعمال شوند.

واسط کاربری یک سیستم نرم‌افزاری روش و سازوکاری است که از آن طریق یک کاربر با سیستم تعامل می‌نماید تا وظایف کسب‌وکاری‌اش را به انجام رساند. کاربران با استفاده از واسط، خدماتی از سیستم درخواست می‌نمایند، ورودی‌ها را ارائه می‌کنند و واکنش‌های سیستم را دریافت می‌نمایند. احساس کاربران در خصوص واسط به شدت بر روی پذیرش سیستم و موفقیت پروژه تاثیر می‌گذارد. اگر یادگیری و استفاده از واسط کاربری آسان باشد، آنگاه احتمال استفاده از سیستم و استفاده‌ی درست از سیستم توسط کاربران زیاد خواهد بود. این منجر به افزایش بهره‌وری و پذیرش سیستم جدید خواهد شد. از سوی دیگر، اگر واسط کاربری پیچیده، گیج‌کننده، یا دشوار برای استفاده باشد، در این‌صورت به احتمال زیاد کاربران احساس خستگی نموده و مرتکب اشتباه می‌شوند. بنابراین، طراحی واسط کاربر یک فعالیت بحرانی در توسعه‌ی نرم‌افزار است. البته نباید اهمیت این فعالیت بیش از حد بزرگ جلوه داده شود.

در فصل ۸ در خصوص مدل‌سازی تعامل کنشگر-سیستم بحث شد. بدین مفهوم که توضیح داده شد که چگونه کاربران با سیستم تعامل می‌کنند تا مورد کاربردها را اجرا نمایند. مدل‌سازی تعامل کنشگر-سیستم توالی جفت‌هایی که شامل درخواست کاربر و به‌دنبال آن واکنش سیستم به آن درخواست می‌باشند را مشخص می‌نمایند. به این‌ها رفتار تعاملی کنشگر-سیستم می‌گویند. این‌ها بخشی از طراحی واسط کاربر، یعنی بخشی از طراحی رفتار واسط کاربر، می‌باشند، در فصل ۸ درخصوص ظاهر واسط کاربر و سایر رفتار تعاملی واسط کاربر که مختص ویجت‌های[۲] گرافیکی است بحثی نشد، برای مثال استفاده از یک فیلد متنی برای ورود داده‌ها با استفاده از یک فهرست انتخابی فرق دارد.

این فصل مفاهیم و گام‌های اولیه برای طراحی واسط کاربر را ارائه می‌نماید. به‌ویژه، شما موارد زیر را خواهید آموخت:

---

[۱] Look and Feel

[۲] Widget

- اجزای طراحی یک واسط کاربر
- اهمیت طراحی یک واسط کاربر
- فرایند طراحی یک واسط کاربر
- اصول طراحی یک واسط کاربر
- رهنمون‌هایی برای طراحی واسط کاربر
- اصول چابکی در طراحی یک واسط کاربر

۱-۱۲ طراحی واسط کاربر چیست؟

همان‌گونه که پیشتر گفته شد، واسط کاربر وسیله و سازوکاری است که از آن طریق کاربران با سیستم تعامل می‌کنند. با گذشت زمان و هر چه فناوری‌های مربوطه پیشرفت می‌کنند این روش‌ها و سازوکارها نیز پیچیده‌تر می‌شوند. چند دهه پیش، واسط کاربر کاملاً ابتدایی بود. برای وارد کردن یک برنامه و دادن داده به آن، از ماشین تایپ مخصوصی استفاده می‌شد که روی یک نوار سیاه یا روی کارت‌های منگنه [۱] سوراخ‌هائی را ایجاد (پانچ) می‌کرد. سپس نوار منگنه روی یک دستگاه نوری نصب می‌شد تا نوار را خوانده و دنباله بیت‌ها را به کامپیوتر ارسال نماید. دستگاه‌های خروجی نیز ماشین تایپ کنسول [۲] و چاپگر خطی [۳] بودند. پردازش دسته‌ای [۴] روش غالب در رایانش بود. برطبق استاندارد امروز، چنین واسط کاربری به هیچ وجه کاربر پسند [۵] نیست، ولی در آن زمان استفاده از کامپیوتر یک افتخار بود؛ لذا کسی در این خصوص شکایتی نداشت.

سیستم‌های نرم‌افزاری امروزی دارای واسط‌های کاربر گرافیکی [۶] بوده و مُد پردازش در آن‌ها تعاملی است، گرچه هنوز هم واسط‌های متنی [۷] مورد استفاده قرار می‌گیرند. مشخصه‌های واسط‌های کاربر گرافیکی عبارتند از:

- چندوظیفگی [۸] مبتنی بر ویندوز: کاربران می‌توانند چندین پنجره‌ی کاری را باز کرده و در یک زمان روی وظایف مختلف کار کرده و آن‌ها را ردیابی نمایند.
- یادگیری و استفاده‌ی آسان [۹]: طراحی مناسب ظاهر و حس با استفاده از ویجت‌های گرافیکی باعث می‌شود تا واسط کاربر طبیعی شده و یادگیری و استفاده از آن آسان باشد.
- نمایش چندرسانه‌ای [۱۰]: امکان استفاده از گرافیک، صوت، پویانمایی [۱۱] و فیلم، نمایش و ارتباط اطلاعات را به شدت بهبود می‌بخشد.

هدف از طراحی واسط کاربر این است که با استفاده از این ویژگی‌ها، تعامل کاربر و ارائه‌ی قابلیت‌های سیستم به او آسان‌تر گردد. برای دست‌یابی به این اهداف، معمولاً طراحی واسط کاربر شامل فعالیت‌های طراحی زیر می‌گردد.

۱. طراحی طرح اولیه‌ای برای پنجره‌ها و جعبه گفت‌وگو. منظور از این کار بخش‌بندی کل صفحه نمایش به پنجره‌ها و جعبه‌های گفت‌وگو است تا تعامل کاربر و کار روی وظایف مختلف را آسان کند. برای مثال، اغلب کاربردهای مبتنی بر ویندوز، صفحه نمایش را به‌طوری بخش‌بندی می‌کنند که یک میله‌ی فهرست [۱۲] و یک میله‌ی ابزار [۱۳] در بالا و یک میله‌ی

---

[۱] Punch Card
[۲] Console Typewriter
[۳] Line Printer
[۴] Batch Processing
[۵] User Friendly
[۶] GUI
[۷] Text-based
[۸] Multitasking
[۹] Easy to Learn and Use
[۱۰] Multimedia Presentation
[۱۱] Animation
[۱۲] Menubar
[۱۳] Toolbar

وضعیت[۱] در پائین قرار می‌گیرد. اکثر محیط‌های توسعه‌ی یکپارچه (IDE) مابقی صفحه نمایش را به بخش‌هایی مانند قاب پروژه[۲]، قاب ناوبری[۳]، و قاب محتوی[۴] تقسیم می‌کنند تا کارکردن روی پروژه برای کاربر راحت شود.

۲. طراحی رفتار تعاملی. هدف از این کار طراحی دنباله‌ای از پیام‌های رد و بدل شده بین کاربر و سیستم است. برای مثال، کاربر IDE گزینه‌ی پرونده[۵] را فشار داده و پروژه‌ی جدید[۶] را انتخاب می‌کند تا جعبه‌ی گفت‌وگوی مشخصات پروژه[۷] باز شود.

۳. طراحی نقشه‌ی نمایش اطلاعات. منظور از این کار طراحی نمایش نتایج پردازش اطلاعات می‌باشد. برای مثال، استفاده از یك نمودار میله‌ای برای برجسته کردن تفاوت فروش ماهیانه‌ی محصولات مختلف، یا یك نمودار دایره‌ای برای نمایش درصد فروش هر محصول از کل فروش.

۴. طراحی پشتیبانی برخط. هدف از این کار طراحی راهنمای برخط کاربران، دفترچه راهنمای کاربران، پیغام‌های خطا، تسهیلات کمك، عدم انجام[۸] و انجام مجدد[۹]، پشتیبان‌گیری و بازاندوزی[۱۰]، ترمیم[۱۱]، و سایر قابلیت‌های پیشتبانی می‌باشد.

۱۲-۲ چرا طراحی واسط کاربر مهم است؟

سازمان‌های دولتی و کسب‌وکاری روی سخت‌افزار و نرم‌افزار کامپیوتر سرمایه‌گذاری می‌کنند، با این انتظار که بهره‌وری و کیفیت خدمات را افزایش داده و در عین حال هزینه‌های اجرایی را پایین بیاورند. بازگشت سرمایه[۱۲] بستگی به طراحی واسط کاربر دارد، زیرا این تنها کانال ارتباطی میان کاربر و سیستم است. از طریق واسط کاربر، کاربران از کامپیوتر استفاده می‌کنند تا وظایف کسب‌وکاری را انجام دهند. اگر استفاده از واسط کاربر آسان باشد، آنگاه بهره‌وری و کیفیت کار افزایش یافته و در عوض هزینه‌های عملیاتی کاهش می‌یابد. از سوی دیگر، چنان‌چه درك و استفاده از واسط کاربری دشوار باشد، آن‌گاه ممکن است کاربران از استفاده از سیستم اجتناب نمایند یا نرخ خطای شغلی ایشان افزایش یابد. بنابراین، بازگشت سرمایه کم خواهد بود. متأسفانه، اغلب اهمیت واسط کاربری دست کم گرفته می‌شود. این امر باعث تولید محصولاتی می‌شود که کارکرد و کارایی آن‌ها عالی است اما این محصولات به‌دلیل واسط کاربری ضعیف فروش نمی‌روند. در این جا چند داستان از دنیای واقعی نقل می‌کنیم.

اولین داستان در مورد یك نرم‌افزار توسعه‌ی کاربرد سریع[۱۳] است که قرار بود مهندس نرم‌افزار، یك سیستم نرم‌افزاری در دامنه کسب‌وکاری معین را با استفاده از قطعات بازاستفاده‌پذیر موجود در یک مخزن، به‌سرعت تولید نماید. ایده و پیاده‌سازی در مطالعات آزمایشی[۱۴] به خوبی پیش رفت. ولی برخلاف تلاش وسیع در بازاریابی، محصول خیلی خوب به‌فروش نرفت. یك مانع بزرگ، واسط گرافیکی کاربر بود که به روش اقتضایی[۱۵] ساخته شده بود. این واسط گرافیکی تمام انواع شمایل‌ها را در خود داشت که باعث خستگی و سردرگمی کاربران می‌شد. داستان دیگر در مورد یك نرم‌افزار فروشگاهی است که هزینه‌های حمل برای ارسال نامه‌ها و بسته‌ها را محاسبه می‌کرد و یك برچسب مخصوص برای بسته چاپ می‌کرد. این نرم‌افزار توسط فروشگاه‌های کوچك و خصوصی حمل‌ونقل که شرکای خصوصی سرویس پستی ایالات متحده[۱۶] بودند، استفاده می‌شد. برخلاف

---

[۱] Status bar
[۲] Project Pane
[۳] Navigation Pane
[۴] Content Pane
[۵] File
[۶] New Project
[۷] Project Specification
[۸] Undo
[۹] Redo
[۱۰] Backup and restore
[۱۱] Recovery
[۱۲] Return of Investment (RoI)
[۱۳] Rapid Application Development
[۱۴] Pilot Studies
[۱۵] Ad hoc
[۱۶] United States Postal Service (USPS)

usps و فروشگاه‌های زنجیره‌ای صاحب برند[۱]، این فروشگاه‌های کوچك به مشتری امکان حمل بسته توسط شرکت‌های مختلفی، از جمله usps ، را می‌دادند. بنابراین، مشتری امکان انتخاب در مورد شرکت حمل‌ونقل، زمینی یا هوایی، عادی یا سریع‌السیر را با نرخ‌های مناسب داشت. خصیصه‌ی دیگر چنین فروشگاه‌هایی، خدمات سریع به‌منظور کاهش زمان انتظار مشتری می‌باشد. نرم‌افزار مربوطه که توسط فروشگاه‌های کوچك استفاده می‌شود نیاز داشت که کارمند فروشگاه چهار پنجره را باز کند تا بتواند یك بسته را حمل کند. در هر پنجره انبوهی از آیکون‌ها و دکمه‌های رنگی وجود داشت که کاربران را گیج کرده و باعث می‌شد که اشتباه کنند. اصلاح یك اشتباه هم مانند یك کابوس است. چنان‌چه شما نیز شاید انتظار داشته باشید، این نرم‌افزار خوب فروش نرفت و مشتریان اندکی داشت.

در مقایسه، یك نرم‌افزار رقیب تنها از یك پنجره استفاده می‌کند. کارمند فروشگاه تنها لازم است وزن بسته و کد پستی مقصد را وارد نماید. وزن بسته می‌تواند توسط یك ترازوی دیجیتال که به کامپیوتر متصل است تعیین شود. برخلاف نرم‌افزار قبلی که کاربر موظف بود شرکت حمل‌ونقل و سرویس مورد نظر را انتخاب نماید تا هزینه‌های حمل محاسبه شود، همان‌طور که در شکل ۱۲-۱ نشان داده شده است. این نرم‌افزار رقیب هزینه‌ی حمل مربوط به تمام سرویس‌ها و تمام شرکت‌ها را در یك صفحه نشان می‌دهد. پس از این که مشتری تصمیم خود را گرفت، کارمند فروشگاه نام و آدرس گیرنده را وارد می‌کند. در این زمان، روی دکمه‌ی مورد نظر می‌زند تا برچسب پستی چاپ شود. فلسفه‌ی طراحی این نرم‌افزار رقیب، ساده و احمقانه فرض کن، و ارزیابی تنبل[۲] - به معنی «بیش از آن چه نیاز است کاری نکن» - می‌باشد. برای مثال، به‌جای چهار پنجره، از یك پنجره استفاده شده است و به‌جای این‌که کاربر مجبور باشد تمام اقلام اطلاعات را، بدون توجه به این‌که لازم است یا خیر، وارد نماید، نرم‌افزار رقیب از مقادیر پیش‌فرض استفاده می‌نماید. وقتی مشتری تصمیم خود را می‌گیرد، این نرم‌افزار فقط نیاز به نام و آدرس گیرنده دارد، به عبارتی ارزیابی تنبل انجام می‌شود. این بدین خاطر است که برخی مشتریان فقط می‌خواهند بدانند هزینه‌ی حمل چقدر است یا بسته چه موقع به مقصد می‌رسد.

۱۲-۳ ویجت‌های واسط گرافیکی کاربر

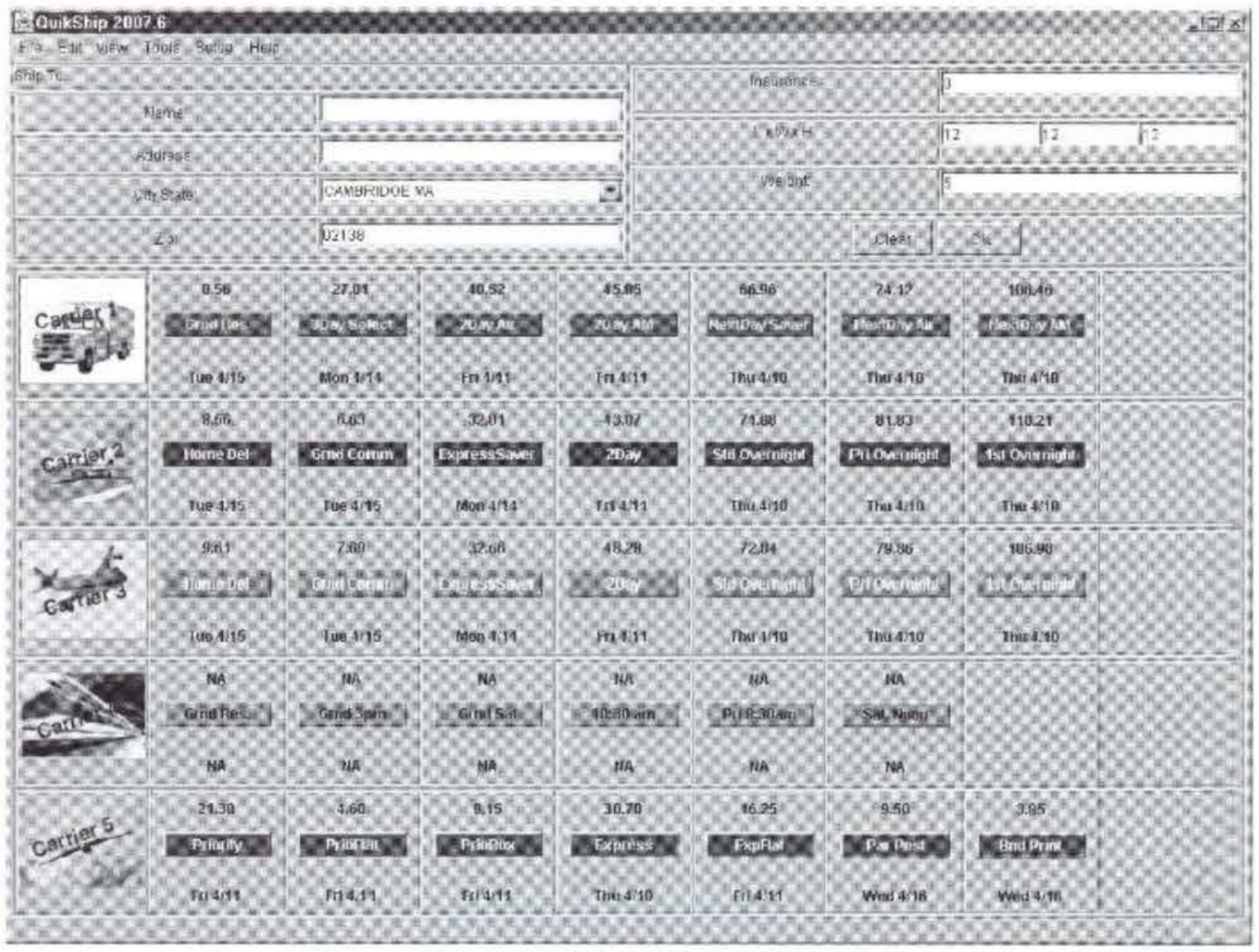
واسط‌های گرافیکی کاربر متشکل از ویجت‌های GUI، یا به‌سادگی «ویجت‌ها»، هستند؛ مانند پنجره‌ها، جعبه‌های گفت‌وگو، فهرست‌ها، گزینه‌ی فهرست‌ها، و خیلی چیزهای دیگر. ویجت‌های مختلف به‌منظورهای متفاوتی به‌کار می‌روند، و استفاده‌ی درست از ویجت‌ها مهم است. برای صرفه‌جویی و سادگی، این بخش فقط ویجت‌هایی که بیشتر استفاده می‌شوند و آن‌هایی که در واسط کاربری کاربردهای مستقل[۳] به‌کار می‌روند را معرفی می‌کند. از نظر ملاحظات طراحی، اکثر ویجت‌های وب کاربردها مشابه همان‌هایی هستند که برای کاربردهای مستقل استفاده می‌شوند. ولی، پیاده‌سازی واسط کاربری وب[۴] متفاوت است.

۱۲-۳-۱- ویجت‌های دربرگیرنده

ویجت‌های دربرگیرنده شامل مواردی چون پنجره، جعبه‌ی گفت‌وگو، قاب طوماری[۵]، قاب زبانه‌دار[۶]، و قاب لایه‌ای[۷] می‌شوند. پنجره، اغلب برای نشان دادن نمایش اصلی یا پنجره‌ی اصلی از یك کاربرد مستقل یا زیرسیستم‌های آن استفاده می‌شود. وقتی سیستم نرم‌افزاری شروع به کار می‌کند، پنجره‌ی اصلی ایجاد شده و در طول کاربرد وجود خواهد داشت. این پنجره می‌تواند هر جایی روی میز کار کاربر نمایش داده شود. بستن پنجره به کاربرد خاتمه می‌دهد و خروج از کاربرد نیز پنجره را می‌بندد. در جاوا، پنجره‌ها با قاب‌هایی تزیین می‌شوند که بر روی آن میله‌ی عنوان، مرز، و سایر دکمه‌ها و فهرست‌ها برای مدیریت پنجره وجود

---

[۱] Brand
[۲] Lazy Evaluation
[۳] Stand–alone
[۴] Web User Interface (WUI)
[۵] Scroll Pane
[۶] Tabbed Pane
[۷] Layered Pane



شکل ۱۲-۱: یك نرم‌افزار حمل‌ونقل که کار کاربر را آسان می‌کند.

دارند. از نقطه نظر کاربر، فرقی بین پنجره و قاب وجود ندارد؛ لذا در این فصل نیز تفاوتی بین این دو نیست. جعبه‌ی گفت‌وگو پنجره‌ای است که توسط پنجره‌ای دیگر فعال می‌شود تا کاربر را در گفت‌وگو درگیر کند. بستن پنجره‌ی گفت‌وگو نرم‌افزار کاربردی را نمی‌بندد.

قاب‌های طوماری متشکل از میله‌های طوماری افقی و عمودی‌اند که دیدن قسمت‌های مختلف از یك شئ بزرگ مانند یك فهرست بلند، یك نمودار یا یك تصویر بزرگ را آسان می‌سازند. قاب‌های زبانه‌دار و قاب‌های لایه‌لایه برای نمایش جنبه‌های مختلف یا نمونه‌های متفاوت از یك موضوع مفیدند و در واقع از اصل طراحی جداسازی دغدغه‌ها پشتیبانی می‌کنند. برای مثال، در یک IDE که منبع هر کلاس در یك قاب زبانه‌دار مجزا نمایش داده می‌شود. در یك جعبه‌ی تنظیم پیکربندی، ممکن است قابهای لبه‌دار متفاوتی استفاده شوند تا جنبه‌های مختلف پیکربندی یك سیستم را تعریف کنند.

فهرست‌ها و گزینه‌های فهرست‌ها، مانند فهرست‌های بازشو[۱]، همراه با پنجره‌ها به‌کار می‌روند تا اعمالی که کاربر می‌تواند فراخواند را سازمان‌دهی نموده و نمایش دهند. دکمه‌ها و دیگر ویجت‌های ورودی/خروجی توسط پنجره‌ها، جعبه‌های گفت‌وگو، صفحات لبه‌دار، و صفحات لایه‌لایه استفاده می‌شوند تا فرم ورودی را شکل داده و نتایج را به کاربر نمایش دهند. ویجت‌های ورودی/خروجی در دو بخش بعدی توصیف می‌شوند. در موقع استفاده از ویجت‌های دربرگیرنده بایستی رهنمون بعدی را مدّنظر قرار داد.

---

[۱] Popup Menu

رهنمون ۱۲-۱

در عین این‌که سعی می‌کنید واسط کاربری از نظر فهم و استفاده آسان باشد، تعداد پنجره‌ها و جعبه‌های گفت‌وگو که لازم است برای تکمیل یك وظیفه‌ی کسب‌وکاری باز شوند را حداقل کنید.

کم کردن تعداد پنجره‌ها تجربه‌ی کاربر را بیشتر می‌کند. البته این نباید منجر به افزایش پیچیدگی و جعبه‌های گفت‌وگو شده و فهم و استفاده از آن‌ها را دشوار نماید. در این رابطه، تفکر خلاقانه نیاز است. GUI نشان داده شده در شکل ۱۲-۱ می‌گوید که چگونه این‌کار انجام می‌شود. به‌عنوان مثال، گرچه تعداد جعبه‌های گفت‌وگو به یك عدد کاهش یافته است، ولی واسط کاربری از نظر فهم و استفاده مشکل نشده است. این بدین دلیل است که گزینه‌ها به روشی خوش‌سازمان‌دهی شده و آسان‌فهم نمایش داده شده‌اند. این نمایش به مشتری کمك می‌کند که یك گزینه برای حمل یا یك کارمند فروشگاه را انتخاب کند تا پاسخ درخواست مشتری را بدهد. نرم‌افزار برای بیمه و نیز ابعاد کالا مقادیر پیش‌فرض را در نظر می‌گیرد که در مورد ۹۵% حمل‌ونقل‌ها یکسان هستند. همچنین کاربر مجبور نیست نام و آدرس گیرنده‌ی کالا را وارد کند، تا زمانی که لازم شود.

۱۲-۳-۲- ویجت‌های ورودی، خروجی و نمایش اطلاعات

شکل ۱۲-۲ برخی ویجت‌های متداول برای ورودی کاربر و نمایش اطلاعات را نشان می‌دهد. این ویجت‌ها به چند دسته تقسیم می‌شوند:

ویجت‌های ورودی/خروجی متن‌گرا

این‌ها روش‌های ورودی/خروجی متون هستند که به کاربر امکان می‌دهند به تعداد از پیش تعیین‌شده‌ای نویسه وارد کند. این روش‌ها به‌طور متداول در جعبه‌های گفت‌وگو یا فرم‌های html استفاده می‌شوند تا اطلاعات ورودی متنی را از کاربر جمع‌آوری نمایند. یك حسن این ویجت‌های ورودی این است که دامنه‌ی ورودی خیلی بزرگ است. این ویجت‌ها به برنامه‌نویس این امکان را می‌دهند که طول مجاز متن‌ها و مجموعه نویسه‌های معتبر را تعیین کند. یك عیب این ویجت‌های ورودی نرخ خطای ورودی است که به‌دلیل اشکالات تایپی[۱] رخ می‌دهد و بیشتر از ویجت‌های ورودی انتخاب‌گرا می‌باشد. همچنین سرعت ورودی کمتر از انواع دیگر ویجت‌های ورودی است. عیب دیگر این است که اگر واسط به‌درستی طراحی نشده باشد به‌طور مداوم بایستی بین مُد صفحه کلید و مُد ماوس تغییر داد. بنابراین، ویجت‌های ورودی متن‌گرا تنها باید برای ورودی‌های «متن آزاد»[۲] به‌کار روند.

ویجت‌های ورودی انتخاب‌گرا

این ویجت‌های ورودی به کاربر امکان می‌دهند با انتخاب یك یا چند گزینه از بین تعداد متناهی از انتخاب‌های مشخص، اطلاعات را وارد نماید. به‌منظور ارضای نیازهای ورودی‌های متفاوت، این‌ها در روش‌های انتخاب فرق می‌کنند. محاسن ویجت‌های ورودی انتخاب‌گرا عبارتند از دقت ورودی زیاد و ورودی سریعتر، زیرا هیچ‌گونه تایپی نیاز نیست. اگر استفاده از ورودی متن‌گرا محدود شود، تغییر بین صفحه کلید و ماوس باید به کلی حذف شود یا کاهش یابد. یك عیب روش ورودی انتخاب‌گرا این است که ممکن است آنچه کاربر می‌خواهد در بین انتخاب‌ها نباشد. این را می‌توان با افزودن یک فیلد متنی

[۱] Typo
[۲] Free Text

به‌صورت «سایر، لطفاً مشخص نمایید» برطرف نمود که به کاربر امکان می‌دهد انتخاب خود را وارد نماید. یك عیب احتمالی دیگر می‌تواند گزینه‌های خسته‌کننده، گیج‌کننده و یا دشوار برای فهمیدن باشد. ورودی انتخاب‌گرا، اگر به درستی به‌کار رود، تجربه‌ی کاربر را بهبود می‌بخشد. این روش باید در طراحی واسط کاربر جزو اولویت‌ها قرار گیرد.

سایر ویجت‌های مشخص

ویجت‌های دیگری که در شکل ۱۲-۲ نشان داده شده‌اند، ویژگی‌های مشخصی دارند که نیازهای مختلف را برآورده می‌سازند.

۱۲-۳-۳- رهنمون‌هایی برای استفاده از ویجت‌های GUI

رهنمون‌های زیر را باید در موقع استفاده از ویجت‌های شکل ۱۲-۲ مدّ نظر قرار داد.

رهنمون ۱۲-۲

برای کاهش خطای انسانی، هرجا که امکان دارد ورودی انتخابی را استفاده کنید.

مثال. افراد ممکن است نام خود را اشتباه تایپ کنند. بنابراین، یك جعبه‌ی ترکیبی[۱] یا ریسنده[۲] مناسب‌تر است. اما، برای ورود کُد پستی، یك فیلد متنی، احتمالاً از نوع قالب‌بندی شده، می‌تواند گزینه مناسب‌تری باشد. زیرا پیدا کردن کد پستی مطلوب از بین هزاران کد پستی، کاری به غایت دشوار می‌باشد.

رهنمون ۱۲-۳

یك فهرست بلند را به درستی به شکل سلسله‌مراتبی از لیست‌های کوتاه‌تر تفکیك کنید تا انتخاب راحت‌تر باشد.

مثال. تقسیم کردن تمام کشورها برحسب قاره‌های مربوطه کار کاربر را در انتخاب کشور ساده می‌کند. برخی مواقع، مقدار بسیار زیادی از اطلاعات بایستی نشان داده شوند. در این‌گونه موارد، روش‌های مجردسازی و دسته‌بندی باید به‌کار روند. منظور از تجرید این است که جزییات غیرضروری را حذف کنیم تا روی یك دید سطح بالا از اطلاعات تمرکز نماییم. این کار امکان ناوبری از یك دید سطح بالا تا سطح جزییات را فراهم می‌آورد. دسته‌بندی، اطلاعات را به تعدادی مقوله[۳] و زیرمقوله تقسیم می‌کند. رهنمون‌های بعدی را باید در موقع طراحی نمایش اطلاعات مدّ نظر قرار داد.

رهنمون ۱۲-۴

وقتی اطلاعاتی که قرار است نمایش داده شوند از یك نوع هستند، از تجربه استفاده کنید.

مثال. مورد کاربرد «جستجوی برنامه‌ی بین‌المللی» در سیستم مدیریت مطالعات برون مرزی (SAMS) ممکن است تمام برنامه‌های موجود در پایگاه‌داده را برگرداند. در این حالت، به‌کار بردن تجربه باعث نمایش فهرستی از خلاصه‌های سطح بالا از برنامه‌ها شده و به کاربر امکان می‌دهد که روی یك پیوند کلیک می‌کند و جزییات برنامه را ببیند.

---

[۱] Combo Box
[۲] Spinner
[۳] Category

مقوله	ویجت	کاربرد	نوع داده	شرح
متن‌گرا	فیلد متنی فیلد متنی قالب‌بندی‌شده فیلد گذرواژه	ورودی	یک خط متن	به کاربر امکان ورود یک خط متن که تعداد مشخصی نویسه داشته باشد را می‌دهد. فیلد متنی قالب‌بندی‌شده به برنامه‌نویس اجازه می‌دهد که مجموعه‌ی نویسه‌های مجاز را نیز تعریف کند. فیلد گذرواژه شبیه فیلد متنی است با این تفاوت که متنی که تایپ می‌شود را مخفی می‌کند.
	محدوده‌ی متنی محدوده‌ی متنی قالب‌بندی‌شده	ورودی/خروجی	چند خط متن	به کاربر امکان ورود، ویرایش، یا مشاهده‌ی چندین خط متن را می‌دهد. محدوده‌ی متنی قالب‌بندی‌شده امکان متون سبک دار را نیز می‌دهد در حالی‌که محدوده‌ی متنی فقط متن خام را پشتیبانی می‌کند. متون سبک‌دار می‌تواند شامل نویسه‌هایی از فونت‌های مختلف و نیز تصاویر باشد.
انتخاب‌گرا	لیست لیست پایین‌ریز جعبه‌ی ترکیبی	ورودی	یک یا چند قلم	به کاربر امکان می‌دهد که از بین لیستی از انتخاب‌ها، انتخاب خود را انجام دهد. چندین انتخاب نیز مجاز می‌باشد. وقتی روی لیست پایین‌ریز کلیک شود، فهرستی از انتخاب‌ها را نشان می‌دهد. جعبه‌ی ترکیبی خصیصه‌هایی از لیست و فیلد متنی را ترکیب می‌کند و به کاربر امکان می‌دهد که از یک لیست انتخاب کند یا این‌که مقداری را تایپ کند.
	جعبه ی بررسی دکمه‌ی رادیویی	ورودی	بولی	به کاربر اجازه می‌دهد یک یا چند قلم را انتخاب کرده یا جواب پرسش‌های بله/خیر را بدهد. انتخاب یکی از دکمه‌های رادیویی از یک گروه به‌طور خودکار دیگر دکمه‌ها را از انتخاب در می‌آورد.
	انتخاب‌کننده‌ی فایل انتخاب‌کننده‌ی رنگ انتخاب‌کننده‌ی مکان تقویم وغیره	ورودی	فایل، قلم، یا مقدار	به کاربر اجازه می‌دهد که با مرورکردن یا کلیک کردن روی یک نقشه یا تقویم یک فایل، رنگ، مکان، یا تاریخی از تقویم را انتخاب نماید.
	ریسنده لغزنده	ورودی	انتخاب یک مورد از یک لیست مرتب یا محدوده‌ای از مقادیر	ریسنده به کاربر امکان می‌دهد که از میان لیست مرتب‌شده‌ای از عناصر یکی را انتخاب نماید، یا مستقیماً یک وروردی معتبر را تایپ کند. برخی مواقع این نسبت به جعبه‌های ترکیبی ترجیح داده می‌شود زیرا یک فهرست پایین‌ریزی نشان داده نمی‌شود که بخواهد نمایش داده‌های مهم را کورکند. لغزنده نیز برای انتخاب ورودی‌هایی از نوع پیوسته، مثل کنترل صدا یا حساسیت، مناسب است.
جدول‌گرا	جدول	ورودی/خروجی	جدول یا فرم	به کاربر امکان مشاهده، ویرایش، یا ورود داده با استفاده از یک جدول یا یک فرم را می‌دهد.
نمودارگرا	صفحه‌ی رسم	ورودی/خروجی	رسم	به کاربر امکان مشاهده یا ویرایش یک نمودار را در یک محدوده‌ی ترسیم می‌دهد.
ساختارگرا	درخت	ورودی/خروجی	سلسله مراتبی	اطلاعات به شکل یک درخت سازماندهی شده و نشان داده می‌شود تا نمایش و ناوبری را آسان سازد.
چارت (نمودار) گرا	نمودار میله‌ای نمودار دایره‌ای نمودار خطی هیستوگرام وغیره	ورودی/خروجی		اطلاعات با استفاده از نمودارهای مناسب نشان داده می‌شود تا اختلاف، توزیع، مسیر، و دیگر ویژگی‌ها را برجسته سازد.
تصویرگرا	تصویر	ورودی/خروجی	تصویری	اطلاعات با کلیک بر روی یک تصویر نشان داده می‌شود، یا این‌که اطلاعات به شکل یک تصویر نمایش داده می‌شود.

شکل ۱۲-۲: ویجت‌هایی برای ورودی کاربر و نمایش نتایج

رهنمون ۱۲-۵

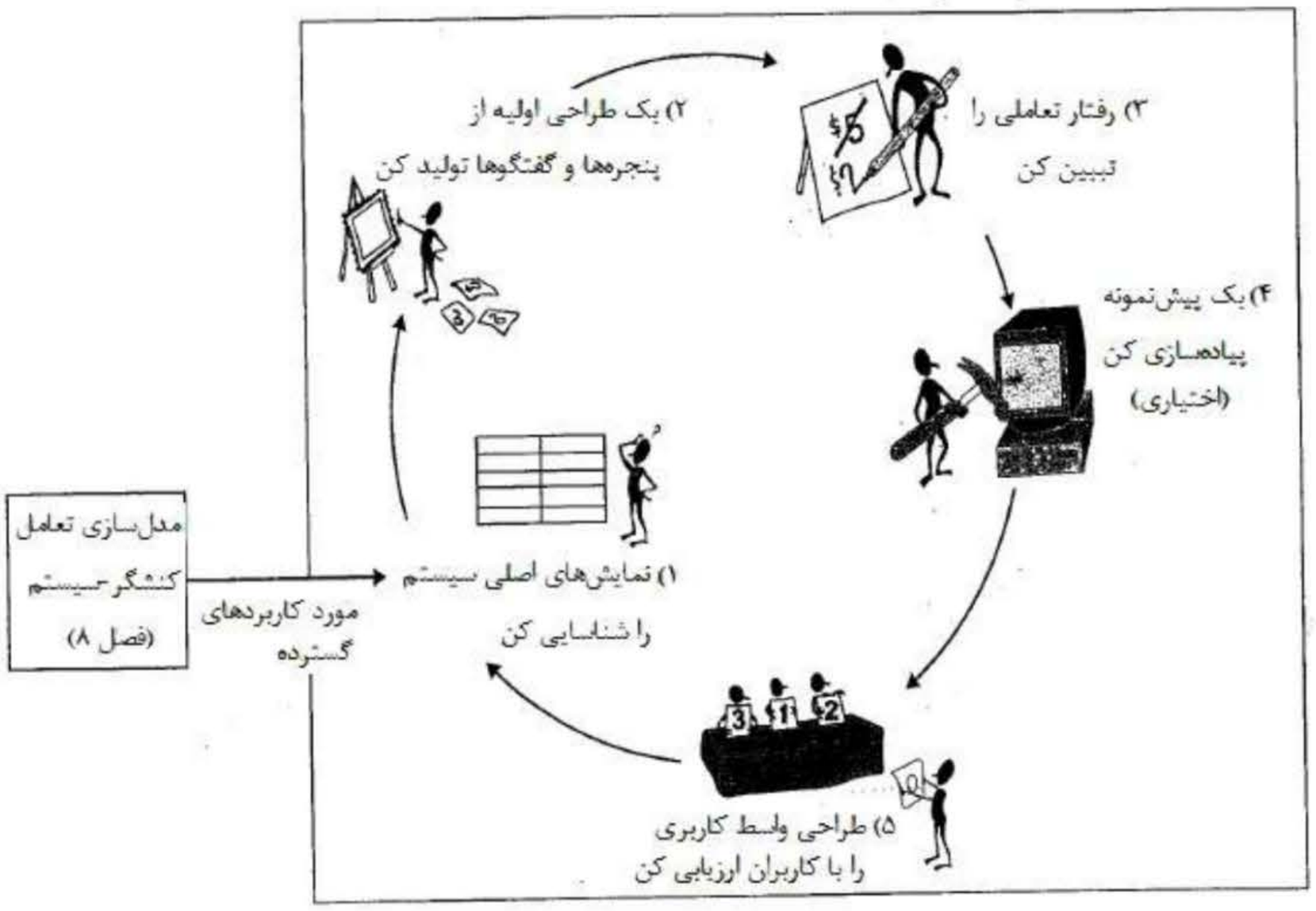
وقتی اطلاعاتی که قرار است نمایش داده شوند از انواع مختلفی هستند یا به مقوله‌های متفاوتی تعلق دارند از دسته‌بندی استفاده کنید.

یك مثال شناخته شده از دسته‌بندی، کاتالوگ کتابخانه است. کاتالوگ، مدارك کتابخانه را بر حسب مقوله‌ها در زیرمقوله‌های موضوعی مختلف دسته‌بندی می‌کند. این دسته‌بندی، معقول به نظر می‌رسد چون اعضای کتابخانه معمولاً در یك مقوله‌ی موضوعی مشخص، دنبال یك مدرك می‌گردند. دسته‌بندی، جستجو به‌دنبال اطلاعات مورد نیاز را تسهیل می‌کند.

در بیشتر موارد، در طراحی نمایش اطلاعات باید نیازهای برنامه و کاربر آن را در نظر گرفت. طراحی هزینه‌های حمل‌ونقل که در شکل ۱۲-۱ نشان داده شده است، نیازهای کسب‌وکار حمل‌ونقل خرده فروشی را مدّ نظر داشته است، بدین مفهوم که به کاربر اجازه داده می‌شود هزینه حمل و تاریخ تحویل برای هر یك از خدمات هر کدام از شرکت‌های حمل‌ونقل اصلی، را در یك پنجره مشاهده نماید. شاید تجرید و دسته‌بندی بهترین روش در این مورد نباشد زیرا کاربر ممکن است مجبور شود چندین پنجره را باز کند.

۱۲-۴ فرایند طراحی واسط کاربر

شکل ۱۲-۳ فرایند طراحی واسط کاربر را نمایش می‌دهد. ورودی این فرایند مورد کاربردهای گسترده تولید شده در افزایش جاری است و خروجی نیز طراحی واسط کاربر می‌باشد. گام‌های این فرایند در زیر ارائه شده و در بخش‌های بعدی تشریح می‌شوند.



شکل ۱۲-۳: فرایند طراحی واسط کاربر

گام ۱- شناسایی نمایش‌های اصلی سیستم
در این گام، نمایش‌های سیستم، ورودی کاربر و کنش‌های کاربر از روی مورد کاربردهای گسترده تولید شده در تکرار جاری شناسایی می‌شوند. این‌ها مبنای طراحی «ظاهر و حس» است که در دو گام بعدی انجام می‌شود.
گام ۲ ـ تولید یك طرح اولیه از پنجره‌ها و جعبه‌های گفت‌وگو
در این گام، یك طراحی چینش اولیه از پنجره‌ها و جعبه‌های گفت‌وگو که متناظر با نمایش‌های سیستم است تولید می‌شود. این گام «ظاهر» واسط کاربر را طراحی می‌کند.
گام ۳ ـ تعیین رفتار واسط
در این گام، یك نمودار حالت تولید می‌شود تا روابط ناوبری بین پنجره‌ها و جعبه‌های گفت‌وگو را مشخص کند. این گام «باطن» واسط کاربری را طراحی می‌کند.
گام ۴ – ساخت یك پیش‌نمونه از واسط کاربر
این گام اختیاری است و هدف آن تولید یك پیش‌نمونه از واسط کاربری است تا ظاهر و حس را، که در دو گام قبلی طراحی شده‌اند، نشان دهد.
گام ۵ – ارزیابی طراحی با کاربران
در این گام، طراحی واسط کاربر، و احتمالاً پیش‌نمونه، به گروهی از نمایندگان کاربر نشان داده شده و بازخورد آن‌ها دریافت می‌شود تا در بهبود طراحی استفاده شود.

۱۲-۴-۱- مطالعه‌ی موردی: طراحی واسط کاربر برای یك نرم‌افزار ویرایشگر نمودار

این مطالعه‌ی موردی عبارت است از طراحی واسط کاربر برای یك نرم‌افزار ویرایشگر نمودار حالت. ویرایشگر نمودار حالت، یك کاربرد مستقل می‌باشد که به کاربر امکان ترسیم یا ویرایش نمودار حالت را می‌دهد. برای سهولت، در این نسخه از ویرایشگر، تنها نمودارهای حالت مسطح[۱] در نظر گرفته می‌شود.

نمودار حالت مسطح نموداری است که در آن تمامی حالت‌ها اتمیک هستند- بدین معنی که شامل حالت‌های دیگری نیستند. حالت‌های مرکب، یا حالت‌هایی که شامل حالت‌های دیگر می‌شوند به‌عنوان گسترش این نرم‌افزار در نظر گرفته شده‌اند.

بخشی از نیازمندی‌های ویرایشگر نمودار حالت که منظور ما را برآورده می‌سازد به شرح زیر می‌باشد: «ویرایشگر نمودار حالت باید به کاربر اجازه دهد یک نمودار جدید ایجاد کند یا یک نمودار موجود را ویرایش کند. ویرایشگر باید به کاربر اجازه دهد حالت‌ها و انتقال‌ها را اضافه، حذف یا ویرایش نموده و نیز این اعمال را Undo و Redo نماید. ویرایشگر باید به کاربر اجازه دهد سایر اعمال ویرایشی از جمله ذخیره‌ی نمودار یا ذخیره به‌عنوان[۲] را انجام دهد.»

همان گونه که در بالا تشریح شده‌است، از روی مورد کاربردی گسترده تولید شده در گام مدل‌سازی تعامل کنشگر–سیستم از تکرار فعلی می‌توان اطلاعات مورد نیاز برای گام‌های طراحی واسط کاربر را استنتاج نمود. برای مثال، شکل ۱۲-۴ مورد کاربردی گسترده «ویرایش نمودار حالت»[۳] مربوط به ویرایشگر نمودار حالت را نشان می‌دهد. برای سادگی، اعمال حذف در این شکل نشان داده نشده‌اند.

۱۲-۴-۲- شناسایی نمایش‌های اصلی سیستم

در این گام، نمایش‌های اصلی سیستم، اطلاعات نمایش داده شده و نیز ورودی کاربر و کنش‌های کاربر مرتبط با آن‌ها از روی مورد کاربردهای گسترده تولید شده در افزایش جاری شناسایی می‌شوند. نمایش اصلی سیستم آن است که نیاز به ورودی کاربر

---
[۱] Flat
[۲] Save as
[۳] Edit State Diagram

کنشگر: کاربر ویرایشگر	سیستم: ویرایشگر نمودار حالت
	۰- سیستم پنجره‌ی اصلی ویرایشگر را نمایش می‌دهد.
۱- TUCBW کاربر ویرایشگر روی گزینه File از میله‌ی فهرست کلیک می‌کند و سپس یکی از موارد زیر را انتخاب می‌کند: ۱-۱-New Diagram، یا ۱-۲- Open Diagram، و ۱-۲-۱- نمودار را یافته و روی دکمه‌ی OK کلیک می‌کند	۲- سیستم به تناسب یکی از موارد زیر را نمایش می‌دهد: ۲-۱- یک نمودار خالی، یا ۲-۲- یک گفتگوی انتخاب نمودار حالت، و ۲-۲-۱- نمودار حالت انتخاب شده را نمایش می‌دهد.
۳- کاربر ویرایشگر یکی از اعمال ویرایشی زیر را مرتب انجام می‌دهد: ۳-۱- کاربر روی دکمه‌ی State کلیک می‌کند. ۳-۱-۱- کاربر در جایی از محدوده‌ی رسم کلیک می‌کند. ۳-۲- کاربر روی دکمه‌ی انتقال کلیک می‌کند. ۳-۲-۱- کاربر ماوس را روی حالت مبدا فشار داده، و آن‌را روی حالت مقصد کشیده و رها می‌کند. ۳-۳- کاربر روی یک حالت یا انتقال کلیک می‌کند. ۳-۳-۱- کاربر حالت یا انتقال را ویرایش کرده و روی دکمه‌ی OK یا Cancel کلیک می‌کند. ۳-۴- کاربر روی گزینه‌ی Edit از میله‌ی فهرست کلیک کرده و Undo یا Redo را انتخاب می‌کند.	۴- سیستم به صورت زیر پاسخ می‌دهد: ۴-۱- سیستم شکل اشاره‌گر را به شکل باضافه تغییر می‌دهد. ۴-۱-۱- سیستم شکل یک حالت را با نامی موقتی رسم می‌کند. ۴-۲- سیستم شکل اشاره‌گر را به شکل باضافه تغییر می‌دهد. ۴-۲-۱- سیستم شکل یک انتقال را با برچسبی موقتی از حالت مبدا به حالت مقصد رسم می‌کند. ۴-۳- سیستم یک جعبه‌ی گفتگوی ویرایش حالت/ویرایش انتقال نمایش می‌دهد. ۴-۳-۱- سیستم نمودار حالت تغییریافته یا دست‌نخورده را نمایش می‌دهد. ۴-۴- سیستم نمودار حالت که عمل قبلی undo/redo شده‌است نمایش می‌دهد.
۵- وقتی کاربر ویرایش را تمام کرد روی File از میله‌ی فهرست کلیک کرده و یکی از موارد ذیل را انتخاب می‌کند: ۵-۱- Save، یا ۵-۲- Save As. ۵-۲-۱- کاربر اطلاعات درخواست شده را تکمیل کرده و روی دکمه‌ی OK کلیک می‌کند.	۶- سیستم به صورت زیر پاسخ می‌دهد: ۶-۱- سیستم پیام "Diagram Saved" را در میله‌ی وضعیت نمایش می‌دهد. ۶-۲- سیستم یک گفتگوی Save State Diagram As... را نمایش می‌دهد.
۷- TUCEW کاربر ویرایشگر سیستم پیام "Diagram Saved" را در میله‌ی وضعیت می‌بیند	

شکل ۱۲-۴: مورد کاربردی گسترده «ویرایش نمودار حالت»

داشته باشد یا نتایج پردازش سیستم را نمایش دهد. یک پیام خطا یا جعبه‌ی تأیید، نمایش اصلی سیستم محسوب نمی‌شوند. قوانین زیر در این گام به‌کار می‌روند. نتایج در شکل ۱۲-۵ نشان داده شده‌اند.

۱. گام‌های موجود در ستون سمت راست هرکدام از مورد کاربردهای گسترده را امتحان کنید. نمایش اصلی سیستم وقتی شناسایی می‌شود که:

الف) این گام نتایج پردازشی سیستم را نمایش دهد، در این حالت، اطلاعات نمایش داده شده نیز شناسایی می‌شود؛ یا

ب) این گام از کاربر درخواست ارائه‌ی ورودی داشته باشد (متن، انتخاب، یا دیگر انواع ورودی).

۲. گام‌های موجود در ستون سمت چپ هرکدام از مورد کاربردهای گسترده را امتحان کنید تا ورودی‌های کاربر و کنش‌های کاربر که متناظر با هرکدام از نمایش‌های سیستم هستند را شناسایی کنید.

مثال ۱۲-۱

نمایش‌های سیستم، اطلاعاتی که نمایش می‌دهند، ورودی کاربر و کنش‌های کاربر برای مورد کاربردی گسترده «ویرایش نمودار حالت» نشان داده شده در شکل ۱۲-۴ را شناسایی کنید.

جواب: به‌کار بردن قوانین فوق باعث شناسایی نمایش‌های سیستم، اطلاعات نمایش داده شده، ورودی کاربر و کنش‌های کاربر می‌شود که در شکل ۱۲-۵ نشان داده شده است.

گام مورد کاربرد گسترده	نمایش سیستم	اطلاعات نمایش داده‌شده	ورودی کاربر	کنش کاربر
۰	پنجره‌ی اصلی ویرایشگر			
۲	• نمودار خالی • نمودار انتخاب‌شده • گفتگوی انتخاب	• پرونده‌ها و پوشه‌ها (استناج‌شده)	• پرونده‌ی نمودار انتخاب‌شده	• روی گزینه‌ی File از میله‌ی فهرست کلیک می‌کند و New Diagram را انتخاب می‌کند
۴	• گفتگوی ویرایش حالت • گفتگوی ویرایش انتقال	• اطلاعات حالت • اطلاعات انتقال	• اطلاعات حالت ویرایش‌شده • اطلاعات انتقال ویرایش‌شده	• روی دکمه‌ی State کلیک می‌کند • روی دکمه‌ی Transition کلیک می‌کند • روی یک حالت یا انتقال کلیک می‌کند • روی Edit از میله‌ی فهرست کلیک می‌کند و Undo یا Redo را انتخاب می‌کند • روی دکمه‌ی Ok یا Cancel کلیک می‌کند
۶	نمودار حالت را به عنوان گفتگو ذخیره می‌کند	• پیام "Diagram Saved" یا "Diagram Saved As.." در میله وضعیت	• اطلاعات مورد درخواست	• روی دکمه‌ی File از میله‌ی فهرست کلیک می‌کند و Save یا Save As • روی دکمه‌ی OK یا Cancel کلیک می‌کند (دکمه‌ی Cancel استناج شده است)

شکل ۱۲-۵: نمایش‌های اصلی سیستم برای مورد کاربردی ویرایش نمودار حالت

توجه کنید که در خیلی از موارد ممکن است مشخصات مورد کاربردی گسترده آن‌گونه که در شکل ۱۲-۴ نشان داده شده است مشروح نباشد. برای مثال، گام‌های تعامل ممکن است گزینه‌های سطح پایین را نشان ندهند. ولی، مورد کاربرد گسترده به‌هرحال نمایش سیستم را نشان می‌دهد گرچه تمام ویجت‌های سطح پایین به وضوح مشخص نشده باشند. در این گونه موارد، ویجت‌های سطح پایین از روی تجربه، پروژه‌های مشابه، یا در حین طراحی چیدمان اولیه شناسایی می‌شوند.

۱۲-۴-۳- تولید یک طراحی چیدمان اولیه

در این گام، دو فعالیت انجام می‌گیرد: (۱) به‌دست آوردن پنجره‌ها و جعبه‌های گفت‌وگو برای نمایش‌های سیستم که در گام پیشین شناسایی شده‌اند، و (۲) تولید یک طراحی چیدمان اولیه برای این پنجره‌ها و جعبه‌های گفت‌وگو. برای به‌دست آوردن پنجره‌ها و جعبه‌های گفت‌وگو از معیارهایی که برای ویجت‌های دربرگیرنده در بخش ۱۲-۳-۱ ذکر گردید استفاده می‌شود. یعنی، پنجره‌ها و جعبه‌های گفت‌وگو از روی نمایش‌های سیستم که در گام قبلی شناسایی شده‌اند برطبق معیارها به‌دست می‌آیند. توجه کنید که ممکن است برخی پنجره‌ها و جعبه‌های گفت‌وگو در گام‌های پیشین شناسایی شده یا در افزایش‌های قبلی پیاده‌سازی شده باشند. از روی اطلاعات نمایش داده شده و ورودی کاربر و کنش‌های کاربر که در مورد کاربردهای گسترده مشخص شده‌اند، ویجت‌های دربرگیرنده، در هر پنجره یا جعبه‌ی گفت‌وگو به‌دست می‌آیند.

مثال ۱۲-۲

پنجره‌ها، جعبه‌های گفت‌وگو، و ویجت‌های مربوطه را از شکل ۱۲-۵ شناسایی کنید.

جواب: نتیجه در شکل ۱۲-۶ نشان داده شده است. پنجره‌ها و جعبه‌های گفت‌وگو پیش‌تر شناسایی شده‌اند. ولی بعضی

از آن‌ها می‌توانند از یک انتخاب کننده‌ی پرونده استفاده کنند. از یک صفحه رسم نمودار جهت نمایش نمودار حالت استفاده شده است.

پنجره/گفتگو	مولفه‌ی GUI/ویجت
پنجره‌ی اصلی ویرایشگر	صفحه‌ی رسم نمودار (برای نمودار خالی و نمودار انتخاب شده) میله‌ی وضعیت میله‌ی فهرست File (New Diagram/Open Diagram/Save/Save As) Edit (Undo/Redo) دکمه‌ها: State, Transition, Pointer (inferred)
انتخاب‌گر پرونده‌ی نمودار حالت	مرورگر پرونده دکمه‌ها: OK, Cancel
گفتگوی ویرایش حالت	فیلدهای متنی نام حالت // سایر فیلدهای متنی برای صفت‌های قابل ویرایش حالت که در مدل دامنه نشان داده شده است محدوده‌های متنی شرط حالت ....
گفتگوی انتقال حالت	فیلدهای متنی نام انتقال // سایر فیلدهای متنی برای صفت‌های قابل ویرایش انتقال که در مدل دامنه نشان داده شده است محدوده‌های متنی کد انتقال ....
انتخاب‌گر ذخیره‌ی نمودار حالت	مرورگر پرونده دکمه‌ها: Save, Cancel

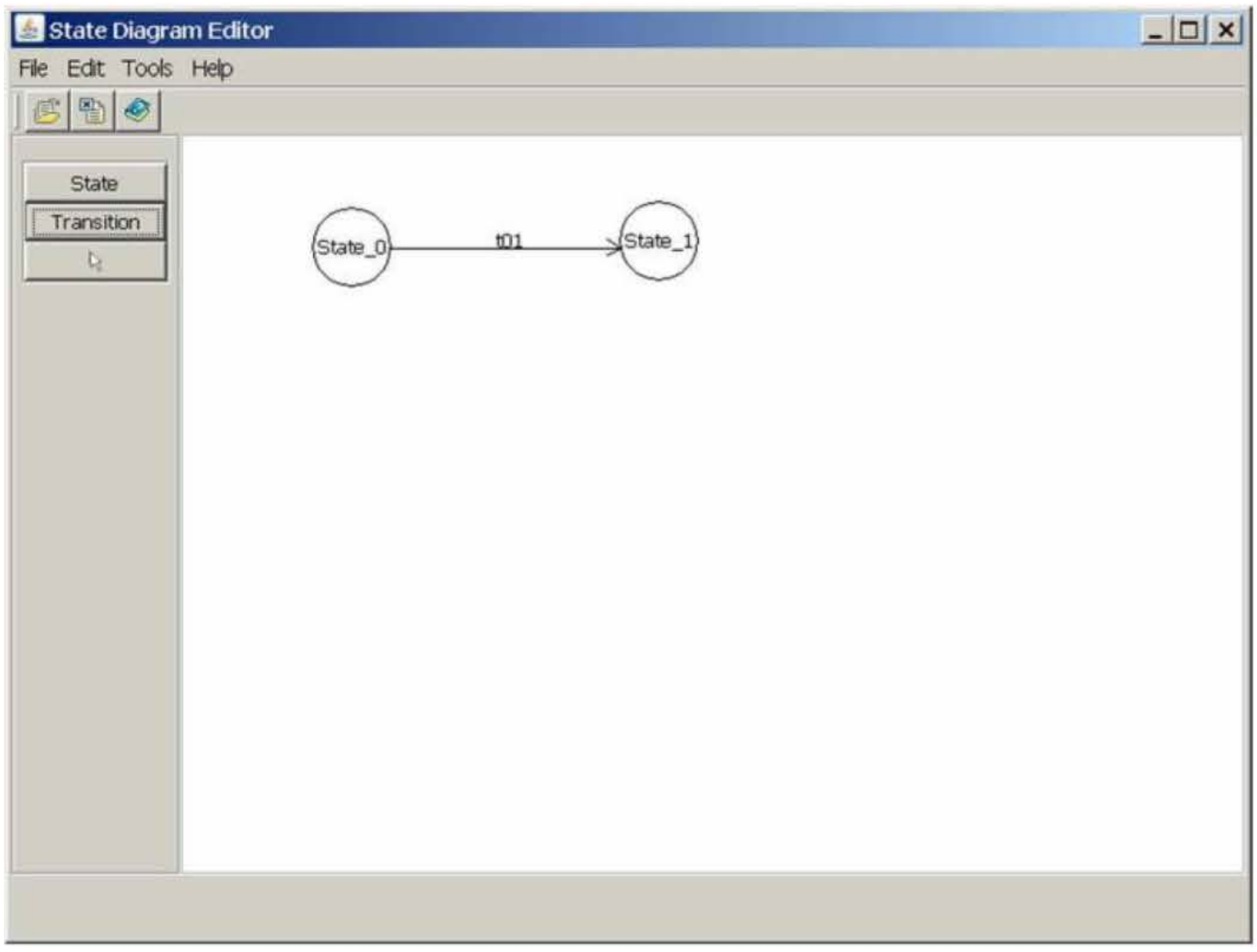
شکل ۱۲-۶: نمایش‌های اصلی سیستم برای مورد کاربردی ویرایش نمودار حالت

دو جعبه‌ی گفت‌وگوی ویرایش استفاده می‌شوند؛ یکی برای ویرایش یک حالت و دیگری برای ویرایش یک انتقال. این جعبه‌های گفت‌وگو دربرگیرنده‌ی فیلدهای متنی و محدوده‌های متنی هستند که به کاربر امکان ویرایش صفت‌های حالت و انتقال را می‌دهند. صفت‌های حالت و انتقال یا از روی مورد کاربردهای گسترده یا از روی مدل دامنه پیدا می‌شوند. جواب نشان داده شده در شکل ۱۲-۶ فرض می‌کند که هر حالت دارای یک نام و یک شرط بوده و هر انتقال نیز دارای نام و کد انتقال می‌باشد. نوع صفت به‌منظور جستجوی ویجت از شکل ۱۲-۲ استفاده شده است.

فعالیت بعدی ایجاد یک طراحی چیدمان اولیه برای پنجره‌ها و جعبه‌های گفت‌وگو می‌باشد. این یک فعالیت خلاقانه است که بستگی به کاربرد در حال ساخت دارد. از روی تجربه می‌توان گفت در مواردی که چیدمان استانداردی موجود باشد طراحی چیدمان باید از آن‌چه به‌طور متداول در صنعت یا دامنه‌ی کاربردی استفاده می‌شود، تبعیت کند. این کار منحنی یادگیری [1] کاربر را کاهش می‌دهد. برای مثال، طراحی‌های چیدمان پذیرفته شده‌ای برای واسط کاربر IDEها و ویرایشگرهای مدرک وجود دارند. برای برخی کاربردهای دیگر، ظاهر و حس سفارشی مورد نیاز است که باید طبیعت کاربرد را نیز مدنظر قرار دهد. طراحی نشان داده شده در شکل ۱۲-۱ یک مثال است.

[1] Learning Curve

طراحی اولیه‌ای که در این گام تولید شده است می‌تواند مجموعه‌ای از ترسیم‌ها روی قطعه‌ای کاغذ باشد که با استفاده از یک ابزار نمایش اسلاید یا واژه‌پرداز تولید شده و با استفاده از یک ابزار پیش‌نمونه‌سازی یا با زبان پیاده‌سازی مورد نظر، پیاده‌سازی شده باشد. شکل ۱۲-۷ پیاده‌سازی Java از طراحی چیدمان مربوط به پنجره‌ی اصلی ویرایشگر را نشان می‌دهد. چنان‌چه کاربر روی یک حالت یا انتقال دو-کلیک کند، یک جعبه‌ی گفت‌وگو ظاهر می‌شود که به کاربر امکان ویرایش آن حالت یا انتقال را می‌دهد. شکل ۱۲-۸ جعبه‌ی ویرایش حالتی را نشان می‌دهد که وقتی کاربر روی حالت State-0 دو-کلیک کرده باشد، ظاهر شده است.



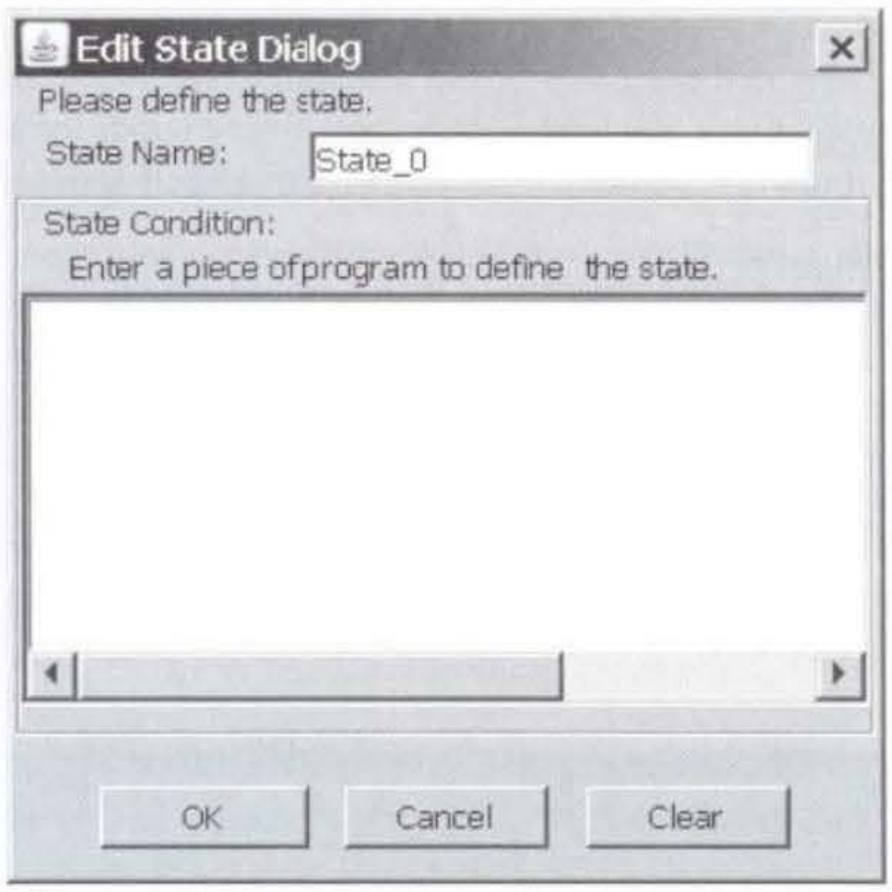
شکل ۱۲-۷: طراحی پنجره‌ی اصلی برای یک ویرایشگر نمودار حالت

۱۲-۴-۴- تبیین رفتار تعاملی

در این گام رفتار تعاملی واسط کاربر طراحی شده و با استفاده از یک نمودار حالت تبیین می‌شود. حالت‌های نمودار بیان‌گر پنجره‌ها و جعبه‌های گفت‌وگو و انتقال‌ها بیان‌گر ورودی کاربر و کنش‌های کاربر هستند که باعث تغییر از یک پنجره یا جعبه‌ی گفت‌وگو به دیگری می شوند.

برای مثال، شکل ۱۲-۹ یک نمودار حالت ناقص را برای مورد کاربرد گسترده شکل ۱۲-۴ نشان می‌دهد که در آن، پیکانی که سر آن نقطه دارد به حالت اولیه اشاره می‌کند. حالت‌ها و انتقال‌های اضافی از سایر مورد کاربردهای گسترده را می‌توان در نمودار حالت آورد. مهم این است که مطمئن شویم بین نمودار حالت و مورد کاربردهای گسترده سازگاری وجود دارد. یعنی، مورد کاربردهای گسترده و نمودار حالت هر دو یک رفتار را توصیف کنند.

همان‌گونه که در شکل ۱۲-۹ نشان داده شده است، به حالت‌ها می‌توان پنجره‌ها یا جعبه‌های گفت‌وگویی که نشان می‌دهند را افزود. این روش چند مزیت به شرح زیر دارد:



شکل ۱۲-۸: طراحی یک جعبه‌ی گفت‌وگو برای ویرایش یک حالت

- به برنامه‌نویس کمک می‌کند انتقال از یک نمایش به نمایش دیگر را تصور کند.
- برای بررسی درستی نمودار حالت مفید است. یعنی این‌که آیا نمودار حالت به درستی انتقال از یک نمایش به دیگری را برطبق ورودی کاربر و کنش کاربر مدل می‌کند؟
- برای بررسی کامل بودن مفید است. یعنی مطمئن شویم که به ازای هر کنش کاربر مهمی که می‌تواند در آن حالت رخ دهد، یک انتقال خروجی وجود دارد. برای مثال، جعبه‌ی گفت‌وگوی ویرایش حالت[۱] سه دکمه دارد، بنابراین، حالت مربوطه نیز باید سه انتقال خروجی داشته باشد. حالت شکل ۱۲-۹ نیز در واقع سه انتقال خروجی متناظر با این سه کنش کاربر دارد.
- می‌توان از ابزارها استفاده کرد یا ابزارهایی ساخت که یک پیش‌نمونه‌ی تصویر متحرک[۲] تولید کند تا رفتار GUI را نشان دهد.

۱۲-۴-۵- ساخت یک پیش‌نمونه

کاربران ممکن است قضاوت‌های متفاوتی از واسط کاربر داشته باشند. این قضاوت‌ها به عوامل مختلفی بستگی دارند، از جمله ترجیحات شخصی، تجربه‌ی گذشته، مسئولیت‌ها، و پیش‌زمینه‌ی فرهنگی. بنابراین، درگیر نمودن کاربران در ارزیابی طراحی یک واسط کاربر امر مهمی است. پیش‌نمونه‌سازی یک راه مؤثر برای درگیر نمودن کاربران در ارزیابی واسط کاربر است، زیرا با این کار هدف «شنیدن کی بود مانند دیدن» برآورده می‌شود.

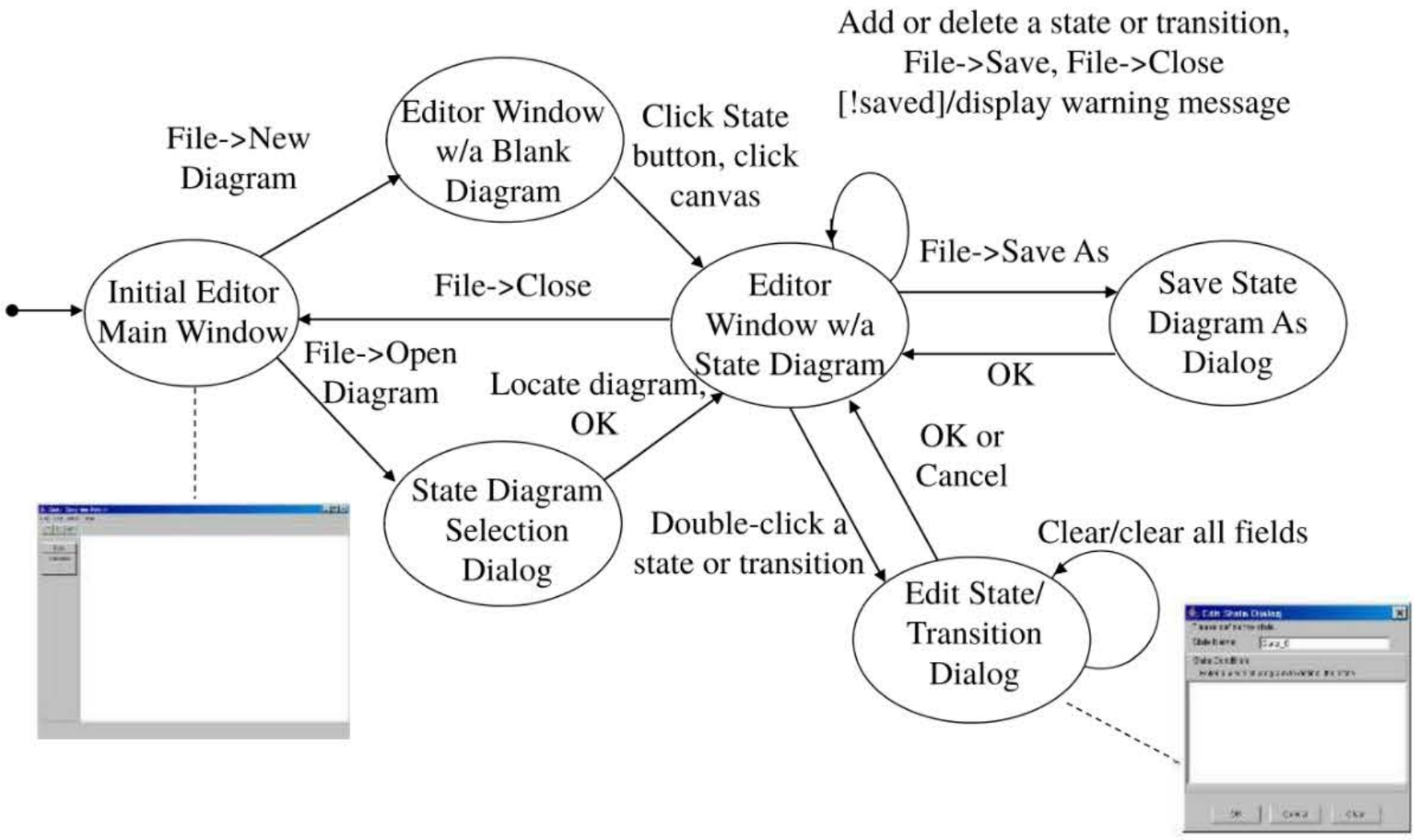
فایده‌ی پیش‌نمونه‌سازی مثل استفاده از خانه‌های مدل در بازار مسکن است. بدون خانه‌ی مدل، برای خیلی از خریداران خانه بسیار دشوار خواهد بود که بفهمند از یک نقشه‌ی طراحی چه انتظاری باید داشته باشند. خانه‌ی مدل به‌طور مؤثری ظاهر و حس خانه را به خریداران بالقوه نشان داده و امکان بیان درخواست تغییرات را برای آن‌ها تسهیل می‌کند.

روش‌های مختلف پیش‌نمونه‌سازی برای واسط کاربر را می‌توان به‌صورت زیر دسته‌بندی نمود.

---

[۱] Edit State Dialog

[۲] Animated



شکل ۱۲-۹: یک نمودار حالت واسط کاربری که تقریباً کامل شده است.

۱. روش‌های ایستا پیش‌نمونه‌های غیر قابل اجرا تولید می‌کنند. یعنی، طراحی چیدمان پنجره‌ها و جعبه‌های گفت‌وگو رو کاغذ، با استفاده از ابزار نمایش اسلاید، یا با کمک یک واژه‌پرداز به‌تصویر کشیده می‌شود. ترسیم‌ها به کاربران نمایش داده می‌شوند درحالی‌که طراح کارکردی و رفتار سیستم را توضیح می‌دهد.

پیش‌نمونه‌های ایستا شبیه مدل‌های معماری در صنعت ساختمان هستند. این مدل‌های فیزیکی مقیاس کوچک در بیش‌تر پروژه‌های ساختمانی ایجاد می‌شوند تا ایده‌های طراحی بین افراد ردوبدل شود. ساخت پیش‌نمونه‌های ایستا و استفاده از آن‌ها برای ارتباط طراحی چیدمان نسبتا ارزان است، ولی برای نشان دادن جنبه‌ی رفتاری سیستم چندان مؤثر نیست. عیب دیگر هم این است که پیش‌نمونه را نمی‌توان کامل کرد تا به سیستم نهایی منجر شود.

۲. روش‌های پویا با استفاده از یک ابزار پیش‌نمونه‌سازی، یا پیاده‌سازی با استفاده از یک زبان اسکریپتی یا یک زبان پیاده‌سازی، پیش‌نمونه‌های اجرایی تولید می‌کنند. پیش‌نمونه‌های پویا می‌توانند رفتار تعاملی در سطوح مختلف پیچیدگی، از رفتار متحرک ساده تا رفتار کاملا پیاده‌سازی شده را به نمایش بگذارند.

خصوصیات اجرایی تجربه‌ی تعاملی واقعی‌تری را فراهم می‌کنند. بعضی روش‌ها به کاربران اجازه می‌دهند با پیش‌نمونه امتحان کنند. این باعث می‌شود نتایج ارزیابی مفیدتر باشند. اگر پیش‌نمونه به زبان مقصد پیاده‌سازی شود، می‌توان پیش‌نمونه را به محصول نهایی تکمیل نمود، ولی طراحی و پیاده‌سازی پیش‌نمونه‌های پویا به زبان و تلاش بیش‌تری نیاز دارند. ضمناً اعمال تغییرات اساسی که در مراحل اولیه‌ی توسعه‌ی پیش‌نمونه، به‌خصوص برای سیستم‌های جدید، اتفاق می‌افتد نیز زمان‌بر خواهدبود.

۳. روش‌های ترکیبی پیش‌نمونه‌های ایستا را در مرحله‌ی اول توسعه‌ی پیش‌نمونه ساخته و پس از این‌که کاربران کم و بیش از ظاهر و حس پیش‌نمونه‌های ایستا رضایت داشتند، به سراغ پیش‌نمونه‌های پویا می‌روند.

نمودار حالتی که در گام آخر ساخته می‌شود پیاده‌سازی پیش‌نمونه را بسیار ساده می‌کند زیرا پیش‌نمونه نمایش سیستم مرتبط با هر حالت را نشان می‌دهد. وقتی یک انتقال رخ دهد، وارد حالت مقصد شده و نمایش سیستم مربوط به آن

حالت را نشان می‌دهد.

۱۲-۴-۶- ارزیابی طراحی واسط کاربر با کاربران

در این گام، طراحی واسط کاربر همراه با کاربران مورد ارزیابی قرار می‌گیرد. این درواقع اعمال اصول چابکی-درگیر نمودن کاربران در فرایند توسعه- است. طراحی واسط کاربر باید عوامل زیادی را درنظر گیرد، که برخی از آن‌ها فقط با کار کردن با کاربران مشخص می‌شوند. عواملی که بر پذیرش یک طراحی واسط کاربر تاثیر می‌گذارند عبارتند از:

- واسط‌های کاربری سیستم‌های موجود. کاربران به دلیل کار کردن با سیستم‌های موجود در سال‌های متمادی به آن‌ها عادت کرده‌اند. لذا ممکن است مایل به تغییر در واسط کاربر جدید نباشند.
- طبعیت کسب‌وکار. برخی کاربردها، از قبیل کسب‌وکار حمل بار که پیش‌تر در مورد آن بحث شد، روی سرعت پردازش تأکید دارند. در ایالات متحده، روز مادر، روز عشاق، و روز کریسمس شلوغ‌ترین فصول حمل بار در سال هستند. پنجاه درصد حجم حمل بارهای یک سال در طول فصل کریسمس اتفاق می‌افتد و سه روز آخر آن شلوغ‌ترین ایام هستند. برای فروشگاه‌های حمل بارهای خرده، سرعت امری بحرانی است زیرا اگر حمل بسته طول بکشد مشتری به سراغ فروشگاه دیگری خواهدرفت تا بسته‌ی خود را ارسال نماید. برخی دیگر از کاربردها روی دقت، قابلیت اطمینان و امنیت تاکید دارند. برای مثال، سیستم‌های مدیریت رکوردهای پزشکی باید درخصوص دقت و امنیت رکوردهای پزشکی اطمینان بدهند. این سیستم‌ها همچنین باید تضمین دهند که پیغام‌های هشدار به‌درستی و به‌موقع زمان‌بندی شده و تحویل دکترها و پرستاران می‌گردند. برخی کاربردها بر روی دیگر جنبه‌ها تاکید دارند. تنها کاربران هستند که می‌توانند بگویند آیا واسط کاربر نیازهای ایشان را برآورده می‌سازد یا خیر.
- روان‌شناسی کاربران. روان‌شناسی کاربران ترکیبی از چندین عامل متفاوت است، از قبیل سنت، فرهنگ اجتماعی، زمینه‌ی تحصیلی، تجربه‌ی کاری، و جنسیت. به‌عنوان مثال، رنگ قرمز یک اثر شدید داشته و در ایالات متحده به‌عنوان رنگ هشدار به‌کار می‌رود. اگر قیمت یک سهام، اوراق بهادار، یا ارز تجاری کم‌تر از روز قبل باشد، قیمت مربوطه به رنگ قرمز نشان داده می‌شود. اما، قرمز در آسیا، به‌خصوص در چین، ژاپن، کره و آسیای جنوب شرقی رنگ خوش‌بختی است. این رنگ برای قیمت‌هایی که در بازار سهام افزایش داشته‌اند به کار می‌رود. این‌گونه موارد، مسائل فرهنگی شناخته‌شده هستند و به‌سادگی قابل تشخیص هستند. تنها کاربران هستند که می‌توانند تعیین کنند که یک واسط کاربر با مسائل مختص یک سازمان ناسازگاری دارد یا آن‌ها را نقض می‌کند. به‌عنوان مثال، کلینیک‌ها و بیمارستان‌ها از رنگ‌های مختلفی برای مشخص کردن رکوردهای پزشکی استفاده می‌کنند. اگر یک طراحی واسط کاربری چنین امکانی را فراهم آورد، دراین‌صورت استفاده از رنگ باید با مسائل موجود سازگار باشد. این تنها توسط کاربران قابل تأیید است.
- ترجیحات شخصی کاربران. کاربران مختلف ترجیحات مختلفی دارند، که بر پذیرش طراحی واسط کاربر تاثیر می‌گذارد. مجدداً تنها کاربرانند که می‌توانند این را تأیید کنند.

بحث بالا نشان می‌دهد که این مهم است که ارزیابی طراحی واسط کاربری همراه با کاربران صورت پذیرد و بازخورد ایشان جمع‌آوری و تحلیل گردد و براساس بازخوردها طراحی اصلاح شود. برخی از روش‌های ارزیابی طراحی واسط با کاربران به شرح زیر است.

۱. ارائه‌ی واسط کاربری. این روش ارزیابی، با پیش‌نمونه‌های ایستا استفاده می‌شود. برنامه‌نویس از حالت اولیه شروع می‌کند و با نشان دادن پنجره‌ها و جعبه‌های گفت‌وگو و نیز دنبال کردن انتقال‌های نمودار حالت کارکردی سیستم را

ارائه می‌نماید. کاربران تشویق می‌شوند که سؤال پرسیده، نظر داده، و درخواست تغییر دهند. این نکات در ادامه بحث خواهندشد. این فرایند تکراری است و پیمایش عمق-اول[۱] انجام می‌شود تا این‌که تمام انتقال‌های نمودار حالت پیمایش شوند.

۲. نمایش واسط کاربری. این روش هم با پیش‌نمونه‌های ایستا و هم با پیش‌نمونه‌های پویا استفاده می‌شود. در این روش برنامه‌نویس خصیصه‌های سیستم را با استفاده از یک پیش‌نمونه توضیح و نمایش می‌دهد. بازخورد کاربر جمع‌آوری و برای بهبود طراحی استفاده می‌گردد.

۳. آزمایش واسط کاربری. این روش برای ارزیابی پیش‌نمونه‌های پویا استفاده می‌شود. برنامه‌نویس پیش‌نمونه‌ها را نمایش داده و اجازه می‌دهد کاربر برای مدت زمانی پیش‌نمونه را امتحان کند. سپس کاربران بازخوردها و درخواست‌های تغییر را می‌دهند. این‌ها متعاقباً توسط تیم توسعه مورد بررسی قرار می‌گیرند. دستورکارهای آزمایشگاه که به خوبی نوشته شده باشد، جلسات آموزشی کاربران، یا نیروی پشتیبان کاربران می‌تواند کارایی این روش را بهتر کند.

۴. جلسه‌ی بازبینی واسط کاربری. در این روش خبرگان طراحی واسط کاربری، خبرگان دامنه، نمایندگان کاربر، و برنامه‌نویس‌ها شرکت دارند. قبل از جلسه‌ی بازبینی، حاضران بایستی در یک ارائه‌ی واسط کاربری شرکت نموده، با پیش‌نمونه کار آزمایشی انجام داده، یا طراحی چیدمان و رفتار را بازبینی کرده باشند. حاضران بایستی به فهرستی از سؤال‌های بازبینی پاسخ دهند (بخش ۱۲-۴-۷ را ببینید). در جلسه، حاضران نتایج بازبینی‌شان را تبادل کرده و روش‌های ممکن برای بهبود طراحی را بحث می‌کنند. موارد قابل انجام شناسایی و به برنامه‌نویسان مشخص واگذار می‌شود و ایشان موظفند درخصوص چگونگی برطرف نمودن این موارد گزارش دهند.

۵. تحقیق در مورد واسط کاربری. در این روش از یک پرسش‌نامه‌ی تحقیق استفاده می‌شود تا بازخورد کاربران، خبرگان طراحی واسط کاربری، و خبرگان دامنه جمع‌آوری گردد. پرسش‌نامه باید خلاصه بوده و بر روی مسائل واسط کاربری تمرکز داشته‌باشد. سؤال‌های بازبینی و تحقیق باید روی به‌دست اوردن بازخورد در مورد مسائل زیر تمرکز داشته باشد: (۱) سادگی فراگیری و استفاده از سیستم، (۲) ظاهر کلی واسط، (۳) بازدهی و مؤثربودن انجام وظایف کسب‌وکار توسط سیستم، (۴) استفاده از زبان و واژگان، (۵) مدیریت خطا و پیغام‌های خطا (بدین مفهوم که آیا مدیریت خطا مؤثر است و آیا پیغام‌های خطا به کاربران کمکی می‌کنند؟)، و (۶) تسهیلات کمک و مستندسازی (مانند آموزش‌ها و جزوات راهنمای کاربران). حاضران باید پاسخ‌های خود را بر طبق یک مقیاس نسبی مانند «خیلی خوب»، «خوب»، «متوسط»، «پایین‌تر از متوسط»، و «ضعیف» انتخاب نمایند.

۱۲-۴-۷- فهرست بررسی بازبینی طراحی واسط کاربر

۱. آیا در کل، طراحی واسط کاربر با طراحی واسط کاربری استاندارد در صنعت سازگار است؟
۲. آیا هرکدام از پنجره‌ها و جعبه‌های گفت‌وگو ظاهری ساده دارد که به کاربر اجازه دهد روی تم[۲] اصلی تعامل تمرکز نماید؟
۳. ایا رفتار واسط کاربر (مثلاً نمودار حالت تولید شده در گام ۳) با رفتار مشخص شده در مورد کاربرد گسترده سازگار است؟
۴. آیا طراحی واسط کاربر یادگیری و استفاده از سیستم را ساده می‌کند؟
۵. آیا ویجت‌های GUI به‌درستی و به‌طور سازگار استفاده شده‌اند، و آیا با استانداردها یا عرف صنعت تطابق دارند؟
۶. آیا برچسب‌گذاری ویجت‌های GUI و متون توصیفی از دید یک کاربر جدید دقیق و اطلاع‌دهنده هستند؟
۷. آیا شمایل‌ها[۳] به‌درستی استفاده شده و راجع به کارکردشان اطلاع‌دهنده هستند؟

---

[۱] Depth First
[۲] theme
[۳] Icons

۸. آیا خروجی سیستم از نظر منطقی و به‌درستی ساخت‌یافته و سازمان‌دهی شده است که درک آن راحت باشد؟
۹. آیا پیغام‌ها واضح هستند و به‌وضوح نمایش داده می شوند؟
۱۰. آیا پیغام‌های خطا عاری از جزییات پیاده‌سازی بوده و برای کاربران اطلاع دهنده و مفید هستند؟
۱۱. آیا رنگ، چشمک‌زدن، و دیگر روش‌های واسط کاربر به‌دقت، به‌درستی، و به‌شکل محدود استفاده شده‌اند که باعث ازدست رفتن تمرکز نشوند؟

۱۲-۵ طراحی قابلیت‌های پشتیبانی از کاربران

قابلیت‌های پشتیبانی از کاربران عبارتند از: مستندات برخط، کمک وابسته به زمینه، پیغام‌های خطا و ترمیم. مستندات برخط حداقل باید شامل یک راهنمای کاربر[۱] و یک دفترچه راهنمای کاربر[۲] باشد. راهنمای کاربر باید توصیف سطح بالایی از کارکرد سیستم و این‌که سیستم برای طبقه‌های مختلف از کاربران چه می‌تواند انجام دهد، ارائه نماید. توصیف کارکرد هرکدام از مورد کاربردها و این‌که چگونه مورد کاربردها را می‌توان با یکدیگر استفاده کرد تا اهداف کسب‌وکاری مشخصی ارضا شوند باید در این دفترچه درج شده باشند. دفترچه راهنمای کاربر را می‌توان تقریباً به‌سادگی از روی مشخصات مورد کاربردهای گسترده تولید کرد. بدین شکل که، گام‌های مورد کاربردهای گسترده به‌صورت یک متن تک ستونی تبدیل می‌شود. عکس صفحه از پنجره‌ها و جعبه‌های گفت‌وگو نیز درج می‌شوند تا پاسخ سیستم را نشان دهند.

یک طراحی کاربر پسند از مستندات برخط باید به کاربر امکان جستجوی اطلاعات مورد نیاز را به‌سادگی و به‌سرعت ازطریق مرور فهرست مندرجات، شاخص‌ها[۳]، ناوبری از یک مکان به مکان دیگر، و جستجو به‌دنبال موضوع موردنظر بدهد. این خصیصه‌ها را می‌توان به‌آسانی با استفاده از صفحات وب به‌دست آورد. راهنمای وابسته به زمینه، اطلاعات راهنمایی که مربوط به زمینه باشد را فراهم می‌آورد. برای مثال، کاربر می‌تواند روی محدوده‌ی متنی شرط حالت در شکل ۱۲-۸ کلیک راست زده و از فهرست بازشو گزینه‌ی کمک[۴] را انتخاب نماید. سیستم باید اطلاعات کمک که مختص به ویرایش یک شرط حالت است را نمایش دهد. اگر اطلاعات کمک در بازه‌ی محدوده‌ی متنی[۵] یافت نشود، سیستم باید اطلاعات کمک درباره‌ی Edit State Dialog یا State Diagram Editor را نمایش دهد.

زیر سیستمی که باید کمک وابسته به زمینه ارائه دهد شامل سلسله مراتبی از مدیریت‌های درخواست کمک است که به‌صورت درخت یا شبکه[۶] سازمان‌دهی شده‌اند. هر مدیر، مسئول ارائه‌ی اطلاعات کمک برای یک موضوع خاص می‌باشد و فقط گره‌ی والد خودش را به‌عنوان گره بعدی می‌شناسد. ریشه‌ی درخت، اطلاعاتی درخصوص سیستم نرم‌افزاری ارائه می‌نماید و برای آن بعدی وجود ندارد. گره‌های برگ درخت، اطلاعاتی درباره‌ی خاص‌ترین موضوعات مانند محدوده‌ی متنی[۷] ارائه می‌کنند. در صورت لزوم، مدیر، اطلاعات کمک را چنان‌چه داشته باشد برمی‌گرداند؛ درغیر این‌صورت، درخواست را به گره‌ی بعدی می‌فرستد. بدین‌ترتیب، مرتبط‌ترین اطلاعات کمک توسط زنجیره‌ای از مدیران ارائه خواهد شد. الگوی زنجیره‌ی مسئولیت‌ها[۸] که در بخش ۴ کتاب (اعمال الگوهای خاص موقعیت) معرفی شده‌است، پیاده‌سازی این قابلیت کمک وابسته به زمینه را تسهیل می‌کند.

طراحی پیغام‌های خطا وظیفه‌ی مهم دیگر طراحی واسط کاربر است. بسیاری از مهندسان نرم‌افزار پیغام خطا را با پیغام اشکال‌زدایی اشتباه می‌گیرند. پیغام‌های خطا مانند «خطایی در تابع اسکریپت ()foo اتفاق افتاده است» یا «عمل غیرمجاز در خط ۴۲۹» برای برنامه‌نویس مفیدند که بتواند اشکال را پیدا کند ولی برای کاربر نهایی به‌هیچ‌وجه اطلاع‌دهنده نیستند.

---

[۱] User Guide
[۲] User Manual
[۳] Indices
[۴] Help
[۵] Text Area
[۶] Lattice
[۷] State Condition
[۸] Chain of Responsibility

پیغام‌های خطا باید کاربرگرا باشند نه برنامه‌نویس‌گرا. این پیغام‌ها باید به کاربر بگوید چه اشتباهی شده و کاربر برای این‌که از آن رد شود چه می‌تواند بکند.

یک قابلیت مهم یک سیستم نرم‌افزاری امکان ترمیم از یک وضعیت نامطلوب است. عمومی‌ترین و دست‌یافتنی‌ترین قابلیت ترمیم، امکان undo و redo کردن برخی اعمال خاص است. این‌کار را با استفاده از الگوی فرمان[۱] می‌توان انجام داد. یک قابلیت پیچیده‌تر در ترمیم امکان پشتیبان‌گیری خودکار[۲] از مجموع داده‌های مهم که توسط کاربر انتخاب می‌شوند می‌باشد. این کار باعث می‌شود که در موقع نیاز، داده‌ها را بتوان بازیابی نمود. این قابلیت را می‌توان با استفاده از الگوی یادآور انجام داد. درنهایت، ممکن است سیستم را با نرم‌افزار تحمل‌پذیر خطا طراحی نمود که قابلیت ترمیم خودکار از یک وضعیت نامطلوب که به دلایل مختلفی مانند استثناها، عملیات خطادار، یا خطای نرم‌افزاری، وارد آن‌ها شده‌است، داشته باشد.

۱۲-۶ رهنمون‌هایی برای طراحی واسط کاربر

در یکی از بخش‌های قبلی رهنمون‌هایی برای استفاده از انواع متفاوتی از ویجت‌ها توصیف گردیدند. این بخش، اصول و رهنمون‌های طراحی واسط کاربر را معرفی می‌نماید. این‌ها قوانین راهنما هستند. تفاوت در این است که اصول، کلی‌تر بوده و طراح باید آن‌ها را به درستی تفسیر و به‌کار ببرد. رهنمون‌ها خاص‌تر بوده و راهنمایی‌های مشخص‌تری درمورد چگونگی انجام یک کار مشخص ارائه می‌دهند. بدین‌ترتیب می‌توان گفت که یک رهنمون یک اصل سطح پایین است. برای مثال، «ساده و احمقانه بگیر» یک اصل طراحی است زیرا راهنمایی‌های عمومی ارائه می‌دهد و این طراح است که باید آن را در زمینه‌های مختلف به‌درستی اعمال نماید. در مقابل، جمله‌ی «پنجره‌ها، جعبه‌های گفت‌وگو، و صفحات وب باید ظاهر و حسی ساده و آسان برای یادگیری داشته باشند» یک رهنمون است چون راهنمایی خاصی درمورد طراحی ظاهر و حس ارائه می‌کند.

اصول طراحی نرم‌افزار از قبیل طراحی برای تغییر، جداسازی دغدغه‌ها، نهان‌کردن اطلاعات، چسبندگی زیاد، پیوستگی کم، و ساده و احمقانه بگیر، درمورد طراحی واسط کاربر قابل استفاده‌اند. این اصول می‌گویند که اشیای GUI باید از اشیای کسب‌وکار مجزا باشند. لذا این مشکل که تغییر در یکی از آن‌ها روی دیگری تاثیر داشته باشد رخ نمی‌دهد. مثالی که این اصول طراحی را نقض می‌کند، یک شئ GUI است که هم مسئول ارائه‌ی ویجت‌ها است و هم مدیریت رویه‌های کنش. این قبیل چسبندگی محکم، تغییر نرم‌افزار جهت پاسخ‌گویی به تغییر در نیازمندی‌ها را مشکل می‌سازد. این‌که چگونه باید GUI را از اشیای کسب‌وکار مجزا نمود، در فصل ۱۰ (اعمال الگوهای واگذاری مسئولیت) به‌تفصیل بیان شد.

به‌عنوان مثال دیگر، اصل جداسازی دغدغه‌ها پیشنهاد می‌کند که دکمه‌ها و اقلام فهرست‌ها در یک واسط کاربر باید برحسب دغدغه‌های مختلف گروه‌بندی شوند. مثلاً، متداول است که اقلام فهرست درون دستورهایی مثل File، Edit، View، Tools، Help، و دیگر دغدغه‌ها دسته‌بندی شوند. اصل «ساده و احمقانه بگیر» به ما می‌گوید که یادگیری و استفاده از واسط کاربری باید آسان باشد. از این لحاظ رهنمون‌های مختلفی باید اتخاذ گردند.

رهنمون ۱۲-۶

طراحی واسط کاربر باید کاربرمحور باشد.

این بدان معنی است که واسط‌های کاربری باید کاربرپسند باشند. به‌عبارت دیگر ظاهر و حس آن‌ها برای کاربر آشنا بوده و از واژگان و زبان کاربر استفاده کرده باشند، یادگیری، فهم و استفاده از آن‌ها آسان باشد، کاربر درجریان باشد که چه کاری در حال انجام است، امکان Undo و Redo فراهم شده باشد، و کاربران در سطوح مختلف نیز پشتیبانی شوند.

---

[۱] Command Pattern

[۲] Automatic Backup

رهنمون ۱۲-۷

واسط کاربر باید سازگار باشد.

منظور رعایت سازگاری در ظاهر و حس پنجره‌ها، گفت‌وگوها، ویجت‌ها، و نیز سازگاری در استفاده از واژگان، سازگاری در فعالیت‌های پردازش اطلاعات و نتایج مربوطه، و سازگاری در سرتاسر سیستم است. برای مثال، گرچه بیش‌تر از یک روش برای انجام عمل کپی و چسباندن[۱] وجود دارد، ولی درخواست‌ها باید به‌طرز یکسان پردازش شوند و نتایج مشابهی تولید کنند. به‌عنوان مثالی از سازگاری در کل سیستم، سیستم مدیریت مطالعات برون‌مرزی (SAMS) را درنظر بگیرید. صفحات وب این سیستم باید ظاهر و حسی مشابه صفحات وب دفتر آموزش‌های بین‌المللی[۲] داشته باشند زیرا OIE مانند SAMS کار می‌کند. این باعث می‌شود که مراجعه کننده ناوبری سازگاری را تجربه کند.

رهنمون ۱۲-۸

کمینه کردن تغییر حالت بین حالت ماوس و حالت صفحه‌کلید.

تغییر حالت‌های متعدد بین کلیک کردن دکمه‌های ماوس و تایپ کردن با صفحه‌کلید درحین اجرای یک وظیفه‌ی کسب‌وکاری، کاربر را مشغول کرده و بهره‌وری او پایین می‌آید. بنابراین، یک واسط کاربری کاربرپسند، باید نیاز به تغییر حالت بین دو روش ورودی را حداقل نماید.

رهنمون ۱۲-۹

یک خصیصه‌ی خوب ممکن است آن‌قدر هم خوب نباشد.

مهندسان نرم‌افزار تمایل دارند به کاربران «خصیصه‌های خوب» تحویل دهند، بدون توجه به این‌که آیا کاربر این خصیصه‌ها را می‌خواهد یا نه! این موضوع هم درمورد کارکردها و هم درمورد واسط کاربری اتفاق می‌افتد.

به‌عنوان مثال، برخی نرم‌افزارها دارای یک ویزارد[۳] دستیار هوشمند می‌باشند که همواره به‌طرز خودکار برای کمک کردن ظاهر می‌شود. واقعیت این است که این ابزار خیلی هم کمک نمی‌کند. یک پیغام ظاهرشونده که دائماً به کاربر یادآوری می‌کند که کاری را انجام دهد، مثلاً بروزرسانی را انجام دهد، آزاردهنده و برضد بهره‌وری است. به‌عنوان مثال، یک نسخه از UNIX دارای تصحیح خودکار خط فرمان است. اما در ۹۹% مواقع، تصحیح پیشنهادی منطقی به‌نظر نمی‌رسد. بدترین چیز این است که هیچ راهی برای خاموش (غیرفعال) کردن این خصیصه «خوب» وجود ندارد.

---

[۱] Copy-and-Paste
[۲] OIE
[۳] Wizard

رهنمون ۱۲-۱۰

دست‌پخت خودتان را بخورید.

بسیاری از محصولات کاربرپسند نیستند چون طراح هرگز به‌طرز جدی از محصول استفاده نمی‌کند. این رهنمون پیشنهاد می‌کند که طراح واسط کاربری و برنامه‌ساز، یا کسی از جانب ایشان، باید به‌طور جدی از نرم‌افزار استفاده کند و قبل از تحویل آن به کاربران نهایی، مشکلات واسط کاربری را برطرف نماید.

۱۲-۷ اعمال اصول چابکی

رهنمون ۱۲-۱۱

مشارکت فعالانه‌ی کاربر ضروری است. یک روش همکوشانه و همکارانه بین تمامی ذی‌نفعان لازم است.

واسط کاربری علاوه بر تعامل تمامی کاربران با سیستم بر روی بهره‌وری کاربر و کیفیت کار نیز تاثیر می‌گذارد، این‌ها هم بر روی تمام ذی‌نفعان، اعم از کارفرمای کاربران تاثیر می‌گذارند. بنابراین، در طراحی واسط کاربر باید قشر گسترده‌ای از کاربران فعالانه مشارکت نمایند. امروزه، تمام کسب‌وکارها به‌شدت رقابتی هستند. بنابراین، همکاری ذی‌نفعان ضروری است. پروژه باید به‌دنبال پشتیبانی ذی‌نفعان باشد تا نمایندگان کاربر مشخص شده و به‌طور فعال با تیم همکاری نمایند تا واسط کاربری مورد نیاز طراحی شود و تیم بازخورد کاربران را جمع‌آوری نماید. این‌کار به‌طرز قابل توجهی نیاز به تغییر واسط کاربری در آینده را کاهش می‌دهد.

رهنمون ۱۲-۱۲

نیازمندی‌ها تغییر می‌کنند ولی مقیاس زمان ثابت است.

تغییر در واسط کاربری دوباره‌کاری قابل ملاحظه‌ای را می‌طلبد. این اصل چابک پیشنهاد می‌کند که مدت تکرار نباید به‌خاطر این دوباره‌کاری‌ها اضافه شود. درعوض، تیم باید با مذاکره با مشتری و کاربران نیازمندی‌های با اولویت پایین را شناسایی و آن‌ها را از دور خارج کند تا جا برای دوباره‌کاری‌ها باز شود. قانون ۸۰-۲۰ می‌گوید که همیشه چنین نیازمندی‌هایی وجود دارند. این موضوع همچنین اهمیت اولویت‌بندی نیازمندی‌های و تصحیح اولویت‌ها پس از هر تکرار را نشان می‌دهد.

رهنمون ۱۲-۱۳

نسخه‌های کوچک و افزایشی تولید کرده و تکرار کنید. به‌علاوه، برروی تحویل مداوم محصولات نرم‌افزار تمرکز کنید.

این اصول کمک می‌کنند که میزان دوباره‌کاری کم شود، زیرا نیازمندی‌های کاربران و نیازمندی‌های واسط کاربر واقعی می‌تواند به‌زودی و به‌سرعت شناسایی شود. نتیجه‌ی این‌که افزایش کوچک باشد این خواهد بود که می‌توان به‌آسانی به‌سراغ مسئله رفت.

رهنمون ۱۲-۱۴

یک واسط کاربری به اندازه‌ی کافی خوب کفایت می‌کند. به نرم‌افزار درحال‌کار بیش از طراحی بها دهید.

شناسایی نیازمندی‌های یک واسط کاربری واقعی یک چالش حقیقی است. گرچه پیش‌نمونه‌ها و طراحی‌های واسط کاربری مفیدند ولی این‌ها نرم‌افزارهای واقعی نیستند. مشکل به‌این‌خاطر نیست که زمان یا تلاش کافی صرف نشده است، بلکه به خاطر طبیعت چگونگی تصمیم‌گیری کاربران درخصوص واسطی است که می‌خواهند. یعنی کاربران از طریق «دیدن، باور کردن است» تصمیم می‌گیرند. لذا، یک طراحی به‌اندازه‌ی کافی خوب، کفایت می‌کند. نرم‌افزار درحال‌کار مهم‌تر است، زیرا به کاربران امکان می‌دهد واسط کاربری که می‌خواهند را مشخص کنند.

رهنمون ۱۲-۱۵

نیازمندی‌ها را در سطح بالا استخراج کنید؛ سبک‌وزن و بصری.

اعمال این اصل به واسط کاربری به معنی استخراج نیازمندی‌های واسط کاربری در سطح بالاست که باعث می‌شود این نیازمندی‌ها سبک وزن و بصری باشند. همان‌گونه که در رهنمون ۱۲-۱۴ بحث شد، کاربران ازطریق «دیدن، باور کردن است» می‌دانند چه می‌خواهند. بنابراین، کسب نیازمندی‌ها در سطح بالا و سبک‌وزن ساختن آن‌ها کفایت می‌کند. طبیعت واسط کاربری می‌گوید که مؤثرتر خواهد بود اگر نیازمندی‌ها با کمک روش‌های بعدی، مانند طرح، پیش‌نمونه‌ی واسط کاربری، و تصویر صفحه به‌دست آورده شوند.

۱۲-۸ پشتیبانی ابزاری برای طراحی واسط کاربر

ابزارهای زیادی برای پشتیبانی طراحی واسط کاربری برنامه‌های رومیزی، مبتنی بر وب[۱]، و موبایل[۲] وجود دارند. نمونه‌هایی از این ابزارها عبارتند از Adobe Flash Builder، Eclipse Window Builder Pro GUI Designer، Microsoft Visual Studio Windows Forms Designer، و Netbeans GUI Builder Plug-in برای برنامه‌های رومیزی؛ Microsoft Visual Studio Web Designer و Adobe Dreamweaver برای طراحی صفحات وب؛ و Netbeans Visual Mobile Designer برای طراحی واسط‌های کاربری لمسی برای دستگاه‌های Java ME. در اکثر این ابزارها می‌توان واسط کاربری را با کشیدن و قراردادن ویجت‌های GUI از یک جعبه بر روی یک نقشه طراحی نمود. بدین معنی که این ابزارها روش «چیزی که می‌بینید چیزی است که خواهید داشت»[۳] را انجام می‌دهند. این ابزارها همچنین به کاربر اجازه می‌دهند بین دید طراحی و دید کد منبع تغییر حالت دهد. دید طراحی به کاربر امکان می‌دهد که GUI را به‌شکل بصری ویرایش کند در حالی که دید کد منبع به کاربر اجازه می‌دهد کد منبع را مستقیماً ویرایش نماید. سرانجام، تمام ابزارها کد را از روی طراحی تولید می‌نمایند.

خلاصه

این فصل با مقدمه‌ای در مورد طراحی واسط کاربر و اهمیت واسط کاربری در موفقیت یک پروژه‌ی نرم‌افزاری شروع می شود. این اهمیت با داستان‌های دنیای واقعی توضیح داده شده‌است. سپس ویجت‌های GUI و فایده‌ی آن‌ها معرفی شده‌اند. این فصل همچنین یک فرایند طراحی واسط کاربر و رهنمون‌های مربوطه را ارائه می‌نماید. فرایند مذکور با یک مطالعه‌ی موردی که یک واسط کاربری برای یک ویرایشگر نمودار حالت تولید می‌کند توضیح داده می‌شود. طراحی GUI مکمل مدل‌سازی تعامل کنشگر-سیستم است که در فصل ۸ ارائه گردید. پیش‌نمونه‌های GUI هم‌راستا با مورد کاربردهای گسترده استفاده می‌شوند تا جمع‌آوری بازخورد کاربر را تسهیل نمایند. طراحی GUI و مدل‌سازی تعامل کنشگر سیستم باید به‌طور همزمان انجام شوند تا از همدیگر سود ببرند.

برای مطالعه‌ی بیش‌تر

کتاب Johnson [93] شامل قوانین و رهنمون‌های اضافه‌تری درخصوص طراحی واسط کاربری می‌باشد. کتاب «راهنمای اصلی برای طراحی واسط کاربر» نوشته‌ی Wilber O. Galitz [66] طراحی واسط کاربر را از دیدگاه یک کاربر معرفی کرده و بیش از ۳۰۰ مطالعه‌ی موردی و نیز رهنمون‌های طراحی برای کاربردهای مبتنی بر وب و دستگاه‌های موبایل دارد. Jenifer Tidwell [150] بهترین روش‌هایی برای طراحی واسط کاربر به‌عنوان الگوهای طراحی معرفی می‌کند که برای توسعه‌ی کاربردها قابل بازاستفاده هستند. مراجع [۸۳، ۸۴] نیز منابع خوبی در خصوص برنامه‌سازی GUI با استفاده از Java Abstract Window Kit (AWT) و Swing هستند.

---

[۱] Web-Based
[۲] Mobile
[۳] What You See Is What You Get (WYSYWIG)

سوال‌های مروری فصل

۱. کارهای یک واسط کاربر چیست؟
۲. چرا طراحی واسط کاربر مهم است؟
۳. واسط‌های کاربر گرافیکی چه خصیصه‌هایی دارند؟
۴. فعالیت‌های طراحی در طراحی واسط کاربر چیست؟
۵. مقوله‌های گسترده‌ی ویجت‌های واسط کاربر گرافیکی کدامند؟ وظایف هرکدام از این مقوله ویجت‌ها چیست؟
۶. مقوله‌های خاص ویجت‌های ورودی، خروجی، و نمایش اطلاعات کدامند؟ فایده‌ی هریک از این‌ها چیست؟
۷. مزایا و معایب ویجت‌های متن-گرا کدامند؟
۸. مزایا و معایب ویجت‌های انتخاب-گرا کدامند؟
۹. رهنمون‌های استفاده از ویجت‌های واسط کاربری گرافیکی کدامند؟
۱۰. گام‌های فرایند طراحی واسط کاربری که در این فصل معرفی شدند کدامند؟
۱۱. پیش‌نمونه‌ی واسط کاربری چیست؟ آیا گونه‌های مختلفی از پیش‌نمونه‌های واسط کاربری وجود دارند؟ اگر بله، آن‌ها کدامند؟
۱۲. فایده‌ی یک پیش‌نمونه‌ی واسط کاربری چیست؟
۱۳. فایده‌ی نمودار حالت در پیش‌نمونه‌سازی برای واسط کاربری چیست؟
۱۴. روش‌های ارزیابی طراحی واسط کاربری همراه با کاربران کدامند؟
۱۵. چرا روش‌های ارزیابی طراحی واسط کاربری همراه با کاربران مهم است؟
۱۶. قابلیت‌های پشتیبانی کاربر که در این فصل تشریح گردیدند کدامند؟

تمرین‌ها

برای هرکدام از مورد کاربردهای زیر، اگر در فصل‌های قبلی این کار را انجام نداده‌اید، ابتدا یک توصیف برای مورد کاربرد گسترده ارائه دهید. سپس گام‌های طراحی واسط کاربر را اجرا کنید و یک طراحی واسط کاربر برای آن مورد کاربرد تولید کنید.

۱-۱۲ مورد کاربرد «رزرو خودرو»[۱] از کاربرد اجازه‌ی برخط خودرو. توصیفی از این کاربرد در پیوست ت آمده است.

۲-۱۲ مورد کاربرد «ورود» Login که در آن کاربر برای اولین بار وارد می‌شود و سیستم کاربر را به صفحه‌ای هدایت می‌کند که سؤال‌های مجوزی که در موقع تغییر گذرواژه در آینده از او پرسیده خواهد شد را تعریف می‌کند. فرض کنید هر کاربر باید پنج سؤال تعریف کرده و پاسخ این سؤال‌ها را نیز ارائه نماید.

۳-۱۲ مورد کاربردهای «خروج کتاب»[۲] و «بازگرداندن کتاب»[۳] از سیستم کتابخانه‌ی برخط که به اعضا اجازه می‌دهد کتاب‌ها را به امانت برده و بازگردانند. پیش‌شرط این دو مورد کاربرد این است که عضو وارد سیستم شده باشد. کتاب‌هایی که خارج می‌شوند برای عضو ارسال (پستی) می‌شوند و عضو نیز کتاب‌های بازگشتی را از طریق پست بازمی‌گرداند.

۴-۱۲ در پیوست ت یک برنامه‌ی ویرایشگر نمودار کلاس تشریح گردید. برای این برنامه، یک طراحی واسط کاربر انجام دهید.

---

[۱] Reserve a Car
[۲] Checkout Books
[۳] Return Books

۱۲-۵ طراحی واسط کاربر فوق را با استفاده از فهرست بررسی بازبینی طراحی واسط کاربر ارائه شده در بخش ۱۲-۴-۷ بررسی کنید. هرگونه مشکلی که کشف می‌کنید را مستند کنید. طراحی را اصلاح کنید تا مشکلات رفع شوند.

پیوست‌ها

پیوست آ فرایند شخصی نرم‌افزار: تخمین، برنامه‌ریزی، و تضمین کیفیت

پیوست ب تکنولوژی‌های جاوا

پیوست پ ابزارهای نرم‌افزاری

پیوست ت شرح پروژه

آ فرایند شخصی نرم‌افزار: تخمین، برنامه‌ریزی و تضمین کیفیت

پیوست آ، ادامه و کامل‌کننده‌ی موضوع فرایند شخصی نرم‌افزار است که در فصل ۲ مطرح گردید.

آ-۱ تخمین تلاش در PSP

در زمان انجام فرایند شخصی نرم‌افزار، مهندس نرم‌افزار از زمان «تا تاریخ » و داده‌ی «درصد تا تاریخ » که در فرم‌های خلاصه برنامه‌ریزی پروژه[۱] ثبت شده است، استفاده می‌کند تا دقت تخمین را افزایش دهد. برای امکان‌پذیرکردن این کار، تمرین‌های برنامه‌نویسی طراحی می‌شود. یعنی تمرین‌هایی طراحی می‌شود که فرض‌های زیر را برآورده سازد: اول، کار لازم برای یک امر جدید مانند امور قبلی توزیع خواهد شد. دوم، هر پروژه به‌حدی کوچک است که هر مرحله‌ی آن تنها شامل انجام یک وظیفه باشد. سوم، کار به‌حدی مشابه کارهای قبلی است که %ToDate تقریباً تخمین دقیقی از وظایف به‌دست بدهد. امید است که مهندس نرم‌افزاری که در PSP تعلیم داده شده است، برای همه‌ی پروژه‌های آینده نیز به پرکردن Time Recording Log و Defect Recording Log ادامه دهد. این داده‌ها در آینده برای برنامه‌ریزی و تضمین کیفیت پروژه‌های جدید به‌کارگرفته خواهد شد. روش‌های مختلفی می‌تواند برای تخمین و برنامه‌ریزی با PSP به‌کار گرفته شود. این روش‌ها شامل مدل COCOMO، روش تخمین نقطه کار، روش تخمین Delphi، نمودار PERT، و نمودار Gantt می‌باشد. این روش‌ها در فصل ۲۳ نشان داده شده‌اند. روشن است که دقت تخمین و برنامه‌ریزی می‌تواند با استفاده از داده‌های Time in Phase که در فرم‌های Project Plan Summary ثبت شده است، بهبود یابد.

علاوه بر روش‌های شناخته شده‌ی تخمین، روش تخمینی مبتنی‌برپروکسی[۲] یا روش PROBE، که به‌تفصیل در [۸۷] شرح داده شده است، می‌تواند با PSP استفاده گردد. یک پروکسی، جانشینی برای نرم‌افزاری است که قرار است تخمین زده شود. روش PROBE بر مبنای این فرض است که تخمین پروکسی کار ساده‌تری است؛ و در نتیجه، به تخمین زمان لازم برای توسعه نرم‌افزار کمک می‌کند.

پروکسی باید شرایط زیر را برآورده سازد، که در [۸۷] شرح داده شده است:

---

[۱] Project Plan Summary

[۲] Proxy-Based Estimation

- معیار اندازه‌ی پروکسی باید رابطه‌ی نزدیکی با تلاش لازم برای توسعه‌ی نرم‌افزار داشته باشد. یعنی باید رابطه‌ی نزدیکی با منابع لازم برای توسعه‌ی نرم‌افزار داشته باشد.
- محتوای پروکسی یک سیستم باید به‌طور خودکار قابل شمارش باشد. یعنی پروکسی باید یک موجودیت فیزیکی باشد که بتواند به‌دقت تعریف شود و به‌طور خودکار شمارش گردد.
- پروکسی باید بتواند به‌سادگی از ابتدای پروژه دیده نشود. برای مثال، کلاس‌ها، ورودی‌ها، خروجی‌ها، فیلدهای پایگاه‌داده، فرم‌ها، و گزارش‌ها می‌توانند به‌سادگی دیده شوند و بنابراین می‌توانند به‌عنوان پروکسی مورد استفاده قرار گیرند.
- پروکسی باید بتواند بر حسب نیازهای هر پروژه و هر توسعه‌دهنده تنظیم‌گردد. یعنی پروکسی باید از داده‌هایی که مربوط به پروژه‌ی جدید است، استفاده کند.
- پروکسی باید نسبت به تغییرات پیاده‌سازی که بر هزینه یا تلاش لازم برای توسعه تاثیر می‌گذارد، حساس باشد. یعنی داده‌های پروکسی باید برای زبان‌های برنامه‌نویسی مختلف، سبک‌های طراحی گوناگون و انواع مختلف کاربرد در دسترس باشد.

روش PROBE از داده‌های پروکسی مربوط به پروژه‌های قبلی و رگرسیون خطی برای ایجاد تابع خطی زیر استفاده می‌کند:

$$T = f(S) = \beta_0 + \beta_1 \times S$$

که در آن، T زمان توسعه، S اندازه‌ی تخمین زده شده‌ی پروکسی، و β_0 و β_1 ضرایب به‌دست‌آمده از رگرسیون خطی می‌باشند. از آنجا که فرض شده که اندازه پروکسی را می‌توان ساده‌تر تخمین زد، و پروکسی با نرم‌افزار مربوط است، این تابع خطی می‌تواند برای تخمین زمان واقعی توسعه‌ی نرم‌افزار نیز مورد استفاده قرار بگیرد.

برای شکستن یک پروژه‌ی بزرگ به سلسله‌مراتبی از مولفه‌ها، از روش تقسیم‌وحل استفاده می‌شود. در این روش، مولفه‌های سطوح بالاتر، توسط مولفه‌های سطوح پایین‌تر پالایش[۱] می‌شوند. این کار، یافتن پروکسی‌ها برای مولفه‌های سطوح پایین‌تر را آسان‌تر می‌نماید. سپس زمان لازم برای ترکیب مولفه‌های سطوح پایین‌تر به زمان لازم برای توسعه‌ی یک مولفه‌ی سطح بالاتر اضافه می‌شود.

برنامه‌ریزی در PSP مشابه برنامه‌ریزی با استفاده از روش‌های دیگر است. تنها تفاوت این است که داده‌های ثبت‌شده‌ی Time in Phase و Interrupt Time به مهندسی نرم‌افزار در ایجاد یک زمان‌بندی واقعی‌تر کمک می‌نماید. فعالیت برنامه‌ریزی شامل گام‌های زیر است:

۱. یک تخمین از زمان کلی موردنیاز برای توسعه نرم‌افزار ایجاد می‌شود. روش انجام این کار در پاراگراف‌های قبلی شرح داده‌شده‌است.
۲. با استفاده از داده‌های %ToDate که در فرم‌های Project Plan Summary ثبت‌شده است، به مراحل توسعه زمان تخصیص بدهید.
۳. با توجه به داده‌های شخصی ثبت‌شده برای پروژه‌های پیشین، ساعت‌های هفتگی وظایف را تخمین بزنید. برای مثال، Interrupt Time صرف‌شده در سایر فعالیت‌ها مانند جلسات، ایمیل زدن‌ها، تماس‌های تلفنی، و مانند آن.
۴. زمان‌های موجودکه می‌توانند در هر هفته برای پروژه صرف شوند را محاسبه کنید. پایان پروژه‌های دیگر را نیز در نظر بگیرید.
۵. وظایف توسعه را بر حسب اولویت‌ها و وابستگی‌های موجود بین آنها مرتب کنید.

---

[۱] Refinement

۶. با توجه به ترتیب انجام وظایف، طول زمانی هر وظیفه، و زمان موجود در هر هفته برای هر وظیفه، یک زمان‌بندی ایجاد کنید.
۷. نقاط عطف پروژه را تعریف کنید و زمان‌های برنامه‌ریزی‌شده برای پایان هر کدام را فهرست نمایید.

آ-۲ تضمین کیفیت نرم‌افزار در PSP

راهبرد PSP برای سیستم‌های پرکیفیت، مدیریت محتوای نقایص همه‌ی قسمت‌های سیستم است. از طریق مرور طراحی و کد می‌توان از راه‌های کارآمدی کیفیت و بهره‌وری نرم‌افزار را افزایش داد. PSP قوانین زیر را برای مرور کُد پیشنهاد می‌نماید:

۱. پیش از آن‌که به مرحله‌ی بعدی توسعه بروید، شخصاً همه‌ی کارهای خود را مرور و بازبینی نمایید.
۲. سعی کنید که همه نقایص کار را پیش از تحویل طراحی یا کد به دیگران رفع نمایید.
۳. از یک فهرست مروری شخصی استفاده کنید و یک فرایند ساخت‌یافته‌ی مرور را در پیش بگیرید. این فهرست مروری شخصی، از فهرست ثبت‌شده‌ی کاستی‌ها ایجاد می‌شود که برای انواعی از نقایص که بیشتر دیده می‌شوند، راه‌هایی برای کشف و رفع خطا پیشنهاد می‌کند. فهرست مروری در بخش‌هایی سازمان‌دهی می‌شود که هر یک بر کشف نقایص مشابه تمرکز می‌نماید. این فهرست از زمانی به زمان دیگر تغییر می‌کند تا پیشرفت حاصل ازکاربرد PSP را نشان دهد.
۴. از روش‌های صحیح مرور استفاده کنید: مرور را از بخش‌های کوچک شروع کنید، مرورها را بر روی کاغذ انجام دهید، و این کار را در زمان‌هایی که خسته نیستید یا استراحت می‌کنید انجام دهید.
۵. زمان مرور، اندازه‌ی مصنوعات مرورشده، و تعداد و انواع نقایصی که یافته اید را اندازه‌گیری نمایید.
۶. از این داده‌ها برای بهبود فرایند شخصی مرور خود استفاده کنید.
۷. مولفه‌های خود را طوری طراحی و پیاده‌سازی کنید که به‌سادگی قابل مرور باشند.
۸. داده‌های مربوط به نقایص را مرور کنید تا راه‌هایی برای پیشگیری از بروز این خطاها کشف نمایید.
۹. با توجه به استانداردهای کدنویسی مرور انجام دهید.

آ-۳ طراحی و کیفیت

PSP به مقوله‌ی طراحی، از نقطه‌نظر اجتناب‌ازخطا توجه می‌نماید و چهار قالب طراحی پیشنهاد می‌دهد. این قالب‌ها، جایگزین‌هایی درشکل جدولی برای قالب‌های معمول طراحی مانند نمودار کلاس، سناریو، شبه‌کد، و نمودار حالت هستند. به‌طور خلاصه، این قالب‌های طراحی عبارتند از :

۱. قالب توصیف کارکردی[۱]
این قالب برای توصیف یک کلاس، کلاس والد آن، و صفت‌ها و عملیات آن کلاس مورد استفاده قرار می‌گیرد. این قالب، یک توصیف جدولی از یک کلاس است.
۲. قالب توصیف عملیاتی[۲]
این قالب یک شرح خط‌به‌خط از یک سناریو است، یعنی هر خط عمل انجام‌شده توسط یک شئ را به‌همراه یک توضیح مختصر شرح می‌دهد.

[۱] Functional Specification Template
[۲] Operational Specification Template

۳. قالب توصیف منطق[۱]
این قالب، چیزی مشابه توصیف منطق برنامه با استفاده از انگلیسی ساخت‌یافته یا شبه‌کد است.
۴. قالب توصیف حالت[۲]
این قالب، حالت‌ها، تغییر حالت‌ها، و شرط‌های نگهبان را توصیف می‌کند.

علاوه بر قالب‌های طراحی، PSP از روش‌های متعددی برای وارسی یک طراحی استفاده می‌کند. برای مثال برای وارسی این‌که آیا یک طراحی کارکرد موردانتظار را برآورده می‌سازد، از استانداردهای طراحی بهره می‌گیرد. یک جدول اجرا یا جدول ردیابی، برای ثبت نتایج میانی یا ردیابی‌های انجام‌شده در زمان اجرای دستی یک منطق مورد استفاده قرار می‌گیرد. توصیف یک حالت، با ردیابی دستی حالت‌ها و تغییر حالت‌ها وارسی می‌شود.

.

---

[۱] Logic Specification Template

[۲] State Specification Template

ب

تکنولوژی‌های جاوا

این پیوست، یک خودآموز کوچک برای برخی از تکنولوژی‌های پر کاربرد جاوا، مانند JDBC[1]، Swing، و JSP[2] ارائه می‌دهد.

ب-۱ شروع با اتصال پایگاه‌داده

تقریباً همه‌ی برنامه‌های کاربردی جهان واقع از یک پایگاه‌داده استفاده می‌کنند. معمول‌ترین نوع پایگاه‌داده، پایگاه‌داده‌ی رابطه‌ای است. یک پایگاه‌داده‌ی رابطه‌ای برای نمایش داده از جداول استفاده می‌کند و یک زبان پرس‌وجو نیز برای دسترسی به پایگاه‌داده در اختیار کاربر قرار می‌دهد. معمول‌ترین زبان پرس‌وجو، زبان پرس‌وجوی ساخت‌یافته[3] است که به‌نام SQL شناخته می‌شود. این بخش نشان می‌دهد که چگونه از یک برنامه‌ی شیءگرا می‌توان به پایگاه‌داده دسترسی پیدا کرد. هدف از این بخش، شروع مبحث اتصال پایگاه‌داده، به‌سادگی و با سرعت است.

ب-۱-۱- اتصال پایگاه‌داده چیست؟

دنیای شیءگرایی کاملاً با دنیای رابطه‌ای متفاوت است. اگر یک پایگاه‌داده‌ی رابطه‌ای برای ذخیره‌ی اشیا استفاده شود، باید سازوکاری فراهم شود که دسترسی برنامه‌ی شیءگرا به پایگاه‌داده برای ذخیره و بازیابی اطلاعات را تسهیل نماید. اتصال پایگاه‌داده این کار را انجام می‌دهد. به‌طورخاص ODBC،[4] که توسط مایکروسافت ارائه‌شده، یک واسط برنامه به زبان C برای دسترسی به یک پایگاه‌داده است. Java DataBase Connectivity یا JDBC یک واسط دسترسی به پایگاه‌داده برای جاوا است.

همان‌طور که در شکل ب-۱ نشان داده شده است، یک برنامه‌ی کاربردی جاوا از JDBC API استفاده می‌کند که خود برای دسترسی به پایگاه‌داده از یک پل JDBC/ODBC و یک درایور ODBC، و یا یک درایور JDBC که توسط شرکت سازنده تولید شده، استفاده می‌نماید.

---

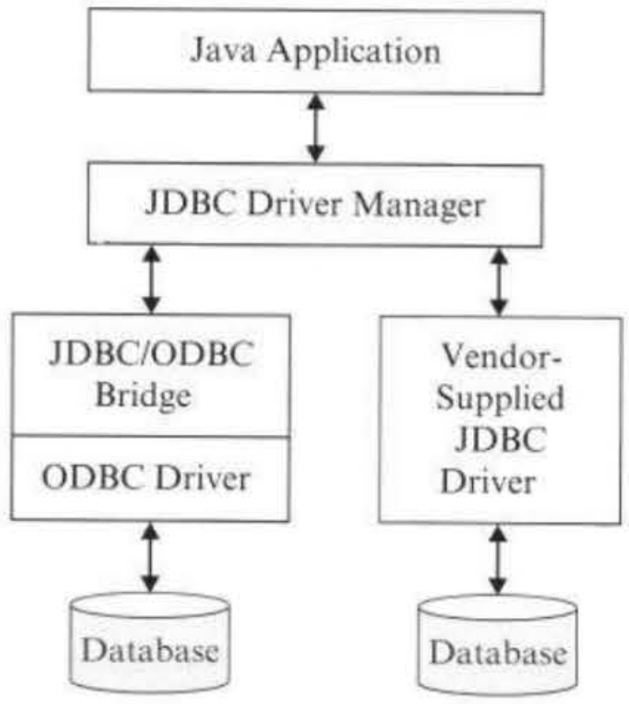
[1] Java DataBase Connectivity
[2] Java Server Pages
[3] Structured Query Language
[4] Open DataBase Connectivity

ب-۱-۲- تنظیم منابع داده

پیش از نوشتن برنامه، ممکن است لازم باشد که منبع داده‌ای که به پایگاه‌داده‌ی مورد استفاده پیوند دارد، تنظیم گردد. این موضوع به سیستم مدیریت پایگاه‌داده‌ی مورد استفاده (DBMS) مربوط است. اگر از Microsoft Access استفاده شود، گام‌های زیر باید انجام گردد که ممکن است در کامپیوترهای مختلف، متفاوت باشد:

۱. بر روی Start، سپس Settings، و سپس Control Panel کلیک نمایید.
۲. بر روی Administrative Tools دو بار کلیک نمایید.
۳. بر روی Data Sources (ODBC) دو بار کلیک کنید.
۴. بر روی زبانه‌ی User DSN و سپس دکمه‌ی Add کلیک نمایید.
۵. در جعبه‌ی گفتگوی Create New Data، درایور متناسب با پایگاه‌داده را انتخاب کنید و سپس بر روی Finish کلیک کنید.
۶. در جعبه‌ی گفتگویی که نشان داده می‌شود، نامی‌که می‌خواهید در فیلد Data Source Name استفاده کنید، وارد نمایید.
۷. دگمه‌ی Select را کلیک نموده، مسیر پایگاه‌داده را مشخص نمایید.
۸. دگمه‌ی OK را کلیک کنید. اکنون باید منبع جدید داده را در فهرست User Data Sources مشاهده کنید.
۹. OK را کلیک کنید و قاب کنترل را ببندید.

اگر از MySQL استفاده می‌شود، نیازی به این گام‌ها نیست.



شکل ب-۱: Java DataBase Connectivity (JDBC)

ب-۱-۳- دسترسی به پایگاه‌داده‌ها از یک برنامه

استفاده از JDBC برای دسترسی به یک پایگاه‌داده، کار نسبتاً ساده‌ای است. همان‌طور که در شکل ب-۲ دیده می‌شود، این کار از چهارگام تشکیل می‌شود:

گام ۱- اتصال به یک پایگاه‌داده

همان‌طور که در خط (۳۱) دیده می‌شود، اولین گام، اتصال به یک پایگاه‌داده است. این کار شامل بارگذاری کلاس درایور JDBC در خط (۱۵) و برقراری یک ارتباط با پایگاه‌داده در خط (۲۱) می‌باشد.

گام ۲- بازیابی رکوردها از پایگاه‌داده

خطوط (۳۲) تا (۳۵) دستورات مربوط به بازیابی رکوردها از پایگاه‌داده را نشان می‌دهند. تابع executeQuery(String sql) از Statement، رکوردها را از پایگاه‌داده بازیابی می‌کند و در یک شئ Result Set برمی‌گرداند. شئ Result Set یک جدول داده است که مشابه یک رابطه در یک پایگاه‌داده‌ی رابطه‌ای است.

تابع executeUpdate(String sql) باید در صورتی استفاده شود که پرس‌وجو از نوع اضافه‌کردن، به‌روزرسانی، حذف، و یا ایجاد جدول باشد. این متد، شماره و تعداد ردیف‌هایی که تحت تاثیر قرار گرفته‌اند را بر می‌گرداند.

```
(1) package database;
(2) import java.sql.*;
(3) import java.util.*;
(4) import java.io.*;
(5) //other import statements
(6) public class DBMgr {
(7)   Connection con = null;
(8)   String driver = "sun.jdbc.odbc.JdbcOdbcDriver";
(9)   String url = "jdbc:odbc:mydb";
(10)  String usename;
(11)  String password;
(12)  Statement stmt = null;
(13)  public void connect(){
(14)    try{//load the JDBC driver
(15)      Class.forName(driver);
(16)      }catch(ClassNotFoundExeption e){
(17)      System.err.println("Failed to load driver.");
(18)      return;
(19)      }
(20)    try{//connect to database
(21)    con=DriverManager.getConnection(url,
              username,password);
(22)    }catch(SQLExeption e){
(23)    System.err.println("Failed to connect");
(24)    return;
(25)    }
(26)  }
```

```
(27)  public ArrayList getClassX(String attrName,
          String attrValue){
(28)  ResultSet rs=null;
(29)  ArrayList result=new ArrayList();
(30)  try{
(31)    connect();
(32)    stmt=con.createStatement();
(33)    rs=stmt.executeQuery(
(34)      "SELECT * FROM RABLEX"+
(35)      "WHERE "+attrName+"="+
              "'"+attrValue+"'");
(36)    while(rs.next()){
(37)      ClassX x=new ClassX();
(38)      x.setAttr1(rs.getString("attr1"));
(39)      x.setAttr2(rs.getString("attr2"));
(40)      //other set attribute statements
(41)      result.add(x);
(42)    }
(43)  stmt.close();con.close();
(44)  }
(45)  catch(SQLException e){
(46)   return null;
(47)  }
(48)  return result;
(49) }
(50) }
```

شکل ب-۲: دسترسی به یک پایگاه‌داده از یک برنامه‌ی جاوا

گام ۳ - پردازش نتیجه‌ی پرس‌وجو

خط‌های (۳۶) تا (۴۲) در مورد ورودی‌های جدول از شئ Result Set تکرار می‌شوند. در بیشتر موارد، ورودی‌های هر ردیف برای تنظیم مقادیر صفت‌های یک شئ مورد استفاده قرار می‌گیرند. اشیا به یک شئ ArrayList که در خط (۴۸) برگردانده می‌شود، اضافه می‌شوند.

گام ۴ - قطع اتصال از پایگاه‌داده

این گام در خط (۴۳) نشان داده‌شده‌است.

هدف از برنامه شکل ب-۲، کمک به خواننده در شروع یادگیری JDBC است. به‌همین دلیل نام کاربر و گذرواژه‌ی آن به‌طور ثابت تعریف شده است. این کار یک مشکل امنیتی را پیش می‌آورد، زیرا خواندن برنامه و کشف نام کاربر و گذرواژه بسیار آسان است. یک راه بهتر، استفاده از رمزگذاری، ذخیره‌ی نام کاربر و گذرواژه در حافظه، و استفاده از آنها برای اتصال به پایگاه‌داده است.

رشته‌ی راه‌انداز و آدرس پایگاه‌داده می‌تواند در یک فایل از ویژگی‌ها تعریف شده، و در یک شئ Properties بارگذاری شود. سپس این ویژگی‌ها می‌تواند با فراخوانی متد getProperty(String propertykey) از شئ Properties بازیابی گردد.

ب-۲ آغاز یادگیری Swing

فصل ۱۲ چگونگی طراحی واسط‌های کاربری برای یک کاربرد مورد نظر را نشان می‌دهد. این بخش، برنامه‌نویسی واسط کاربری گرافیکی (GUI) با Swing، یک ابزار ویجت برای جاوا، را معرفی می‌نماید. Swing گسترشی از ابزار Java Abstract Window Toolkit (AWT) می‌باشد. برای تمایز این دو ابزار، همه‌ی قطعه‌های Swing با حرف J بزرگ شروع می‌شوند، مانند JFrame، JComponent که گسترش‌یافته‌ی AWT Frame و AWT Component می‌باشند. Swing برای پیاده‌سازی واسط‌های کاربری گرافیکی برای کاربردهای متکی‌به‌خود ساخته شده است. آنچه در ادامه می‌آید از GUI ویرایشگر نمودار حالت نشان داده شده در شکل ۱۶-۳ برای معرفی برنامه‌نویسی با Swing استفاده کرده‌است. IDEهای بسیاری برنامه‌نویسی Swing را تسهیل می‌نمایند. برای مثال، تکه‌برنامه‌های نشان‌داده‌شده در این بخش می‌تواند به‌راحتی با استفاده از امکانات طراحی تعاملی GUI از یک IDE ایجاد شود.

ب-۲-۱- ایجاد پنجره‌ی اصلی با JFrame

ابتدا باید JFrame برای ایجاد پنجره‌ی اصلی گسترش یابد. کد زیر یک پنجره با عنوان "State Diagram Editor" و یک نوار منو ایجاد می‌کند.

```
package gui;
import javax.swing.*;
public class SDEFrame extends JFrame {
        JMenuBar jMenuBar1 = new JMenuBar();
        JMenu jMenuFile = new JMenu("File");
        JMenu jMenuEdit = new JMenu("Edit");
        JMenu jMenuTools = new JMenu("Tools");
        JMenu jMenuHelp = new JMenu("Help");

        public SDEFrame() {
                try{ initComp();
                } catch (Exception ex) {
                        ex.printStackTrace();
                }
        }
        private void initComp() throws Exception {
                setTitle("State Diagram Editor");
                jMenuBar1.add(jMenuFile); jMenuBar1.add(jMenuEdit);
```

```
            jMenuBar1.add(jMenuTools); jMenuBar1.add(jMenuHelp);
            setJMenuBar(jMenuBar1); setSize(800,600);
        }
        /** exit application when the window is closed */
        protected void processWindowEvent(windowEvent e){
            super.processWindowEvent(e);
            if (e.WINDOW_CLOSING==e.getID()) System.exit(0);
        }
}
```

برای نشان دادن پنجره، به یک کلاس Main نیاز است که یک رویداد از SDEFrame ایجاد کند و آن را قابل دیدن نماید:

```
import java.awt.Dimension;
public class Main {
    public Main() {
        SDEFrame frame = new SDEFrame();
        frame.validate();
        //Center the Window
        Dimension screenSize = Toolkit.getDefaultToolkit().
            getScreenSize();
        Dimension frameSize = frame.getSize();
        frame.setLocation( (screenSize.width - frameSize.width) / 2,
                           (screenSize.height - frameSize.height) / 2);
        frame.setVisible(true);
    }
    public static void main(String[] args) {
        try { UIManager.setLookAndFeel(
            UIManager.getSystemLookAndFeelClassName());
        } catch (Exception exception) {
          exception.ptintStackTrace();
        }
        new Main();
    }
}
```

خواننده می‌تواند این برنامه‌ها را کامپایل و اجرا نماید تا نتیجه را ببیند.

ب-۲-۲- استفاده از مدیر چیدمان (Layout) برای مرتب‌سازی مولفه‌ها

این بخش شرح می‌دهد که چگونه از مدیرزمینه‌های JPanel و Swing برای مرتب‌سازی مولفه‌ها استفاده شود، به‌طوری که بتوانند در محل‌های مناسب نشان داده شوند. یک شئ JPanel، یک دربردارنده است. اجزای GUI و سایر اشیا JPanel می‌توانند به‌یک شئ JPanel اضافه شوند. یک مدیر چیدمان یا به‌طور ساده یک چیدمان، مولفه‌های یک شئ JPanel را به‌شکلی از پیش تعریف شده مرتب می‌نماید. سه چیدمان پایه وجود دارد که می‌توانند به‌راحتی مورد استفاده قرارگیرند و بسیاری از مشکلات چیدمان را حل نمایند. این سه چیدمان عبارتند از : Layout Border، Grid Layout، Flow Layout. یک چیدمان حاشیه (border)، ناحیه‌ی مستطیلی JPanel را به پنج ناحیه تقسیم می‌کند: ناحیه‌ی مرکزی، و چهار ناحیه‌ی حاشیه‌ای شمال، جنوب، شرق، و غرب. برای مثال، جهت افزودن یک نوار ابزار، دکمه‌ها، و ناحیه‌ی رسم به پنجره‌ی اصلی (مانند شکل ۱۶-۳)، می‌توان از چیدمان حاشیه برای قرار دادن نوار ابزار در ناحیه‌ی شمالی قاب محتوا استفاده کرد، که این دربردارنده‌ی پیش‌فرض یک JFrame است:

```
public class SDEFrame extends JFrame {
        //...
        JButton stateButton=new JButton("  State ");
        JButton transitionButton=new JButton("Transition");
        JButton pointerButton=new JButton();
        private void initComp() throws Exception {
                //...
                JPanel westPanel=new JPanel(new GridLayout(3,1));
                JPanel stateBtnPanel=new JPanel();
                stateBtnPanel.setLayout(new FlowLayout());
                JPanel transitionBtnPanel=new JPanel();
                transitionBtnPanel.setLayout(new FlowLayout());
                ...
                getContentPane().add(westPanel,BorderLayout.WEST);
                westPanel.add(stateBtnPanel);
                westPanel.add(transitionBtnPanel);
                stateBtnPanel.add(stateButton);
                transitionBtnPanel.add(transitionButton);
                ...
        }
}
```

یک چیدمان شبکه‌ای[۱] مولفه‌ها را در یک شبکه‌ی مستطیلی مرتب می‌نماید. برای مثال، چیدمان سه دگمه‌ی رسم در شکل ۱۶-۳ را در نظر بگیرید. یک JPanel به‌نام West Panel به ناحیه‌ی غربی قاب محتوا اضافه می‌شود و از یک چیدمان شبکه‌ای برای مرتب‌سازی دگمه‌ها در West Panel استفاده می‌گردد. با این پیاده‌سازی، دکمه‌ها کل West Panel را اشغال خواهند کرد،

[۱] Grid Layout

به‌طوری که هیچ فضایی بین آن‌ها باقی نماند. این امر چندان زیبا به‌نظر نمی‌رسد. چیدمان جریان[۱] می‌تواند برای اضافه‌کردن فاصله بین دگمه‌ها و حاشیه‌های West Panel مورد استفاده قرار گیرد. یک چیدمان جریان، مولفه‌ها را از چپ به‌راست مرتب می‌نماید. به‌طور خاص‌تر می‌توان سه شئ JPanel را به West Panel اضافه کرد که چیدمان‌های جریان داشته باشند و هر یک حاوی یک دکمه‌ی رسم باشند.

ب-۲-۳- پردازش رویدادهای دکمه‌ها با Action Listener

در Swing، وقتی که کاربر بر روی یک دکمه‌ی GUI کلیک می‌کند، یک شئ Action Event ایجاد می‌شود. برنامه‌ی کاربردی باید چنین رویدادهایی را تشخیص داده، مورد رسیدگی قرار دهد. در Swing، Action Events توسط اشیا Action Listener پردازش می‌شود. برای مثال، شکل ۱۶-۳۲ می‌تواند در Swing به‌شکل زیر پیاده‌سازی شود:

```
//package and import statements
public class EditorGUI {
        EditController controller=new EditController();
        //Create the State and Transition buttons
        JButton stateButton=new JButton(" State ");
        JButton transButton=new JButton("Transition");
        //Create and add action listeners
        stateButton.addActionListener(new StateButtonListener());
        stateButton.addActionListener(new TransButtonListener());

        //Define the action listeners as inner classes
        class StateButtonListener implements ActionListener {
                public void actionPerformed(ActionEvent e) {
                        controller.addState(e);
                }
        }
        class TransButtonListener implements ActionListener {
                public void actionPerformed(ActionEvent e) {
                        controller.addTrans(e);
                }
        }
}
```

---

[۱] Flow

ب-۲-۴- پیاده‌سازی قابلیت‌های رسم

ناحیه‌ی رسم یا بوم نقاشی، یک JPanel است. این ناحیه ممکن است واسط‌های Mouse Listener و Mouse Motion Listener را برای دریافت و رسیدگی به رویدادهای ماوس پیاده‌سازی کند. به‌جای‌این‌کار ممکن است اشیا Mouse Listener و Mouse Motion Listener برای رسیدگی به رویدادهای ماوس ایجاد و اضافه شوند.

همان‌طور که در فصل ۱۶ نشان داده شد، طراحی ویرایشگر نمودار حالت، به Edit Controller اجازه می‌دهد که در خصوص رویدادهای مربوط به ماوس، عمل مناسب را انجام دهد. شکل ب-۳ یک نمونه کد را نشان می‌دهد که در آن ناحیه‌ی رسم یک شنونده‌ی ماوس و یک شنونده‌ی حرکت ماوس را ایجاد می‌کند و رویدادهای ماوس را مورد رسیدگی قرار می‌دهد. این شنونده‌ها داده‌های مختصات ماوس را جمع آوری می‌کنند و توابع مربوط از Edit Controller را صدا می‌زنند تا رویدادهای ماوس را پردازش نمایند.

همان‌گونه که در فصل ۱۶ شرح داده‌شد، Edit Controller از یک ماشین حالت برای دنبال‌کردن حالت‌های ویرایش استفاده می‌کند و مسئولیت‌های رسیدگی به رویدادهای GUI را به ماشین حالت می‌سپارد. به‌علاوه، برای درک رفتار حالت‌ها، از الگوی حالت استفاده می‌گردد. پیاده‌سازی الگوی حالت .... است. برای مثال، متدهای کلاس ریشه‌ی State خود را بر می‌گردانند (مانند this در C++، Java، وC)، به‌جز SelectBtnPressed() که Init. getInstance() را بر می‌گرداند. یک پیاده‌سازی از زیر کلاس‌های الگوی حالت شکل ۱۶-۲۶ در شکل ب-۴ دیده می‌شود.

```
//package and imports
public class Vanvas extends JPanel{
  private Point2D p1 = null, p2 = null;
  private Line2D line = null;
  public canvas(){
   super();
   addMouseListener(new MouseAdapter(){
    public void mouseClicked(MouseEvent event){
      Point2D p = event.getPoint();
      if(event.getClickCount()== 2){
          State s = Controller.getInstance().
            getDiagram().find(p);
          EditStateDialog d = new EditStateDialog();
          d.display(s); d.setVisible(true);
          }
      Controller.getInstance().mouseClicked(p);
      setCursor(Cursor.getPredefinedCursor(
        Cursor.DEFAULT_CURSOR));
      repaint();
     }
     public void mouseReleased(MouseEvent event){
        p2 = event.getPoint();
        Controller.getInstance().mouseReleased(p1,p2);
        repaint();
        setCursor(Cursor.getPredefinedCursor(
          Cursor.DEFAULT_CURSOR));
     }
     public void mousePressed(MouseEvent event){
        p1 = event.getPoint();
        Controller.getInstance().mousePressed(p1);
     }});
   addMouseMotionListener(new MouseMotionListener(){
     public void mouseDragged(MouseEvent event){
        p2 = event.getPoint();
        line = newLine2D.Double(p1,p2);
        repaint();
      }
     public void mouseMoved(MouseEvent event){}
     });
}
public void paintComponent(Graphics g){
    super.paintComponent(g);
    //draw rubber line while mouse is dragged
    if(line != null){
      g.drawLine((int)line.getP1().getX(),
        (int)line.getP1().getY(),
        (int)line.getP2().getX(), (int)line.getP2().getY());
      line = null;
    }
   //ask the state diagram to draw itself
    Controller.getInstance().getDiagram().draw((Graphics2D)g);
    }
}
```

شکل ب-۳: پیاده‌سازی ناحیه‌ی رسم

```
public class Init extends ControllerState {
  private static Init instance;
  private Init() { }
  public static Init getInstance() {...}
  public ControllerState addState() {
    return AddState.getInstance();
  }
  public ControllerState addTransition() {
    return AddTransition.getInstance();
  }
}
public class AddState extends ControllerState{
  private static AddState instance;
  private AddState() {}
  public static AddState getInstance() {...}
  public ControllerState mouseClicked(Point2D p) {
    Controller c= Controller.getInstance();
    Diagram d = c.getDiagram();
    if(!d.contains(p)) d.addState(new State(p));
    return Init.getInstance();
  }
}
```

```
public class AddTransition extends ControllerState {
  private static AddTransition instance = null;
  private AddTransition() {}
  public static AddTransition getInstance(){...}
  public ControllerState mousePressed(Point2D p) {
    Controller c= Controller.getInstance();
    if(c.getDiagram().find(p)!=null)
      return AssSrcSelected.getInstance();
    return this;
  }
}
public class TransSrcSelected extends ControllerState {
  private static TransSrcSelected instance;
  private TransSrcSelected() {}
  public static TransSrcSelected getInstance() {...}
  public ControllerState mouseDragged (Point2D p1, Point2D p2)
  { return null; }
  public ControllerState mouseReleased (Point2D p1, Point2D p2)
  { Controller c=Controller.getInstance();
    Diagram d=c.getDiagram();
    State s1 = d.find(p1), s2 = d.find(p2);
    if(s1!=null && s2!=null) {
      d.addTransition(...); return Init.getInstance();
    } else { return this; }
  }
}
```

شکل ب-۴: پیاده‌سازی زیرکلاس‌های الگوی حالت

ب-۳ شروع به‌کار با صفحات سرور جاوا

ب-۳-۱- صفحات سرور جاوا چه هستند؟

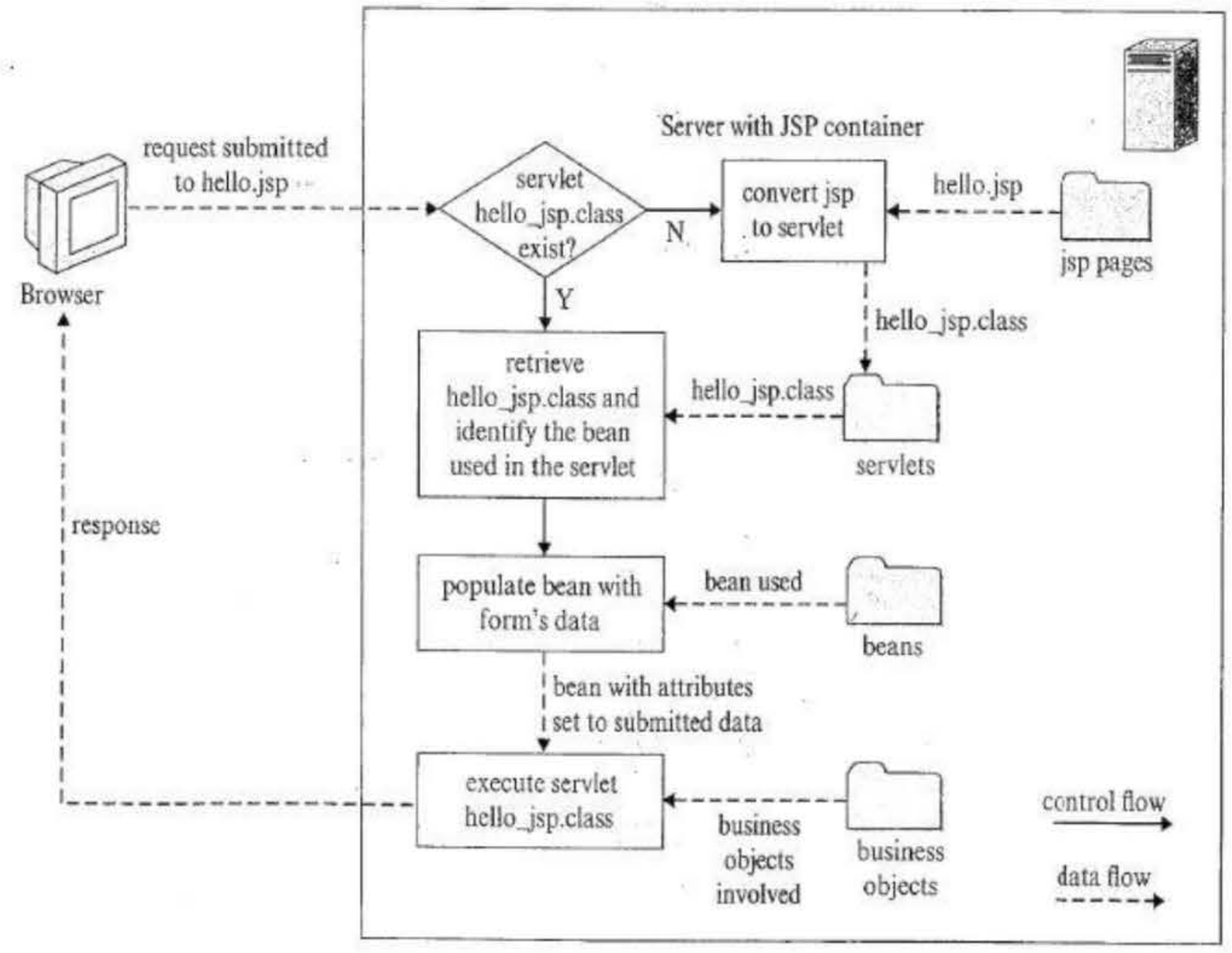
امروزه کاربردهای مبتنی‌بروب بیشتر و بیشتر عمومی‌شده‌اند. این کاربردها بر پایه‌ی صفحات سرور هستند تا درخواست‌های پردازش ارسال‌شده از یک مشتری را پردازش نمایند. در سکوی جاوا، صفحات سرور به‌نام Java Server Page (JSP) شناخته می‌شوند. اگر ساده صحبت کنیم، یک صفحه‌ی JSP حاوی کد html با قطعه برنامه‌های جاوا با نام Scriptlets می‌باشد. یعنی یک فایل JSP، پسوند JSP دارد (مثال: Login.JSP).

پیش از معرفی JSP، درخواست‌ها از مرورگر و پاسخ‌های به آن توسط Servletها پردازش می‌شده است. Servletها کلاس‌های جاوا با چنین وظایف می‌باشند. برنامه‌نویسی Servletها بسیار ساده است، زیرا برای ایجاد پاسخ‌ها تنها به جملات Println(...) نیاز است. یعنی صفحات html به مرورگر مشتری فرستاده می‌شود. JSP به برنامه‌نویس اجازه می‌دهد که صفحه‌ی وب را با استفاده از html و اطلاعات فرایند، و با کمک Scriptletها طراحی نماید.

ب-۳-۲- جریان کار JSP

شکل ب-۵ چگونگی کار JSP را نشان می‌دهد. درخواست‌ها، حاوی یک فرم که به یک صفحه‌ی JSP فرستاده شده، توسط موتور JSP و یا دربردارنده‌ی روی سرویس دهنده‌ی وب پردازش می‌شود. وقتی یک صفحه‌ی JSP، مانند hello.jsp، برای اولین

بار دیده می‌شود، به یک Servlet مانند hello-jsp.class تبدیل می‌شود، و در غیراین‌صورت، Servlet درون دربردارنده‌ی JSP بارگذاری می‌شود و دانه‌های (Beans) مورد استفاده‌ی Servlet شناسایی می‌شود. سپس موتور JSP داده‌های فرم را استخراج می‌کند و از آنها برای تنظیم ویژگی‌های bean استفاده می‌کند. این کار بدان‌دلیل امکان‌پذیر است که bean بر اساس قراردادهای کد نویسی جاوا نوشته شده است و متدهای Setکننده‌ای فراهم می‌کند که به نام‌های فیلدهای ورودی در فرم اشاره می‌کنند. سپس موتور JSP، Servlet را اجرا می‌کند که به‌نوبه‌ی خود از bean پرس‌وجو کند و با اشیای کاری مربوطه تعامل می‌نماید تا درخواست را پردازش نموده و پاسخ را ایجاد کند. سپس پاسخ به مشتری فرستاده می‌شود.



شکل ب-۵: شرح جریان کاری JSP

ب-۳-۳- نصب یک سرویس‌دهنده‌ی وب با JSP Container

Apache Tomcat و یا به‌بیان ساده‌تر Tomcat، که یک محصول متن باز از سازمان نرم‌افزاری آپاچی (ASF) است، یک دربردارنده‌ی Servlet بسیار پرکاربرد است. ASF، JSP را پشتیبانی می‌کند و یک محیط سرویس دهنده‌ی وب HTTP از جاوای خالص برای اجرای جاوا را فراهم می‌سازد. Tomcat و دستورات مربوط به نصب آن می‌تواند از http://tomcat.apache.org/ دریافت گردد.

ب-۳-۴- استفاده از صفحات سرور جاوا

این بخش با یک مثال ساده نشان می‌دهد که چگونه از صفحات JSP استفاده کنیم و ویژگی‌های پایه‌ی JSP را نیز شرح می‌دهد. در این مثال یک دانشجو باید با استفاده از دو فیلتر موضوع و ترم برای برنامه‌های تبادل دانشجو جستجو نماید. برنامه‌ی کاربردی مبتنی‌بروب، یک خلاصه از برنامه‌های پوشش دهنده‌ی شرایط جستجو را به‌عنوان خروجی بر می‌گرداند. در ادامه،

گام‌های مربوط به توسعه‌ی این مثال شرح داده می‌شوند:

گام ۱- یک صفحه‌ی HTML حاوی یک فرم ایجاد کنید.

در این گام، یک صفحه‌ی html با یک فرم برای ورود اطلاعات ورودی لازم توسط کاربر ایجاد می‌شود. ویژگی action از فرم، به صفحه‌ی JSP که درخواست را پردازش خواهد کرد، تغییر می‌کند. برای مثال ویژگی method به "Post" تغییر می‌یابد. شکل ب-۶-الف یک نمونه کد html برای این صفحه را نشان می‌دهد.

گام ۲- یک کلاس bean متناظر ایجاد کنید.

در این گام، یک کلاس جاوا، به نام Javabean، ایجاد و یا از فایل html تولید می‌گردد. نام‌های ویژگی‌های bean متناظر با فیلدهای ورودی فرم است. این نام‌ها به کوچکی و بزرگی حروف حساس هستند و از قواعد نام گذاری جاوا که در بخش «استانداردهای کدنویسی» بیان شد، پیروی می‌نمایند. به‌علاوه، bean باید متدهای معمول دریافت‌کننده و مقداردهنده (getter & setter) را که بر اساس قواعد کدنویسی جاوا نوشته شده‌اند را داشته باشد. شکل ب-۶-ب، bean ایجادشده را نشان می‌دهد.

```
<html><head><title>Search  for Oversea Programs</title></head>
 <body><form name="search" action=".../pages/search.jsp" method="post">
  <table><tr><td>Search Criteria</td></tr>
   <tr><td colspan="2">
   <table>
    <tr class="blackheading">
     <td align="right">Subject : </td>
     <td colspan="4" align="left"  class="normaltext2" >
     <input type="text" name="subject“ id=“subject” size="25" maxlength="20"/>
   </td>
   </tr>
   <tr class="blackheading">
  <td align="right">Term : </td>
  <td colspan="4" class="normaltext2" align="left">
  <input type="text" name="term" id="term" size="25" maxlength="45"/>
  </td></tr>
  <tr height="20">
  <td colspan="2" align="center">
  <input type="submit" value="Submit"> 
   <input type="button" value="Cancel">
   </td>
  </tr>
  </table></td>
  </tr>
  </table>
  </form>
 </body>
</html>a
```

(a) HTML page for entering search criteria

```
package beans;
public class SearchBean{
    String subject,term;
    public SearchBean(){...}
    public String getSubject(){
        return subject;
    }
    public void setSubject(String  subject){
        this.subject = subject;
    }
    public String getTerm(){
        return term;
    }
    public void setTerm(String term){
        this.term = term;
    }
}
```

(b) corresponding Java bean

شکل ب-۶: یک فرم و Java bean متناظر آن

گام ۳- یک صفحه‌ی JSP برای پردازش درخواست ایجاد کنید.

در این گام، یک صفحه‌ی JSP برای پردازش درخواست ایجاد می‌شود. صفحه‌ی JSP می‌تواند از bean پرس‌وجو کند و داده‌ی ورودی را پردازش نماید، و یا bean را برای پردازش درخواست به یک شئ کسب‌وکار ارجاع دهد. صفحه‌ی JSP می‌تواند حاوی

Scriptletها و کد html برای پردازش درخواست و نمایش اطلاعات خروجی باشد. شکل ب-۷ یک صفحه‌ی JSP نمونه را نشان می‌دهد که در آن برای شرح منظور از شماره‌ی خطوط استفاده شده است.

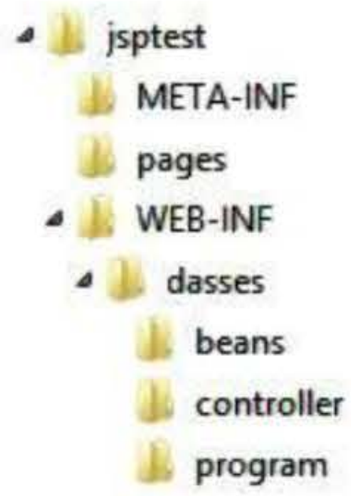
سه خط اول صفحه‌ی JSP کلاس‌های موردنیاز را وارد می‌کند. خط (۴) به موتور JSP می‌گوید که یک رویداد از SearchBean به‌نام searchBean ایجاد نماید و از آن برای ذخیره‌ی داده‌ی ورودی استفاده کند. خط (۵) باعث می‌شود که داده‌ی ورودی همه‌ی فیلدهای فرم در bean ذخیره شود، که با (*) نشان داده‌شده‌است.

ویژگی‌های مستقل می‌توانند با خطوط متعدد SetProperty نشان داده شوند که در هر یک از آنها ستاره با نام ویژگی جایگزین شده است.

خط‌های (۶) و (۷)، scriptletهایی هستند که یک رویداد از SearchController ایجاد می‌کنند و متد search(searchBean) آن را فراخوانی می‌نمایند، که به‌طور طبیعی یک آرایه از اشیای Program برمی‌گرداند. در این حالت، شئ searchBean شرایط جستجو را منتقل می‌کند. به استفاده از یک جفت رشته‌ی %> و <% برای دربرگرفتن کد منبع جاوا در فایل html توجه کنید. خط‌های (۸) تا (۲۳) کد html با scriptlet هستند که کنترل و محاسبات لازم برای ایجاد پاسخ به مشتری را حمل می‌کنند. اپراتور تخصیص با علامت "=" که قبل از یک فراخوانی تابع می‌آید، مانند =p.getName() در خط (۱۷)، باعث می‌شود که مقدار بازگشتی در صفحه‌ی html نشان داده شود.

```
(1) <%@page import = "controller.SearchController"%>
(2) <%@page import = "beans.SearchBean"%>
(3) <%@page import = "program.*"%>
(4) <jsp:useBean id="searchBean" scope="session" class="beans.SearchBean" />
(5) <jsp:setProperty name="searchBean" property="*"/>
(6) <%SearchController sc=new SearchController();
(7)     Program[] programs=sc.search(searchBean);%>
(8) <html><head> <title>Search Result</title></head>
(9) <body><h2>Search Results</h2><hr width="90%" />
(10) <%if(programs == null || programs.length == 0 ){%>
(11) <p>No programs found, please <a href="../pages/search.html">try again</a>.</p>
(12) <% } else { %>
(13) <table wodth="90%" class="program" cellpadding="0" cellspacing="0">
(14) <tr> <td> Program Name </td><td> Description </td> <td> Housing </td><td> Fees </td> </tr>
(15) <% } for(int i=0; i<programs.length; i++){
(16)       Program p=programs[i]; %>
(17)       <tr> <td><%=p.getName()%> </td> <td><%=p.getDescription()%></td>
(18)       <td><%=p.getHousing()%> </td><td> <%=p.getFees()%></td></tr>
(19) <%}%>
(20) </table>
(21) <a class="back_link relink" href="../pages/search.html">&lt;&lt; Back to Search</a>
(22) </body>
(23) </html>
```

شکل ب-۷: یک صفحه‌ی JSP برای پردازش یک درخواست و نمایش خروجی



شکل ب-۸: ساختار کشویی برای یک فایل war.

گام ۴- سایر کلاس‌ها را پیاده‌سازی کنید.

در این گام، سایر کلاس‌های مرتبط، مانند SearchController و Program پیاده‌سازی می‌شوند.

گام ۵- برنامه‌ی کاربردی وب را بسازید و مستقر نمایید.

در این گام، کلاس‌های bean و کلاس‌های اشیای کسب‌وکار به byteCode کامپایل می‌شوند. برای استقرار برنامه‌ی کاربردی وب، فایل‌های byteCode، فایل‌های html، فایل‌های JSP، و فایل‌های تصویر در کشوهای مناسب مرتب می‌شوند (شکل ب-۸). فایل‌های حاوی این کشوها دسته بندی می‌شوند تا یک فایل آرشیو وب، مثلاً jsptest.war ایجاد کنند. یک IDE مانند eclipse یا NetBeans به‌طور خودکار فایل war را پس از کامپایل فایل‌های Java ایجاد می‌نماید.

سپس فایل jsptest.war بسته به‌دستورات داده‌شده توسط تولیدکننده‌ی وب سرور و موتور JSP، در پوشه‌ی مناسب قرار داده می‌شوند. برای Tomcat، فایل باید در کشوی webapps زیر کشویی که Tomcat در آن نصب شده، قرار بگیرد. به‌عنوان مثال:

C:\...*Tomcat*6.0*webapps**jsptest.war*

در پایان، وب سرور با توجه به دستورات داده شده توسط تولیدکننده، باید اجرا شود و شروع به‌کار می‌کند.

گام ۶- مثال را اجرا کنید.

یک مرورگر وب را باز کنید و در فیلد آدرس آن را وارد کنید:

“http://localhost:8080/jsptest/pages/search.html”

سپس کلید Enter را فشار دهید. یک صفحه‌ی جستجو نشان داده می‌شود. موضوع و عبارت مورد جستجو را وارد کنید و Submit را کلیک نمائید. اگر پیغام خطایی نشان داده نشود، مرورگر باید یک فهرست از برنامه‌هایی که شرایط جستجو را برآورده می‌کنند، نشان دهد.

.

پ

ابزارهای نرم‌افزاری

این پیوست، برخی از ابزارهای نرم‌افزاری مورد استفاده در زمان پیاده‌سازی و تست را معرفی می‌نماید . این ابزارها شامل NetBeans ، Juint، Cobertura، و Emma می‌باشد.

پ-۱ NetBeans

بسیار دشوار است که از یک ویرایشگر متن برای نوشتن برنامه‌های جاوا، و از دستورات Javac و Java برای کمپایل و اجرای برنامه‌های جاوا از اعلان دستور استفاده شود. با استفاده از محیط‌های توسعه‌ی جامع [۱] بر این مشکلات غلبه می‌گردد. یک IDE یک ابزار نرم‌افزاری است که از انواع مختلفی از فعالیت‌های توسعه‌ی نرم‌افزار شامل تحلیل، طراحی، پیاده‌سازی، کمپایل، اجرا، آزمون، عیب‌یابی، تضمین کیفیت نرم‌افزار، مدیریت پیکربندی نرم‌افزار و مدیریت پروژه پشتیبانی می‌نماید. به‌عنوان یک مثال، شکل پ-۱ نمایی از نسخه NetBeans 6-5-1 را نشان می دهد. NetBeans یک IDE منبع باز است که تحت لیسانس CDDL[۲] و GPL[۳] می باشد. این ابزار کاملا با جاوا نوشته شده است و به همین دلیل می تواند روی بسترهای مختلف اجرا گردد.

پس از نصب، کاربر برای اجرای NetBeans دو بار بر روی آیکون آن کلیک مینماید. اولین کاری که باید انجام شود، ایجاد یک پروژه است. این کار با انتخاب File و سپس انتخاب New Project انجام می‌گردد. یک جعبه‌ی گفت‌وگو باز می‌شود که به کاربر اجازه می‌دهد دسته و نوع پروژه را انتخاب نماید. برای ایجاد یک پروژه‌ی جاوا، کاربر در فهرست Categories بر روی Java ، و سپس در فهرست Projects بر روی Java Application کلیک می‌نماید. سپس کاربر برای پرکردن اطلاعات خاص پروژه بر روی دکمه Next کلیک می‌نماید. پنجره‌ی نشان‌داده‌شده در شکل پ-۱ یک پنجره‌ی معمول IDE است. این پنجره دارای سه پنل پروژه ، ویرایشگر و پیمایشگر می‌باشد. پنل پروژه در بالاترین گوشه سمت چپ ناحیه‌ی اصلی دارای سه زبانه است. زبانه‌ی Projects به کاربر امکان می‌دهد که بسته‌ها و کتابخانه‌های هر پروژه را مشاهده نماید. زبانه Files به کاربر اجازه می‌دهد که فایل‌های هر پروژه را مشاهده کند. این فایل‌ها شامل فایل‌های منبع، فایل‌های Class، و فایل های xml است که ویژگی‌های متفاوتی از پروژه را شرح می‌دهند. زبانه‌ی Services، سرویس‌هایی که هر کاربر می‌تواند برای دسترسی به پایگاه‌داده‌ها و سرویس‌های وب مورد استفاده قراردهد، را نشان می‌دهد.

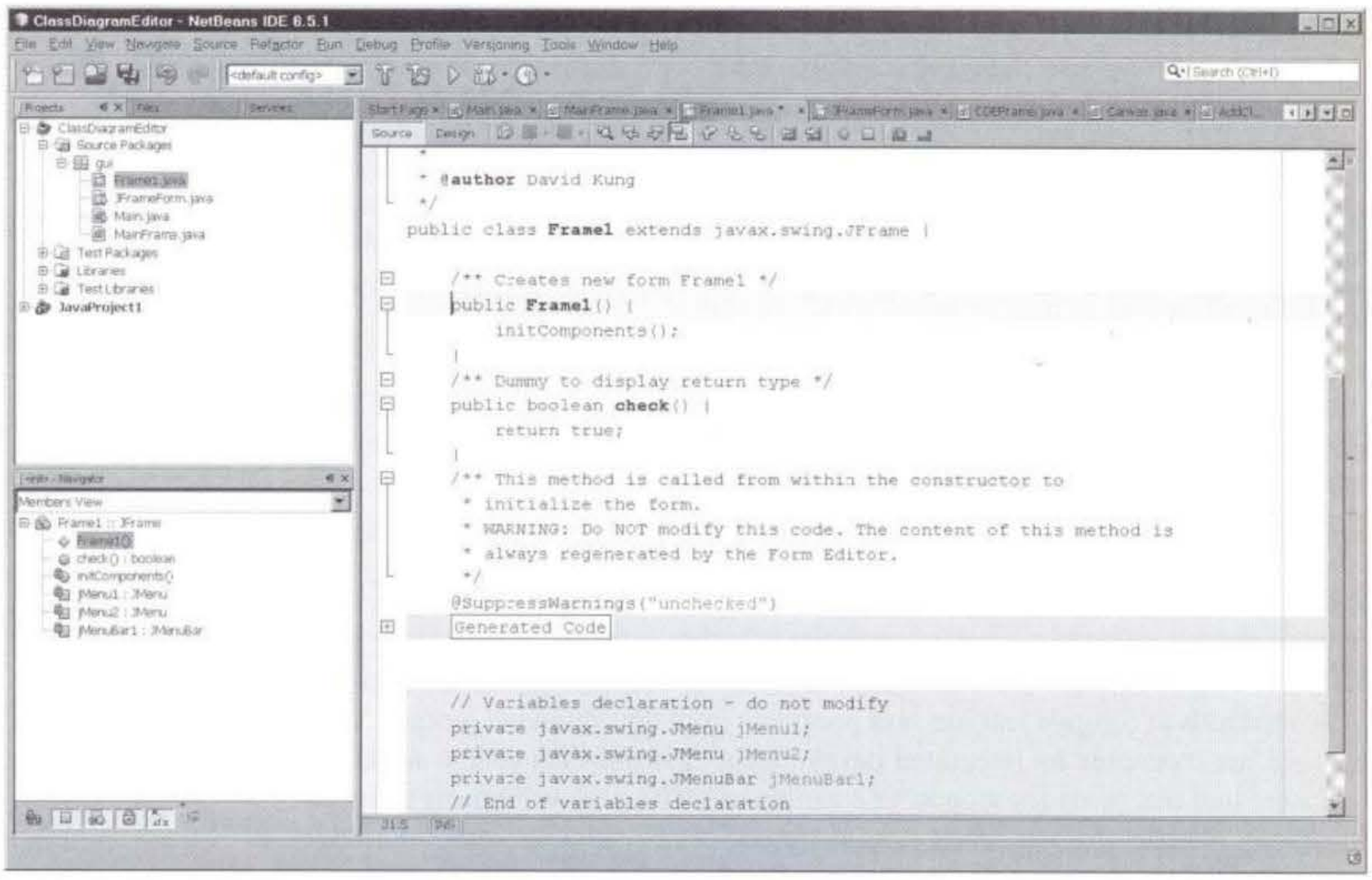
---

[۱] Integrated Development Environments (IDE)

[۲] Common Development Distribution License

[۳] General Public License

پنل سمت چپ پایین، توابع و صفات کلاس انتخاب‌شده در پنل بالای سمت چپ را نشان می‌دهد. به‌علاوه انواع برگردانده‌شده از توابع و انواع صفت‌ها نیز نمایش داده می‌شود. اگر کاربر روی یک تابع یا صفت دوبار کلیک کند، نشانگر در پنل ویرایشگر به آن تابع یا صفت منتقل خواهد شد تا کاربر بتواند روی آن ویژگی کار کند. ویرایشگر دارای خصوصیات کمکی بسیاری است که برنامه‌نویسی را بسیار ساده‌تر و دلپذیرتر می‌سازد. برای مثال، کاربر می‌تواند روی دکمه‌ی Run کلیک کند و Build Main Project ، Run Main Project ، و Test Project را انتخاب کند تا پروژه را کمپایل، اجرا و یا تست نماید. معرفی بیشتر IDE فراتر از محدوده‌ی این کتاب است. خوشبختانه IDEهای بسیاری در زمان دانلودکردن، یک خودآموز ساده نیز در اختیار کاربر قرار می‌دهند تا بتواند از آن برای آشنایی بیشتر با IDE استفاده نماید.



شکل پ–۱: نواحی کاری معمول در NetBeans

پ–۲ استفاده از JUnit

JUnit عضوی از خانواده‌ی XUnit از ابزارهای آزمون است که برای آزمون واحد و آزمون بازگشت برنامه‌های جاوا ساخته شده است. اعضاي ديگر خانواده‌ي XUnit، شامل CUnit براي C، CppUnit براي C++، NUnit براي .Net، HtmlUnit براي html و HttpUnit براي http مي‌باشند. در اين بخش Junit و در يكي از بخش‌هاي آينده HttpUnit معرفي مي‌گردد. ابزارهاي ديگر نيز مشابه هستند و آشنايي با آنها به خواننده واگذار مي‌شود. JUnit یک چارچوب آزمون واحد متن باز برای جاوا است که توسط توسط اریک گاما و کنت بک نوشته شده است. این چارچوب، پیاده‌سازی و اجرای موارد کاربرد و بررسی نتایج آزمون را بسیار ساده می‌نماید. وقتی موارد آزمون پیاده‌سازی شدند، می‌توانند دوباره و دوباره اجرا شوند و این‌کار باعث خودکارسازی فرایند آزمون بازگشت می‌گردد. پیاده‌سازی و اجرای موارد کاربرد در JUnit شامل شش گام ساده است. مثال پاک‌سازی که در

فصل‌های قبل شرح داده شد، برای شرح این گام‌ها مورد استفاده قرار می‌گیرد. به‌طور خاص، چهار مورد آزمون به‌دست‌آمده در بخش ۲۰-۳-۱ پیاده‌سازی می‌شوند. برای استفاده از JUnit، باید تابع ()purge در کلاس Purge کپسوله[1] شود که یک لیست LinkedList را به‌عنوان یک عضو داده شامل می‌شود.

```
package test.example;
import junit.framework.*;
import java.util.LinkedList;
import example.*;
public class PurgeTest extends TestCase {
  Purge purge1;
  public PurgeTest(String name){super(name);}
  public void setUp(){purge=new Purge();}
  public void tearDown() {} //do nothing
  public void testEmptyList() {
   purge1.purge(); int size=purge1.getList().size();
   assertTrue("output list size: expected==0, actual=="+size,
      size=0);
 }
 public void testSingleItem(){
  purge1.getList().add(new Item("item1"));
  purge1.purge(); int size=purge1.getList().size();
  assertTrue("output list size: expected==1,actual=="+size,
     size==1);
 String items = purge1.getList().toString();
  assertTrue("output list: expected==item1,actual=="+items,
     items.compareTo("item1")==0);
 }
```

```
public void testNoDuplicate(){
 LinkedList list = purge1.getList();
 list.add(new Item("item1"));
 list.add(new Item("item2"));
 purge1.purge();int size=purge1.getList().size();
 assertTrue("output list size: expected==2,
    actual=="+size , size==2);
 String items = purge1.getList().toString();
 assertTrue("output list: expected==item1,item2,
   actual=="+items,items.compareTo("item1,item2")
   ==0);
 }
public void testDuplicate(){
 LinkedList list=purge1.getList();
 list.add(new Item("item1"));
 list.add(new Item("item1"));
 purge1.purge(); int size=purge1.getList().size();
 assertTrue("output list: expected==1,
  actual=="+size, size==1);
 String items = purge1.getList().toString();
 assertTrue("output list: expected==item1,
  actual=="+items,items.compareTo("item1")==0);
}
public static Test suite(){
  return new TestSuite(PurgeTest.class);
 }
}
```

شکل پ-۲: کلاس TestCase برای آزمودن Purge

گام ۱- یک زیر کلاس از TestCase ایجاد کنید.

همه‌ی موارد آزمون یک کلاس، در کلاسی کپسوله شده است که کلاس TestCase را توسعه می‌دهد. کلاس TestCase توسط چارچوب JUnit فراهم شده است. این موضوع در شکل پ-۲ نشان داده شده است که در آن یک کلاس به نام PurgeTest به‌عنوان زیرکلاسی از TestCase ایجاد شده است. کلاس PurgeTest رویداد Purgel از Purge را تعریف می‌کند. براساس قواعد نام‌گذاری JUnit، این یک «ثابت آزمون» نامیده می‌شود. این ثابت، مربوط به کلاس تحت آزمون[2] است و دسترسی موارد آزمون به CUT را تسهیل می‌نماید.

گام ۲- ()setUp و ()teardown قبلی را لغو (Override) کنید.

توابع ()setUp و ()teardown، توابعی از کلاس TestCase هستند. زیرکلاس‌ها باید این دو تابع را لغو کنند و setUp و tearDown خاص آزمون پیاده‌سازی نمایند. وقتی Junit موارد آزمون را اجرا می‌کند، تابع ()setUp قبل از فراخوانی هر یک از متدهای آزمون که موارد آزمون را پیاده‌سازی می‌کنند، فراخوانی می‌شود. تابع ()tearDown پس از اجرای هر متد آزمون فراخوانی می‌شود. در این مثال، متد ()SetUp، به‌سادگی ثابت آزمون را با یک رویداد جدید از Purge مقداردهی مجدد می‌کند. به‌این‌ترتیب، هرمتد آزمون، کپی خود از CUT را تست می‌کند.

گام ۳- موارد آزمون را پیاده‌سازی کنید.

---

[1] Encapsulate

[2] Class Under Test (CUT)

در JUNIT، موارد آزمون با توابعی پیاده‌سازی می‌شوند که براساس قانون نام‌گذاری JUNIT نام‌گذاری شده‌اند. یعنی نام هر تابع با یک «test» شروع می‌شود، مانند test EmptyList()، test SingleItem()، test NoDuplicate()، و test Duplicate() که در شکل پ-۲ نشان داده شده است. این مثال، چهار مورد آزمونی که در بخش ۲۰-۳-۱ شرح داده شد را پیاده‌سازی می‌کند. هر یک از این تابع‌ها، لیست ورودی را مقداردهی اولیه می‌کند، تابع Purge() را فراخوانی می‌کند، ولیست نتایج را بررسی می‌کند. برای مثال، متد test Single Item() یک عضو به‌لیستی که در ابتدا خالی بوده اضافه می‌کند، متد purge() را فراخوانی می‌کند، و سپس بررسی می‌کند که لیست تغییر نیافته باشد. این بررسی با دو جمله‌ی assert True انجام می‌شود.

تابع assert True (string message) ، Boolean Condition وقتی که شرط بولی آن « نادرست» ارزیابی شود، یک پیام چاپ می‌کند. اولین جمله از این دو جمله، اندازه ی لیست را بررسی می‌کند و اطمینان می‌یابد که این اندازه، یک باشد. دومین جمله اطمینان می‌یابد که تنها المان لیست، «iteml» نامیده شده باشد . علاوه بر assertTrue(...)، توابع متعدد چندریختی دیگری می‌تواند مورد استفاده قرارگیرد. به‌عنوان مثال، assertEqual(...)، assertFalse(...)، assertNull(...)، و مواردی مشابه این قابل‌استفاده اند. صفحه‌ی API JavaDoc برای کلاس Assert این توابع را به‌تفصیل شرح می‌دهد.

گام ۴ –یک متد عمومی و ایستای Test suite() بنویسید.

این متد مشخص می‌کند که کدام یک از متدهای آزمون پیاده‌سازی‌شده در گام ۳، باید توسط اجراکننده‌ی آزمون که توسط Junit فراهم شده است، اجرا گردد. دو راه برای انجام این کار وجود دارد : (۱) روش دستی به ما اجازه می‌دهد که انتخاب کنیم کدام مورد آزمون اجرا گردد، و (۲) روش خودکار به اجراکننده‌ی آزمون اجازه می‌دهد که همه‌ی موارد آزمون را اجرا نماید. متد suite() در شکل پ-۲ روش خودکار را پیاده‌سازی می‌کند. اگر اجرای همه‌ی آزمون‌ها وقت‌گیر یا هزینه‌بر باشد، روش دستی ترجیح داده می‌شود. در این حالت، متد suite() به‌شکل زیر پیاده‌سازی می‌شود. این متد، یک رشته آزمون تهی ایجاد می‌کند و مواردآزمون دلخواه را با فراخوانی متد addTest رشته آزمون اضافه می‌نماید. سپس رشته آزمون به فراخواننده بازگردانده می‌شود.

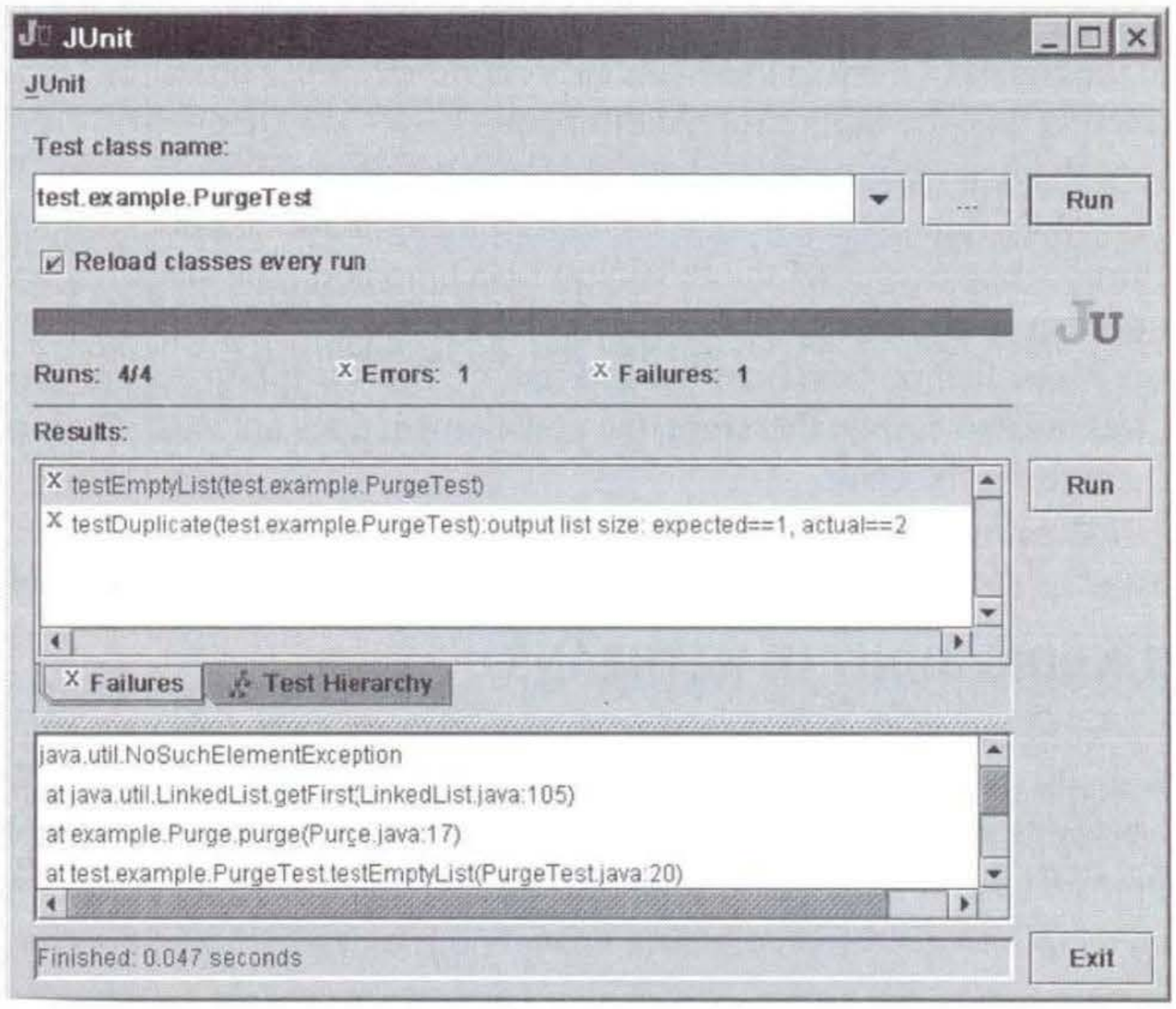
```
Public static Test suite()  {
        TestSuite suite= new TestSuite ();
        Suite.addTest (new PurgeTest ( testEmptyList  )  {
                Protected void runTest ()
                {  testEmtyList ();  }
        }};
        // other suite.addTest ( ) to add more test cases
        Return suite;
}
```

گام ۵- موردهای آزمون و CUT را کامپایل کنید.

قبل از اجرای اجراکننده‌ی آزمون، باید موردهای آزمون و CUT کامپایل گردد. این کار مانند کامپایل کردن سایر برنامه‌های جاوا انجام می‌شود، به‌جز این‌که در این‌جا فایل junit.jar باید در مسیر کلاس اضافه شود و فایل(های) جاوای موارد آزمون باید به فایل‌های مبدایی که باید کامپایل شود افزوده گردد. استفاده از یک IDE کار را بسیار ساده خواهد کرد، زیرا دیگر نیازی به تایپ‌کردن مسیر کلاس و فایل‌های مبدا نیست.

گام ۶- موردهای آزمون را اجرا کنید.

سه اجرا کننده‌ی آزمون وجود دارد:



شکل پ-۳: واسط کاربری اجراکننده‌های آزمون JUnit

۱. junit.awtui.TestRunner
۲. junit.swingui.TestRunner
۳. junit.textui.TestRunner

هریک از این‌ها یک کلاس جاوا هستند و بنابراین می‌توانند مانند هر برنامه‌ی جاوای دیگری اجرا شوند. این اجراکننده‌های آزمون همان‌طور که در نام بسته‌ها متفاوتند، واسط‌های کاربری متفاوتی را نیز از خود نشان می‌دهند. شکل پ-۳ واسط کاربری اجراکننده‌ی آزمون Swing را پس از اجرای موارد کاربردی که در شکل پ-۲ نشان داده شده است نمایش می‌دهد. واسط کاربری اجراکننده‌ی آزمون awt مشابه است و واسط کاربری text، حاصل اجرا را روی خروجی استاندارد چاپ می‌کند. واسط کاربری text یکپارچه‌سازی JUnit با یک ابزار دیگر را ساده می‌نماید. هم‌چنین این واسط، در زمانی که می‌خواهید واسط گرافیکی خود را برای نمایش حاصل اجرا پیاده‌سازی کنید، مفید است. به عنوان مثال، نمایش نتایج به صورت یک نمودارگرد را ساده می‌کند.

JComboBox و browse JButton، در بالای Swing GUI به کاربر امکان می‌دهد که رشته‌ی آزمون دلخواه را برای اجرا انتخاب کند. پس از انتخاب، کاربر دکمه‌ی Run را برای اجرای آزمون کلیک می‌نماید. برای مثال، GUI نشان می‌دهد که test.example.PurgeTest انتخاب شده است. اجرای چهار آزمون، یک خطا و یک شکست را ایجاد می‌کند. نوار پیشرفت افقی بلند، قرمز رنگ می‌شود تا نشان دهد که خطا و شکستی رخ داده است. اگر هیچ خطا یا شکستی تشخیص داده نشود، این نوار سبز خواهد بود. جزییات خطا و شکست در Results JScrollPane نشان داده‌شده‌است. این بدان معنی است که خطا در زمان اجرای متد testEmptyList() و شکست در زمان اجرای متد testDuplicate(...) رخ داده است.

در JUnit، شکست‌ها شرایط شکست‌خورده‌ی پیش‌بینی‌شده هستند، یعنی یک آزمون به‌دلیل آن‌که یک assert True(…) به «نادرست» ارزیابی شده، با شکست مواجه شده‌است. اگر مورد آزمون به‌درستی طراحی و پیاده‌سازی شده باشد، شکست نشان می‌دهد که یک خطا در نرم‌افزار شناسایی شده‌است. برای بررسی این مطلب، اجازه دهید که CUT را که تابع purge از کلاس Purge است، امتحان کنیم. خط ۵ و ۶ در شکل ۶-۲۰-الف، مسئول حذف اجزای تکراری است. خط ۵ دو مرجع p و q را به جای نام دو فقره‌ای که به آن ارجاع می‌کنند، مقایسه می‌کند. این‌کار باعث شکست می‌شود. پس از اصلاح این مشکل، شکست باید ناپدید شود. خطاها شرایط پیش‌بینی‌شده‌ای هستند که استثناهای زمان اجرا را ایجاد می‌کنند (مانند تقسیم بر صفر). خطای ایجادشده توسط اجرای رشته‌ی آزمون Purge Test، یک استثنای NoSuchElement است که در زمان اجرای متد testEmptyList() رخ می‌دهد. در بخش پایینی Swing GUI، حاصل ردگیری پشته دیده می‌شود که نشان می‌دهد محل دقیق بروز خطا در خط ۱۷ کلاس Purge است: "p=(Item)list.getFirst()". استثنا به این دلیل رخ می‌دهد که لیست به شکل خالی ایجاد شده است، پس جزء اول وجود ندارد. اصلاح این مشکل، خطا را برطرف خواهد کرد.

پ-۳ اجرای JUINT در NetBeans

اجرای JUnit در NetBeans، یک راه مناسب برای توسعه‌ی آزمون-رانده(TDD) است. اینکار، با اجرای سایر برنامه‌های کاربردی جاوا در NetBeans تفاوتی ندارد. گام‌های زیر چگونگی پیکربندی NetBeans برای اجرای اجراکننده‌ی آزمون JUnit را شرح می‌دهد:

۱. روی Run > Set Project Configuration > Customize… کلیک کنید.
۲. در جعبه‌گفت‌وگوی Propject Properties که نشان داده می‌شود، درفهرست Categories، گزینه‌ی Run را انتخاب کنید.
۳. در فیلد متنی Main Class، اجراکننده‌ی آزمونی که می‌خواهید استفاده کنید را وارد نمایید، به عنوان مثال، "junit.swingui.TestRunner".
۴. در فیلد متنی Arguments، نام کامل زیرکلاس TestCase را وارد کنید، به عنوان مثال، "test.example.PurgeTest".
۵. بر روی OK کلیک کنید.

برای این که JUnit، موارد آزمون را در NetBeans اجرا کند، آیکون Run را از نوار ابزار کلیک کنید. واسط گرافیکی اجراکننده ی آزمون نشان داده می شود و نتیجه ی اجرا نمایش داده می شود.

پ-۴ ابزار پوشش Emma

همان‌طور که در شکل ۲۰-۱۱ نشان داده شده است، یکی از گام‌های فرایند آزمون، بررسی پوشش آزمون است. یکی از پوشش‌های معمول مورد بررسی، پوشش کد است. ابزارهای بسیاری برای بررسی پوشش کد وجود دارد. برای مثال، Emma به‌عنوان افزونه‌ای برای NetBeans نصب و فعال‌سازی می‌شود و اطلاعات مربوط به پوشش کلاس، پوشش خط، و پوشش بسته را تولید می‌نماید. برای نصب و فعال‌سازی Emma در NetBeans، گام‌های زیر را انجام دهید:

۱. افزونه را نصب کنید: از منونی Tools، Plugins را انتخاب کنید. در فهرست افزونه‌های موجود، گزینه‌ی Code Coverage Plugin را علامت بزنید. Install را کلیک کنید و طبق دستورات نصب عمل کنید.
۲. افزونه‌ی پوشش کد را برای یک پروژه فعال کنید: در بخش Projects، روی نام پروژه کلیک راست کنید و از بخش Coverage، Activate Coverage Collection را انتخاب نمایید. ابزار، فعال می‌شود.

۳. موردهای آزمون را با JUnit در NetBeans اجرا کنید: مانند بخش قبل عمل کنید.
۴. پوشش آزمون را مشاهده کنید: روی نام پروژه کلیک راست کنید و از بخش Coverage، Show Project Coverage Statistics را انتخاب کنید. برای آن که ببینید که کدام خط‌های برنامه آزموده می‌شوند و کدام خط‌ها آزموده نمی‌شوند، در بخش Project روی نام کلاس دوبار کلیک کنید. خطوط سبز کاملاً پوشش داده شده‌اند، خطوط زرد نسبتاً پوشش داده شده‌اند، و بقیه پوشش داده نشده‌اند.

پ-۵ ابزار پوشش Cobertura

بسیاری از شرکت‌های نرم‌افزاری، به پوشش شاخه به‌عنوان یک معیار پوشش آزمون واحد نیاز دارند. به‌هرحال، در زمان نگارش این کتاب، Emma از پوشش شاخه پشتیبانی نمی‌کند. برای پوشش شاخه، می‌توان از نرم‌افزار رایگان Cobertura استفاده کرد. این ابزار، پیچیدگی سیکلوماتیک را نیز محاسبه می‌نماید. این نرم‌افزار را می‌توان از http://cobertura.sourceforge.net/ دانلود کرد. نصب آن به‌راحتی، با باز کردن آن در یک پوشه‌ی خالی انجام می‌شود. قبل از اجرای Cobertura در NetBeans، باید فایل‌های اسکریپت سازنده را نصب کنید:

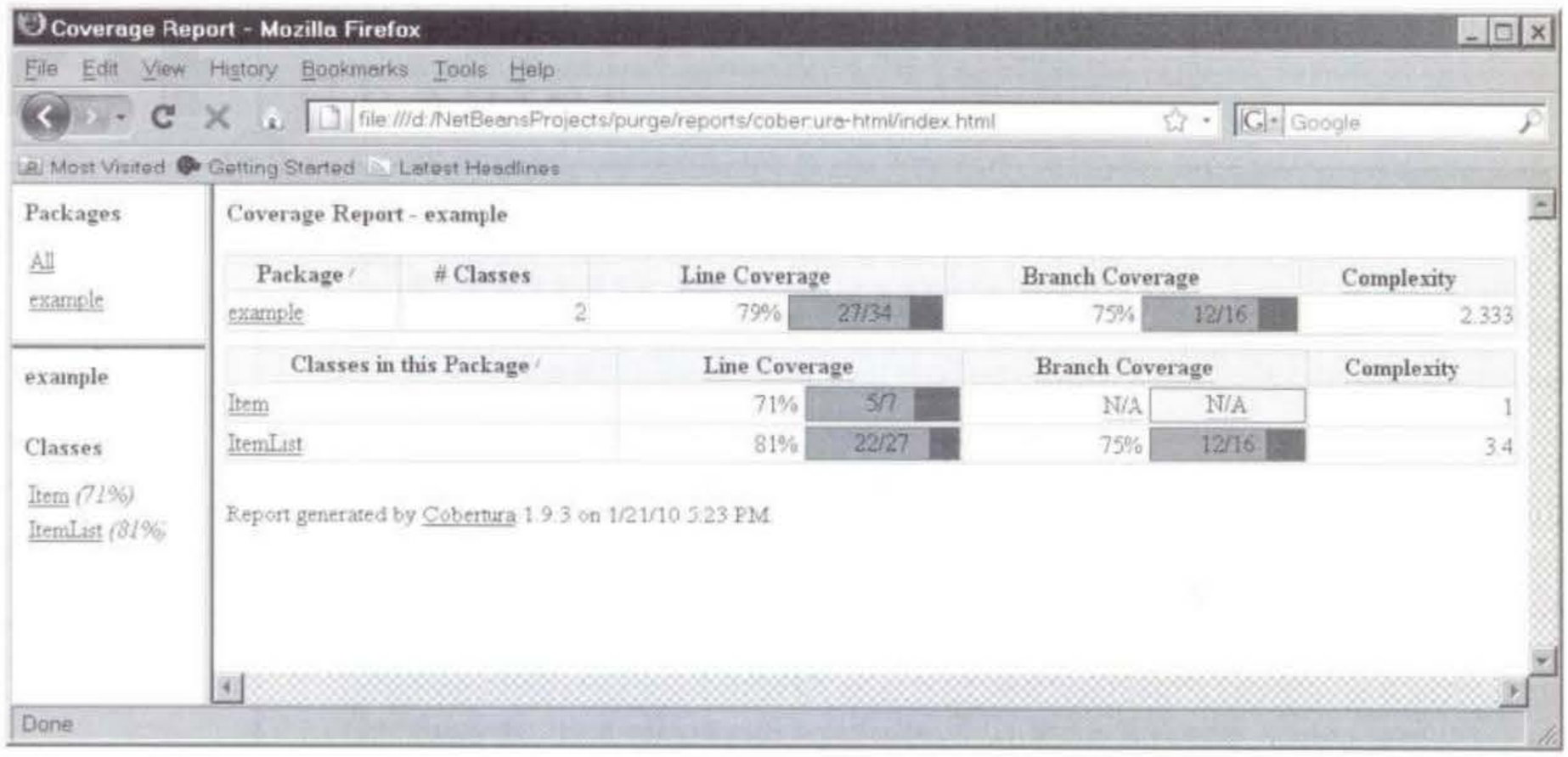
۱. فایل‌های Build.xml و build.properties را در پوشه‌ی cobertura-1.9.3/examples/basic کپی کنید و نام آنها را به‌ترتیب، cobertura-build.xml و cobertura-build.properties بگذارید.
۲. فایل‌های تغییرنام‌یافته را برداشته و به پوشه‌ی پروژه‌ی NetBeans خود منتقل کنید.
۳. با استفاده از یک ویرایشگر متن، فایل cobertura-build.properties را باز کنید. در سطر Cobertura.dir=../..، ”../..“ را با مسیر حاوی فایل cobertura.jar جایگزین کنید. این باید مسیر پوشه‌ی cobertura-1.9.3 باشد. فایل به‌روزشده را ذخیره کنید.

یک گام مهم دیگر، انتقال فایل‌های جاوای مورد آزمون خود از پوشه‌ی project test به پوشه‌ی project src است. این کار به دلیل آن است که Cobertura تنها پوشه‌ی src برای CUT و فایل‌های منبع مورد آزمون را بررسی می‌کند. شما می‌توانید برای جلوگیری از انتقال فایل‌های آزمون، فایل cobertura-build.xml را تغییر دهید.

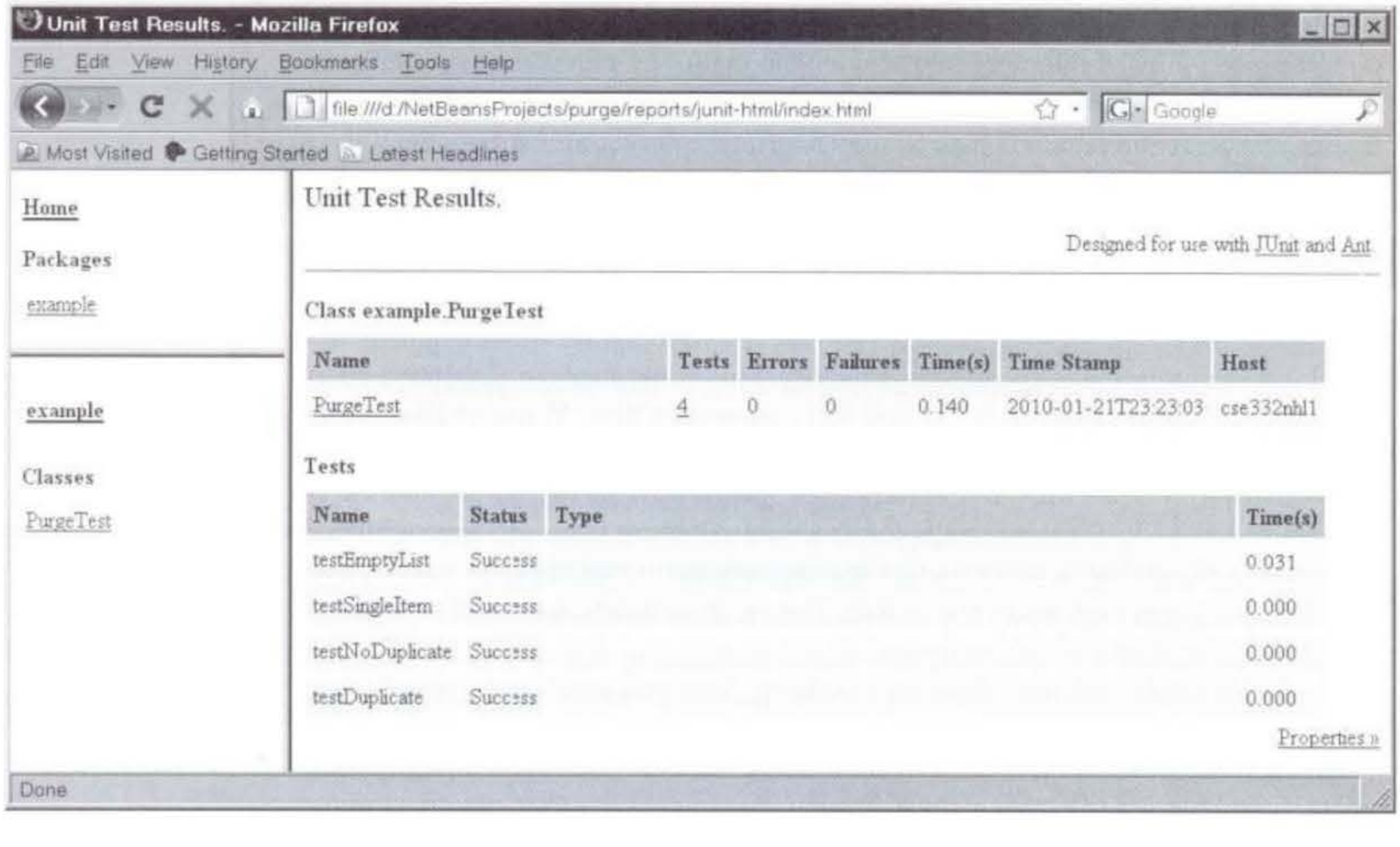
شما باید CUT و موردهای آزمون را کامپایل کنید، از بایت‌کد CUT استفاده کنید، آزمون را اجرا کنید، و اطلاعات مربوط به پوشش آزمون را جمع‌آوری نمایید. همه‌ی این کارها به‌سادگی قابل انجام است: در نمای File، فایل coberturta-build.xml را باز می‌کنید، روی وظیفه‌ی main کلیک راست می‌کنید و Run Target را انتخاب می‌نمایید. حال باید بتوانید اجرای هر وظیفه و وضعیت اجرا را مشاهده نمایید.

اگر همه‌ی وظایف Cobertura با موفقیت انجام شود، می‌توانید اطلاعات پوشش را در پوشه‌ی گزارش‌ها، زیر پوشه‌ی پروژه ببینید. برای مثال، می‌توانید پوشه‌ی cobertura-html را باز کنید تا فهرستی از فایل‌های html مشاهده نمایید. اگر روی فایل index.html کلیک راست کنید و view را از منوی بازشده انتخاب کنید، مانند شکل پ-۴ گزارش پوشش را خواهید دید.

اگر عملیات مشابهی را برای فایل index.html زیر پوشه‌ی junit-html انجام دهید، نتایج آزمون واحد را که در شکل پ-۵ نشان داده شده است، مشاهده می‌نمایید.



شکل پ-۴: گزارش نمونه‌ای از پوشش Cobertura



شکل پ-۵: مشاهده‌ی نتایج تست واحد در NetBeans

پ-۶ آزمون برنامه‌های کاربردی تحت وب با HttpUnit

HttpUnit گسترش‌یافته‌ی JUnit برای برنامه‌های کاربردی تحت وب است. این ابزار از رفتار مرورگر تقلید می‌کند و اجازه می‌دهد که یک مورد آزمون، درخواست‌هایی به یک وب‌سرور ارسال کند و پاسخ‌های آن‌را دریافت نماید. اجزای یک صفحه‌ی وب، مانند فرم‌ها، جدول‌ها، و پیوندها به‌عنوان اشیا در نظر گرفته می‌شوند. این کار پیاده‌سازی و اجرای موارد آزمون، و تحلیل نتایج آزمون را ساده‌تر می‌کند. این بخش به اختصار نشان می‌دهد که چگونه برای این کارها از HttpUnit استفاده شود.

پ-۶-۱- پیکربندی یک IDE برای استفاده از HttpUnit

HttpUnit را می‌توانید از http://sourceforge.net دانلود کنید. برای نصب، به‌سادگی فایل دانلود شده را در یک پوشه‌ی خالی باز نمایید. سپس یک IDE، مانند NetBeans را اجرا کنید و یک پروژه‌ی جاوا ایجاد نمایید. برای استفاده از HttpUnit در این پروژه، فایل‌های jar موجود در پوشه‌های jars و lib را به کتابخانه‌ی پروژه اضافه کنید. با NetBeans کارهای زیر را انجام دهید:

۱. در نوار منو، روی Run، سپس Set Project Configuration، و سپس Customize کلیک کنید.
۲. در جعبه‌گفتگوی Project Properties که ظاهر می‌شود، در جعبه‌ی Categories بر روی Libraries کلیک کنید و سپس در میانه‌ی جعبه گفتگو، زبانه‌ی Compile را انتخاب نمایید.
۳. در زبانه‌ی Compile، روی دکمه‌ی Add Jar/Folder در سمت راست کلیک کنید و سپس همه‌ی فایل‌های jar را از پوشه‌ای که فایل ابزار را در آن باز کرده بودید، به پوشه‌های jars و lib اضافه نمایید.
۴. Ok را بزنید. پس از آن‌که فایل‌های jar به ابزار اضافه شد، نصب کامل می‌شود.

پ-۶-۲- پیاده‌سازی موردهای آزمون در HttpUnit

پیاده‌سازی موردهای آزمون در HttpUnit مشابه JUnit است، ولی با آن در پیاده‌سازی متدهای آزمون متفاوت است. این موضوع با یک مثال شرح داده می‌شود. مثال در مورد نوشتن موردهای آزمون HttpUnit برای آزمودن یک صفحه‌ی JSP logon است.

فرض کنید که یک کاربر وب برای ورود به یک برنامه‌ی کاربردی تحت وب به یک صفحه‌ی logon.html دسترسی دارد. پس از رفتن به این صفحه، کاربر نام کاربری و کلمه‌ی عبور را وارد می‌کند و روی دکمه‌ی Submit کلیک می‌کند تا فرم را به یک صفحه‌ی logon.jsp بفرستد. اگر نام کاربری و کلمه‌ی عبور صحیح بودند، وب‌سایت، صفحه‌ی خانگی شخصی کاربر را نمایش می‌دهد، درغیر این‌صورت، کاربر را دوباره به صفحه‌ی logon.html ارجاع می‌دهد. مورد آزمون در شکل پ-۶، کاربرد HttpUnit برای آزمودن logon.jsp را نشان می‌دهد. این شکل، تنها حالت ورود معتبر کاربر را نمایش می‌دهد. سناریوهای ورود نامعتبر نیز باید به همین‌ترتیب آزموده شوند.

ابتدا، یک WebConversation در خط ۷ ایجاد می‌شود تا رفتار مرورگر را تقلید نماید. در خط ۸، یک شئ WebRequest برای دسترسی به صفحه‌ی logon.html ایجاد می‌شود. فرض می‌شود که برنامه‌ی کاربردی تحت وب با استفاده از درگاه ۸۰۸۰ روی میزبان محلی اجرا می‌شود و صفحه‌ی logon.html روی پوشه‌ی اصلی آن قرار گرفته است. در زمان آزمودن یک کاربرد واقعی، رشته‌ی url باید با آدرس واقعی صفحه‌ی مورد درخواست جایگزین شود. در خط ۱۰، درخواست به وب‌سرور فرستاده می‌شود و یک شئ WebResponse از آن دریافت می‌گردد. حال می‌توان از طریق شئ پاسخ به فرم ورود دسترسی داشت و در خط‌های ۱۱ تا ۱۳ نام کاربری و کلمه‌ی عبور را تنظیم کرد. سپس فرم به سرور فرستاده می‌شود و یک شئ WebResponse

دیگر در خط ۱۴ از سرور دریافت می‌شود. در خط ۱۵، بررسی می‌شود که آیا پاسخ حاوی رشته‌ی "Welcome" به همراه testUserName هست یا خیر، تا وارسی شود که صفحه‌ی خانگی شخصی بازگشته است.

برای اجرای مورد آزمون در NetBeans، فقط باید همان‌گونه که در بخش پ-۳ شرح داده شد، Logon Test به عنوان کلاس اصلی تنظیم شود. در هرحال، کد نشان‌داده‌شده در شکل پ-۶ کامل نیست. این کد باید کامل شود و سرور باید قبل از کامپایل و اجرای مورد آزمون، کاملاً تنظیم گردد.

```
(01)    Package webtest:
(02)    Import junit.framework.*
(03)    Import com.meterware.httpunit.*
(04)    Public class LogonTest extends TestCase  {
(05)      Public void testLogon()  {
(06)        try  {
(07)          WebConversation wc=new WebConversation();
(08)          WebRequest request = new GetMethodWebRequest
(09)             (“http://localhost:8080/sams/logon.html”);
(10)          WebResponse response = wc.sendRequest(request);
(11)          WebForm form=response.getFormWithName(“logonform”);
(12)           Form.setParameter(“login”,testUserName);
(13)           Form.setParameter(“password”,testPassword);
(14)           WebResponse logonResponse=form.submit();
(15)           assertTrue(“Logon failed: “,
(16)              logonResponse.getText().indexOf(“Welcome “+testUserName)>=0);
(17)        } catch(Exception e)  {
(18)             e.printStackTrace();
(19)        }
(20)      }
(21)      Public static void main(String[] args)  {
(22)        Junit.swingui.TestRunner.run(LogonTest.class);
(23)      }
(24)    }
```

شکل پ-۶: آزمون یک صفحه‌ی سرور با استفاده از HttpUnit

پ-۷ استفاده از CVS و SubVersion در NetBeans

بسیاری از IDEها مانند NetBeans و Eclipse از ابزارهای کنترل نسخه پشتیبانی می‌کنند. روال نصب و استفاده از یک ابزار کنترل نسخه در یک IDE، مشابه است. بنابراین در این بخش تنها به چگونگی استفاده از CVS و SubVersion در NetBeans می‌پردازیم.

پ-۷-۱- ایجاد یک مخزن راه‌دور CVS

شکل پ-۷ برخی از دستورات پراستفاده از CVS را نشان می‌دهد. پیش از استفاده از CVS، شما باید یک مخزن راه دور ایجاد کنید و اجازه‌ی دسترسی به آن را به گروهی از توسعه‌دهندگان بدهید. در بیشتر موارد، مخزن روی یک ماشین یونیکس یا لینوکس ایجاد می‌شود. پس شما باید یک حساب کاربری ممتاز یونیکس یا لینوکس مانند root داشته باشید. در ادامه، فرض شده است که شما به عنوان root وارد شده‌اید و یا یک پوسته با دستور جایگزین root را اجرا کرده‌اید (su – root). برای ایجاد یک مخزن CVS با استفاده از csh، گام‌های زیر را انجام دهید:

دستور	شرح
cvs init	با افزودن زیرپوشه‌ی CVSROOT و فایل‌های کنترلی پیش‌فرض، یک انباره را مقداردهی اولیه می‌کند.
cvs checkout modules ...	برای ماژول‌های شامل حداقل یک زیرپوشه، یک کپی اختصاصی از مبدا ایجاد می‌کند.
cvs update	پوشه‌ی کاری شما را با استفاده از تغییرات انباره، به‌روزرسانی می‌کند. این دستور باید در پوشه‌ی اختصاصی مبدا اجرا شود.
cvs commit file ...	تغییراتی که شما روی فایل‌ها داده‌اید را روی انباره‌ی مبدا اعمال می‌کند.
cvs add file ...	برای ماژول‌هایی که به پوشه‌ی کاری شما اضافه شده است، فایل‌های جدیدی به رکوردهای cvs اضافه می‌کند. این کار در زمانی‌که دستور cvs commit را اجرا می‌کنید، تاثیر خود را خواهد گذاشت.
Cvs remove file ...	فایل‌ها را از انباره پاک می‌کند. این کار در زمانی‌که دستور cvs commit را اجرا می‌کنید، تاثیر خود را خواهد گذاشت.
Cvs export modules ...	کپی‌هایی از یک مجموعه از فایل‌های مبدا را برای ارسال از سایت آماده می‌کند.
cvs import	یک مجموعه به‌روزرسانی از یک سایت دیگر را بر روی انباره‌ی مبدا اعمال می‌کند.
cvs diff	تفاوت‌های بین فایل‌های پوشه‌ی کاری شما و انباره‌ی مبدا را نشان می‌دهد.
cvs status	وضعیت فایل را نشان می‌دهد.
cvs –H, or cvs -help	فهرستی از دستورات موجود cvs را نشان می‌دهد، و یا اگر نام دستور مشخص شود، اطلاعات مربوط به نحوه‌ی استفاده از آن را نمایش می‌دهد.
cvs –V, or cvs -version	اطلاعات مربوط به نسخه‌ی جاری cvs را نشان می‌دهد.

شکل پ-۷: دستورات پراستفاده CVS

۱. در cshrc. پوشه‌ی اصلی خانگی خود، “CVS rootrepository$_p$$_a$$_t$$_h$setenv CVSROOT /repository$_p$$_a$$_t$$_h$.” است.
۲. در پوشه‌ی خانگی root، دستور را اجرا کنید.

source .cshrc

این کار، طوری متغیر محیطی CVSROOT را تنظیم می‌کند که به مسیر مخزن مورد انتخاب شما اشاره کند.

۳. دستور زیر را اجرا کنید.

cvs init

این کار، مخزن CVS را مقداردهی اولیه و تنظیم می‌کند.

۴. با اجرای دستورات زیر، CVS را به عنوان یک کاربر و یک گروه تنظیم کنید. عددهای صحیح زیر، شماره شناسایی منحصربه‌فرد گروه و کاربر هستند که می‌توانند هر عدد صحیحی باشند که قبلاً به منظور مشابه استفاده نشده باشد:

Groupadd –g 50 cvsgrp

Useradd –u 125 –g cvsgrp –d /u/cvs –s /bin/sch –c ‘ ‘ CVS Repository ‘ ‘

۵. مدهای ownership و access را برای کاربر و گروه cvs، برای پوشه‌های CVSROOT و زیرپوشه‌های آن تنظیم کنید.

Chown –R cvs $CVSROOT

Chgrp –R cvsgrp $CVSROOT

Chmod –R g+s $CVSROOT

۶. بررسی کنید که ورودی cvspserver در فایل etc/services/باشد:

Grep cvs /etc/services

که باید خطوط زیر را نشان دهد:

Cvspserver 2401/tcp # CVS client / server operations

Cvspserver 2401/udp # CVS client / server operations

اگر دستورgrep این خطوط را چاپ نکرد، آنها را به فایل /etc/services اضافه کنید.

۷. یک ورودی به /etc/inetd.conf اضافه کنید تا CVS را به‌عنوان یک سرور فعال کند.

CVS server

Cvspserver stream tcp nowait root /user/bin/cvs cvs

–allow-root=/usr/local/src/cvsroot pserver

۸. به inetd daemon بگویید که فایل پیکربندی را دوباره بخواند:

Killall -HUP inetd

CVS در NetBeans IDE پیش‌نصب شده است، پس برای استفاده از آن در NetBeans IDE نیازی به نصب و تنظیم خاصی نیست.

پ–۷–۲– تنظیم Subversion در NetBeans

Subversion یک ابزار متن‌باز است که حاصل پروژه‌ی Apache Sopftware Foundation می‌باشد. این ابزار در ابتدا طراحی شد تا نسخه‌ی بهتری از CVS برای جایگزین کردن آن باشد. بنابراین، این ابزار همه‌ی توانمندی‌های CVS را به‌همراه دارد. اما Subversion فراتر از جایگزینی برای CVS حرکت کرده است و اکنون مجموعه‌ی کاملی از ویژگی‌های پیشرفته را دربر می‌گیرد. برای مثال، پوشه‌ها، عملیات کپی، به‌روزسازی، و حذف، و پیوندهای نمادین را هم نسخه‌بندی می‌کند. یک کاربر می‌تواند به‌دلخواه، یک فایل را قبل از به‌روزرسانی قفل کند تا از تغییرات هم‌زمان جلوگیری نماید. کاربران دیگر می‌توانند فایل قفل‌شده را بخوانند ولی نمی‌توانند آن را تغییر دهند. برای اطلاع از سایر ویژگی‌های این ابزار، مستندات برخط آن را مطالعه نمایید.

نرم‌افزار Subversion Client می‌تواند از هریک از دو سایت زیر دانلود شود و بر اساس دستورالعمل‌های موجود نصب شود:

http://subversion.apache.org/

http://www.collab.net/downloads/netbeans

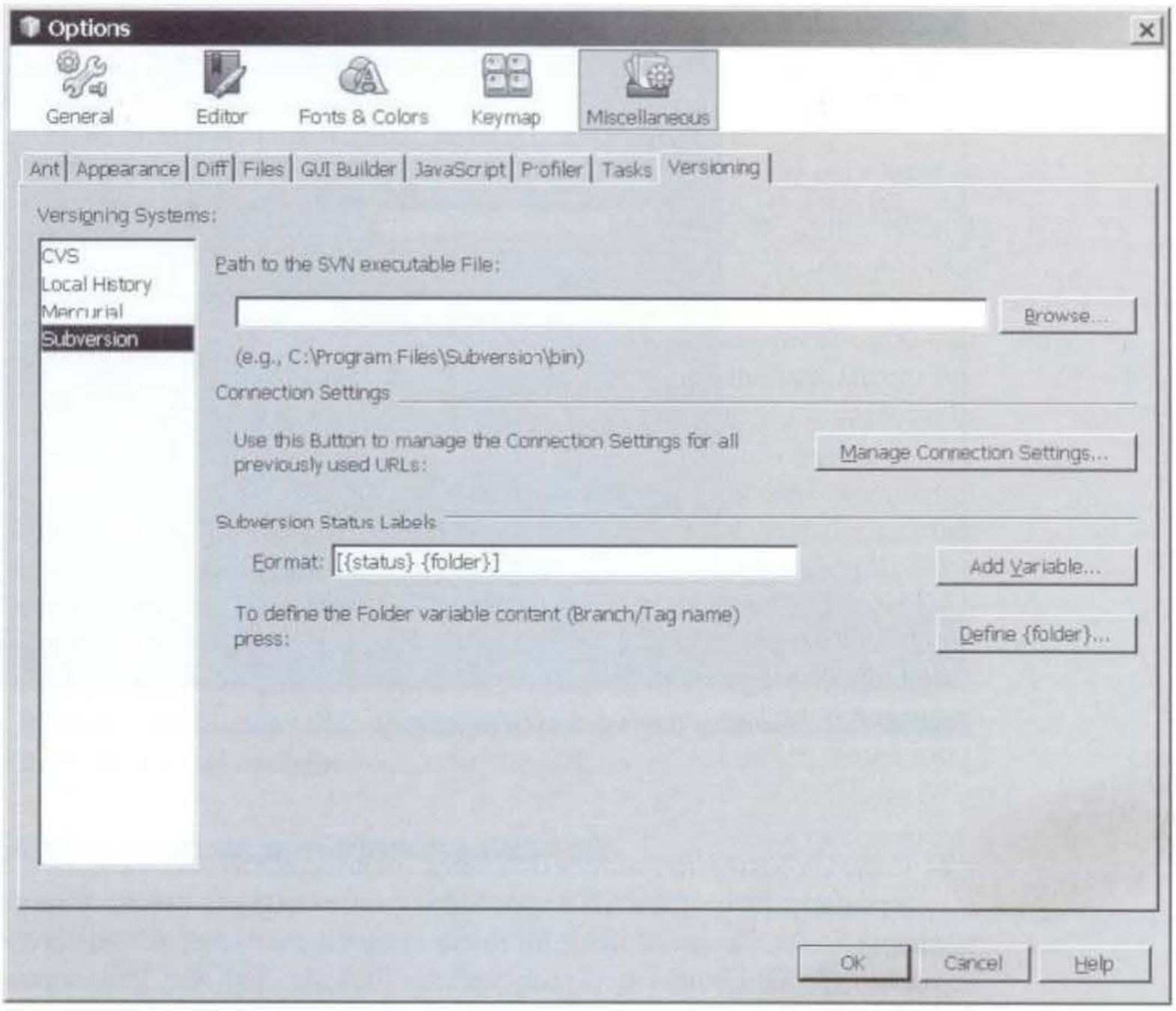
همچنین، کاربران NetBeans می‌توانند این نرم‌افزار را به‌عنوان یک افزونه‌ی NetBeans نصب نمایند. برای انجام این کار در NetBeans 6.5.1، برروی Tools و سپس Plugins کلیک کنید و زبانه‌ی Available Plugins را برای نمایش فهرستی

از افزونه‌های موجود، انتخاب نمایید. از این فهرست، Subversion را انتخاب کرده، دکمه‌ی Install را کلیک کنید. برای کامل شدن فرایند نصب، دستورات نشان‌داده‌شده را دنبال کنید.

شما باید مسیر فایل اجرایی ابزار را مشخص کنید. برای سیستم عامل ویندوز، این فایل اجرایی "svn.exe" نام دارد. شما می‌توانید برای یافتن پوشه‌ی حاوی این فایل، از جستجوی سیستم استفاده کنید. وقتی مسیر شناسایی شد، از منوی Tools گزینه‌ی Options را انتخاب کنید و سپس بر روی آیکون Miscellaneous در بالای پنجره‌ی گفت‌وگوی Options کلیک کنید. زبانه‌ی Versioning را انتخاب کرده، و در فهرست Versioning Systems در سمت چپ، Subversion را انتخاب کنید. IDE شکلی مشابه شکل پ-۸ را نشان خواهد داد. مسیر را وارد کنید و یا پوشه‌ی حاوی فایل اجرایی را با فشردن دکمه‌ی Browse پیدا کنید. OK را بزنید و NetBeans IDE را دوباره شروع کنید.

شما باید به مخزن Subversion دسترسی داشته باشید. شما می‌توانید با استفاده از ابزار svnadmin که در نرم‌افزار اصلی قرار دارد، یک مخزن محلی ایجاد کنید. در محل خط دستور، عبارت زیر را تایپ کنید:

```
Svnadmin create /path/to/your/repository
```



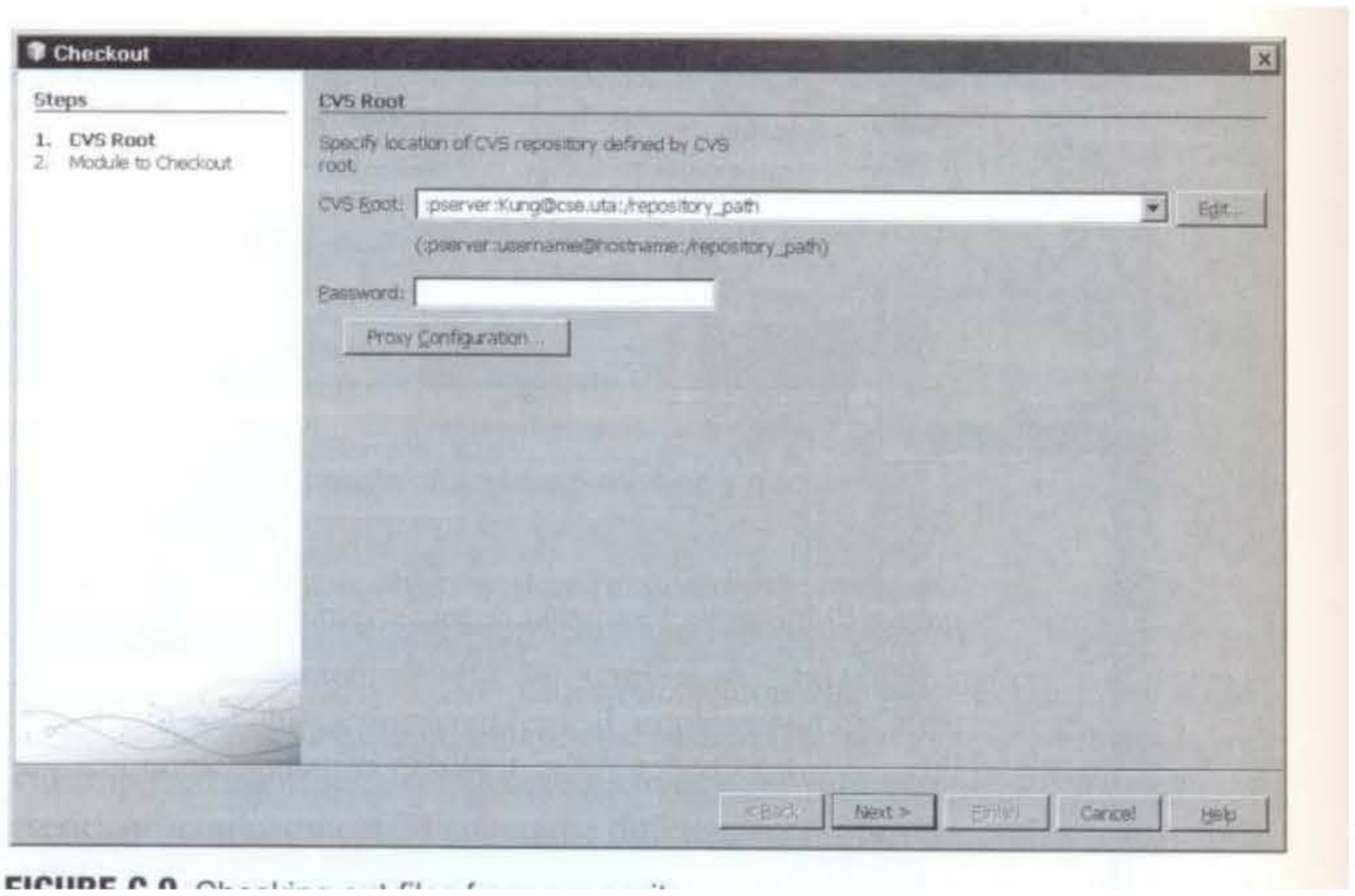
شکل پ-۸: مشخص کردن مسیر فایل اجرایی Subversion Client

پ-۷-۳- به امانت گرفتن فایل‌ها از یک مخزن

در دو بخش قبل، شرح داده شد که چگونه CVS و Subversion در NetBeans نصب شوند. در این بخش و بخش‌های آینده نشان داده خواهد شد که چگونه از این سیستم‌های کنترل نسخه برای انجام عملیات کنترلی در NetBeans IDE 6.5.1 استفاده گردد. از آن‌جا که گام‌های انجام این عملیات، برای CVS و Subversion مشابه است، هر دو به همراه هم توضیح داده می‌شود. برای جداکردن اصطلاحات مربوط به این دو ابزار، که با فونت ایتالیک نشان داده می‌شوند، از یک "/" استفاده می‌شود. به‌عنوان مثال، "choose Versioning > CVS/Subversion > Checkout" به معنی "choose Versioning > CVS > Checkout" یا "choose Versioning > Subversion > Checkout" می‌باشد.

در بیشتر موارد، اولین کاری که باید انجام شود، به‌امانت‌گرفتن فایل‌های اشتراکی از یک مخزن راه‌دور است. این عمل، کپی‌های محلی از فایل‌ها و پوشه‌هایی که به امانت می‌گیرید ایجاد می‌کند. سپس شما این فایل‌ها را ویرایش می‌کنید و وقتی انجام تغییرات به‌پایان رسید، فایل تغییریافته را به مخزن تحویل می‌دهید. گام‌های زیر باید برای به‌امانت‌گرفتن فایل از یک مخزن CVS یا Subversion انجام گردند:

۱. از نوار منوی پنجره‌ی اصلی NetBeans، گزینه‌های Versioning > CVS/Subversion > Checkout را انتخاب کنید. یک ویزارد Checkout مانند شکل پ-۹ باز می‌شود.



شکل پ-۹: به امانت گرفتن فایل از یک مخزن

۲. در فیلد CVS Root/Repository URL نام کاربری واقعی، نام میزبان، و مسیر مخزن پوشه‌ی CVS root/Subversion repository روی میزبان راه دور را وارد کنید. در فیلد Password، کلمه‌ی عبور را وارد کنید. به‌جای این کار می‌توانید دکمه‌ی Edit را کلیک کنید تا یک جعبه‌ی گفت‌وگو نشان داده شود، و یا فلش سمت پایین را کلیک کنید تا پروتکل انتخاب گردد. این کار ساده‌تر است، زیرا شما ناچار نیستید که با ترکیب جملات آشنا باشید. NetBeans چهار فرمت مختلف CVS را پشتیبانی می‌کند که دو فرمت آخر، به یک فایل اجرایی خارجی CVS نیاز دارد:

(آ) فرمت سرور با کلمه‌ی عبور راه دور (Remote Password server format)، که فرمت مورد استفاده در شکل پ-

۹ است. ترکیب جمله به‌صورت زیر است:

“:pserver:username@hostname:/repository_path”

(ب) دسترسی با استفاده از پوسته‌ی راه دور (Access using Remote Shell (RSH)) یا فرمت پوسته‌ی امن (Secure Shell (SSH)). ترکیب جمله به‌صورت زیر است:

“:ext:username@hostname:/repository_path”

(ج) فرمت دسترسی به یک مخزن محلی (Access to a local repository). ترکیب جمله به‌صورت زیر است:

“:local:/repository_path”

(د) فرمت دسترسی به یک مخزن محلی با استفاده از یک پروتکل راه دور (Access to a local repository using a remote protocol). ترکیب جمله به‌صورت زیر است:

“:fork:/repository_path”

برای Subversion، NetBeans از پنج نوع پروتکل ارتباطی پشتیبانی می‌کند:

(آ) دسترسی مستقیم به مخزن روی دیسک محلی:
فرمت URL به‌صورت file://repository_path است.

(ب) دسترسی از طریق پروتکل WebDAV به یک سرور آگاه به Subversion:
فرمت URL به‌صورت http://hostname/repository_path است.

(ج) دسترسی از طریق پروتکل HTTP با رمزگذاری SSL:
فرمت URL به‌صورت https://hostname/repository_path است.

(د) دسترسی ازطریق پروتکل custom به یک سرور svnserv:
فرمت URL به‌صورت svn://hostname/repository_path است.

(ه) دسترسی از طریق پروتکل SVN از راه یک تونل SSH خارجی:
فرمت URL به‌صورت svn+ssh://hostname/repository_path است.

بسته به فرمت CVS یا پروتکل Subversion مورد استفاده، شما ممکن است به اطلاعات اضافی مانند کلمه‌ی کاربری و کلمه‌ی عبور نیاز داشته باشید. پس از وارد کردن اطلاعات لازم، دکمه‌ی Next را برای ادامه‌ی کار کلیک نمایید.

۳. روی پنل Modules/Folders to checkout، پیمانه/پوشه‌ای که می‌خواهید به امانت بگیرید را در فیلد Module/Repository Folder(s) وارد کنید. به‌جای این کار می‌توانید با کلیک کردن روی دکمه‌ی Browse این پیمانه‌ها/پوشه‌ها را از مخزن انتخاب نمایید. برای CVS، اگر پیمانه‌ای انتخاب نشود، کل مخزن به‌امانت گرفته می‌شود. در فیلد Branch/Repository Revision، یک شاخه، شماره‌ی مرور، یا علامت برای امانت‌گیری تعیین کنید. دکمه‌ی Browse/Search به شما امکان می‌دهد که از فهرستی از همه‌ی شاخه‌ها در مخزن، انتخاب نمایید. سپس پوشه‌ی کاری محلی را انتخاب کنید و یا با کلیک روی دکمه‌ی Browse به پوشه‌ی محلی موردنظر بروید. برای Subversion، پس از آن‌که گزینه‌ی Checkout را

انتخاب کردید، از Scan for NetBeans Projects خارج شوید. دکمه‌ی Finish را کلیک کنید تا فرایند امانت‌گیری آغاز گردد.

اگرمنابع به‌امانت‌گرفته‌شده شامل یک پروژه باشند، IDE به شما اطلاع می‌دهد تا آنها را در IDE باز کنید. درغیر این‌صورت، IDE به شما اطلاع می‌دهد که یک پروژه‌ی جدید از منابع ایجاد کنید و منابع را در IDE باز نمایید.

پ-۷-۴- ویرایش منابع و مشاهده‌ی تغییرات

به‌شکلی که در بخش آخر گفته شده است، شما می‌توانید یک پروژه‌ی نسخه‌بندی‌شده ایجاد نمایید. همچنین می‌توانید پروژه‌ی نسخه‌بندی‌شده‌ای را که خارج از NetBeans ایجاد کرده‌اید، همانند یک پروژه‌ی NetBeans باز کنید. وقتی پروژه باز شد، خواهید توانست فایل‌های آن را ویرایش کنید. تغییراتی که می‌دهید با سه رنگ آبی، سبز، و قرمز مشخص می‌شوند، که به‌ترتیب، خطوطی را که نسبت به نسخه‌ی قبل تغییر کرده‌اند، اضافه شده‌اند، و حذف شده‌اند را نشان می‌دهد. پنجره‌ی نسخه‌بندی، یک نمای بلادرنگ از تغییرات انجام‌شده برروی کپی محلی پوشه‌های انتخاب‌شده را نمایش می‌دهد. این پنجره، به‌طور پیش‌فرض در پنل پایینی IDE نشان داده می‌شود و فایل‌های اضافه‌شده، حذف‌شده، یا تغییریافته را نشان می‌دهد. با انتخاب Window از منوی اصلی و سپس انتخاب Versioning > CVS/Subversion این پنجره باز می‌گردد. این کار را می‌توان با کلیک راست بر روی فایل یا پوشه‌ی نسخه‌بندی‌شده، و انتخاب CVS/Subversion > Show Changes نیز انجام داد. در بالای پنجره‌ی نسخه‌بندی، چهار آیکون عملیاتی برای پرکاربردترین وظایف وجود دارد. شکل پ-۱۰ این آیکون‌ها را معرفی می‌نماید.

Icon	Name	Description
	Refresh Status	Refresh the status of the selected files and folders to reflect any changes that may have been made externally.
	Diff All	Open the Diff Viewer to display a side-by-side comparison of local copies and the versions in the repository.
	Update All	Update all selected files from the repository.
	Commit All	Commit local changes to the repository.

شکل پ-۱۰: آیکون‌های عملیاتی در پنجره‌ی نسخه‌بندی

پ-۷-۵- مشاهده‌ی وضعیت فایل

IDE برای نشان‌دادن وضعیت فایل‌ها، نام آنها را با رنگ‌های مختلف نمایش می‌دهد:

- آبی: نشان می‌دهد که فایل به‌طور محلی تغییر یافته است.
- سبز: نشان می‌دهد که فایل به‌طور محلی اضافه شده است.
- قرمز: نشان می‌دهد که نسخه‌ی کپی محلی فایل با نسخه‌ی روی مخزن، در تضاد است.
- خاکستری: نشان می‌دهد که از این فایل صرف‌نظر شده‌است و فایل با عملیات به‌روزرسانی و اعلام به مخزن پردازش نمی‌شود. درصورتی از یک فایل صرف‌نظر می‌شود که آن فایل نسخه‌بندی نشده باشد.
- خط‌زده‌شده: نشان می‌دهد که فایل از دستور اعلام به مخزن کنار گذاشته شده است. این فایل‌ها هنوز تحت تاثیر عملیاتی مانند به‌روزرسانی هستند.

پ-۷-۶- مقایسه‌ی نسخه‌های مروری فایل

مقایسه‌ی کپی‌های محلی با نسخه‌ی روی مخزن، یک کار معمول در مدیریت مرور است. سه راه برای این کار وجود دارد: (۱) کلیک راست روی فایل محلی نسخه‌بندی‌شده در قاب Files در گوشه‌ی سمت چپ بالای NetBeans IDE، و انتخاب CVS/Subversion > Diff، (۲) دوبار کلیک برروی فایل در پنجره‌ی Versioning، یا (۳) کلیک آیکون Diff All در نوار ابزار بالای پنجره‌ی Versioning. انجام هریک از این کارها، یک Diff Viewer گرافیکی را در پنجره‌ی اصلی IDE باز می‌کند. Viewer، نسخه‌ی محلی و کپی‌های روی مخزن از فایل را در دو پنل موازی طوری باز می‌کند که جدیدترین نسخه در سمت راست قرار بگیرد. همان‌طور که گفته شد، قسمت‌هایی از کد که تغییریافته، اضافه شده، و یا حذف گشته است، با رنگ‌های آبی، سبز، و قرمز مشخص می‌شوند.

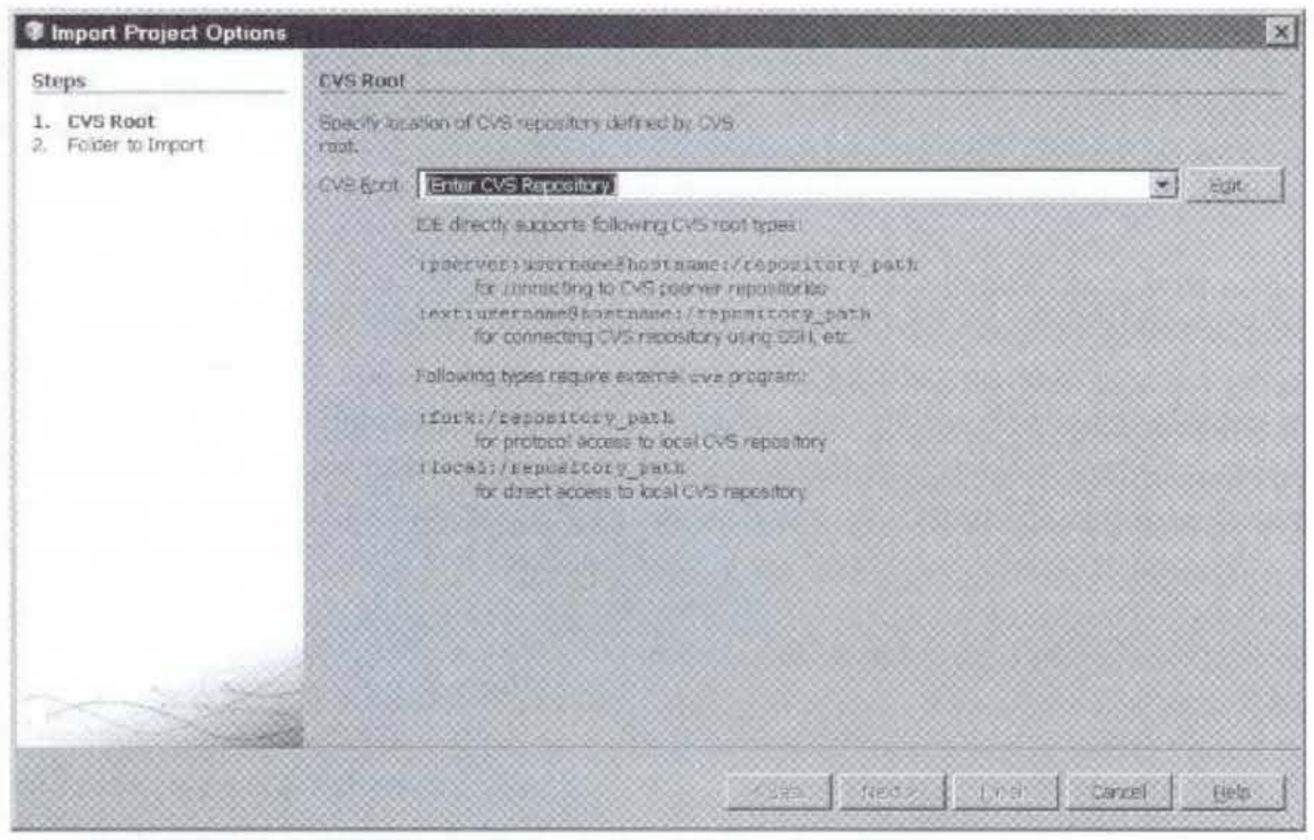
پ-۷-۷- ادغام تغییرات از مخزن

شما می‌توانید فایل‌های موجود برروی مخزن را با کپی‌های محلی خود ادغام کنید. برای CVS، فایل‌ها یا پوشه‌هایی که می‌خواهید با نسخه‌ی محلی خود در پنجره‌ی Files، Projects، Favorites، و یا Versioning تلفیق شود را انتخاب کنید و CVS/Merge Changes را انتخاب نمایید. در جعبه‌ی گفت‌وگو، شاخه‌ی پوشه‌ای که می‌خواهید تغییرات از آن را ادغام کنید انتخاب نمایید و یا با کلیک بر روی Browse، آن را از یک فهرست انتخاب کنید. می‌توانید برچسب را در فیلد Tag Name مشخص کنید، و یا آن را با کلیک بر روی Browse از یک فهرست انتخاب کنید. برای افزوده شدن تغییرات یافته‌شده در شاخه به کپی محلی فایل، Merge را کلیک نمایید. در این زمان، اگر ناسازگاری یا برخوردی وجود داشت، رنگ وضعیت فایل به قرمز تغییر می‌کند.

برای Subversion، بر روی فایل‌ها یا پروژه‌ها در پنجره‌ی Projects یا Files کلیک راست کنید و Subversion > Merge Changes را انتخاب نمایید. در جعبه‌ی گفت‌وگو، در فیلد Merge From، یکی از موارد One Repository Folder، Tow Repository Folder، و یا Since its Origin One Repository Folder را انتخاب کنید. آخرین گزینه، تغییرات رخ‌داده از زمان ایجاد پوشه و شماره‌ی مرور که در Ending Revision Field مشخص می‌کنید را اعمال می‌نماید. در فیلد(های) Repository Folder، پوشه(هایی) که می‌خواهید از آنها تغییرات را اعمال کنید مشخص نمایید. در فیلدهای Starting Revision و Ending Revision، نقطه‌ی شروع و پایان دامنه‌ی تغییراتی که می‌خواهید اعمال شود را وارد کنید. برای انجام تغییرات، Merge را کلیک نمایید. اگر ناسازگاری یا برخوردی وجود داشت، رنگ وضعیت فایل قرمز می‌گردد.

پ-۷-۸- حل برخوردها

پنجره‌ی نسخه‌گذاری، ناسازگاری‌ها و برخوردهای بین کپی‌های محلی و کپی‌های مخزن را در ستون Status نشان می‌دهد. همه‌ی این برخوردها باید پیش از اعمال تغییرات محلی بر روی مخزن، برطرف گردند. شما می‌توانید بر روی یک فایل مشمول برخورد کلیک راست کنید و CVS/Subversion > Resolve Conflicts را برای فعال کردن Merge Conflicts Resolver انتخاب کنید. با این کار، دو مرور دارای برخورد در دوستون کنار هم نشان داده می‌شود و بخش‌هایی از کد که متفاوتند با تغییر رنگ مشخص می‌شوند. حال می‌توانید با کلیک روی گزینه‌ی Accept، و یا Accept Next مربوط به یکی از مرورها، برخورد را حل کنید.



شکل پ-۱۱: وارد کردن فایل‌ها به یک مخزن راه دور CVS

پ-۷-۹- به‌روزرسانی کپی‌های محلی

توصیه می‌شود که همه‌ی فایل‌های محلی، پیش از نوشتن آنها روی مخزن، به‌روزرسانی شوند. شما می‌توانید کپی‌های محلی‌خود را با اجرای دستور به‌روز رسانی، با تغییراتی که سایر برنامه‌نویسان روی مخزن اعمال کرده‌اند، به‌روز نمایید. برای این کار، در قاب File، روی یک فایل نسخه‌بندی‌شده کلیک راست کنید و CVS/Subversion > Update را انتخاب نمایید.

پ-۷-۱۰- اعمال فایل‌های محلی بر مخزن

شما می‌توانید فایل‌ها یا پوشه‌های محلی خود را با کلیک راست بر روی آنها در قاب File و انتخاب CVS/Subsystem > Commit، بر روی مخزن اعمال کنید. IDS یک جعبه‌گفت‌وگوی Commit با یک ناحیه‌ی متنی Commit Message و یک جدول Files to Commit نشان می‌دهد. جدول، شامل هر فایل و پوشه، به همراه وضعیت آن، عمل Commit، و مسیر مخزن است. شما می‌توانید در ستون Commit Action روی یک فیلد کلیک کنید تا عمل مربوطه را تغییر دهید. برای مثال، می‌توانیدیک فایل یا پوشه را از فهرست تغییراتی که قرار است روی مخزن اعمال شوند حذف کنید.

پ-۷-۱۱- وارد کردن فایل‌ها به یک مخزن

شما می‌توانید فایل‌های یک پروژه‌ی نسخه‌بندی‌نشده را به یک مخزن راه‌دور ارسال نمایید. برای این کار، یک پروژه، یا پوشه، یا فایل نسخه‌بندی‌نشده را انتخاب کنید، و در منوی اصلی، Versioning > CVS/Subsystem > Import into Repository را برگزینید. یک ویزارد Import مانند شکل پ-۱۱ باز می‌شود که به شما امکان می‌دهد مخزن، پوشه، و فایل‌های محلی مورد انتقال را انتخاب کنید. وقتی همه‌ی این موارد مشخص شد، برای آغاز فرایند انتقال، بر روی Finish کلیک کنید.

ت

شرح پروژه‌ها

ت-۱ سیستم کرایه‌ی خودرو

یک موسسه‌ی کرایه‌ی خودرو، تعدادی محل برای کرایه‌ی خودرو در سطح پایتخت ایجاد کرده است. از جایی‌که این موسسه مدل کسب‌وکار بزرگی دارد و خدمات کاربرپسندی را ارائه می‌دهد، کسب‌وکار آن در سال‌های اخیر به‌سرعت توسعه یافته است. به‌دلیل رشد این کسب‌وکار، هزینه‌های گرداندن آن نیز افزایش یافته است. در مجموع، با داغ‌تر شدن بازار کار این موسسه، هزینه‌های کاری آن در سال‌های اخیر دوبرابر شده است. این موسسه می‌خواهد راه‌حلی برای کاهش هزینه‌های عملیاتی خود پیدا کند. عملیات کسب‌وکار موسسه در ادامه شرح داده می‌شود. این شرح کامل نیست و موسسه برای اعمال هر گونه پیشنهاد خوب برای ارتقای آن، انعطاف‌پذیر است.

خودروها از محلی گرفته می‌شوند و یا به همان نقطه برگردانده می‌شوند و یا با یک هزینه‌ی اضافی به محل دیگری تحویل داده می‌شوند. اگرچه در حال حاضر خدمات موسسه فقط در حوزه‌ی خودروهای مسافرتی است، ولی ممکن است در آینده شامل گونه‌های دیگر کرایه‌ی خودرو هم بشود و در این صورت باید بتوان از همان سیستم کرایه‌ی خودرو[1] استفاده نمود. موسسه دارای انواع مختلف خودرو از برندهای مختلف در ناوگان خود می‌باشد. هر نوع خودرو ممکن است در مدل‌های مختلف موجود باشد. برای مثال تویوتا انواع کرولا، کمری، و ... را تولید می‌کند. مدل‌ها به چند رده‌ی قیمتی گروه‌بندی شده‌اند. مشتری باید بتواند نوع و مدل اتومبیلی که می‌خواهد را انتخاب نماید. اگر خودرو انتخاب‌شده در دسترس نبود، سیستم باید پیامی نمایش دهد که به مشتری بگوید که خودرو پیش‌تر کرایه داده‌شده و به او امکان بدهد که نوع و مدل دیگری انتخاب نماید و یا اجازه دهد که سیستم، مدل‌های مشابه از انواع دیگر را پیشنهاد دهد.

این موسسه برای مشتریان خود تعدادی برنامه‌ی مختلف کرایه دارد. برای مثال، یک «برنامه‌ی روزانه با مسافت نامحدود» و یک «برنامه‌ی مربوط به آخر هفته با تخفیف ۱۰٪» موجود است. موسسه تشخیص داده است که ارائه‌ی اطلاعاتی در مورد مدل‌های اتومبیل، مانند دنده‌ی دستی یا خودکار، دو درب یا چهار درب بودن، و سدان یا بدون صندوق بودن مهم است و یک مشتری مایل است که چنین اطلاعاتی را در زمان اجاره‌ی خودرو داشته باشد. در حال حاضر، مشتریان کرایه‌ی خودرو را از طریق تماس رودررو و یا تلفنی با موسسه انجام می‌دهند. فروشندگان، کرایه‌های درخواست‌شده را به‌طور دستی و با استفاده از یک فرم رزرواسیون پردازش می‌کنند و آن‌ها را در یک کمد فایل آرشیو می‌کنند. برای رزرو کردن یک خودرو نیازی به پرداخت

---

[1] Car Rental System

بیعانه نیست. اگر مشتری تا یک محدوده‌ی زمانی مشخص برای امضای فرم قرارداد مراجعه نکند، رزرو او باطل می‌شود. این رزرواسیون فقط درصورتی معتبر است که هنوز اتومبیل درخواستی مشتری موجود باشد.

گاهی ممکن است یک مشتری یک مجموعه‌ی رزرواسیون داشته باشد و مایل باشد صورت‌حساب آنها یک‌جا پردازش شود. به‌محض این‌که یک خودرو به یک مشتری تخصیص یافت، یک صورت‌حساب برای او باز می‌شود. یک صورت‌حساب می‌تواند یک یا تعداد بیشتری مورد کرایه را شامل گردد. به‌طور معمول یک مشتری درهنگام باز گرداندن خودرو تسویه‌حساب می‌کند، ولی در برخی موارد، صورت‌حساب باید به یک موسسه (مانند محل کار مشتری) فرستاده شود. وقتی مشتری با کارت اعتباری پرداخت خود را انجام می‌دهد، هزینه‌ی کرایه توسط یک موسسه‌ی پردازش کارت‌های اعتباری پردازش می‌گردد.

یک خودرو ممکن است برای یک روز خاص در دسترس باشد یا نباشد. ماشین‌های کرایه‌ها به نگهداری مدام نیاز دارند، و به‌علاوه هرگونه آسیب به آنها باید در سریع‌ترین زمان تعمیر گردد. موسسه مایل است اطلاعات مربوط به خرید، نگهداری، و استفاده‌ی یک خودروی کرایه‌ای را برای مقاصد کاری و مالیاتی نگهداری و دنبال نماید.

ت-۲ سیستم ملی خدمات نمایشگاهی

سیستم ملی خدمات نمایشگاهی[1] یک سیستم فراهم‌کننده‌ی خدمات است که به مشتریان صاحب کسب‌وکارهای مختلف کمک می‌کند که نمایشگاه‌های ملی و بین‌المللی مربوط به حرفه‌ی خود را ایجاد، معرفی، سازمان‌دهی، و اجرا نمایند. صاحبان کسب‌وکار برای خدماتی که نیاز دارند با NTSS تماس می‌گیرند . ایجاد یک نمایشگاه، با طراحی نمایشگاه در ارتباط است. این کار شامل خدمات تخصصی، از انتخاب یک تم و یک شعار، تا انتخاب محل و مدت برگزاری می‌گردد. معرفی نمایشگاه شامل فعالیت‌های تبلیغاتی می‌باشد. سازمان‌دهی یک نمایشگاه شامل همه‌ی فعالیت‌های لازم برای انجام قبل از برگزاری نمایشگاه، به‌جز ایجاد و معرفی آن می گردد. برای مثال، معرفی سخنرانان و ثبت‌نام شرکت‌کنندگان و نمایش‌دهندگان در جهت سازمان‌دهی انجام می‌شود. اجرای یک نمایشگاه فعالیتی شامل ثبت‌نام در محل برگزاری، برپاکردن غرفه‌ها ، سامان‌دهی اتاق‌های کنفرانس برای سمینارها، پذیرش، و توزیع مواد مصرفی نمایشگاه است. NTSS یک حساب کاربری برای هر مشتری ایجاد می کند تا هزینه‌های خدماتی و پرداخت‌های انجام‌شده را ثبت نماید و تراز مالی حساب‌ها را نگهداری نماید.

یک نمایشگاه می‌تواند به‌عنوان یک رویداد مورد توجه قرارگیرد. برای مثال، نمایشگاه صنایع الکترونیکی سال ۲۰۰۸ از ۷ تا ۱۰ ژانویه در مرکز همایش‌های لاس‌وگاس برگزار گردید، و کنفرانس Macworld از ۱۴ تا ۱۸ ژانویه ۲۰۰۸ در مرکز MOs cone برگزار شد. یک رویداد یک موسس دارد که می‌تواند حقیقی یا حقوقی باشد. برای مثال CES توسط Consumer Electronics Association ، و Macworld توسط Apple تاسیس و سازمان‌دهی گردید. یک رویداد می‌تواند به یک یا چند دامنه‌ی از پیش‌تعریف‌شده تعلق داشته باشد. برای مثال، CES و MacWorld هر دو به تکنولوژی و صنایع الکترونیک تعلق داشتند. فهرست دامنه‌های جدید ازپیش‌تعریف‌شده می‌تواند از زمانی تا زمان دیگر تغییر نماید، زیرا دامنه‌های جدید باید به آن اضافه شوند و دامنه‌های خارج‌ازرده از آن حذف شوند.

طیف وسیعی از شرکت‌کنندگان، شامل پرسنل برگزاری رویداد، سخنرانان مدعو یا انتخاب‌شده، نمایش‌دهندگان، و بازدید-کنندگان در نمایشگاه حضور می‌یابند. به‌غیراز پرسنل برگزاری رویداد و سازمان‌دهی کننده‌ی آن، همه‌ی شرکت‌کنندگان باید برای شرکت در این رویداد ثبت‌نام نمایند. نرخ ثبت‌نام برای رویدادهای مختلف متفاوت است. پرسنل برگزاری رویداد، آن را ایجاد، آماده‌سازی، و اجرا می‌نمایند. سخنرانان مدعو از چهره های مشهور و شناخته‌شده در آن حوزه هستند که دعوت شده‌اند تا یک سخنرانی کلیدی، که رویداد را معرفی می‌کند، ارایه نمایند. سخنرانان انتخاب‌شده دعوت شده‌اند تا بر مبنای ارزیابی پیشنهادیه‌های خود در این رویداد سخنرانی نمایند. پیشنهادیه‌ها معمولاً توسط یک کمیته‌ی داوران، که در این زمینه متخصص

[1] National Trade Show Service System (NTSS)

بوده و مایل به کمک هستند، بررسی می‌گردد. وضعیت یک پیشنهادیه، شامل در حال مرور، پذیرفته‌شده، و ردشده است. نمایش‌دهندگان به نمایشگاه می‌آیند تا محصولات و خدمات خود را به‌نمایش بگذارند. این افراد باید بسته به اندازه‌ی غرفه‌ی خود(بزرگ، متوسط، یا کوچک) هزینه پرداخت نمایند. غرفه‌ها برای کل زمان برگزاری رویداد درخواست و اجاره می‌شود. در پایان، بازدیدکنندگان به منظورهای مختلف برای بازدید از نمایشگاه می‌آیند.

ت-۳ سیستم مدیریت تحصیل در خارج‌ازکشور

در این پروژه، یک برنامه‌ی کاربردی مبتنی بر وب، به‌طور افزایشی برای دفتر تحصیلات بین‌الملل یک دانشگاه، طراحی، پیاده‌سازی، آزمون، و نصب می‌گردد.

این برنامه‌ی کاربردی به دانشجویان اجازه خواهد داد که با استفاده از انواع شرایط جستجو، برای برنامه‌های تبادل دانشجوی برون‌مرزی جستجو نمایند. وب‌گاه مربوطه، نتایج جستجو را نمایش خواهد داد. این نتایج به‌صورت سلسله‌مراتبی سازمان‌دهی می‌گردند تا پیمایش و یافتن اطلاعات موردنظر آسان‌تر گردد. در این سند، فهرستی از نیازمندی‌های این نرم‌افزار ذکر شده است. این فهرست، کامل و نهایی نیست و پیش‌بینی می‌گردد موارد جدیدی به آن افزوده گردد (دانشجویان به پیشنهاد نیازمندی‌های جدید تشویق می‌گردند). از دانشجویان انتظار می‌رود که از فرایند چابک یکنواخت و متدولوژی‌های مربوط به آن برای انجام تحلیل، طراحی، پیاده‌سازی، آزمون، و استقرار سیستم‌های مدیریت تحصیل در خارج‌ازکشور[۱] استفاده نمایند. سه تکرار وجود خواهد داشت که هریک در حدود یک ماه به‌طول می‌انجامد. انتظار نمی‌رود که این پروژه در یک ترم تحصیلی به پایان برسد. پروژه می‌تواند در یک درس دیگر ادامه یابد.

نیازمندی‌های عملیاتی نرم‌افزار (RS) را در ادامه مشاهده می‌کنید:

R1) برنامه‌ی کاربردی مبتنی‌بروب باید یک امکان جست‌وجو برای برنامه‌های تبادل دانشجو، با شرایط متنوع جست‌وجو فراهم نماید.

R2) وب‌گاه باید امکان نمایش سلسله‌مراتبی نتایج جست‌وجو، برای کمک به کاربر دررسیدن از یک خلاصه‌ی سطح بالا تا جزییات برنامه‌های تبادل دانشجو را فراهم نماید. گروه‌های پروژه باید با همکاری OIE، شکل مناسب برای صفحه‌ی نمایش را به‌دست آورده، تعریف نمایند.

R3) سیستم باید برای کارمندان OIE، امکان پردازش تعاملی یا یک‌جای اطلاعات برنامه‌های تبادل دانشجو را فراهم آورد. این اطلاعات در یک پایگاه داده ذخیره می‌شوند.

R3.1) برای وضعیت تعاملی:

R3.1.1) واسط کاربری باید مبتنی بر وب باشد.

R3.1.2) واسط کاربری باید وقتی شرط کاربرپسند بودن آن مطرح نیست، از کم‌ترین پنجره‌های ممکن استفاده نماید.

R3.1.3) واسط کاربری باید از جابه‌جایی‌های ناخواسته بین عملیات مربوط به صفحه‌کلید و عملیات مربوط به ماوس در زمان ورود اطلاعات جلوگیری نماید.

R3.2) برای وضعیت پردازش یک‌جا:

R3.2.1) سیستم باید بتواند حداقل یک یا دو فرمت از پرونده‌های ساختاریافته‌ی لازم که توسط OIE فراهم‌شده است را پردازش نماید.

---

[۱] Study Abroad Management System

R3.2.2) سیستم باید یک واسط کاربری گرافیکی (GUI) برای یک کارمند OIE فراهم نماید که اطلاعات مربوط به پرونده‌های داده‌ای که باید پردازش شود را وارد نماید. این واسط گرافیکی باید کاربرپسند باشد و به تلاش کمی از جانب کاربر نیاز داشته باشد.

R3.2.3) سیستم باید جهت اضافه‌کردن فرمت پرونده‌ها و داده‌ساختارهای دیگر، قابل‌گسترش باشد.

R4) عملکرد برخط.

R4.1) برنامه‌ی کاربردی مبتنی‌بروب باید برای دانشجویان، قابلیت ارسال برخط درخواست‌ها، به برنامه‌های تبادل دانشجو را فراهم نماید. فرایند می‌تواند با نمایش یک فهرست چک‌لیست/کارهای موجود برای انجام آغاز شود. جهت توضیحات بیشتر، R6 را ببینید.

R4.2) سیستم باید درخواست‌های ارسال‌شده را به‌طور امن در پایگاه‌داده ذخیره نماید.

R4.3) سیستم باید به کمک یک پیام، که وضعیت فرایند درخواست را نشان می‌دهد، پاسخ‌گو باشد. یک پیام ممکن، فهرست کارهای موجود برای انجام است که نشان می‌دهد هنوز برای تکمیل فرایند درخواست به چه چیزهایی نیاز است.

R4.4) سیستم باید برای اطلاع‌رسانی به پرسنل OIE ، در چارچوب ترجیحات فعلی اطلاع‌رسانی، یک ایمیل تولید و ارسال نماید.

R5) استفاده‌ی پرسنل و مدیر سیستم از آن.

R5.1) سیستم باید در ابتدا یک حساب کاربری مدیریت با کلمه‌ی عبور مشخص ایجاد نماید. مدیر سیستم اجازه خواهد داشت که کارگران دیگر از پرسنل را ایجاد، به‌روزرسانی، و یا حذف نماید. مدیر سیستم نیز خود یک کاربر از پرسنل است.

R5.2) سیستم باید یک سیستم کاربری مبتنی بر وب، و محافظت‌شده با کلمه‌ی عبور برای کارکنان فراهم نماید تا بتوانند از طریق آن Login نموده، ترجیحات شخصی خود برای اطلاع‌رسانی را مشخص نمایند.

R5.3) سیستم باید به مدیر اجازه بدهد که ترجیحات عمومی اطلاع‌رسانی در سطح سیستم را، که بر همه‌ی تنظیمات انجام‌شده توسط کاربران دیگر ترجیح خواهد داشت، تعریف نماید.

R6) سیستم باید یک چک‌لیست و یک فهرست کارهای موجود برای انجام، برای تسهیل پی‌گیری فرایند درخواست از جانب دانشجویان فراهم نماید.

R6.1) فهرست(ها) باید آنچه انجام‌شده است و آنچه باید انجام بشود را نشان بدهند. این فهرست می‌تواند برای دانشجویان مختلف و برنامه‌های تحصیلی مختلف، متفاوت باشد. گروه‌ها باید برای نهایی کردن این فهرست با OIE وارد مذاکره گردند.

R6.2) هر جزء از فهرست باید فقط یکی از کارهایی که باید انجام شود را نشان دهد.

R6.3) برای هریک از کارهایی که باید انجام شود، باید در صورت امکان یک پیوند به صفحه‌ی مناسب برای انجام آن کار وجود داشته باشد.

R6.4) برای هر کار انجام‌شده، در صورت امکان باید یک پیوند به صفحه‌ی مناسب برای به‌روزرسانی اطلاعاتی که قبلا ارسال‌شده وجود داشته باشد.

R7) وب‌سایت باید به یک کارمند OIE اجازه دهد که کاربردهای برخط را مرور و پردازش نماید.

R7.1) باید بر روی صفحه‌ی شخصی کارمند (که پس از ورود کاربر به سیستم باز می‌شود)، یک پیوند "Review Online Applications" با یک علامت "+" وجود داشته باشد که وقتی باز می‌شود، چند انتخاب ممکن باشد: (۱) مرور کاربردهای تازه‌تخصیص‌یافته، (۲) مرور کاربردهای پردازش شده، (۳) مرور همه‌ی کاربردها. (ممکن است نیاز باشد که تیم‌ها برای نهایی کردن این نیازمندی، با OIE مشورت نمایند.

R7.2) پس از آن‌که یک کارمند کاربردهای برخط را برای مرور انتخاب می‌کند، سیستم باید جدولی را برای نشان دادن خلاصه‌ی اطلاعات کاربران نمایش دهد: هر کاربرد برخط، در هر ردیف. هر ردیف باید شامل یک پیوند به یک صفحه‌ی قابل ویرایش باشد، به‌طوری‌که کاربر بتواند کاربر برخط را به‌روزرسانی نماید.

R7.3) پس از آن‌که کارمند به‌روزرسانی خود را اعلام کرد، سیستم باید پیامی بدهد که وضعیت آن به‌روزرسانی را نشان دهد.

در این پروژه باید برخی از نیازمندی‌های کیفی، حداقل شامل موارد زیر، نیز در نظر گرفته شود:

- قابلیت اطمینان: سیستم باید کاملاً آزموده شود تا اطمینان حاصل شود که نسخه‌های تحویلی به OIE کاملاً عملیاتی و در دسترس هستند.
- قابل گسترش بودن: سیستم باید بتواند برای پوشاندن نیازمندی‌های جدید، به‌راحتی گسترش یابد.
- استقلال از سکو: سیستم باید روی سکوهای مختلف اجرا شود و سیستم‌های مختلف مدیریت پایگاه داده را پشتیبانی نماید.
- واسط کاربرپسند: سیستم باید واسط کاربرپسندی فراهم کند که با قراردادهای معمول مربوط به ظاهر و حس، و نیز قوانین تعامل انسان و ماشین برای کاربردهای تحت وب سازگار باشد.
- امنیت: سیستم باید محتوای وب‌سایت را از حمله‌های بدخواهانه محافظت نموده، از حریم شخصی کاربران خود حفاظت نماید.

ت-۴ ویرایشگر نمودار کلاس UML

این پروژه‌ی تیمی، تحلیل، طراحی، پیاده‌سازی و تست یک ویرایشگر گرافیکی(GED) برای ایجاد و ویرایش نمودارهای کلاس UML را شامل خواهد شد. این پروژه باید توسط تیم‌های شرکت‌کننده انجام شود. فهرست نیازمندی‌های نرم‌افزار را در ادامه می‌بینید. تیم پروژه می‌تواند این فهرست را با افزودن نیازمندی‌های اضافی گسترش دهد:

۱. GED باید به کاربر اجازه دهد که به شکل زیر با پروژه‌ها کار کند:

(آ) GED به کاربر اجازه می‌دهد که یک پروژه‌ی جدید ایجاد نماید. وقتی این گزینه انتخاب می‌گردد، GED باید از کاربر برای اطلاعات لازم پروژه سؤال کند و پوشه‌ها و فایل‌های لازم برای پروژه‌ی جدید را ایجاد نماید.

(ب) GED باید به کاربر اجازه دهد که با انتخاب یک فایل پروژه از یک پوشه‌ی مفروض، یک پروژه‌ی موجود را باز کند.

(ج) GED باید به کاربر اجازه دهد که یک پروژه را ذخیره نماید.

(د) GED باید به کاربر اجازه دهد که یک پروژه را حذف نماید، سپس یک پیام تایید نیز نشان داده شود.

(هـ) GED باید به کاربر اجازه دهد که یک پروژه را ببندد، درصورتی‌که پروژه‌ی ذخیره نشده‌ای وجود داشته باشد، GED به کاربر اطلاع می‌دهد.

۲. GED باید به کاربر اجازه دهد که به شکل زیر با نمودارهای کلاس UML کار کند:

(آ) GED باید به کاربر اجازه دهد که در یک پروژه‌ی مفروض، یک نمودار جدید ایجاد نماید.GED از کاربر می‌خواهد که برای نمودار، یک نام معتبر و یک بسته(package) یا پیمانه(module) انتخاب نماید.

(ب) GED به کاربر اجازه می‌دهد که یک نمودار را ذخیره نماید.

(ج) GED به کاربر اجازه می‌دهد که یک نمودار را با استفاده از یک نام و/ یا محل دیگر ذخیره نماید.

(د) GED به کاربر اجازه می‌دهد که یک نمودار موجود را در یک پروژه باز کند.

(هـ) GED به کاربر اجازه می‌دهد که یک نمودار بازشده را با استفاده از عملیات مختلف ویرایش، شامل موارد زیر ویرایش نماید:

۱. اضافه کردن یک کلاس جدید: GED یک جعبه‌ی گفتگو نشان می‌دهد تا کاربر بتواند اطلاعات کلاس مانند نام کلاس، صفت‌ها و انواع آن‌ها، عملیات و پارامترها و انواع و انواع برگشتی را در آن وارد نماید.
۲. به‌روزرسانی یک کلاس: GED به کاربر اجازه می‌دهد که اطلاعات کلاس را تغییر دهد.
۳. حذف یک کلاس: GED به کاربر اجازه می‌دهد که با اطلاع‌رسانی به کاربر از طریق یک جعبه‌ی گفتگوی نیازمند به تایید، یک کلاس را حذف کند.
۴. اضافه کردن یک رابطه: GED راه کاربرپسندی فراهم می‌نماید که به کاربر اجازه دهد رابطه هایی بین یک یا چند کلاس ایجاد نماید.
۵. انتخاب یک یا چند عنصر گرافیکی: GED راه کاربرپسندی فراهم می نماید که به کاربر اجازه دهد یک یا چند عنصر گرافیکی (کلاس‌ها، رابطه‌ها، یا ترکیب آنها) از یک نمودار UML را انتخاب نماید.
۶. جابه‌جا کردن عناصر گرافیکی انتخاب‌شده:GED به کاربر اجازه می‌دهد که المان‌های منتخب خود را با کشیدن آنها به محل دلخواه جابه‌جا نماید.
۷. کپی کردن، برش، و چسباندن: GED توانمندی‌های معمول «Copy Paste» و «Cut Paste» را فراهم می سازد.
۸. حذف المان‌های انتخاب‌شده: GED به کاربر اجازه می‌دهد که المان‌های انتخاب‌شده را حذف نماید.
۹. باطل‌کردن و دوباره‌انجام‌دادن کارها: GED به کاربر اجازه می دهد که تا حداکثر ۱۰ مرتبه‌ی ویرایش، عملیات Undo و Redo را انجام دهد (در نسخه‌های آینده، کاربر می‌تواند تعداد عملیات قابل Undo و Redo را مشخص نماید).

(و) GED با استفاده از یک پایگاه‌داده یا سیستم‌فایل، از ماندگاری اشیا پشتیبانی می‌کند.GED انباره‌ی مانا را از دید کاربر مخفی می‌نماید تا بتواند در آینده برای پشتیبانی پایگاه‌داده‌ها از انواع مختلف و پایگاه‌داده‌های چندگانه، سیستم را توسعه دهد.

(ز) GED باید با استفاده از فرایند نرم‌افزار و متدولوژی‌های نرم افزاری پوشش‌داده‌شده در این درس و الگوهای طراحی، طراحی گردد.

(ح) GED باید با استفاده از متدهای آزمون آموزش‌داده‌شده و ابزارهای آزمون دردسترس، در سطح آزمون واحد، آزمون یکپارچگی و آزمون پذیرش، تست گردد. موارد آزمون باید توسعه داده شود و نتایج تست ثبت گردد.

۳. GED باید به انتخاب کاربر، چارچوب کد منبع C++ یا جاوا تولید نماید.

ت-۵ شبیه‌ساز ارتباطات رادیویی

این پروژه در مورد فرستنده/گیرنده ی موبایلی کامپیوتری است که سیستم های جاسازی شده‌ی زمان حقیقی هستند. یک فرستنده/گیرنده‌ی کامپیوتری از سخت‌افزار و نرم‌افزار تشکیل می گردد. موبایل‌ها (که به آنها واحدها گفته می‌شود) از طریق یک ایستگاه مبنا و تحت‌کنترل آن با هم ارتباط برقرار می‌نمایند. به این ایستگاه، «پایگاه» گفته می شود. سخت‌افزار هر واحد می تواند سیگنال‌های رادیویی پیمانه بندی شده را دریافت و ارسال نماید و با توجه به وضعیت کانال (مشغول یا غیرمشغول) و دستورات و شماره‌های دریافت‌شده از پایگاه، اطلاعات نرم‌افزاری سیستم را تهیه کند.

سیستم نرم‌افزاری باید اطلاعات (مانند ID واحد و شماره‌های ANI[۱] را ذخیره و پردازش نماید، و سخت‌افزار را به شکل مناسبی کنترل نماید. پایگاه سیگنال‌های رادیویی را از واحدها دریافت می‌کند و بررسی می‌نماید که واحدهای تماس‌گیرنده و تماس‌پذیرنده، مشترکین قانونی سیستم باشند. همچنین برای مطلع کردن واحد مورد تماس، سیگنال‌های رادیویی منتشر می‌نماید . تماس‌ها می‌تواند از نوع موبایل به موبایل، موبایل به تلفن، و یا تلفن به موبایل باشد. پایگاه تعداد محدودی خط تلفن دارد که طوری به تلفن‌ها وصل می‌شود که تماس‌ها بین واحدهای موبایل و تلفن‌ها بتواند آرشیو گردد.

پروژه با استفاده از متدولوژی چابک یکنواخت، تحلیل و طراحی را انجام می‌دهد و یک پیش‌نمونه برای شبیه‌سازی سیستم در جاوا پیاده‌سازی می‌نماید. هدف از پروژه، مدل‌سازی نرم‌افزار و شبیه‌سازی رفتارهای سخت‌افزار است. سیستم نرم‌افزاری یک فرستنده / گیرنده‌ی موبایل به کاربر اجازه می‌دهد که با استفاده از دستورات زیر فرستنده / گیرنده را برنامه‌ریزی نماید. متن محصور شده در > و < پاسخ‌های سیستم را نشان می‌دهد.

Command	Description
0# <3 beeps, pause, 3 beeps>	Exit programming mode
1# <beep> 4 digit ID	Enter this transceiver's ID number
2# <beep> 4 digit ID	Enter group 1 ID number
3# <beep> 4 digit ID	Enter group 2 ID number
4# <beep> 4 to 7 digit ANI number # <3 beeps>	Enter this transceiver's automatic number Identification number
5# <beep> 0 (disable) or 1 (enable)	Disable/enable this user a priority user
6# <beep> 2 digit code (00-41)	Define CTCSS tone
7# <beep> 0 (carrier) or 1 (CTCSS tone)	Define busy channel detection mode
8# <beep> 4 digit code	Define manual operation access code
9# <beep> 2 digit (01-16)	Define number of radio channels available
10# <beep> 1 to 4 digit reset code # <3 beeps>	Define transceiver reset code

[۱] Automatic Number Identification

نرم‌افزار واحد عملیات زیر را انجام می‌دهد:

۱-مونیتور کردن تماس های دریافتی

این وضعیت در حال اسکن یک واحد است. سخت افزار واحد به‌طور خودکار کانال‌های رادیویی را اسکن می‌نماید تا یک کانال مشغول پیدا کند. نرم افزار این را حس می‌کند و شماره شناسایی(ID) دریافتی از کانال را می‌خواند. اگر شماره‌ی شناسایی برابر با شماره شناسایی واحد بود، سخت افزار روی کانال متمرکز می‌شود و زنگ (بلندگو) را به صدا درمی‌آورد. وقتی کاربر کلید پاسخگویی (PTT) را فشار می دهد، عدد ANI واحد به پایگاه منتقل می‌شود و ارتباط برقرار می‌گردد. اگر عدد ID برابر با عدد ID واحد نبود، نرم‌افزار به سخت‌افزار فرمان می‌دهد که به کانال بعد برود و اسکن‌کردن را دوباره آغاز نماید.

۲- تماس از یک موبایل به یک خط ثابت

کاربر کلید پاسخگویی (PTT) و '*' را هم زمان فشار می‌دهد. واحد، تحت کنترل نرم افزار، برای یک کانال صاف (مثلاً یک کانال آزاد و بدون نویز) جستجو می‌کند تا یکی را پیدا کند و یا همه‌ی کانال ها را امتحان کرده باشد. اگر یک کانال صاف پیدا شد، واحد روی آن کانال متمرکز می‌شود و'*' و شماره‌ی ANI واحد را به پایگاه می‌فرستد. اگر یک کانال صاف دیگر توسط پایگاه پیدا شد، واحد یک آوای شماره‌گیری دریافت می‌کند. سپس کاربر می‌تواند کلید PTT را فشار دهد و آن را تا پایان شماره‌گیری تلفن ثابت نگاه دارد. عدد توسط نرم افزار خوانده می‌شود و برای انتقال به سخت افزار فرستاده می‌شود. وقتی طرف مورد تماس گوشی را بردارد، تماس برقرار می‌شود. اگر پایگاه یک کانال صاف پیدا نکرد، واحد یک آوای خطا دریافت خواهد کرد و نرم‌افزار، سخت‌افزار را به حالت در حال اسکن برخواهدگرداند، کاربر ممکن است دوباره اقدام به برقراری تماس نماید.

۳-تماس از موبایل به موبایل

در حالت اسکن کردن، کاربر می‌تواند کلید های PTT و '۰' را هم‌زمان فشار دهد و رها کند. نرم‌افزار این عمل را تشخیص می‌دهد و شروع به جستجوی یک کانال صاف برای ارسال این درخواست می‌نماید. اگر یک کانال صاف پیدا شد، نرم‌افزار شماره‌ی ANI واحد را برای انتقال، به سخت‌افزار می‌فرستد، در غیراین‌صورت، نرم‌افزار به سخت‌افزار فرمان می‌دهد که یک آوای خطا پخش نماید. پس از ارسال شماره‌ی ANI واحد، کاربر یک صدای بوق می‌شنود و در این زمان، کاربر باید دکمه‌ی PTT را فشار دهد و در زمان شماره‌گیری شماره‌ی ID موبایل مقصد، آن را نگاه دارد، وقتی موبایل مقصد پاسخ گفته شد، تماس برقرار می‌شود. اگر پایگاه، دستگاه را پیدا نکرد، یک آوای خطا به واحد تماس‌گیرنده می‌فرستد و نرم‌افزار واحد، سخت‌افزار را به حالت اسکن کردن برمی‌گرداند.

۴-پایان یک تماس

کاربر کلیدهای PTT و'' را به‌طور هم‌زمان فشار می‌دهد. نرم‌افزار به سخت‌افزار فرمان می‌دهد که کلید ''و سپس شماره‌ی ANI واحد را ارسال کند.

۵-دسترسی به حالت سنتی

کاربر کلید PTT را فشار می‌دهد و هم‌زمان کد دسترسی به حالت سنتی، که کدی چهاررقمی است، را وارد می‌نماید. نرم‌افزار سه آوای بوق ایجاد می‌کند و سخت‌افزار آنها را پخش می‌کند.

۶-خروج از حالت سنتی

در این حالت دو وضعیت وجود دارد. کاربر می‌تواند کلید PTT و کلید ''را فشاردهد و رادیو این را با سه بوق اطلاع دهد. به‌علاوه می‌توان با دریافت کد reset از پایگاه از حالت سنتی خارج شد.

ت-۶ محیط آزمون حالت اشیا

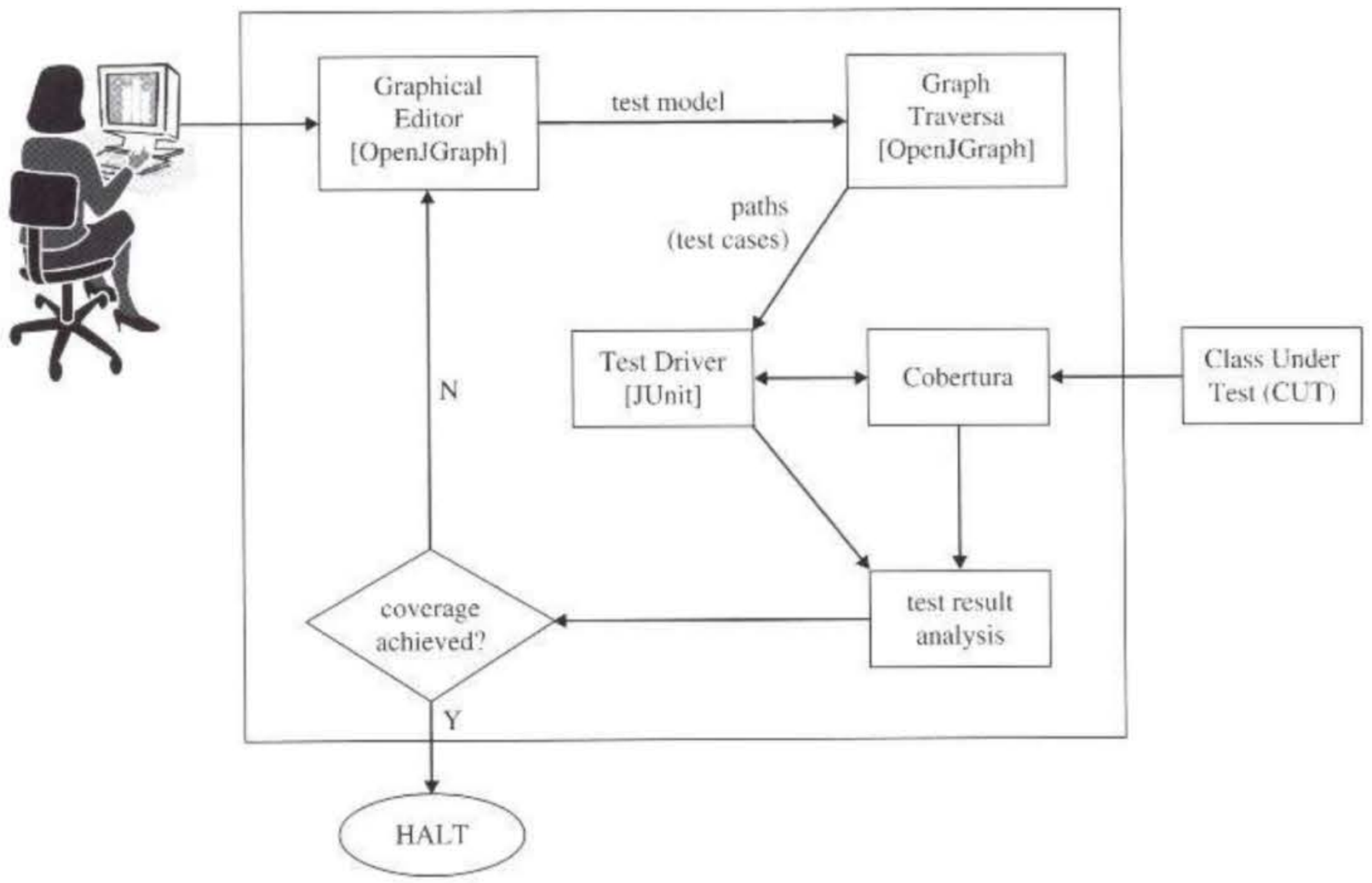
این پروژه‌ی تیمی، یک ابزار ساده برای آزمون حالت شی برمبنای روش Class Bench که در فصل ۲۰ شرح داده شده است را پیش‌نمونه‌سازی می‌نماید. این پروژه بیشتر گام‌های آزمون را خودکارسازی خواهد کرد.

این ابزار سعی می‌کند که سه ابزار موجود را یکپارچه نماید:

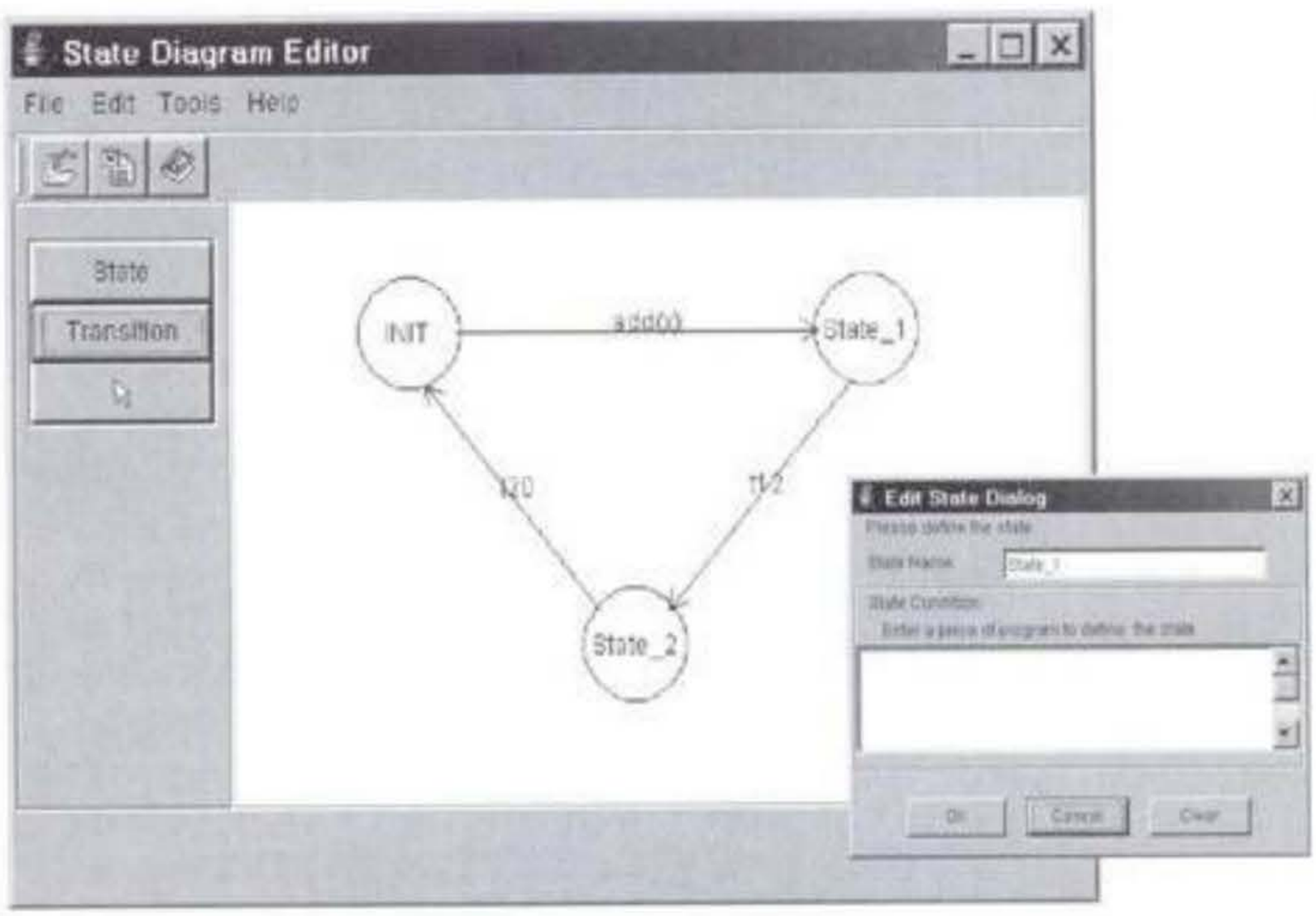
۱. OpenJGraph (قابل دسترسی از Source Forge)
۲. Junit (http://www.junit.org/index.htm)
۳. Cobertura یا JCoverage (قابل دسترسی از Source Forge)

به علاوه، Eclipse، NetBeans، ویا ant می‌توانند برای یکپارچه‌سازی ابزارهای فوق مورد استفاده قرار گیرند. پیاده‌سازی در جاوا خواهد بود، زیرا این ابزارها خود در جاوا پیاده‌سازی شده‌اند. شکل ت-۱ جریان ابزار پیشنهادی را نشان می‌دهد. ابزار دارای نیازمندی های زیر است:

۱. ابزار باید یک GUI برای کاربر فراهم سازد که بتواند یک مدل آزمون شاخه را با استفاده از Open JGraph ویرایش نماید. کاربر باید شرایط حالت و عملیات انتقال را توصیف نماید. این‌ها به ترتیب، شرایط جاوا و جملات جاوا هستند. شکل ت-۲ یک GUI نمونه‌ی پیاده‌سازی شده در جاوا را نشان می‌دهد . در این پروژه، برای سادگی، پیاده‌سازی از Open JGraph استفاده‌ی مجدد می‌نماید. جعبه گفت‌وگوی Edit State به کاربر اجازه می‌دهد که شرایط حالت را توصیف نماید. به همین‌ترتیب جملات جاوا می‌توانند برای عملیات یک انتقال مورد استفاده قرار گیرد.
۲. ابزار عملیات زیر را بنا به درخواست کاربر، در یک گام یا در گام‌های متعدد انجام خواهد داد.
   (آ) ابزار به‌طور خودکار از مدل آزمون، پیمایش‌های Open JGraph را تولید خواهد کرد.
   (ب) ابزار به طور خودکار از مدل پیمایش‌ها، زیر کلاسJUnit Test Case شامل متدهای آزمون را تولید خواهد کرد.
   (ج) ابزار CUT را اجرا خواهد کرد.
   (د) ابزار آزمون های JUnit را اجرا خواهد نمود و اطلاعات پوشش را نشان خواهد داد.
۳. اگر دانشجویان الگوی حالت را آموخته‌اند از الگوی حالت برای طراحی و پیاده‌سازی استفاده کنند.



شکل ت-۱: جریان کار ابزار



شکل ت-۲: یک واسط کاربری گرافیکی برای ویرایش مدل آزمون ClassBench

مراجع

1. M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti, "Control-flow Integrity," in *Proceedings of the 12th ACM Conference on Computer and Communications Security* (CCS), 2005, pp. 340–353.
2. P. Abrahamsson, J. Warsta, M. Siponen, and J. Ronkainen, "New Directions on Agile Methods: A Comparative Analysis," in *Proceedings of the 25th International Conference on Software Engineering* (ICSE'03), Oregon, 2003, pp. 244–254.
3. Mathur Aditya, *Foundations of Software Testing* (Boston: Addison-Wesley Professional, 2008).
4. K. K. Aggarwal and Yogesh Singh, *Software Engineering* (3rd ed.), (New Delhi, India: New Age International Publishers, 2007).
5. Robert Aiello and Leslie Sachs, *Configuration Management Best Practices: Practical Methods That Work in the Real World* (Boston: Addison-Wesley Professional, 2010).
6. P. Akritidis, C. Cadar, C. Raiciu, M. Costa, and M. Castro, "Preventing Memory Error Exploits with WIT," in *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, Washington, DC, 2008. IEEE Computer Society, pp. 263–277.
7. J. Aldrich, C. Chambers, and D. Notkin, "Architectural Reasoning in ArchJava," in *Proceedings of the European Conference on Object-Oriented Programming* (ECOOP), vol. 2374, 2002, pp. 185–193.
8. Roger T. Alexander and A. J. Offutt, "Analysis Techniques for Testing Polymorphic Relationships," in *Proceedings of 30th IEEE International Conference on Technology of Object-Oriented Languages and Systems* (TOOLS 30), Santa Barbara, CA, August 1–5, 1999, pp. 104–114.
9. Ian Alexander, "Misuse Cases: Use Cases with Hostile Intent," *IEEE Software*, vol. 20, no. 1, January/February 2003, pp. 58–66.
10. Paul Ammann and Jeff Offutt, *Introduction to Software Testing* (Cambridge, UK: Cambridge University Press, 2008).
11. Scott Ambler, *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process* (Hoboken, NJ: Wiley 2002).
12. D. J. Anderson, "Stretching Agile to Fit CMMI Level 3—The Story of Creating MSF for CMMI? Process Improvement at Microsoft Corporation," in *Proceedings of 2005 Agile Conference*, Denver, CO, July 24–29, 2005, pp. 193–201.
13. K. Arnold, J. Gosling, and David Holmes, *The Java Programming Language*, 4th ed. (Upper Saddle River, NJ: Prentice Hall, 2005).
14. David Astels, *Test-Driven Development: A Practical Guide*, 2nd ed. (Upper Saddle River, NJ: Prentice Hall, 2003).
15. David Avison and Guy Fitzgerald, "Where Now for Development Methodologies," *Communications of the ACM*, vol. 46, no. 1 (January 2003), pp. 79–82.
16. Victor R. Bassili, "Viewing Maintenance as Reuse-oriented Software Development," *IEEE Software*, vol. 7, no. 1 (1990), pp. 19–25.
17. Victor R. Bassili, Lionel C. Briand, and Walcelio L. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators," *IEEE Transactions on Software Engineering*, vol. 22, no. 10 (October 1996), pp. 751–761.
18. Len Bass, Paul Clements, and Rick Kazman, *Software Architecture in Practice* (Boston: Addison-Wesley, 2003).
19. Kent Beck and Martin Fowler, *Planning Extreme Programming* (Boston: Addison-Wesley Professional, 2000).
20. Kent Beck, *Test-Driven Development: By Example* (Boston: Addison-Wesley Professional, 2002).
21. Kent Beck, *Extreme Programming Explained: Embrace Change*, 2nd ed. (Boston: Addison-Wesley, 2004).
22. B. Beizer, *Software Testing Techniques*, 2nd ed. (Hoboken, NJ: Van Nostrand Reinhold, 1990).

23. David Bellagio and Tom J. Milligan, *Software Configuration Management Strategies and IBM Rational ClearCase: A Practical Introduction,* 2nd ed. (Indianapolis, IN: IBM Press, 2005).

24. S. P. Berczuk and B. Appleton, *Software Configuration Management Patterns: Effective Teamwork, Practical Integration* (Boston: Addison-Wesley, 2003).

25. Berzins and Luqi, *Software Engineering with Abstractions* (Boston: Addison-Wesley, 1991).

26. T. Bhat, et al., "Evaluating the Efficacy of Test-Driven Development: Industrial Case Studies," *Proceedings of International Symposium on Empirical Software Engineering,* Rio de Janiero, Brazil, September 21–22, 2006, pp. 356–363.

27. R. Binder, "Design for Testability in Object-Oriented Systems," *CACM,* vol. 37, no. 9 (September 1994), pp. 87–101.

28. Robert Binder, *Testing Object-Oriented Systems: Models, Patterns, and Tools* (Boston: Addison-Wesley, 2000).

29. Kurt Bittner and Ian Spence, *Use Case Modeling* (Boston: Addison-Wesley Professional, 2002).

30. Michael R. Blaha and James R. Rumbaugh, *Object-Oriented Modeling and Design with UML,* 2nd ed. (Upper Saddle River, NJ: Prentice Hall, 2004).

31. Benjamin S. Blanchard, *System Engineering Management,* 4th ed. (Hoboken, NJ: Wiley, 2008).

32. Boehm, B. "A Spiral Model of Software Development and Enhancement," *IEEE Computer,* vol. 21, no. 5 (May 1988), pp. 61–72.

33. Barry Boehm, "Software Risk Management: Principles and Practices," *IEEE Software,* January 1991, pp. 32–41.

34. Barry W. Boehm, Chris Abts, A. Winsor Brown, Sunita Chulani, Bradford K. Clark, Ellis Horowitz, Ray Madachy, Donald J. Reifer, and Bert Steece, *Software Cost Estimation with COCOMO II* (Upper Saddle River, NJ: Prentice Hall, 2000).

35. Barry Boehm and Richard Turner, "Observations on Balancing Discipline and Agility," *Proceedings of the Conference on Agile Development,* IEEE Computer Society, 2003, pp. 32–39.

36. G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide,* 2nd ed. (Boston: Addison-Wesley, 2005).

37. L. Briand, Y. Labiche, and Y. Wang, "Revisiting Strategies for Ordering Class Integration Testing in the Presence of Dependency Cycles," in *Proceedings of the 12th International Symposium on Software Reliability Engineering* (ISSRE), November 2001, pp. 287–296.

38. L. C. Briand, Y. Labiche, and Y. Wang, "An Investigation of Graph-Based Class Integration Test Order Strategies," *IEEE Transactions on Software Engineering,* vol. 29, no. 7 (July 2003), pp. 594–607.

39. F. P. Brooks Jr., *The Mythical Man-Month,* 2nd ed. (Boston: Addison-Wesley Professional, 1995).

40. Bernd Bruegge and Allen H. Dutoit, *Object-Oriented Software Engineering: Using UML, Patterns, and Java,* 3rd ed. (Upper Saddle River, NJ: Prentice Hall, 2009).

41. C. Cadar, V. Ganesh, P. Pawlowski, D. Dill, and D. Engler, "EXE: Automatically Generating Inputs of Death," in *Proceedings of the 13th ACM Conference on Computer and Communications Security* (CCS'06), 2006, pp. 322–335.

42. C. Cadar, D. Dunbar, and D. Engler, "KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex System Programs," in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation* (OSDI), 2008.

43. W. Chang, B. Streiff, and C. Lin, "Efficient and Extensible Security Enforcement Using Dynamic Data Flow Analysis," in *Proceedings of the 15th ACM Conference on Computer and Communications Security* (CCS), New York, 2008, pp. 39–50.

44. Brian Chess and Jacob West, *Secure Programming with Static Analysis* (Boston: Addison Wesley, 2007).

45. S. Chidamber and C. F. Kemerer, "A Metrics Suite for Object-Oriented Design," *IEEE Trans. on Software Engineering,* vol. 20, no. 6 (June 1994), pp. 476–493.

46. Elliot J. Chikofsky and James H. Cross, "Reverse Engineering and Design Recovery: A Taxonomy," *IEEE Software,* vol. 7, no. 1 (1990), pp. 13–17.

47. J. Clause and A. Orso, "Penumbra: Automatically Identifying Failure-Relevant Inputs Using Dynamic Tainting," in *Proceedings of the 18th International Symposium on Software Testing and Analysis* (ISSTA), New York, 2009, pp. 249–260.

48. Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Paulo Merson, Robert Nord, and Judith Stafford, *Documenting Software Architectures: Views and Beyond,* 2nd ed. (Boston: Addison-Wesley Professional, 2010).

49. Alistair Cockburn, *Writing Effective Use Cases* (Boston: Addison-Wesley Professional, 2000).

50. Alistair Cockburn, *Agile Software Development: The Cooperative Game* (Boston: Addison-Wesley Professional, 2006).

51. Mike Cohn, *Agile Estimating and Planning* (Upper Saddle River, NJ: Prentice Hall, 2006).

52. Mike Cohn, *Succeeding with Agile: Software Development Using Scrum* (Boston: Addison-Wesley Professional, 2009).

53. Lisa Crispin and Janet Gregory, *Agile Testing: A Practical Guide for Testers and Agile Teams* (Boston: Addison-Wesley Professional, 2009).

54. http://netbeans.org/kb/docs/ide/cvs.html#checking

55. Marco D'Ambros, Alberto Bacchelli, Michele Lanza, "On the Impact of Design Flaws on Software Defects," in *Proceedings of IEEE 10th International Conference on Quality Software* (QSIC), Zhangjiajie, China, July 14–15, 2010, pp. 23–31.

56. Alan A. Davis, *Software Requirements: Objects, Functions and States,* rev. ed. (Upper Saddle River, NJ: Prentice Hall, 1993).

57. Alan A. Davis, et al., "Identifying and Measuring Quality in a Software Requirements Specification," in *Proceedings of the First International Software Metrics Symposium,* IEEE, Baltimore, MD, 1993, pp. 141–152.

58. Chad Dougherty, Kirk Sayre, Robert C. Seacord, David Svoboda, and Kazuya Togashi, "Secure Design Patterns," Technical Report, CMU/SEI-2009-TR-010, March 2009.

59. M. Dowson, "The Ariane 5 Software Failure," *Software Engineering Notes,* vol. 22, no. 2 (March 1997), p. 84.

60. H. Erdogmus, M. Morisio, and M. Torchiano, "On the Effectiveness of the Test-First Approach to Programming," *IEEE Transactions on Software Engineering,* vol. 31, no. 3 (2005), pp. 226–237.

61. R. Eshuis and R. Wieringa, "Tool Support for Verifying UML Activity Diagrams," *IEEE Transactions on Software Engineering,* vol. 30, no. 7 (2004), pp. 437–447.

62. Donald Firesmith, "Engineering Security Requirements," *Journal of Object Technology,* vol. 2, no. 1 (January–February 2003), pp. 53–68.

63. Martin Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language,* 3rd ed. (Boston: Addison-Wesley Professional, 2003).

64. Elisabeth Freeman, Eric Freeman, Bert Bates, Kathy Sierra, and Elisabeth Robson, *Head First Design Patterns* (Sebastopol, CA: O'Reilly Media, 2004).

65. Sanford Friedenthal, Alan Moore, and Rick Steiner, *A Practical Guide to SysML,* 2nd ed. (Burlington, MA: Morgan Kaufmann, 2011).

66. Wilbert O. Galitz, *The Essential Guide to User Interface Design* (Hoboken, NJ: Wiley, 2007).

67. E. Gamma, Richard Helm, Ralph Johnson and John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software* (Boston: Addison-Wesley, 1995).

68. V. Ganesh, T. Leek, and M. Rinard, "Taint-Based Directed Whitebox Fuzzing," in *Proceedings of 31st IEEE International Conference on Software Engineering,* Washington DC, 2009, pp. 474–484.

69. M. Ganis, D. Leip, F. Grossman, and J. Bergin, "Introducing Agile Development (XP) into a Corporate Webmaster Environment—An Experience Report," in *Proceedings of Agile Conference,* Denver, CO, July 24–29, 2005, pp. 145–152.

70. J. Gao, C. Chen, T. Yasufumi, D. Kung, and P. Hsia, "Identifying Polymorphism Change and Impact in Object-Oriented Programs," *Journal of Software Maintenance,* 1997.

71. P. Godefroid, M. Y. Levin, and D. Molnar, "Automated Whitebox Fuzz Testing," *Network and Distributed Systems Security,* 2008, pp. 151–166.

72. Mark G. Graff and Kenneth R. van Wyk, *Secure Coding* (Sebastopol, CA: O'Reilly Media, 2003).

73. Penny Grubb and Armstrong A. Takang, *Software Maintenance: Concepts and Practice*, 2nd ed. (Singapore: World Scientific Publishing Company, 2003).

74. Charles B. Haley, Robin Laney, Jonathan D. Moffett, and Bashar Nuseibeh, "Security Requirements Engineering: A Framework for Representation and Analysis," *IEEE Transactions on Software Engineering,* vol. 34, no. 1 (January/February 2008), pp. 133–153.

75. Spyros T. Halkidis and Nikolaos Tsantalis, "Architectural Risk Analysis of Software Systems Based on Security Patterns," *IEEE Transactions on Dependable and Secure Computing,* vol. 5, no. 3 (July–September, 2008), pp. 129–142.

76. D. Harel, "On Visual Formalisms," *CACM,* vol. 31, no. 5 (May 1988), pp. 514–531.

77. M. J. Harrold, J. D. McGregor, and K. J. Fitzpatrick, "Incremental Testing of Object-Oriented Class Structure," in *Proceedings of 14th International Conference on Software Engineering*, 1992, pp. 68–80.

78. David Hay, *Requirements Analysis: From Business Views to Architecture* (Upper Saddle River, NJ: Prentice Hall 2011).

79. James Herbsleb, David Zubrow, Dennis Goldenson, Will Hayes, and Mark Paulk, "Software Quality and the Capability Maturity Model," *Communications of the ACM*, vol. 40, no. 6 (1997), pp. 30–40.

80. James A. Highsmith, *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems* (New York: Dorset House Publishing Company, 1999).

81. D. Hoffman and P. Strooper, "ClassBench: A Framework for Automated Class Testing," *Software Practice and Experience*, vol. 27, no. 5 (1997), pp. 573–597.

82. G. Hoglund and G. McGraw, *Exploiting Software* (Boston: Addison-Wesley, 2004).

83. Cay S. Horstmann and Gary Cornell, *Core Java, Volume 1: Fundamentals*, 8th ed. (Upper Saddle River, NJ: Prentice Hall, 2007).

84. Cay S. Horstmann and Gary Cornell, *Core Java, Volume 2: Advanced Features*, 8th ed. (Upper Saddle River, NJ: Prentice Hall, 2008).

85. P. Hsia, X. Li, C. Hsu, and D. Kung, "A Technique for Selective Revalidation of OO Software," *Journal of Software Maintenance*, 1997, pp. 217–233.

86. Watts S. Humphrey, *Introduction to the team software process* (Boston: Addison-Wesley, 2000).

87. Watts S. Humphrey, *PSP: A Self-Improvement Process for Software Engineers* (Boston: Addison-Wesley, 2005).

88. Ming Huo, J. Verner, Liming Zhu, and M. A. Babar, "Software Quality and Agile Methods," in *Proceedings of the IEEE International Conference on Computer Software and Applications*, Hong Kong, 2004. pp. 520–525.

89. IEEE Std. 1219-1998, *Standard for Software Maintenance* (Los Alamitos, CA: IEEE Computer Society Press, 1998).

90. Ivar Jacobson, *Object-Oriented Software Engineering: A Use Case Driven Approach* (Reading, MA: Addison-Wesley, 1992).

91. Ivar Jacobson, James Rumbaugh, and Grady Booch, *Unified Software Development Process* (Boston: Addison-Wesley, 1999).

92. Jim Johnson, *Chaos: Charting the Seas of Information Technology* (Boston: The Standish Group, 1994).

93. Jeff Johnson, *Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Rules* (Burlington, MA: Morgan Kaufmann, 2010).

94. C. Jones, *Assessment and Control of Software Risks* (Upper Saddle River, NJ: Prentice Hall, 1994).

95. Christopher M. King, Curtis E. Dalton, and Ertem Osmanoglu, *Security Architecture: Design, Development and Operations* (New York, NY: McGraw-Hill, 2001).

96. Juha Koskela, "Software Configuration Management in Agile Methods," 2003, www.vtt.fi/inf/pdf/publications/2003/P514.pdf.

97. Lasse Koskela, *Test Driven: Practical TDD and Acceptance TDD for Java Developers* (Greenwich, CT: Manning Publications, 2007).

98. Alexander Kossiakoff, William N. Sweet, Sam Seymour, and Steven M. Biemer, *Systems Engineering Principles and Practice*, 2nd ed. (Hoboken, NJ: Wiley-Interscience, 2011).

99. D. Kung, J. Gao, P. Hsia, J. Lin, and Y. Toyoshima, "Design Recovery for Software Testing of Object-Oriented Programs," in *Proceedings of the Working Conference on Reverse Engineering*, Baltimore, MD, May 21–23 (Los Alamitos, CA: IEEE Computer Society Press, 1993), pp. 202–211.

100. D. Kung, "On Decision Tree Verification and Consolidation," *Journal of Information and Software Technology*, vol. 36, no. 7 (1994), pp. 485–494.

101. D. Kung, J. Gao, P. Hsia, Y. Toyoshima, C. Chen, Y. S. Kim, and Y. Song, "Developing an Object-Oriented Software Testing and Maintenance Environment," *Communications of the ACM*, vol. 38, no. 10 (October 1995), pp. 75–87.

102. D. Kung, J. Gao, P. Hsia, F. Wen, Y. Toyoshima, and C. Chen, "On Regression Testing of Object-Oriented Programs," *Journal of Systems and Software*, vol. 32, no. 1 (January 1996), pp. 21–40.

103. Y. Labiche, P. Thevenod-Fosse, H. Waeselynck, and M. H. Durand, "Testing Levels for Object-Oriented Software," in *Proceedings of the International*

Conference on Software Engineering, 2000, pp. 136–145.

104. Craig Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, 3rd ed. (Upper Saddle River, NJ: Prentice Hall, 2005).

105. N. Leveson, S. Cha, and T. Shimeall, "Safety Verification of Ada Programs Using Software Fault Trees," *IEEE Software* (July 1991), pp. 48–59.

106. Rudiger Lincke, Tobias Gutzmann, and Welf Lowe, "Software Quality Prediction Models Compared," in *Proceedings of the IEEE 10th International Conference on Quality Software* (QSIC), Zhangjiajie, China, July 14–15, 2010, pp. 82–91.

107. M. Linton, "Composing User Interfaces with InterViews," *IEEE Computer* vol. 22 (February 1989), pp. 8–22.

108. Mike Mason, *Pragmatic Guide to Subversion*, (Pragmatic Bookshelf, 2010).

109. T. J. McCabe and C. W. Butler, "Design Complexity Measurement and Testing," *CACM*, vol. 32, no. 12, (December 1989), pp. 1415–1425.

110. J. McDermott, "Abuse-Case-Based Assurance Argument," in *Proceedings of the 17th Computer Security and Applications Conference*, Washington DC, 2001, pp. 366–374.

111. John McDermott and Chris Fox, "Using Abuse Case Models for Security Requirements Analysis," in *Proceedings of the 15th Annual Computer Security and Applications Conference*, Phonenix, AZ, 1999, pp. 55–64.

112. J. McDonald and Paul Strooper, "Testing Inheritance Hierarchies in the ClassBench Framework," in *Proceedings of the Technology of Object-Oriented Languages and Systems*, Melbourne, Australia, November 23–26, 1998.

113. Gary McGraw, "Software Security Testing," *IEEE Security and Privacy*, 2004, pp. 32–36.

114. Gary McGraw, *Software Security: Building Security In* (Boston: Addison-Wesley, 2006).

115. Tom Mens, "A State-of-the-Art Survey on Software Merging," *IEEE Transactions on Software Engineering*, vol. 28, no. 5 (May 2002), pp. 449–462.

116. N. Nagappan, E. M. Maximilien, T. Bhat, and L. Williams, "Realizing Quality Improvement through Test Driven Development: Results and Experiences of Four Industrial Teams," *Empirical Software Engineering*, vol. 13, no. 3 (2008), pp. 289–302.

117. Douglas W. Nance, *Fundamentals of Pascal, Understanding Programming and Problem Solving*, 3rd ed. (Course Technology, 1997).

118. Nils J. Nilsson, "Artificial Intelligence: A New Synthesis" (Burlington, MA: Morgan Kaufmann, 1998).

119. Stephen R. Palmer and John M. Felsing, *A Practical Guide to Feature-Driven Development* (Upper Saddle River, NJ: Prentice Hall, 2002).

120. C. Ravindranath Pandian, *Applied Software Risk Management: A Guide for Software Project Managers* (Boca Raton, FL: Auerbach Publications, 2006).

121. Mark C. Paulk, "A Comparison of ISO 9001 and the Capability Maturity Model for Software," Technical Report, CMU/SEI-94-TR-12, ESC-TR-94-12, July 1994.

122. D. E. Perry and G. E. Kaiser, "Adequate Testing and Object-Oriented Programming," *Journal of Object-Oriented Programming*, vol. 2 (January–February 1990), pp. 13–19.

123. Shari Lawrence Pfleeger and Joanne M. Atlee, *Software Engineering: Theory and Practice*, 4th ed. (Upper Saddle River, NJ: Prentice Hall, 2009).

124. Mary Poppendieck and Tom Poppendieck, *Lean Software Development: An Agile Toolkit* (Boston: Addison-Wesley, 2003).

125. Roger S. Pressman, *Software Engineering: A Practitioner's Approach*, 6th ed. (New York, NY: McGraw-Hill, 2004).

126. A. Qumer and B. Henderson-Sellers, "A Framework to Support the Evaluation, Adoption, and Improvement of Agile Methods in Practice," *Journal of Systems and Software*, vol. 81, no. 11 (2008), pp. 1899–1919.

127. Joseph Raynus, *Software Process Improvement with CMM* (Boston: Artech House, 1999).

128. F. Ricca and P. Tonella, "Analysis and Testing of Web Applications," in *Proceedings of the 23rd International Conference on Software Engineering*, 2001, pp. 25–34.

129. M. F. Ringenburgand and D. Grossman, "Preventing Format-String Attacks via Automatic and Efficient Dynamic Checking," in *Proceedings of the 12th ACM Conference on Computer and Communications Security* (CCS), New York, 2005, pp. 354–363.

130. G. Rothermel and M. J. Harrold, "Selecting Regression Tests for Object-Oriented Software," in *Proceedings of the IEEE International Conference On Software Maintenance,* Monterey, CA, 1994, pp. 14–25.

131. J. Rumbaugh, M. Blaha, W. Premerlani, and F. Eddy, *Object-Oriented Modeling and Design* (Upper Saddle River, NJ: Prentice Hall, 1991).

132. J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual* (Boston: Addison-Wesley, 1998).

133. S. R. Schach, *Classical and Object-Oriented Software Engineering with UML and Java* (New York: McGraw-Hill Companies, 1999).

134. Hossain Shahriar and Mohammad Zulkernine, "Mitigating Program Security Vulnerabilities: Approaches and Challenges," *ACM Computing Surveys,* vol. 44, no. 3, Article 11, June 2012. pp. 11:1–11:46.

135. Alan C. Shaw, *Real-Time Systems and Software* (Hoboken, NJ: John Wiley and Sons, 2001).

136. Guttorm Sindre and Andreas L. Opdahl, "Eliciting Security Requirements by Misuse Cases," in *Proceedings of Technology for Object-Oriented Language and Systems,* Sydney, Australia, November 20–23, 2000, pp. 120–131.

137. S. Sinha and M. J. Harrold, "Analysis and Testing of Programs with Exception Handling Constructs," *IEEE Transactions on Software Engineering,* vol. 26, no. 9 (September 2000).

138. Jon Skeet, *C# in Depth* (Greenwich, CT: Manning Publications, 2008).

139. Ian Sommerville, *Software Engineering,* 9th ed. (Boston: Addison-Wesley, 2010).

140. Jennifer Stapleton and Peter Constable, *DSDM: Dynamic Systems Development Method: The Method in Practice* (Boston: Addison-Wesley Professional, 1997).

141. B. Stroustrup, *The C++ Programming Language: Special Edition,* 3rd ed. (Boston: Addison-Wesley Professional, 2000).

142. B. Stroustrup, *Programming: Principles and Practice Using C++,* 2nd ed. (Boston: Addison-Wesley Professional, 2008).

143. http://netbeans.org/kb/docs/ide/subversion.html#checking

144. K. Sureshchandra and J. Shrinivasavadhani, "Moving from Waterfall to Agile," in *Proceedings of 2008 Agile Development Conference,* Toronto, Canada, 2008, pp. 97–101.

145. J. Sutherland, C. R. Jakobsen, and K. Johnson, "Scrum and CMMI Level 5: The Magic Potion for Code Warriors," in *Proceedings of Agile Conference,* 2007, pp. 272–278.

146. M. Sutton, A. Greene, and P. Amini, *Fuzzing: Brute Force Vulnerability Discovery* (Boston: Addison-Wesley Professional, 2007).

147. K. C. Tai and F. J. Daniels, "Interclass Test Order for Object-Oriented Software," *Journal of Object-Oriented Programming,* vol. 12, no. 4 (1999), pp. 18–25.

148. Richard N. Taylor, N. Medvidovic, and E. M. Dashofy, *Software Architecture: Foundations, Theory, and Practice* (Hoboken, NJ: Wiley, 2009).

149. R. H. Thayer, "Software System Engineering: A Tutorial," *IEEE Computer,* vol. 35, no. 4 (2002), pp. 68–73.

150. Jenifer Tidwell, *Designing Interfaces,* 2nd ed. (Sebastopol, CA: O'Reilly Media, 2011).

151. Y. L. Traon, T. Jeron, J. M. Jezequel and P. Morel, "Efficient Object-Oriented Integration and Regression Testing," *IEEE Transactions on Reliability,* vol. 49, no. 1 (2000), pp. 12–25.

152. Andrew Troelsen, *Pro C# 2008 and the .NET 3.5 Platform,* 4th ed. (Windows.Net), (New York: Apress, 2007).

153. R. Tvedt, P. Costa, and M. Lindvall, "Does the Code Match the Design? A Process for Architecture Evaluation," in *Proceedings of the International Conference on Software Maintenance,* Montreal, Canada, 2002, pp. 393–401.

154. Jennifer Vesperman, *Essential CVS* (Sebastopol, CA: O'Reilly Media, 2006).

155. J. Viega, J. T. Bloch, Y. Kohno, and G. McGraw, "ITS4: A Static Vulnerability Scanner for C and C++ Code," in *Proceedings of the 16th Annual Computer Security Applications Conference,* New Orleans, LA, 2000, pp. 257–267.

156. John Viega and Gary McGraw, *Building Secure Software: How to Avoid Security Problems the Right Way* (Boston: Addison-Wesley, 2002).

157. P. T. Ward, "The Transformation Schema: An Extension of the Data Flow Diagram to Represent Control and Timing," *IEEE Transactions on Software Engineering,* vol. SE-12 (February 1986), pp. 198–210.

158. E. J. Weyuker, "Axiomatizing Software Test Data Adequacy," *IEEE Transactions on Software Engineering,* vol. SE-12, no. 12 (1986), pp. 1128–1138.

159. E. J. Weyuker, "The Evaluation of Program-Based Software Test Data Adequacy Criteria," *CACM,* vol. 31, no. 6 (1986), pp. 668–675.

160. J. Whittaker and H. Thompson, *How to Break Software Security* (Boston: Addison-Wesley, 2003).

161. Karl Wiegers, *Software Requirements 2,* 2nd ed. (Redmond, WA: Microsoft Press, 2003).

162. Laurie Williams and Robert Kessler, *Pair Programming Illuminated* (Boston: Addison-Wesley Professional, 2002).

163. Rebecca Wirfs-Brock and Alan McKean, *Object Design: Roles, Responsibilities, and Collaborations* (Boston: Addison-Wesley, 2001).

164. Ye Wu, Mei-Hwa Chen, and Howard M. Kao, "Regression Testing on Object-Oriented Programs," *Proceedings of the 10th IEEE International Symposium on Software Reliability Engineering,* Boca Raton, FL, November 1–4, 1999.

165. Joseph W. Yoder and Jeffrey Barcalow, "Architectural Patterns for Enabling Application Security," in *Proceedings of the Fourth Conference on Patterns Languages of Programs* (PLoP '97), Monticello, IL, September 1997.

166. D. Zhang, D. Liu, J. Lei, D. Kung, C. Csallner, and W. Wang, "Detecting Vulnerabilities in C Programs Using Trace-Based Testing," in *Proceedings of the IEEE International Conference on Dependable Systems and Networks,* Chicago, IL, 2010, pp. 241–250.

واژه‌نامه انگلیسی به فارسی

A

Abstract ..... مجرد
Acceptance Testing ..... آزمون پذیرش
Accuracy ..... دقت
Action ..... کنش
Activity ..... فعالیت
Actor ..... کنشگر
Adaptive ..... تطبیقی
Aggregate ..... اجتماع
Agile ..... چابک
Agile Unified Methodology ..... متدولوژی چابک یکنواخت
Analysis ..... تحلیل
Anonymous ..... بی‌نام
Application ..... کاربرد
Approach ..... رویکرد
Architecture ..... معماری
Artifact ..... دست‌ساخته
Association ..... انجمن
Asynchronous ..... ناهمگام
Attribute ..... صفت

B

Backup ..... پشتیبان‌گیری
Baseline ..... خط مبنا
Batch Process ..... فرایند دسته‌ای
Benchmarking ..... محک زنی
Best Practice ..... بهترین روش
Blueprint ..... اوزالید
Brainstorming ..... طوفان فکری
Branch Coverage ..... پوشش شاخه
Bridge Pattern ..... الگوی پل
Bug ..... اشکال
Business ..... کسب‌وکار

C

Canvas ..... صفحه‌ی رسم
Characteristic ..... مشخصه
Check-box ..... جعبه‌ی بررسی
Classification ..... دسته‌بندی
Classifier ..... دسته‌بند
Client-Server Architecture ..... معماری مشتری خادم
Coach ..... مربی
Cognitive ..... شناختی
Cohesion ..... چسبندگی
Collaboration ..... همکوشی
Combo Box ..... جعبه‌ی ترکیبی
Command Pattern ..... الگوی فرمان
Commercial off-the-shelf ..... تجاري آماده فروش
Communication Diagram ..... نمودار ارتباط
Compliance ..... پیروی
Component ..... مولفه
Computational ..... رایانشی
Conceptualization ..... مفهوم‌سازی
Concrete ..... عینی
Conform ..... وفق‌داشتن
Consistency ..... سازگاری

Constraint ..... قید
Containment ..... دربرگیری
Controller ..... کنترلگر
Coupling ..... جفت شدگی
Crash ..... فروپاشی

D

Debugging ..... اشکال‌زدایی
Decomposition ..... تجزیه
Defect ..... نقص
Deliverable ..... تحویلی
Dependency ..... وابستگی
Deployment ..... استقرار
Derive ..... استنتاج
Design ..... طراحی
Design Patterns ..... الگوهای طراحی
Development ..... توسعه
Dialog Box ..... جعبه گفتگو
Discipline ..... رشته‌ی تحصیلی
Domain ..... دامنه
Drop-down List ..... فهرست پایین ریز

E

Effectiveness ..... مؤثر بودن
Efficiency ..... بازدهی
Elaboration ..... تشریح
Elicitation ..... استخراج
Embedded System ..... سیستم جاسازی شده
Enhancement ..... بهبود
Estimation ..... تخمین
Ethnography ..... قوم‌نگاری
Evaluator ..... ارزیاب
Event ..... رویداد
Event-Driven System ..... سیستم رویداد رانده

تکاملی ..... Evolutionary
گسترده ..... Expanded
توسعه‌ای ..... Extensional
برنامه‌نویسی افراطی ..... Extreme Programming (XP)

F

نمای خارجی ..... Façade
امکان‌پذیری ..... Feasibility
خصیصه ..... Feature
بازخورد ..... Feedback
سفت‌افزار ..... Firmware
انشعاب ..... Forking
رسمی ..... Formal
قالب‌بندی ..... Formatting
چارچوب ..... Framework
نیازمندی کارکردی ..... Functional Requirement

G

تعمیم ..... Generalization
رهنمون ..... Guideline

H

جدول درهم‌ساز ..... Hash Table

I

شمایل ..... Icon
پیاده‌سازی ..... Implementation
آغاز ..... Inception
افزایش ..... Increment
شاخص ..... Index
ارث‌بری ..... Inheritance
بازرسی ..... Inspection

Instantiation ..... نمونه سازی
Integrated Development Environment (IDE) ..... محیط توسعه‌ی یکپارچه
Interaction ..... تعامل
Interfacing ..... واسطگری
Interrupt ..... وقفه
Intractability ..... سرکش بودن
Isolated ..... منفرد
Iterative ..... تکراری
Iterator Pattern ..... الگوی تکرارکننده

J

Joint Development ..... توسعه‌ی مشترک

L

Layered Pane ..... قاب لایه‌ای
Layout ..... چیدمان
Legacy ..... موروثی
Lexical ..... لغوی
Lifeline ..... زیست‌خط
Link ..... پیوند
Load ..... بارگزاری
Log ..... ثبت
log-on ..... ورود
Look and Feel ..... ظاهر و حس

M

Maintenance ..... نگهداری
Master Plan ..... برنامه‌ی کلان
Mechanism ..... سازوکار
Memento Pattern ..... الگوی یادآور
Menu Bar ..... میله‌ی فهرست
Message Passing ..... تبادل پیغام
Method ..... روش

نقطه عطف ..... Milestone
مأموریت بحرانی ..... Mission-Critical
معماری مدل، نما، کنترلگر ..... Model-View-Controller (MVC) Architecture
پیمانه ..... Module
تعدد ..... Multiplicity
چندوظیفگی ..... Multitasking

N

قاب ناوبری ..... Navigation Pane
نیاز، خواسته ..... Need
نیازمندی غیرکارکردی ..... Non-Functional Requirement
نماد ..... Notation
انگاره ..... Notion
Nتایی ..... N-tier

O

عینی ..... Objective
شیءگرا (OO) ..... Object-Oriented (OO)
عمل ..... Operation
بهینه‌سازی ..... Optimization
برون‌مرزی ..... Overseas
نمای کلی ..... Overview

P

بسته ..... Package
قاب ..... Pane
نگرش ..... Paradigm
عضو (در کتابخانه) ..... Patron
الگو ..... Pattern
نظیر ..... Peer
ادراک ..... Perception
کارایی ..... Performance
ماندگاری ..... Persistence

Phase ..... مرحله
Pipeline ..... خط لوله
Planning ..... برنامه‌ریزی
Platform ..... سکو
Plugin ..... افزونه
PolyMorphism ..... چندریختی
Popup Menu ..... فهرست بازشو
Post-Condition ..... پس‌شرط
Practice ..... روش
Pre-Condition ..... پیش‌شرط
Predictable ..... پیش‌بینی‌پذیر
Presentation ..... نمایش
Principles ..... اصول
Procedure ..... رویه
Process ..... فرایند
Product Backlog ..... پشته محصول
Productionizing ..... محصول‌سازی
Productivity ..... بهره‌وری
Property ..... ویژگی
Prototype ..... پیش‌نمونه
Pseudo-Code ..... شبه کد

Q

Quality Assurance ..... تضمین کیفیت

R

Reactive ..... واکنشی
Recovery ..... ترمیم
Redo ..... انجام‌مجدد
Redundancy ..... افزونگی
Reengineering ..... بازمهندسی
Refactoring ..... بهسازی
Refine ..... پالایش
Regression ..... بازگشت

نسخه ..... Release
قابلیت اطمینان ..... Reliability
مخزن ..... Repository
بازنمایش ..... Representation
نیازمندی ..... Requirement
جمع‌آوری نیازمندی‌ها ..... Requirements Acquisition
تخصیص مسئولیت ..... Responsibility Assignment
مسئولیت-رانده ..... Responsibility-Driven
بازاندوزی ..... Restore
بازگشت سرمایه ..... Return of Investment
بازاستفاده ..... Reuse
مهندسی معکوس ..... Reverse Engineering
بازبینی ..... Review
حل و فصل ریسک ..... Risk Resolution
محکم ..... Robust
روال ..... Routine

S

ایمنی ..... Safety
زمان‌بندی ..... Schedule
قلمرو ..... Scope
قاب طوماری ..... Scroll Pane
امن ..... Secure
امنیت ..... Security
معنا ..... Semantics
جداسازی دغدغه‌ها ..... Separation of Concerns
نمودار توالی ..... Sequence Diagram
راه‌حل جادویی ..... Silver Bullet
لغزنده ..... Slider
نرم افزار ..... Software
تضمین کیفیت نرم‌افزار ..... Software Configuration Management (SCM)
مهندسی نرم افزار ..... Software Engineering
تبیین نیازمندی‌های نرم‌افزار ..... Software Requirements Specification
فضای پاسخ ..... Solution Space
ذی‌نفع ..... Stakeholder

Standalone ..... مستقل
State Diagram ..... نمودار حالت
Stateless ..... بدون حالت
Story Board ..... تخته‌ی داستان
Strategy ..... راهبرد
Structured ..... ساختمند
Style ..... سبک
Swimlane ..... خط‌شنا
Symbol ..... نشانه
Syntax ..... نحو

T

Tabbed Pane ..... قاب زبانه‌دار
Tailored ..... سفارشی
Tame Problem ..... مسئله خوش‌رفتار
Ternary ..... سه‌سه‌یی
Test Case ..... مورد آزمون
Test Script ..... اسکریپت آزمون
Test Stub ..... بدنه‌ی آزمون
Test-Driven Development ..... توسعه‌ی آزمون رانده
Throughput ..... بازدهی
Time to Market ..... زمان تا تحویل محصول به بازار
Toolbar ..... میله‌ی ابزار
Toolsmith ..... کارشناس ابزار
Traceability ..... ردیابی‌پذیری
Tracking ..... ردگیری
Transform ..... تبدیل
Transition ..... انتقال
Trial-end-Error ..... آزمون و خطا

U

Unified Modeling Language (UML) ..... زبان مدل‌سازی یکنواخت
Unified Process (UP) ..... فرایند یکنواخت
Unit Testing ..... آزمون واحد

مورد کاربرد ..... Use-Case
واسط کاربر ..... User Interface
سودمندی ..... Utility

V

اعتبارسنجی ..... Validation
وارسی ..... Verification

W

گذار ..... Walkthrough
مسئله بدرفتار ..... Wicked Problem
کاربرگ ..... Worksheet

واژه‌نامه فارسی به انگلیسی

الف

اجتماع ..... Aggregate
ادراک ..... Perception
ارث‌بری ..... Inheritance
ارزیاب ..... Evaluator
استخراج ..... Elicitation
استقرار ..... Deployment
استنتاج ..... Derive
اسکریپت آزمون ..... Test Script
اشکال ..... Bug
اشکال‌زدایی ..... Debugging
اصول ..... Principles
اعتبارسنجی ..... Validation
افزایش ..... Increment
افزونگی ..... Redundancy
افزونه ..... Plugin
الگو ..... Pattern
الگوهای طراحی ..... Design Patterns
الگوی پل ..... Bridge Pattern
الگوی تکرارکننده ..... Iterator Pattern
الگوی فرمان ..... Command Pattern
الگوی یادآور ..... Memento Pattern
امکان‌پذیری ..... Feasibility

امن ..... Secure
امنیت ..... Security
Nتایی ..... N-tier
انتقال ..... Transition
انجام‌مجدد ..... Redo
انجمن ..... Association
انشعاب ..... Forking
انگاره ..... Notion
اوزالید ..... Blueprint
ایمنی ..... Safety
آزمون پذیرش ..... Acceptance Testing
آزمون و خطا ..... Trial-end-Error
آزمون واحد ..... Unit Testing
آغاز ..... Inception

ب

بارگزاری ..... Load
بازاستفاده ..... Reuse
بازاندوزی ..... Restore
بازبینی ..... Review
بازخورد ..... Feedback
بازدهی ..... Efficiency
بازدهی ..... Throughput
بازرسی ..... Inspection
بازگشت ..... Regression
بازگشت سرمایه ..... Return of Investment
بازمهندسی ..... Reengineering
بازنمایش ..... Representation
بدنه‌ی آزمون ..... Test Stub
بدون حالت ..... Stateless
برنامه‌ی کلان ..... Master Plan
برنامه‌ریزی ..... Planning
برنامه‌نویسی افراطی ..... Extreme Programming (XP)
برون‌مرزی ..... Overseas
بسته ..... Package

بهبود ..... Enhancement
بهترین روش ..... Best Practice
بهره‌وری ..... Productivity
بهسازی ..... Refactoring
بهینه‌سازی ..... Optimization
بی‌نام ..... Anonymous

پ

پالایش ..... Refine
پس‌شرط ..... Post-Condition
پشته محصول ..... Product Backlog
پشتیبان‌گیری ..... Backup
پوشش شاخه ..... Branch Coverage
پیاده سازی ..... Implementation
پیروی ..... Compliance
پیش‌بینی‌پذیر ..... Predictable
پیش‌شرط ..... Pre-Condition
پیش‌نمونه ..... Prototype
پیمانه ..... Module
پیوند ..... Link

ت

تبادل پیغام ..... Message Passing
تبدیل ..... Transform
تبیین نیازمندی‌های نرم‌افزار ..... Software Requirements Specification
تجاری آماده فروش ..... Commercial off-the-shelf
تجزیه ..... Decomposition
تحلیل ..... Analysis
تحویلی ..... Deliverable
تخته‌ی داستان ..... Story Board
تخصیص مسئولیت ..... Responsibility Assignment
تخمین ..... Estimation
ترمیم ..... Recovery
تشریح ..... Elaboration

تضمین کیفیت ..... Quality Assurance
تضمین کیفیت نرم‌افزار ..... Software Configuration Management (SCM)
تطبیقی ..... Adaptive
تعامل ..... Interaction
تعدد ..... Multiplicity
تعمیم ..... Generalization
تکاملی ..... Evolutionary
تکراری ..... Iterative
توسعه ..... Development
توسعه‌ای ..... Extensional
توسعه‌ی آزمون رانده ..... Test-Driven Development
توسعه‌ی مشترک ..... Joint Development

ث

ثبت ..... Log

ج

جداسازی دغدغه‌ها ..... Separation of Concerns
جدول درهم‌ساز ..... Hash Table
جعبه گفتگو ..... Dialog Box
جعبه ی ترکیبی ..... Combo Box
جعبه‌ی بررسی ..... Check-box
جفت شدگی ..... Coupling
جمع‌آوری نیازمندی‌ها ..... Requirements Acquisition

چ

چابک ..... Agile
چارچوب ..... Framework
چسبندگی ..... Cohesion
چندریختی ..... PolyMorphism
چندوظیفگی ..... Multitasking
چیدمان ..... Layout

ح

حل و فصل ریسک ..... Risk Resolution

خ

خصیصه ..... Feature
خط لوله ..... Pipeline
خط مبنا ..... Baseline
خط‌شنا ..... Swimlane

د

دامنه ..... Domain
دربرگیری ..... Containment
دست‌ساخته ..... Artifact
دسته‌بند ..... Classifier
دسته‌بندی ..... Classification
دقت ..... Accuracy

ذ

ذی‌نفع ..... Stakeholder

ر

راهبرد ..... Strategy
راه‌حل جادویی ..... Silver Bullet
رایانشی ..... Computational
ردگیری ..... Tracking
ردیابی‌پذیری ..... Traceability
رسمی ..... Formal
رشته‌ی تحصیلی ..... Discipline
رهنمون ..... Guideline
روال ..... Routine
روش ..... Method
روش ..... Practice

رویداد ..... Event
رویکرد ..... Approach
رویه ..... Procedure

ز

زبان مدل‌سازی یکنواخت ..... Unified Modeling Language (UML)
زمان تا تحویل محصول به بازار ..... Time to Market
زمان‌بندی ..... Schedule
زیست‌خط ..... Lifeline

س

ساختمند ..... Structured
سازگاری ..... Consistency
سازوکار ..... Mechanism
سبک ..... Style
سرکش بودن ..... Intractability
سفارشی ..... Tailored
سفت‌افزار ..... Firmware
سکو ..... Platform
سه‌سه‌یی ..... Ternary
سودمندی ..... Utility
سیستم جاسازی شده ..... Embedded System
سیستم رویداد رانده ..... Event-Driven System

ش

شاخص ..... Index
شبه کد ..... Pseudo-Code
شمایل ..... Icon
شناختی ..... Cognitive
شیءگرا (OO) ..... Object-Oriented (OO)

ص

صفت ..... Attribute
صفحه‌ی رسم ..... Canvas

ط

طراحی ..... Design
طوفان فکری ..... Brainstorming

ظ

ظاهر و حس ..... Look and Feel

ع

عضو (در کتابخانه) ..... Patron
عمل ..... Operation
عینی ..... Concrete
عینی ..... Objective

ف

فرایند ..... Process
فرایند دسته‌ای ..... Batch Process
فرایند یکنواخت ..... Unified Process (UP)
فروپاشی ..... Crash
فضای پاسخ ..... Solution Space
فعالیت ..... Activity
فهرست بازشو ..... Popup Menu
فهرست پایین ریز ..... Drop-down List

ق

قاب ..... Pane
قاب زبانه‌دار ..... Tabbed Pane
قاب طوماری ..... Scroll Pane

قاب لایه‌ای ..... Layered Pane
قاب ناوبری ..... Navigation Pane
قابلیت اطمینان ..... Reliability
قالب‌بندی ..... Formatting
قلمرو ..... Scope
قوم‌نگاری ..... Ethnography
قید ..... Constraint

ک

کارایی ..... Performance
کاربرد ..... Application
کاربرگ ..... Worksheet
کارشناس ابزار ..... Toolsmith
کسب‌وکار ..... Business
کنترلگر ..... Controller
کنش ..... Action
کنشگر ..... Actor

گ

گذار ..... Walkthrough
گسترده ..... Expanded

ل

لغزنده ..... Slider
لغوی ..... Lexical

م

ماندگاری ..... Persistence
مأموریت بحرانی ..... Mission-Critical
متدولوژی چابک یکنواخت ..... Agile Unified Methodology
مجرد ..... Abstract
محصول‌سازی ..... Productionizing

محک زنی ..... Benchmarking
محکم ..... Robust
محیط توسعه‌ی یکپارچه ..... Integrated Development Environment (IDE)
مخزن ..... Repository
مربی ..... Coach
مرحله ..... Phase
مسئله بدرفتار ..... Wicked Problem
مسئله خوش‌رفتار ..... Tame Problem
مستقل ..... Standalone
مسئولیت-رانده ..... Responsibility-Driven
مشخصه ..... Characteristic
معماری ..... Architecture
معماری مدل، نما، کنترلگر ..... Model-View-Controller (MVC) Architecture
معماری مشتری خادم ..... Client-Server Architecture
معنا ..... Semantics
مفهوم‌سازی ..... Conceptualization
منفرد ..... Isolated
مهندسی معکوس ..... Reverse Engineering
مهندسی نرم افزار ..... Software Engineering
مورد آزمون ..... Test Case
مورد کاربرد ..... Use-Case
موروثی ..... Legacy
مولفه ..... Component
مؤثر بودن ..... Effectiveness
میله‌ی ابزار ..... Toolbar
میله‌ی فهرست ..... Menu Bar

ن

ناهمگام ..... Asynchronous
نحو ..... Syntax
نرم افزار ..... Software
نسخه ..... Release
نشانه ..... Symbol
نظیر ..... Peer
نقص ..... Defect

نقطه عطف ..... Milestone
نگرش ..... Paradigm
نگهداری ..... Maintenance
نماد ..... Notation
نمای خارجی ..... Façade
نمای کلی ..... Overview
نمایش ..... Presentation
نمودار ارتباط ..... Communication Diagram
نمودار توالی ..... Sequence Diagram
نمودار حالت ..... State Diagram
نمونه سازی ..... Instantiation
نیاز، خواسته ..... Need
نیازمندی ..... Requirement
نیازمندی غیرکارکردی ..... Non-Functional Requirement
نیازمندی کارکردی ..... Functional Requirement

و

وابستگی ..... Dependency
وارسی ..... Verification
واسط کاربر ..... User Interface
واسطگری ..... Interfacing
واکنشی ..... Reactive
ورود ..... log-on
وفق‌داشتن ..... Conform
وقفه ..... Interrupt
ویژگی ..... Property

ه

همکوشی ..... Collaboration