



دانشگاه صنعتی شریف

دانشکده‌ی مهندسی کامپیوتر

پایان‌نامه‌ی کارشناسی
گرایش نرم‌افزار

عنوان

مستقل‌سازی سناریو
از آزمون مبتنی بر رفتار

نگارش

سید مهران خلدی، محمد حسین سخاوت

استاد راهنما

دکتر سید حسن میریان

شهریور ۱۳۹۵

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

قدردانی

از استاد بزرگوار دکتر میریان که ما را در انجام این پروژه راهنمایی کردند، تشکر و قدردانی می‌کنیم.

همچنین از جناب آقای مهندس مهدیه به خاطر مشورت‌ها و پیگیری‌های عالمانه و دلسوزانه‌ی ایشان قدردانی می‌نماییم.

مستقل سازی سناریو از آزمون مبتنی بر رفتار

چکیده

توسعه‌ی مبتنی بر رفتار^۱ یکی از تکنیک^۲های جدید برای توسعه‌ی چابک نرم‌افزار^۳ است که استفاده از آن در صنعت نرم‌افزار رواج دارد. در این پایان‌نامه ابتدا به بررسی دقیق توسعه‌ی مبتنی بر رفتار و مقایسه‌ی آن با تکنیک‌های مشابه می‌پردازیم.

در ادامه یک چارچوب^۴ آزمون^۵ جهت استفاده در تکنیک توسعه‌ی مبتنی بر رفتار ارائه می‌کنیم، که با اتکا بر مستقل‌سازی^۶ ساختار آزمون، توسعه‌دهنده را در افزایش کیفیت آزمون‌های تألیفی و کاهش هزینه‌ی نگهداری آن‌ها یاری می‌کند.

نهایتاً و به عنوان نمونه، با استفاده از این چارچوب آزمون برای یک سامانه‌ی کوچک نرم‌افزاری، آزمون‌هایی تألیف می‌کنیم.

کلیدواژه‌ها: توسعه‌ی مبتنی بر رفتار، مستقل‌سازی، کیفیت آزمون نرم‌افزار، چارچوب آزمون.

^۱Behavior Driven Development

^۲Technique

^۳Agile Software Development

^۴Framework

^۵Test

^۶Decoupling

سرخ‌ها

۱	۱. مقدمه	۱
۱	۱.۱. اهمیت موضوع	۱
۲	۲.۱. تعریف مسأله	۲
۲	۳.۱. اهداف پروژه	۲
۳	۴.۱. ساختار پایان‌نامه	۳
۴	۲. پیش‌زمینه	۴
۴	۱.۲. تاریخچه	۴
۶	۲.۲. ادبیات	۶
۶	۱.۲.۲. آزمون، مورد آزمون ^۷ و مجموعه‌ی آزمون ^۸	۶
۷	۲.۲.۲. دسته‌بندی مدل V	۷
۷	۳.۲.۲. هرم آزمون ^۹	۷
۹	۴.۲.۲. متدولوژی ^{۱۰} ها و تکنیک‌های توسعه‌ی نرم‌افزار	۹
۱۰	۵.۲.۲. توسعه‌ی اول-آزمون ^{۱۱}	۱۰
۱۰	۶.۲.۲. توسعه‌ی مبتنی بر آزمون ^{۱۲}	۱۰
۱۲	۷.۲.۲. توسعه‌ی آزمون مبتنی بر پذیرش ^{۱۳}	۱۲
۱۴	۳.۲. تکنیک توسعه‌ی مبتنی بر رفتار	۱۴
۱۴	۱.۳.۲. مشخصات توسعه‌ی مبتنی بر رفتار	۱۴

^۷Test Case

^۸Test Suite

^۹Test Pyramid

^{۱۰}Methodology

^{۱۱}Test First Development

^{۱۲}Test Driven Development

^{۱۳}Acceptance Test Driven Development

۲۱	۲.۳.۲. قالب ^{۱۴} توصیف داستان‌های کاربر ^{۱۵} و سناریو ^{۱۶} ها
۲۱	۳. طراحی و توسعه‌ی چارچوب
۲۱	۱.۳. معماری چارچوب
۲۴	۲.۳. فواید
۲۴	۱.۲.۳. کاهش حجم آزمون‌ها
۲۴	۲.۲.۳. کاهش هزینه‌ی نگهداری
۲۵	۳.۲.۳. افزایش احتمال یافتن خطا
۲۵	۳.۳. استفاده در سطح سیستمی
۲۵	۴.۳. استفاده خارج از محیط آزمون
۲۶	۵.۳. پیاده‌سازی بر مبنای چارچوب جنگو ^{۱۷}
۲۶	۱.۵.۳. نحوه‌ی استفاده
۲۷	۲.۵.۳. اجزاء پیاده‌سازی
۲۹	۳.۵.۳. کد منبع
۳۰	۴. مطالعه‌ی موردی
۳۰	۱.۴. معرفی سامانه ثبت نام
۳۰	۱.۱.۴. موجودیت‌های سامانه
۳۲	۲.۱.۴. رابط کاربری ^{۱۸} سامانه
۳۷	۲.۴. تألیف سناریو برای سامانه
۴۰	۵. جمع‌بندی و کارهای آتی
۴۰	۱.۵. جمع‌بندی
۴۰	۲.۵. کارهای آتی

^{۱۴}Template
^{۱۵}User Stories
^{۱۶}Scenario
^{۱۷}Django
^{۱۸}User Interface

۴۴	آ. پیاده‌سازی بخش جمع‌آوری سناریوها در اینسنیتی ^{۱۹}
۴۶	ب. پیاده‌سازی دکوراتور ^{۲۰} های مربوط به تشخیص و اجرای سناریوها در اینسنیتی
۴۹	پ. متن برنامه‌ی منطق تجاری ^{۲۱} سامانه‌ی آموزش
۵۲	ت. متن برنامه‌ی تعامل گر سامانه‌ی ثبت‌نام
۵۵	واژه‌نامه‌ی فارسی به انگلیسی
۵۶	واژه‌نامه‌ی انگلیسی به فارسی

^{۱۹}Insanity

^{۲۰}Decorator

^{۲۱}Business Logic

فهرست شکل‌ها

۷	۱.۲. نمودار سطوحی آزمون در مدل چرخه عمر V- شکل ^{۲۲}
۸	۲.۲. نمودار سطوح آزمون در هرم آزمون
۹	۳.۲. رابطه‌ی میان آزمون واحد ^{۲۳} و واحدهای برنامه
۱۱	۴.۲. نمودار فعالیت مراحل توسعه‌ی اول-آزمون
۱۲	۵.۲. نمودار حالت وضعیت‌های مجموعه‌ی آزمون و نحوه‌ی گذار بین آنها
۱۳	۶.۲. نمودار فعالیت مراحل توسعه‌ی آزمون مبتنی بر پذیرش
۱۶	۷.۲. مدل مفهومی در تحلیل توسعه‌ی مبتنی بر رفتار
۱۷	۸.۲. فرآیند توسعه‌ی ویژگی ^{۲۴} ها در توسعه‌ی مبتنی بر رفتار
۲۰	۹.۲. یک داستان کاربر ^{۲۵} به همراه شرط پذیرش ^{۲۶}
۲۲	۱.۳. مقایسه معماری پیشنهادی با معماری متداول آزمون
۲۶	۲.۳. افزودن چارچوب به جنگو
۲۷	۳.۳. ورود تنظیمات سنتری ^{۲۷}
۲۸	۴.۳. نمایش جزییات شکست یک سناریو در سنتری
۲۸	۵.۳. گزارش پوشش سناریوهای تعریف شده
۲۹	۶.۳. نمودار توالی عملکرد decorator
۳۱	۱.۴. نمودار موجودیت‌های سامانه‌ی ثبت نام آموزش
۳۳	۲.۴. صفحه‌ی ورود سامانه‌ی ثبت نام
۳۳	۳.۴. صفحه‌ی کاربری کارمند در سامانه‌ی ثبت نام

^{۲۲}V-Shaped Software Lifecycle Model

^{۲۳}Unit Test

^{۲۴}Feature

^{۲۵}User Story

^{۲۶}Acceptance Criteria

^{۲۷}Sentry

۴.۴	صفحه‌ی ویرایش مشخصات ارائه توسط کارمند	۳۴
۵.۴	تغییر ظرفیت درس توسط کارمند	۳۵
۶.۴	دسترسی مدیر سامانه به صفحه‌های کاربرها	۳۵
۷.۴	صفحه‌ی کاربری استاد در سامانه‌ی ثبت نام	۳۶
۸.۴	صفحه‌ی کاربری دانشجو در سامانه‌ی ثبت نام	۳۶
۹.۴	صفحه‌ی ثبت‌نام دانشجو در درس‌های ارائه‌شده توسط سامانه	۳۷
۱۰.۴	خطای پر بودن ظرفیت هنگام ثبت‌نام دانشجو در سامانه	۳۸
۱۱.۴	کد معادل سناریو	۳۹

فصل ۱

مقدمه

۱.۱. اهمیت موضوع

در عصر اینترنت، سرعت تحول نرم افزار موضوعی بسیار مهم است. تا اواخر قرن بیستم، هر پروژه‌ی نرم افزاری برای چندین سال بدون تغییرات عمده استفاده می شد و اجرای هر فاز از پروژه، چند ماه به طول می انجامید. اما امروزه، روش های توسعه‌ی چابک نرم افزار در بسیاری از پروژه های نرم افزاری به کار گرفته می شوند؛ مدت زمان اجرای کل پروژه به چند ماه و مدت زمان اجرای هر فاز، به چند هفته یا حتی چند روز کاهش پیدا کرده و مقاومت پروژه های نرم افزاری در مقابل تغییرات بسیار کاهش یافته است [۱].

با این نرخ بالای تغییرات، مستندات پروژه خیلی سریع منسوخ می شوند؛ به روز نگه داشتن توصیف نیازمندی ها و اجرای آزمون به شیوه های سنتی، بسیار پرهزینه محسوب شده و عملاً بی فایده است. در روش های توسعه‌ی چابک نرم افزار، بیشتر از تولید درست محصول^۱، به تولید محصول درست^۲ توجه می شود و لازم است هزینه‌ی مستندسازی و آزمون را کاهش داده و بر تولید محصول درست تمرکز شود [۲].

در تکنیک توسعه‌ی مبتنی بر رفتار، فرآیند توصیف نیازمندی ها به صورت تکراری^۳ انجام می شود و با توجه به زبان مشترک توصیف نرم افزار بین مشتری و تیم توسعه و همچنین قابلیت تبدیل توصیف به آزمون، از توصیف به عنوان مواد اولیه‌ی آزمون و مستندات پروژه نیز بهره گرفته می شود. این امر باعث کاهش هزینه‌ی نگهداری، افزایش سرعت توسعه آزمون و انطباق پذیری با تغییرات با وجود حفظ کیفیت نرم افزار می شود. همچنین وجود چارچوبی کارا برای پیاده سازی این تکنیک، بر سرعت توسعه و به کارگیری صحیح اصول این تکنیک تأثیر مستقیم دارد.

^۱Building the product right

^۲Building the right product

^۳Iterative

۲.۱. تعریف مسأله

توسعه‌ی مبتنی بر رفتار یکی از تکنیک‌های جدید برای توسعه‌ی چابک نرم‌افزار است که استفاده از آن در صنعت نرم‌افزار رواج دارد [۲]. این تکنیک بر مراحل مختلف فرآیند توسعه‌ی نرم‌افزار^۴ از جمله برنامه‌ریزی، تحلیل، طراحی، پیاده‌سازی و آزمون تاثیرگذار است (ر.ک. به ۱.۳.۲). از جمله مسأله‌هایی که در مرحله‌ی آزمون از فرآیند توسعه‌ی مبتنی بر رفتار وجود دارد، این است که با توجه به اینکه آزمون پذیرش^۵ در این روش، مستقیماً از روی توصیف بدست می‌آید، لازم است هنگام توصیف، جزییاتی را تعیین نماییم که در مرحله‌ی توصیف کاربردی ندارند، بلکه در مرحله‌ی تولید آزمون پذیرش از روی توصیف به آن‌ها نیاز داریم. این در حالی است که توصیه می‌شود جزییات غیر ضروری در توصیف ذکر نشوند [۲].

برای مثال، در یک سامانه‌ی انتخاب واحد، ممکن است برای توصیف قابلیت ثبت‌نام کاربر در یک ارائه از یک درس، صرف وجود یک دانشجو برای توصیف عمل انتخاب واحد کافی باشد؛ اما برای تولید آزمون پذیرش از روی توصیف این قابلیت، لازم است جزییاتی غیر ضروری مانند «نام» و «نام خانوادگی» از دانشجو در توصیف این قابلیت ذکر شوند. در مستندات یکی از چارچوب‌های فعلی توسعه‌ی مبتنی بر رفتار [۳] نیز به این مسأله اشاره شده است.

۳.۱. اهداف پروژه

ایده‌ی ابتدایی این پروژه از همکاری مؤلفان در یک پروژه‌ی صنعتی شکل گرفت. به عنوان توسعه‌دهندگان سامانه‌ی^۶ مذکور، همیشه حس می‌کردیم که بخش قابل توجهی از کدهایی که برای آزمون نرم‌افزار می‌نویسیم شباهت‌های زیادی به یک‌دیگر دارند. از سوی دیگر، به دلیل تغییر نیازمندی‌های سامانه، همیشه لازم بود تا آزمون‌ها به‌روزرسانی شوند و این تغییرات معمولاً در جای‌جای پروژه منتشر می‌شد. تمام این مشکلات باعث می‌شد تا کیفیت آزمون‌ها در طول زمان کاهش یافته و اثر مثبت آن‌ها در فرآیند توسعه‌ی نرم‌افزار کاهش یابد. بنابراین سعی کردیم الگوهای مشترکی بین آزمون‌ها پیدا کنیم و با فاکتوربندی مجدد^۷ مکرر آن‌ها به ساختاری رسیدیم که بسیاری از مشکلات فوق را کم‌رنگ کرده بود. در این ساختار، برای آزمون هر قسمت، فقط اطلاعاتی که مرتبط با عملکرد همان قسمت بودند حفظ می‌شد و جزییات مرتبط با نیازمندی‌های آزمون، به بخش‌های دیگر منتقل شده بود. تلاش کردیم تا ساختار نهایی را تعمیم دهیم تا در پروژه‌های دیگر نیز قابل استفاده باشد. ساختار حاصل، الهام‌بخش چارچوب آزمونی شد که در این پروژه ارائه می‌کنیم.

^۴ Software Development Process

^۵ Acceptance Test

^۶ System

^۷ Refactoring

هدف اصلی از ارائه‌ی چارچوب آزمون پیشنهادی، کاهش هزینه‌ی نگهداری آزمون‌ها و افزایش اثرگذاری آن‌ها در کیفیت نرم‌افزار است. در بخش ۲.۳ فواید مطرح شده پس از استفاده از چارچوب مورد نظر را با دقت بیشتری بررسی خواهیم کرد.

۴.۱. ساختار پایان نامه

این پایان نامه حاوی پنج فصل است.

در فصل نخست مقدمه‌ای از موضوع پایان نامه و اهمیت آن ارائه شده است. در فصل دوم پیش‌زمینه‌ای از کارهای قبلی که در این زمینه صورت گرفته آمده، و در ادامه توسعه‌ی مبتنی بر رفتار به تفصیل شرح داده شده است. فصل سوم چارچوب آزمون پیشنهادی را از جنبه‌ی معماری و پیاده‌سازی بررسی می‌کند. فصل چهارم به ارائه‌ی گزارشی از یک نمونه‌ی استفاده از چارچوب پیاده‌سازی شده در فصل سوم می‌پردازد. و نهایتاً در فصل پنجم جمع‌بندی و کارهای آتی که انجام آن‌ها در آینده امکان‌پذیر خواهد بود مطرح می‌شود.

فصل ۲

پیش زمینه

۱.۲. تاریخچه

روند تکامل رویکرد رایج در آزمون نرم/فزار^۱، از سال ۱۹۵۷ تا سال ۲۰۰۰ به پنج دوره‌ی رفع/شکال-محور^۲، اثبات-محور^۳، تخریب-محور^۴، ارزیابی-محور^۵ و پیشگیری-محور^۶ تقسیم‌بندی می‌شود [۴]. برخی فعالیت‌های بارز این دوره‌ها به شرح زیر است:

۱. تا سال ۱۹۵۷، برنامه نویس‌ها پس از نوشتن برنامه، آن را اجرا نموده تا از کارکرد صحیح آن اطمینان یابند و اگر باگ^۷ در اجرا پیدا می‌شد، آن را پیدا کرده و رفع می‌نمودند. این فرآیند ادامه پیدا می‌کرد تا زمانی که احساس کنند باگی باقی نمانده است.

۲. در سال ۱۹۵۷، بیکر^۸ برای اولین بار آزمون نرم‌افزار را، به‌عنوان روشی برای «اثبات عملکرد صحیح» برنامه، از باگ‌زادی^۹ تمیز داد [۵].

۳. دیستر^{۱۰} در سال ۱۹۶۹ متذکر شد که کاربرد آزمون، «یافتن باگ‌ها» است، نه اثبات عملکرد صحیح برنامه [۶]. در سال ۱۹۸۳، راهنمایی برای اعتبارسنجی^{۱۱}، واریسی^{۱۲} و آزمون نرم‌افزار منتشر شد [۷]

^۱Software Test

^۲Debugging-oriented

^۳Demonstration-oriented

^۴Destruction-oriented

^۵Evaluation-oriented

^۶Prevention-oriented

^۷Bug

^۸Charles L. Baker

^۹Debug

^{۱۰}Edsger Dijkstra

^{۱۱}Validation

^{۱۲}Verification

که «تشخیص باگ‌های تحلیل و طراحی» را، علاوه بر تشخیص باگ‌های پیاده‌سازی، با آزمون نرم‌افزار میسر می‌ساخت [۸].

۴. در سال ۱۹۸۷، معیار/تکاپذیری^{۱۳} به‌عنوان عاملی کلیدی در «اندازه‌گیری کیفیت نرم‌افزار^{۱۴}» معرفی شد [۹].

۵. در سال ۱۹۹۰، آزمون نرم‌افزار به‌عنوان یکی از موثرترین عوامل «پیشگیری از باگ» معرفی شد [۱۰].

پس از رویکردهای مذکور در اواخر دهه ۱۹۹۰، با ظهور متدولوژی‌های جدید مانند اکس.پی.^{۱۵}، که در دسته‌بندی متدولوژی‌های توسعه‌ی چابک نرم‌افزار قرار می‌گیرد، رویکردهای جدیدی در آزمون نرم‌افزار ایجاد شد.

در متدولوژی‌های سنتی با مدل آبشاری^{۱۶}، آزمون یکی از فازهای فرآیند توسعه‌ی نرم‌افزار بود که فقط یک بار انجام می‌شد و پیش‌نیاز آن، اتمام فازهای تحلیل^{۱۷}، طراحی^{۱۸} و پیاده‌سازی^{۱۹} بود. در متدولوژی‌های آبشاری، هر کدام از فازهای تحلیل، طراحی، پیاده‌سازی و آزمون به‌صورت مجزا اجرا می‌شد [۱۱] و آزمون نقش موثری در فازهای تحلیل، طراحی و پیاده‌سازی نداشت.

اما در متدولوژی‌های چابک، مراحل تحلیل، طراحی، پیاده‌سازی و آزمون به‌صورت تکراری اجرا می‌شود. در هر تکرار^{۲۰}، نیازمندی کامل نرم‌افزار مشخص نیست و در تکرار بعد، ممکن است تغییر کند. نقش آزمون در «پاسخگویی به تغییرات»، که یکی از چهار ارزش بیانیه‌ی توسعه‌ی چابک نرم‌افزار^{۲۱} است [۱۲]، بسیار کلیدی است. در آزمون چابک، رویکرد «دستیاری در کیفیت^{۲۲}» بر رویکرد «اطمینان از کیفیت^{۲۳}» غلبه می‌کند [۱۳].

بک^{۲۴} در سال ۲۰۰۲، با معرفی تکنیک توسعه‌ی مبتنی بر آزمون، آزمون را به‌عنوان «محرک توسعه‌ی نرم‌افزار» معرفی می‌کند و نوشتن آزمون را پیش‌نیاز پیاده‌سازی می‌داند. وی «بهبود طراحی نرم‌افزار» را از نتایج بکارگیری این تکنیک می‌داند [۱۴]. همچنین با بهره‌گیری از این نگرش در متدولوژی توسعه‌ی چابک مبتنی

^{۱۳}Reliability

^{۱۴}Software Quality

^{۱۵}Extreme Programming

^{۱۶}Waterfall Model

^{۱۷}Analysis

^{۱۸}Design

^{۱۹}Implementation

^{۲۰}Iteration

^{۲۱}Manifesto for Agile Software Development

^{۲۲}Quality Assistance

^{۲۳}Quality Assurance

^{۲۴}Kent Beck

بر مبنای^{۲۵}، از آزمون نرم‌افزار به‌عنوان «توصیف^{۲۶} نرم‌افزار» بهره گرفته می‌شود [۱۵].

۲.۲. ادبیات

اصطلاح‌های مرتبط با توسعه‌ی نرم‌افزار، ابهام‌ها و برداشت‌های گوناگون دارند برای مثال در [۱۶]، هر پنج اصطلاح «توسعه‌ی آزمون مبتنی بر پذیرش»، «توسعه‌ی مبتنی بر رفتار»، «توصیف با مثال^{۲۷}»، «آزمون پذیرش چابک^{۲۸}» و «تست داستان کاربر^{۲۹}»، هم‌معنی تلقی شده‌اند. برای پیشگیری از ابهام و شفاف‌سازی اصطلاح‌های استفاده‌شده در این پایان‌نامه، در این قسمت توضیحی اجمالی و بدون ارزیابی جنبه‌های مختلف، برای هر اصطلاح ارائه شده است.

۱.۲.۲. آزمون، مورد آزمون و مجموعه‌ی آزمون

یک مورد آزمون، از ورودی و خروجی مورد انتظار قسمتی از برنامه برای آن ورودی تشکیل می‌شود. هر عملکرد مورد آزمون قسمت مشخصی از سامانه را که «اس.یو.تی.^{۳۰}» نامیده می‌شود، مورد آزمون قرار می‌دهد. برای معتبر بودن یک مورد آزمون، ممکن است لازم باشد ورودی اس.یو.تی. دارای پیش-شرط^{۳۱}هایی و خروجی دارای پس-شرط^{۳۲}هایی نیز باشد. از واژه‌ی «آزمون» نیز می‌توان به جای واژه‌ی «مورد آزمون» استفاده کرد [۱۷].

وضعیت اجرای یک مورد آزمون دو حالت دارد؛ سبز^{۳۳} (موفقیت‌آمیز) به معنی سازگاری خروجی برنامه با خروجی مورد انتظار و قرمز^{۳۴} (شکست) به معنی عدم سازگاری خروجی برنامه با خروجی مورد انتظار می‌باشد [۱۸].

یک «مجموعه‌ی آزمون» شامل مجموعه‌ای از آزمون‌ها می‌باشد که وضعیت اجرای آن مجموعه‌ی آزمون، در صورتی سبز است که تک‌تک مورد آزمون‌های آن در وضعیت سبز باشند [۱۷].

^{۲۵}Agile Model Driven Development

^{۲۶}Specification

^{۲۷}Specification by Example

^{۲۸}Agile Acceptance Testing

^{۲۹}User Story Testing

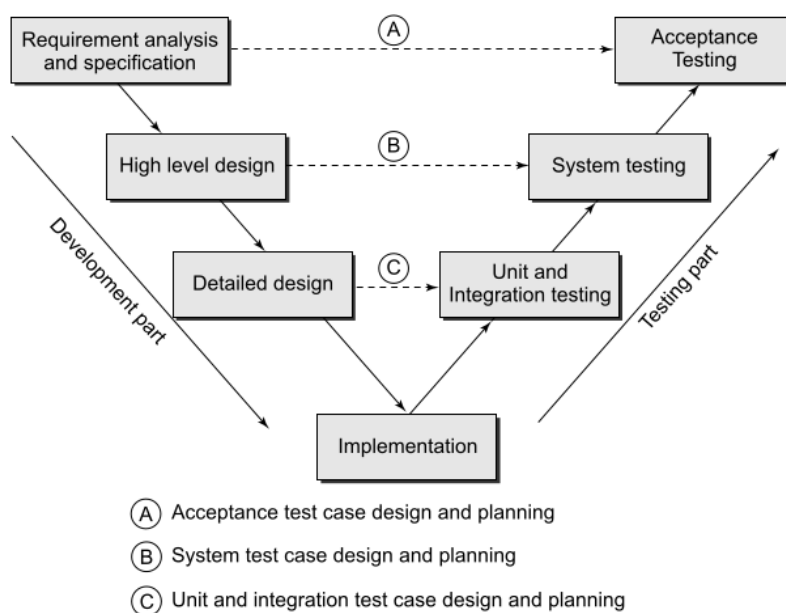
^{۳۰}System Under Test

^{۳۱}Pre-condition

^{۳۲}Post-condition

^{۳۳}Success

^{۳۴}Fail



شکل ۱.۲: نمودار سطوحی آزمون در مدل چرخه عمر V- شکل [۱۷]

۲.۲.۲. دسته‌بندی مدل V

یکی از دسته‌بندی‌های مشهور برای سطوحی آزمون، دسته‌بندی مدل V می‌باشد. این دسته‌بندی، پیش از پیدایش متدولوژی‌های چابک، در متدولوژی «مدل چرخه عمر V- شکل»، که با کمی تغییر در متدولوژی مدل آبشاری و تمرکز بیشتر بر آزمون بوجود آمده، تعریف شده است [۱۷].

وجه تمایز سطوح در این دسته‌بندی، مراحل تولید نرم‌افزار و تقدم زمانی فازهای تحلیل، طراحی و پیاده‌سازی می‌باشد. همانطور که در شکل ۱.۲ نشان داده شده، پس از هر فاز، آزمونی برای ساخته^{۳۵} آن فاز نوشته می‌شود [۱۷].

در این دسته‌بندی چهار سطح «آزمون پذیرش»، «آزمون سامانه^{۳۶}»، «آزمون یکپارچه‌سازی^{۳۷}» و «آزمون واحد» به ترتیب برای مراحل تحلیل، طراحی سطح بالا، طراحی سطح پایین و پیاده‌سازی معرفی شده‌اند.

۳.۲.۲. هرم آزمون

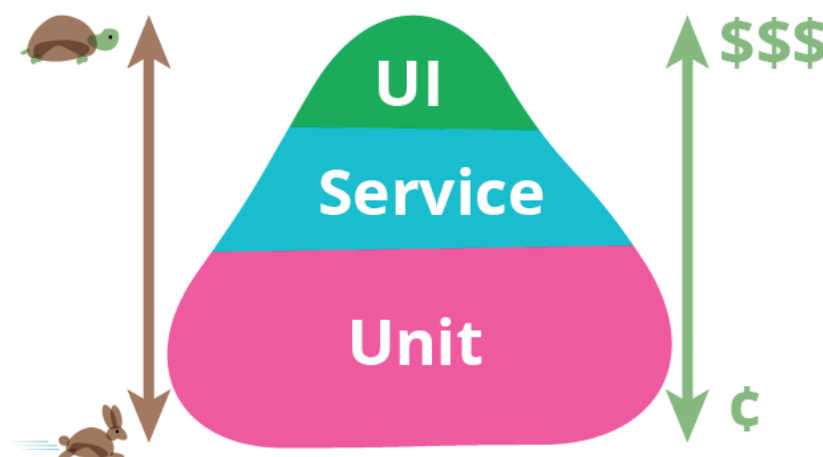
هرم آزمون اصطلاحی است که اولین بار توسط کوهن^{۳۸} برای دسته‌بندی سطوح مختلف آزمون در متدولوژی‌های چابک معرفی شد. وجه تمایز سطوح در این دسته‌بندی، هزینه، زمان اجرا و تعداد آزمون‌های مورد نیاز می‌باشد [۱۹].

^{۳۵}Artifact

^{۳۶}System Test

^{۳۷}Integration Test

^{۳۸}Mike Cohn



شکل ۲.۲: نمودار سطوح آزمون در هرم آزمون [۱۹]

همانطور که در نمودار ۲.۲ نشان داده شده است، در این دسته‌بندی هرچه سطح آزمون بالاتر باشد، هزینه و زمان اجرای آزمون در آن سطح بیشتر، و تعداد آزمون‌های مورد نیاز آن سطح، کمتر می‌شود. با این معیار، سه سطح زیر در این دسته‌بندی تعریف شده است [۱۹]:

۱. **سطح واحد:** پایین‌ترین سطح هرم آزمون، آزمون واحد می‌باشد. هدف اصلی آزمون واحد، این است که واحد کوچکی از نرم‌افزار را در نظر گرفته، آن را از سایر واحدها مجزا ساخته (مانند شکل ۳.۲) و بررسی نماییم که این واحد وظیفه‌ی خود را مطابق انتظار انجام دهد. هر آزمون واحد، به طور مستقل، و قبل از یکپارچه‌سازی^{۳۹} با سایر واحدهای ماثول^{۴۰}، اجرا می‌شود [۲۰].

آزمون واحد تا حد ممکن در سطوح پایین معماری نرم‌افزار مورد استفاده قرار می‌گیرد و به قسمت کوچکی از متن برنامه متمرکز است، عموماً توسط خود برنامه‌نویس‌ها به تعداد زیاد نوشته می‌شود و باید بسیار سریع اجرا شود [۲۱].

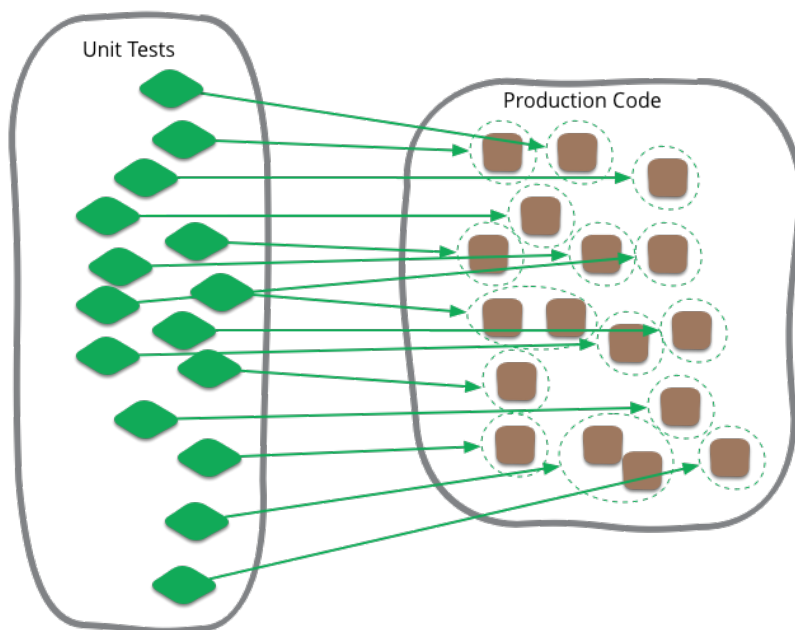
این‌که منظور از یک «واحد» چه باشد، به نیازمندی‌ها و شرایط پروژه بستگی دارد. معمولاً در برنامه‌نویسی شیء‌گرا، کلاس^{۴۱} به عنوان واحد انتخاب می‌شود [۲۱].

۲. **سطح سرویس:** هر نرم‌افزار، از چند سرویس تشکیل شده است که پس از دریافت ورودی از لایه‌ی رابط کاربری، عملیات مورد انتظار سامانه را انجام داده و نتیجه را در اختیار رابط کاربری قرار می‌دهند. منطق برنامه توسط سرویس‌ها پیاده‌سازی می‌شود. این سطح آزمون، عملکرد صحیح سرویس‌های سامانه را می‌آزماید [۲۲].

^{۳۹}Integration

^{۴۰}Module

^{۴۱}Class



شکل ۳.۲: رابطه‌ی میان آزمون واحد و واحدهای برنامه [۲۱]

آزمون سرویس در لایه‌ی میانی هرم آزمون قرار می‌گیرد. آزمون در این سطح، هزینه‌های آزمون سطح رابط کاربری را ندارد، اما تیم را از یکپارچه‌سازی صحیح واحدها در کنار هم مطمئن می‌سازد [۲۲]. ریزدانگی آزمون‌ها در این سطح تقریباً معادل سطح آزمون سامانه از دسته‌بندی مدل V می‌باشد.

۳. سطح رابط کاربری: بالاترین سطح هرم آزمون، سطح رابط کاربری است. در این سطح، مطابقت عملکرد نهایی سامانه با نیازمندی‌هایی که مشتری مشخص می‌نماید، در محصول نهایی بررسی می‌شود. این سطح از آزمون در مقابل تغییرات بسیار شکننده است و هزینه‌ی زیادی صرف نوشتن و اجرای آن می‌شود [۲۲]. ریزدانگی آزمون‌ها در این سطح تقریباً معادل سطوح آزمون سامانه و آزمون پذیرش از دسته‌بندی مدل V می‌باشد.

۴.۲.۲. متدولوژی‌ها و تکنیک‌های توسعه‌ی نرم‌افزار

متدولوژی توسعه‌ی نرم‌افزار^{۴۲} چارچوبی را برای اعمال تجربه‌ی^{۴۳} های مهندسی نرم‌افزار با هدف «فراهم نمودن ابزارهای لازم برای توسعه‌ی سامانه‌های نرم‌افزار-متمرکز^{۴۴}» فراهم می‌آورد. متدولوژی شامل دو عنصر اصلی زیر است [۲۳]:

^{۴۲}Software Development Methodology

^{۴۳}Practice

^{۴۴}Software-intensive

۱. مجموعه‌ای از عرف^{۴۵}ها که شامل زبان مدل‌سازی^{۴۶} (صرفی^{۴۷} و معنایی^{۴۸}) است.
۲. یک فرآیند، که ترتیب فعالیت^{۴۹}ها را مشخص می‌سازد، ساخته‌هایی که با زبان مدل‌سازی توسعه می‌یابند را شرح می‌دهد، وظیفه‌مندی افراد توسعه‌دهنده و تیم را روشن ساخته و شرایطی برای پایش و اندازه‌گیری محصول‌ها و فعالیت‌ها فراهم می‌آورد.
- هر متدولوژی می‌تواند از چند تکنیک، که تجربه هم نامیده می‌شود، در فرآیندها و عرف‌های خود بهره گیرد. برای مثال، متدولوژی اکس.پی.، از تکنیک‌هایی مانند توسعه‌ی مبتنی بر آزمون، برنامه‌نویسی دوتفره^{۵۰} و یکپارچه‌سازی مداوم^{۵۱} بهره می‌گیرد [۱].
- هر تکنیک نیز جنبه‌ای از فرآیندها و/یا عرف‌ها را تعیین می‌نماید. تکنیک‌ها مختص یک متدولوژی خاص نیستند و ممکن است توسط چندین متدولوژی استفاده شوند.

۵.۲.۲. توسعه‌ی اول-آزمون

توسعه‌ی اول-آزمون یک تکنیک است که ترتیب فعالیت‌های توسعه‌ی سامانه توسط توسعه‌دهنده‌ی فنی را مشخص می‌سازد.

همانطور که در نمودار فعالیت شکل ۴.۲ نشان داده شده، برای اعمال هر تغییر در متن برنامه، ابتدا آزمون نوشته می‌شود، تنها به مقداری که آزمون در وضعیت قرمز قرار گیرد. سپس ترجیحا کل مجموعه‌ی آزمون اجرا می‌شود تا مطمئن شویم بقیه‌ی آزمون‌ها در وضعیت سبز قرار دارند و فقط آخرین آزمون در وضعیت قرمز قرار دارد. سپس تغییر لازم در متن برنامه اعمال می‌شود تا کل مجموعه‌ی آزمون به وضعیت سبز برسد [۲۴].

۶.۲.۲. توسعه‌ی مبتنی بر آزمون

توسعه‌ی مبتنی بر آزمون تکنیکی است که برنامه‌نویس را به استفاده از دو تکنیک توسعه‌ی اول-آزمون و فاکتوربندی مجدد با دو هدف اصلی زیر ملزم می‌کند [۲۴]:

۱. با بهره‌گیری از توسعه‌ی اول-آزمون، برنامه‌نویس موظف است قبل از پیاده‌سازی هر واحد از کد، به نیازمندی آن واحد و طراحی آن واحد فکر کرده و در حقیقت با نوشتن آزمون، نیازمندی آن واحد را نیز توصیف می‌کند.

^{۴۵}Convention

^{۴۶}Modeling Language

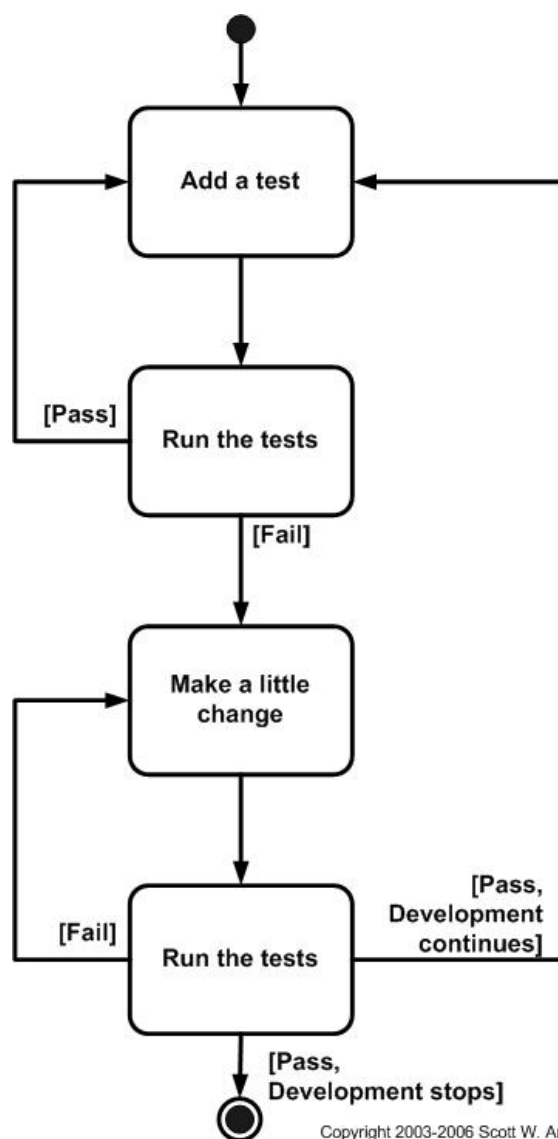
^{۴۷}Syntax

^{۴۸}Semantics

^{۴۹}Activity

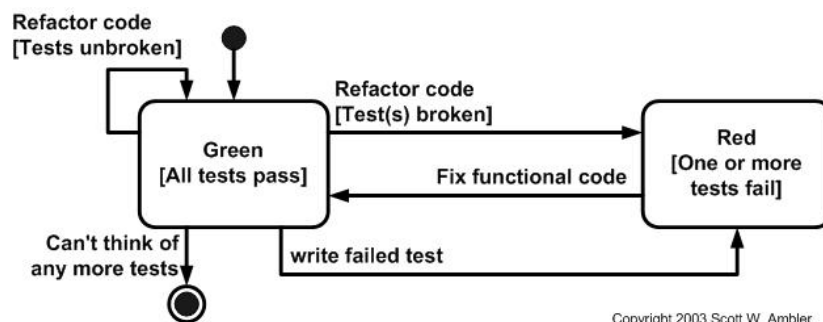
^{۵۰}Pair Programming

^{۵۱}Continuous Integration



Copyright 2003-2006 Scott W. Ambler

شکل ۴.۲: نمودار فعالیت مراحل توسعه‌ی اول-آزمون [۲۴]



شکل ۵.۲: نمودار حالت وضعیت‌های مجموعه‌ی آزمون و نحوه‌ی گذار بین آنها با استفاده از افزودن آزمون، فاکتوربندی مجدد و تغییر متن برنامه [۲۴]

۲. با توجه به اینکه برنامه‌نویس موظف است پس از افزودن هر واحد، فاکتوربندی مجدد را به عنوان مرحله‌ای اجباری از فرآیند توسعه‌ی نرم‌افزار لحاظ کند، کیفیت طراحی نرم‌افزار همواره در بهترین حالت حفظ می‌شود.

همانطور که در نمودار حالت شکل ۵.۲ نمایش داده شده‌است، در حالت سبز، فقط امکان فاکتوربندی مجدد هست و برای تغییر متن برنامه، باید ابتدا با نوشتن یک آزمون برای آن، به حالت قرمز رفته و سپس امکان تغییر متن برنامه را داریم.

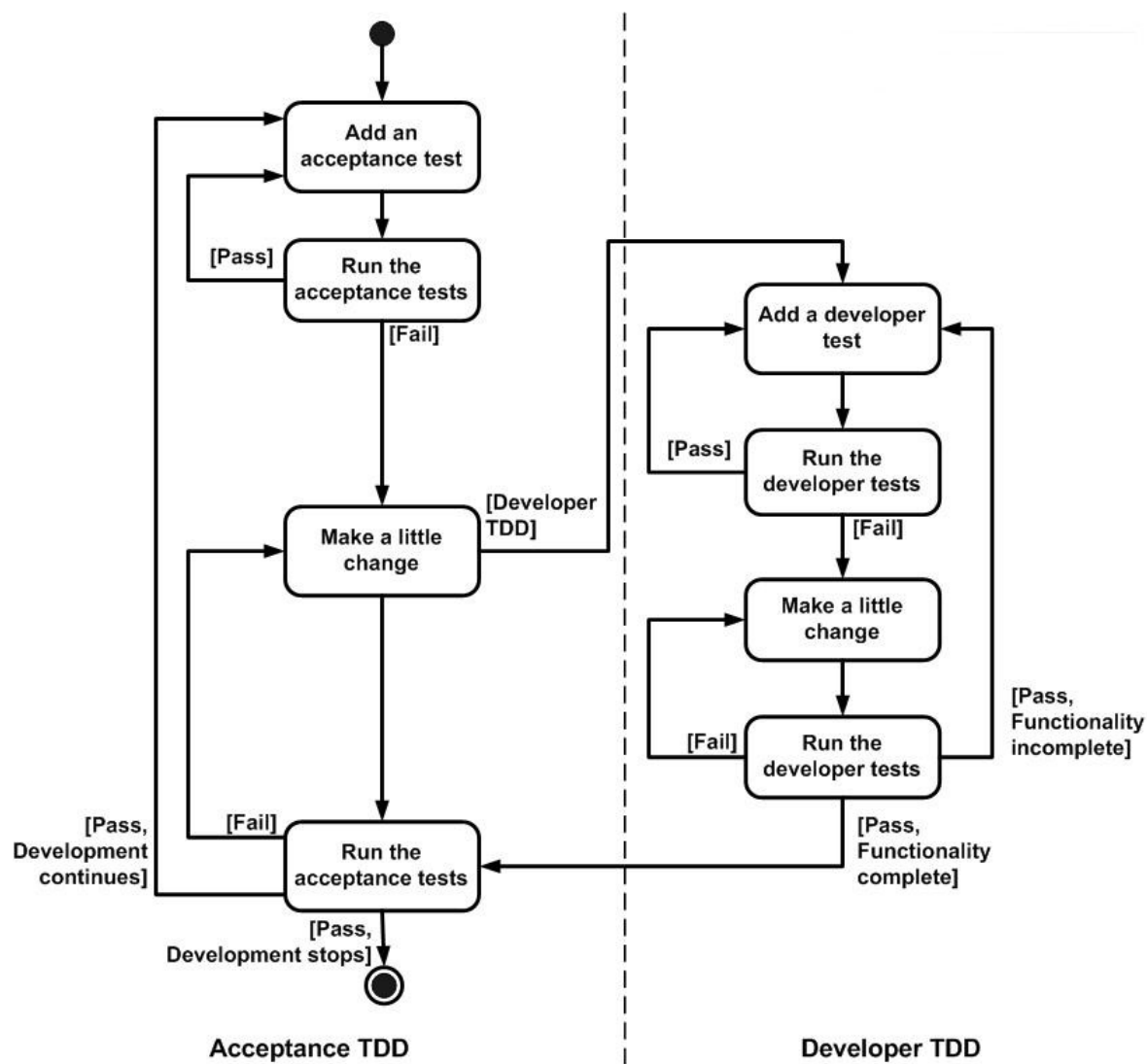
۷.۲.۲. توسعه‌ی آزمون مبتنی بر پذیرش

توسعه‌ی آزمون مبتنی بر پذیرش تکنیکی است که بر دو تجربه زیر استوار است [۱۶]:

۱. پیش از پیاده‌سازی هر قابلیت، اعضای تیم با مشتری درباره مثال‌های واقعی استفاده از آن قابلیت در عمل گفتگو و همکاری می‌کنند. سپس تیم توسعه، این مثال‌ها را به آزمون پذیرش ترجمه می‌کند.
۲. این آزمون‌ها بخش مهمی از توصیف دقیق آن قابلیت محسوب می‌شوند و پیاده‌سازی یک قابلیت زمانی «انجام‌شده»^{۵۲} محسوب می‌شود که این آزمون‌ها در وضعیت سبز قرار گرفته باشند.

همانطور که در نمودار فعالیت ۶.۲ نمایش داده شده، در این تکنیک مشابه با روش توسعه‌ی مبتنی بر آزمون، قبل از پیاده‌سازی هر قابلیت، آزمون پذیرش برای آن نوشته می‌شود. سپس، از اینکه فقط آزمون اخیر در وضعیت قرمز قرار دارد اطمینان حاصل شده و چرخه‌ی توسعه‌ی مبتنی بر آزمون ادامه پیدا می‌کند تا اینکه آزمون پذیرش مربوط به قابلیت مورد نظر در وضعیت سبز قرار گیرد.

^{۵۲} Done



شکل ۶.۲: نمودار فعالیت مراحل توسعه‌ی آزمون مبتنی بر پذیرش [۱۶]

۳.۲. تکنیک توسعه‌ی مبتنی بر رفتار

توسعه‌ی مبتنی بر رفتار یکی از تکنیک‌های توسعه‌ی چابک نرم‌افزار است که در سال ۲۰۰۶ توسط نورث^{۵۳} برای پاسخ‌گویی به ابهام‌ها و مشکل‌هایی که در توسعه‌ی مبتنی بر آزمون بوجود آمده بود، معرفی شد. [۲۵]. تأکید توسعه‌ی مبتنی بر رفتار بر بهینه‌سازی ارتباط میان نقش‌های برنامه‌نویس، آزمون‌گر و خبره‌ی حوزه^{۵۴} می‌باشد و برای تیمی که تمام اعضای آن برنامه‌نویس باشند، توسعه‌ی مبتنی بر رفتار سودی نسبت به توسعه‌ی مبتنی بر آزمون ندارد [۲۶].

این تکنیک، که اخیراً هم در عمل و هم در پژوهش بسیار متداول شده است، حاصل ترکیب و بهبود تجربه‌های معرفی شده در تکنیک‌های توسعه‌ی مبتنی بر آزمون و توسعه‌ی آزمون مبتنی بر پذیرش و علاوه بر آن در نظر گرفتن اصول زیر است [۲۷]:

- لازم است هدف هر داستان کاربر و سودی که برای مشتری دارد، مشخص باشد.
- با نگرش «از بیرون به درون»^{۵۵}، فقط رفتارهایی از سامانه پیاده‌سازی می‌شوند که بیشترین سود را به مشتری می‌رسانند. با داشتن این نگرش، اتلاف هزینه کمینه می‌شود.
- رفتارهای مورد انتظار از سامانه، بین خبره‌ی حوزه، توسعه‌دهنده‌ی سامانه و آزمون‌گر به یک زبان مشترک توصیف می‌شوند. در نتیجه ارتباط ضروری میان این نقش‌ها بهبود می‌یابد.
- این اصول، در تمام سطوح انتزاعی توصیف برنامه، تا پایین‌ترین سطح که پیاده‌سازی یک واحد است، رعایت می‌شوند.

۱.۳.۲. مشخصات توسعه‌ی مبتنی بر رفتار

در حال حاضر، تکنیک توسعه‌ی مبتنی بر رفتار، هنوز در حال توسعه است و تعریف روشنی از توسعه‌ی مبتنی بر رفتار که بر آن اجماع باشد، وجود ندارد. با توجه به اینکه فرآیند توسعه‌ی مبتنی بر رفتار، از ابتدا به صورت انتزاعی معرفی شده و جزئیاتی برای آن ارائه نشده، مشخصات توسعه‌ی مبتنی بر رفتار مبهم و پراکنده هستند. همچنین چارچوب‌ها و ابزارهایی که برای توسعه‌ی مبتنی بر رفتار ارائه شده‌اند، بیشتر به بخش «پیاده‌سازی آزمون» از فرآیند توسعه‌ی مبتنی بر رفتار تمرکز دارند؛ در صورتی که توسعه‌ی مبتنی بر رفتار بر حوزه‌ی گسترده‌تری از فرآیند توسعه‌ی نرم‌افزار تاثیر دارد [۲۸].

^{۵۳}Dan North

^{۵۴}Domain Expert

^{۵۵}From the outside in

در مقاله‌ی [۲۸]، ۶ مورد از مشخصات اصلی توسعه‌ی مبتنی بر رفتار که بر کل فرآیند توسعه‌ی نرم‌افزار، نه فقط قسمت «پیاده‌سازی آزمون»، تاثیر گذار اند، ارائه شده است. در اینجا به صورت خلاصه به آن‌ها اشاره می‌کنیم:

۱. زبان مشترک^{۵۶}:

«زبان مشترک»، هسته‌ی توسعه‌ی مبتنی بر رفتار را تشکیل می‌دهد. زبان مشترک، زبانی است که برآمده از حوزه تجاری^{۵۷} می‌باشد و ابهام را از مکالمه‌ی بین مشتری و تیم توسعه کاهش می‌دهد. همچنین یک واژه‌نامه در ابتدای پروژه ایجاد شده، اکثر واژه‌های آن در مرحله‌ی تحلیل افزوده می‌شوند و در مراحل بعد امکان گسترش دارد.

هر حوزه تجاری، زبان مشترک خاص خود را لازم دارد. توسعه‌ی مبتنی بر رفتار یک قالب ساده برای مرحله‌ی تحلیل ارائه نموده است که مستقل از حوزه تجاری است و از آن در زبان مشترک بهره‌گیری می‌شود. این قالب در بخش ۲.۳.۲ شرح داده شده است.

۲. فرآیند تفکیک^{۵۸} تکراری:

توقع مشتری از یک پروژه‌ی نرم‌افزاری، کسب ارزش تجاری^{۵۹} می‌باشد. معمولاً تشخیص و روشن‌سازی ارزش تجاری، دشوار است. به همین دلیل، ارزش تجاری به اجزای ملموس‌تر تفکیک می‌شود. در شکل ۷.۲ رابطه‌ی این اجزا نسبت به هم نمایش داده شده است.

مرحله‌ی تحلیل در توسعه‌ی مبتنی بر رفتار، با شناسایی رفتار^{۶۰}‌های مورد انتظار از سامانه آغاز می‌شود. در مراحل ابتدایی، شناسایی رفتارهای مورد انتظار آسان‌تر از شناسایی ارزش‌های تجاری سامانه است. هر رفتار تجاری، تعدادی نتیجه‌ی تجاری^{۶۱} محسوس را محقق می‌سازد.

هر نتیجه‌ی تجاری با یک مجموعه‌ی ویژگی^{۶۲} اعمال می‌شود. یک مجموعه‌ی ویژگی با گفتگو بین تیم توسعه و مشتری است برای تحقق نتیجه‌ی تجاری در سامانه تدوین گشته و اولویت‌بندی و تبیین صریح ارتباط آن با نتیجه‌ی تجاری، ضروری است.

در نهایت محدوده^{۶۳} هر مجموعه‌ی ویژگی، با استفاده از چند داستان کاربر معین می‌شود. هر داستان کاربر، یک تعامل بین کاربر و سامانه را توصیف می‌کند. هر داستان کاربر، به سه سوال زیر پاسخ می‌دهد:

^{۵۶} Ubiquitous Language

^{۵۷} Business Domain

^{۵۸} Decomposition

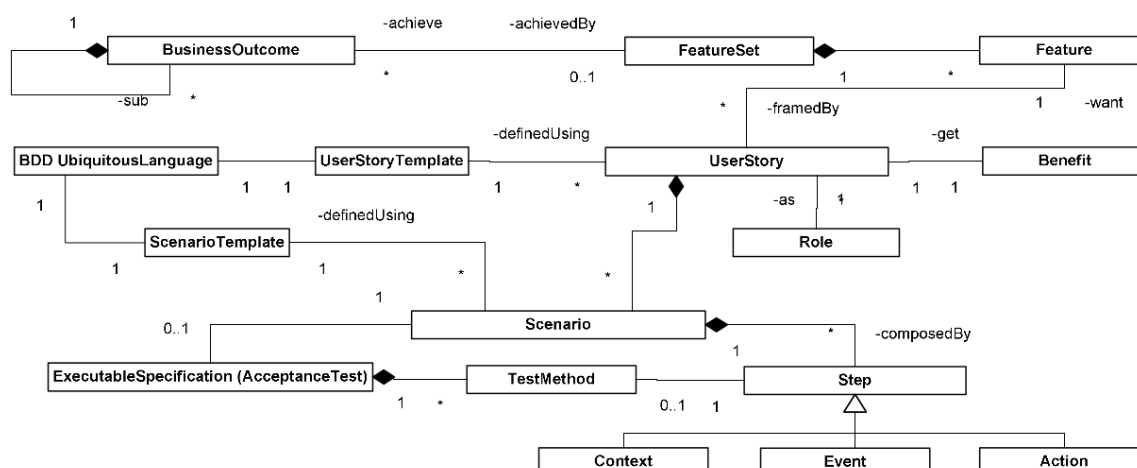
^{۵۹} Business Value

^{۶۰} Behavior

^{۶۱} Business Outcome

^{۶۲} Feature Set

^{۶۳} Scope



شکل ۷.۲: مدل مفهومی در تحلیل توسعه‌ی مبتنی بر رفتار [۲۸]

• نقش کاربر در داستان کاربر چیست؟

• کاربر از این ویژگی چه می‌خواهد؟

• اگر سامانه این ویژگی را داشته باشد، چه سودی به کاربر می‌رسد؟

همچنین برای هر داستان کاربر، چند شرط پذیرش به زبان مشتری تعیین می‌گردد که اگر برآورده شوند، آن داستان کاربر به درستی محقق شده است. در توسعه‌ی مبتنی بر رفتار، شرط پذیرش در قالب «سناریو» توصیف می‌شود (ر.ک. به ۲.۳.۲).

اجرای تکراری فرآیند در توسعه‌ی مبتنی بر رفتار ضروری است. در هر مرحله از توصیف نیازمندی‌های تا حدی پیش می‌رویم که بتوانیم ویژگی جدیدی را پیاده‌سازی نماییم.

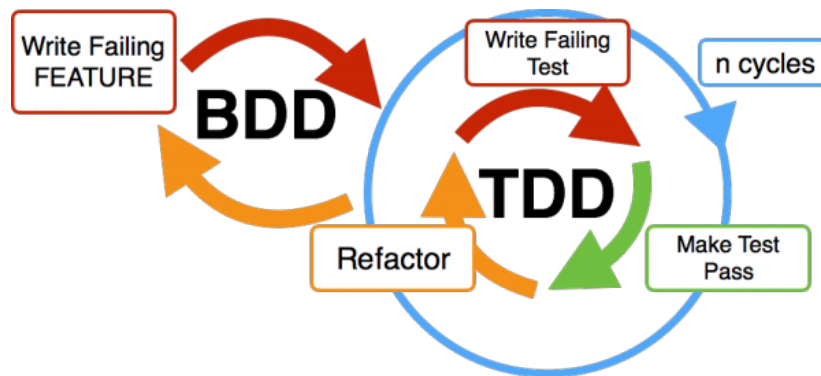
۳. قالب مشخص برای توصیف داستان کاربر و سناریو:

در توسعه‌ی مبتنی بر رفتار، توصیف ویژگی‌ها و داستان‌های کاربر در قالبی مشخص صورت می‌گیرد و دلخواه نیست. این قالب‌ها، قسمتی از زبان مشترک را تشکیل می‌دهند و ساختار متن ساده^{۶۴} دارند (ر.ک. به ۲.۳.۲). ساختار متن ساده باعث می‌شود هم محدودیتی برای تعریف زبان مشترک بوجود نیاید، هم زبان برای تمام اعضای پروژه قابل فهم باشد، در عین اینکه قالبی دارد که توسعه‌دهنده‌ها بتوانند آن به راحتی آن را به آزمون تبدیل نمایند.

۴. فرآیند توسعه‌ی آزمون مبتنی بر پذیرش:

توسعه‌ی مبتنی بر رفتار تکنیک‌های استفاده شده در فرآیند توسعه‌ی آزمون مبتنی بر پذیرش (ر.ک. به

^{۶۴} Plain Text



شکل ۸.۲: فرآیند توسعه‌ی ویژگی‌ها در توسعه‌ی مبتنی بر رفتار [۲۹]

۷.۲.۲) را به ارث می‌برد. همانطور که در شکل ۸.۲ نمایش داده شده، مشابه تکنیک توسعه‌ی آزمون مبتنی بر پذیرش، پیش از آغاز به توسعه‌ی هر ویژگی برنامه، آزمون پذیرش برای آن نوشته می‌شود. این آزمون پذیرش، مستقیماً از روی شرط پذیرش به زبان مشترک با مشتری نوشته شده، تولید می‌شود. سپس فرآیند توسعه‌ی مبتنی بر آزمون (ر.ک. به ۶.۲.۲) ادامه پیدا می‌کند تا آنکه آزمون پذیرش با موفقیت انجام شود.

۵. توصیف خوانا و اجرایی^{۶۵}:

در توسعه‌ی مبتنی بر رفتار، با توجه به اینکه آزمون پذیرش مستقیماً از روی شرط پذیرش که شرایط پذیرش داستان کاربر به زبان کاربر است نوشته می‌شود، توصیفی خوانا برای برنامه است که قابلیت اجرایی نیز دارد.

در توسعه‌ی مبتنی بر رفتار، توصیه می‌شود متن برنامه نیز به صورت رفتار-محور نوشته شود؛ متن برنامه باید توصیف رفتار کلاس‌های باشد و نام تابع^{۶۶}ها و کلاس‌ها، از زبان مشترک گرفته شده باشند. گرچه رعایت این موارد در متن برنامه در بعضی موارد ممکن نیست، توصیه شده تا جای ممکن، نام کلاس‌ها نمایانگر عنوان داستان کاربر باشد و نام تابعها، نمایانگر سناریوها باشند.

۶. رعایت مشخصات مبتنی بر رفتار در مراحل مختلف:

مشخصات ذکر شده درباره‌ی توسعه‌ی مبتنی بر رفتار، درباره‌ی مرحله‌های مختلف از توسعه‌ی نرم‌افزار صادق‌اند. در مرحله‌ی برنامه‌ریزی اولیه، موارد نتیجه‌ی تجاری مشخص می‌شوند. در مرحله‌ی تحلیل، مجموعه‌ی ویژگی حاصل می‌شود و در مرحله‌ی پیاده‌سازی و آزمون، قواعد توسعه‌ی مبتنی بر آزمون و آزمون پذیرش رعایت می‌شوند.

با وجود اینکه توسعه‌ی مبتنی بر رفتار برای تمام فازها توصیه‌ی رعایت توسعه‌ی مبتنی بر رفتار را دارد،

^{۶۵}Executable

^{۶۶}Method

ابزار موجود برای توسعه‌ی مبتنی بر رفتار، در تمام مراحل، برای آن قابلیت ندارند. برای مثال، در حال حاضر، هیچ کدام از ابزاری که به‌عنوان ابزار توسعه‌ی مبتنی بر رفتار یا چارچوب توسعه‌ی مبتنی بر رفتار موجود هستند، قابلیت برای رعایت توسعه‌ی مبتنی بر رفتار در مرحله‌ی برنامه‌ریزی اولیه ارائه نمی‌کنند.

۲.۳.۲. قالب توصیف داستان‌های کاربر و سناریوها

در مرحله‌ی تحلیل از توسعه‌ی مبتنی بر رفتار، برای هر مجموعه‌ی ویژگی، تعدادی داستان کاربر نوشته می‌شود. هر داستان کاربر، با قالب زیر تدوین می‌شود:

Story: [Story Title]

As a [Role]

I want to [Feature]

So that [Benefit]

این قالب، چهار قسمت برای پرکردن دارد که گاهی به‌صورت زیر نیز بیان می‌شود و این دو قالب کاملاً هم‌ارز هستند:

Feature: [Story Title]

In order to [Benefit]

As a [Role]

I need to [Feature]

قسمت‌هایی که در [] مشخص شده‌اند، برای هر داستان کاربر با زبان مشترک مطابق توضیحات زیر تکمیل می‌شوند:

۱. در سطر اول، در قسمت «عنوان داستان^{۶۷}» عنوانی دلخواه برای شناسایی آسان‌تر داستان کاربر نوشته می‌شود.

۲. در قسمت «نقش^{۶۸}»، نقشی که این داستان کاربر به آن سود می‌رساند را مشخص می‌سازد. مشخص‌سازی نقش، هنگام مرحله‌ی تحلیل، به توسعه‌دهنده و مشتری کمک می‌کند هنگام تعیین جزئیات و سطح دسترسی، نقش کاربر را در نظر بگیرند و از پیاده‌سازی ویژگی‌هایی که لازم نیستند، جلوگیری نمایند.

۳. در قسمت «ویژگی^{۶۹}»، فعالیتی که کاربر می‌تواند توسط سامانه انجام دهد تعیین می‌شود.

۴. در قسمت «سود^{۷۰}»، هدف از اضافه نمودن این داستان کاربر به سامانه و سودی که به کاربر می‌رسد

^{۶۷}Story Title

^{۶۸}Role

^{۶۹}Feature

^{۷۰}Benefit

مشخص می‌شود. نوشتن این قسمت، کمک می‌کند تا هر ویژگی از سامانه طوری پیاده‌سازی شود که به کاربر سود برساند و ویژگی‌هایی که سود قابل توجهی به کاربر نمی‌رسانند، در اولویت کمتر قرار گیرند. از هر ویژگی، در شرایط متفاوت، رفتارهای متفاوتی انتظار داریم. نحوه‌ی اداره‌کردن خطاهای مختلف و جریان‌های جایگزین اجرای برنامه، لازم است توسط مشتری تبیین شوند. برای تبیین این شرایط، برای هر داستان کاربر تعدادی شرط پذیرش نوشته می‌شود که در توسعه‌ی مبتنی بر رفتار، «سناریو» نام دارند. هر سناریو، رفتار مورد انتظار سامانه که توسط این داستان کاربر توصیف شده را هنگام رخ دادن یک رویداد، در یک شرایط اولیه‌ی خاص تبیین می‌کند.

برای هر سناریو، قالب زیر رعایت می‌شود:

Scenario 1: [Scenario Title]

Given [Context]

When [Event]

Then [Outcome]

And [Some more outcomes]

Scenario 2:...

قسمت‌هایی که در [] مشخص شده‌اند، برای هر سناریو با زبان مشترک مطابق توضیحات زیر تکمیل می‌شوند:

۱. قسمت «عنوان سناریو^{۷۱}»، عنوانی دلخواه برای شناسایی آسان‌تر سناریو نوشته می‌شود.
۲. قسمت «زمینه^{۷۲}» که به قسمت «اگر^{۷۳}» نیز معروف است، پیش‌شرایطی را برای وضعیت سامانه، پیش از رخداد رویداد مشخص می‌کند. این سناریو و نتایج مورد انتظار آن از سیستم، تنها در صورتی معتبر هستند که این پیش‌شرایط هنگام رخ دادن رویداد برقرار باشند.
۳. قسمت «رویداد^{۷۴}» که به قسمت «وقتی^{۷۵}» نیز معروف است، رویدادی را مشخص می‌سازد که این سناریو، رفتار مورد انتظار سامانه را در پاسخ به آن رویداد تبیین نموده است.
۴. قسمت «پیامد^{۷۶}» که به قسمت «آنگاه^{۷۷}» نیز معروف است، وضعیت مورد انتظار از سامانه پس از رویداد

^{۷۱}Scenario Title

^{۷۲}Context

^{۷۳}Given

^{۷۴}Event

^{۷۵}When

^{۷۶}Outcome

^{۷۷}Then

```
Feature: ls
  In order to see the directory structure
  As a UNIX user
  I need to be able to list the current directory's contents

Scenario: List 2 files in a directory
  Given I am in a directory "test"
  And I have a file named "foo"
  And I have a file named "bar"
  When I run "ls"
  Then I should get:
    """
    bar
    foo
    """
```

شکل ۹.۲: یک داستان کاربر به همراه شرط پذیرش [۳۰]

را تبیین می‌کند.

۵. قسمت «و^{۷۸}»، به هر کدام از بخش‌های زمینه یا پیامد می‌تواند اضافه شود تا شرایط آن قسمت را دقیق‌تر توصیف نماید.

به عنوان مثال، در شکل ۹.۲، یک سناریو برای داستان کاربر عملیات لیست پرونده‌های یک پوشه نمایش داده شده است. در این شکل، تنها یک سناریو برای این داستان کاربر بیان شده؛ اما می‌توان سناریوهای دیگری مانند رفتار مورد انتظار از این دستور هنگامی که دسترسی گرفتن لیست فایل‌ها وجود نداشته باشد یا هنگامی که پوشه‌ی فعلی حذف شده باشد را نیز به مجموعه‌ی سناریوهای تبیین کننده‌ی این داستان کاربر اضافه نمود.

^{۷۸}And

فصل ۳

طراحی و توسعه‌ی چارچوب

آزمون برنامه راهی برای افزایش کیفیت نرم‌افزار است. پیش از این رویکردهایی در توسعه‌ی آزمونهای نرم‌افزاری را دیدیم. با ایده گرفتن از نقاط قوت آنها، چارچوب آزمون جدیدی را جهت انجام آزمون پیشنهاد می‌کنیم و فواید استفاده از آن را بررسی می‌کنیم.

در انتهای فصل نیز گزارشی از پیاده‌سازی چارچوب مربوطه بر پایه‌ی جنگو ارائه می‌کنیم.

۱.۳. معماری چارچوب

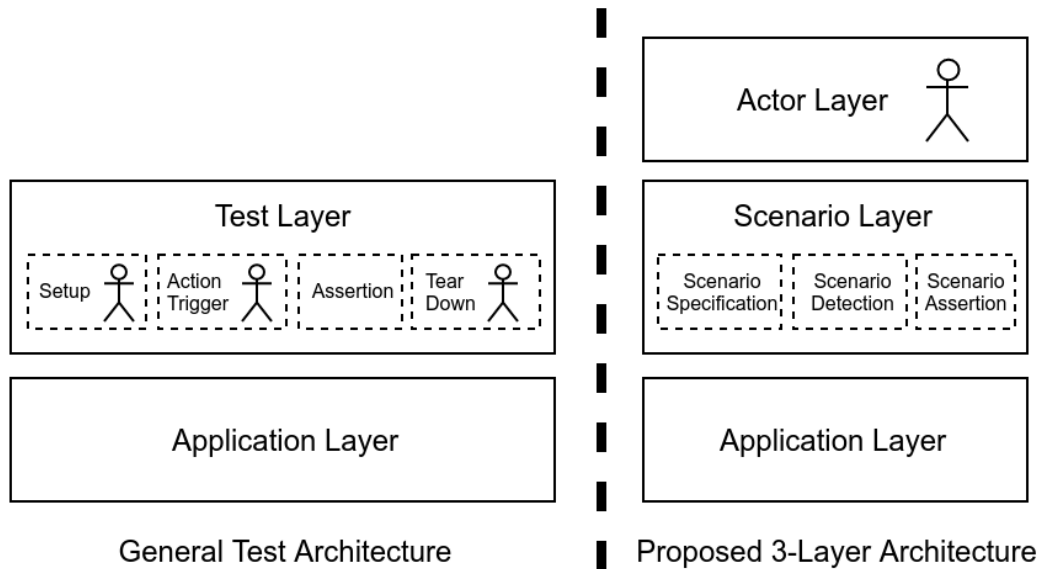
در بخش ۲.۳.۲ با نحوه‌ی بیان یک سناریو آشنا شدیم. یک سناریو، در واقع پاسخ صحیح یک زیرمسئله از مسئله‌ی اصلی را بیان می‌کند.

چارچوب پیشنهادی ما، حول ایده‌ی مستقل‌سازی سناریوها از مصادیق صحت‌سنجی آن شکل گرفته است. در این معماری، سناریوهایی توسط برنامه‌نویس تعبیه می‌شود که مطابق تعریف، انتظار می‌رود همواره برقرار باشند. مجموعه‌ی این سناریوها را به عنوان یک لایه از معماری چارچوب آزمون خود در نظر می‌گیریم و آن را «لایه‌ی سناریو» می‌نامیم. در این لایه، نویسنده‌ی آزمون تنها به توصیف نیازمندی‌های صحت‌سنجی برنامه می‌پردازد و دغدغه‌ی تولید مصادیق سناریوها را نخواهد داشت.

با توجه به تعریف، خود سناریوها تنها مجموعه‌ای از گزاره‌ها هستند که در یک پیاده‌سازی درست، برقرار خواهند بود. حال آن که سنجش صحت این گزاره‌ها می‌تواند متریک^۱ خوبی برای کیفیت کد باشد. در چارچوب پیشنهادی، لایه‌ی دیگری از معماری آزمون به این موضوع تخصیص دارد که آن را «لایه‌ی تعامل‌گر»^۲ می‌نامیم. در واقع، لایه‌ی تعامل‌گر موظف است تا با تغییر مداوم وضعیت سامانه، پیش-شرط‌های سناریوهای مختلف را

^۱Metric

^۲Actor



شکل ۱.۳: مقایسه معماری پیشنهادی با معماری متداول آزمون

فراهم کند تا صحت آن‌ها سنجیده شود.

همانطور که در شکل ۱.۳ نشان داده شده، دو لایه‌ی سناریو و تعامل‌گر، ارکان اصلی معماری پیشنهادی (معماری سمت راست در شکل) جهت پیاده‌سازی چارچوب آزمون هستند. این دو لایه، جایگزین لایه‌ی «آزمون» در معماری رایج (معماری سمت چپ در شکل) می‌شوند. لایه‌ی کاربرد^۳، در هر دو معماری مشترک است و متن برنامه‌ی اصلی در این لایه نوشته می‌شود.

در معماری رایج، لایه‌ی آزمون عملیات زیر را انجام می‌دهد:

- **آماده‌سازی^۴:** همانطور که در بخش ۱.۲.۲ اشاره شد، برای بررسی عملکرد اس.یو.تی. در هر مورد آزمون لازم است ورودی‌ها و پیش‌شرایط آن مهیا شده باشند. مثلاً برای بررسی عملکرد قسمت «ثبت‌نام» از یک سامانه‌ی آموزشی، لازم است یک دانشجو و یک ارائه از درس در سامانه موجود باشند تا بتوان صحت عملکرد ثبت‌نام را بر روی آن سنجید.
- **تحریک کنش^۵ در سامانه^۶:** پس از آماده‌سازی حالت سامانه، دستوری که باعث اعمال رفتار مورد نظر توسط اس.یو.تی. می‌شود را فراخوانی می‌نماییم. برای مثال، برای بررسی عملکرد ثبت‌نام در سامانه‌ی آموزش، تابعی که مربوط به عمل ثبت‌نام است را فراخوانی می‌نماییم.
- **اثبات صحت عملکرد^۷:** پس از تحریک کنش در سامانه، باید حالت نهایی سامانه را بررسی نماییم و

^۳ Application

^۴ Setup

^۵ Action

^۶ Action Trigger

^۷ Assertion

برقراری پسا-شرط‌هایی که توسط مشتری برای این عمل تعریف شده را در سامانه بررسی نماییم. در سامانه‌ی آموزش، باید مواردی مانند کم شدن ظرفیت ارائه‌ی درس پس از ثبت‌نام و قرار گرفتن دانشجو در لیست دانشجویان ارائه‌ی درس را بررسی نماییم.

- **خنثی‌سازی تغییرات^۸:** پس از اطمینان از بررسی صحت عملکرد اس.یو.تی.، تغییراتی که در قسمت‌های آماده‌سازی و تحریک، در حالت سامانه ایجاد شده را خنثی‌سازی می‌نماییم تا در عملیات سایر آزمون‌ها تداخل ایجاد نشود.

برای مثال، هنگام نوشتن آزمون پذیرش برای سناریوی «اگر یک دانشجو و یک ارائه‌ی درس موجود باشند؛ وقتی دانشجو در ارائه ثبت‌نام می‌کند؛ آن‌گاه باید ظرفیت ارائه یک واحد کم شده باشد»:

- قسمت آماده‌سازی، باید موجودیت‌های مربوط به نیم‌سال تحصیلی، درس، استاد، ارائه‌ی درس توسط استاد و دانشجو را در سامانه ایجاد نماید.

- قسمت تحریک کنش، باید تابع مربوط به ثبت‌نام دانشجو در ارائه‌ی درس را فراخوانی نماید.

- قسمت اثبات صحت عملکرد، باید ظرفیت ارائه را قبل و بعد از فراخوانی تابع ثبت‌نام، مقایسه نماید.

- قسمت خنثی‌سازی تغییرات، باید موجودیت‌هایی که در قسمت‌های آماده‌سازی و تحریک کنش در سامانه ایجاد شدند را حذف نماید

اما در معماری سه‌لایه‌ی پیشنهادی، برخلاف معماری رایج، هنگام تبدیل شرط پذیرش (متن ساده) به آزمون پذیرش (آزمون اجراپذیر)، به جای اعمال شرایط ذکر شده در قسمت «اگر» سناریو بر روی سامانه، آن شرایط را صرفاً توصیف می‌نماییم؛ همچنین به جای تحریک کنش مشخص شده در قسمت «وقتی» سناریو، رخداد آن کنش در سامانه را توصیف می‌نماییم. بدین ترتیب، آزمون پذیرش‌های به‌دست آمده، کاملاً توصیفی بوده و نه‌تنها تغییری در وضعیت سامانه بوجود نمی‌آورند، بلکه به‌تنهایی قابل اجرا نیز نمی‌باشد. این آزمون پذیرش‌ها در لایه‌ی سناریو نوشته می‌شوند و قسمت «توصیف سناریو^۹» را تشکیل می‌دهند.

از طرفی، تمام عملیاتی که در قسمت‌های آماده‌سازی، تحریک کنش و خنثی‌سازی تغییرات از معماری رایج انجام می‌شدند، به لایه‌ی تعامل‌گر منتقل می‌شوند. همانطور که در شکل ۱.۳ با علامت آدمک مشخص شده، این قسمت‌ها همگی با سامانه در تعامل هستند و در حالت سامانه تغییر ایجاد می‌کنند.

لایه‌ی تعامل‌گر، مستقل از اینکه چه سناریوهایی در سامانه وجود دارد، با سامانه تعامل نموده و کنش‌های مختلف را بر روی آن اعمال می‌کند. ممکن است این تعامل توسط کاربر با استفاده از رابط کاربری در محصول نهایی^{۱۰} صورت پذیرد یا اینکه با فراخوانی توابع داخلی سامانه توسط برنامه‌نویس وضعیت سامانه تغییر یابد.

^۸Tear Down

^۹Scenario Specification

^{۱۰}Production

قسمت «تشخیص سناریو»^{۱۱} از لایه‌ی سناریو، موظف است کنش‌هایی که توسط لایه‌ی تعامل‌گر بر روی سامانه صورت می‌پذیرد را واری نمود و با توجه به اینکه توصیف بخش‌های «اگر»، «وقتی» و «آنگاه» هر سناریو موجود است، حالت فعلی سامانه را با توصیف «اگر» مطابقت داده و اگر کنشی که در حال رخداد است، با «وقتی» مطابق بود، پس از اجرای کنش، قسمت «اثبات سناریو»^{۱۲} را جهت بررسی برقراری «آنگاه» و ثبت نتیجه و تولید گزارش فعال می‌سازد.

۲.۳. فواید

۱.۲.۳. کاهش حجم آزمون‌ها

یکی از دلایل عمده‌ی عدم علاقه‌ی برنامه‌نویس‌ها به تألیف آزمون، حجم زیاد آزمون در مقایسه با میزان کد پوشیده شده توسط آن است. هر چه سطح آزمون به سطح سیستمی نزدیک‌تر باشد، حجم این کد نیز به دلیل آماده کردن شرایط اولیه‌ی آزمون افزایش می‌یابد.

با توجه به این که در معماری پیشنهادی، تعریف سناریوها مستقل از صحت‌سنجی آن‌ها انجام می‌شود، امکان بازبهره‌گیری کد^{۱۳} وضعیت یک سامانه سناریوهای متفاوت وجود دارد که این موضوع منجر به کاهش حجم آزمون‌ها می‌شود.

۲.۲.۳. کاهش هزینه‌ی نگهداری

از آن جا که تغییر سریع در بسیاری از سامانه‌های نرم‌افزاری ضروری است، حجم زیاد آزمون‌ها (ر.ک. ۱.۲.۳) در هنگام تغییر نیز نیاز به هم‌گام‌سازی دارند. در معماری تک‌لایه، با توجه به اینکه شرایط اجرای آزمون توسط خود آن فراهم می‌شود، وابستگی^{۱۴} به مؤلفه^{۱۵}‌های خارج از محدوده‌ی آزمون وجود دارد و هنگام تغییر رفتار هر مؤلفه، این تغییرات در میان تمام آزمون‌های درگیر انتشار می‌یابد.

با توجه به جدا شدن لایه‌ی سناریو و تعامل‌گر در معماری پیشنهادی، تغییرات محدود به بخشی از آزمون‌ها خواهند بود که واقعاً تغییر کرده و به بخش‌های دیگر انتشار نمی‌یابند.

^{۱۱} Scenario Detection

^{۱۲} Scenario Assertion

^{۱۳} Code Reuse

^{۱۴} Coupling

^{۱۵} Component

۳.۲.۳. افزایش احتمال یافتن خطا

یکی از تفاوت‌های اساسی معماری پیشنهادی و معماری‌های تک‌لایه در کلی بودن تعریف سناریوهاست؛ به این معنا که یک سناریو می‌تواند روی داده‌های مختلفی قابل اعمال باشد. بنابراین، هر سناریو عملاً معادل یک مجموعه‌ی آزمون عمل می‌کند.

این عمومی بودن تعریف سناریو کمک می‌کند تا صحت آن نه فقط روی یک وضعیت سامانه، بلکه برای چندین حالت متفاوت بررسی شود. این موضوع احتمال یافتن خطاهای موجود را افزایش داده و در نتیجه موجب افزایش تأثیرگذاری فرآیند آزمون می‌گردد.

۳.۳. استفاده در سطح سیستمی

اگرچه معماری پیشنهادی ما مستقل از سطح آزمون است، با این حال برداشت مؤلفان این است که تأثیر مثبت آن در سطح آزمون سامانه مشهودتر خواهد بود. در این لایه به دلیل سطح بالا بودن آزمون‌ها، حجم آماده‌سازی‌هایی که یک مورد آزمون می‌بایست انجام دهد تا به پیش-شرط دلخواه برسد بسیار بیشتر بوده و با توجه به آن‌چه در ۱.۲.۳ آمد، معماری پیشنهادی از طریق جدا کردن تعامل‌گر و بازبه‌کارگیری کد آن، به کاهش حجم و هزینه‌ی نگهداری این‌گونه آزمون‌ها کمک شایانی می‌کند.

۴.۳. استفاده خارج از محیط آزمون

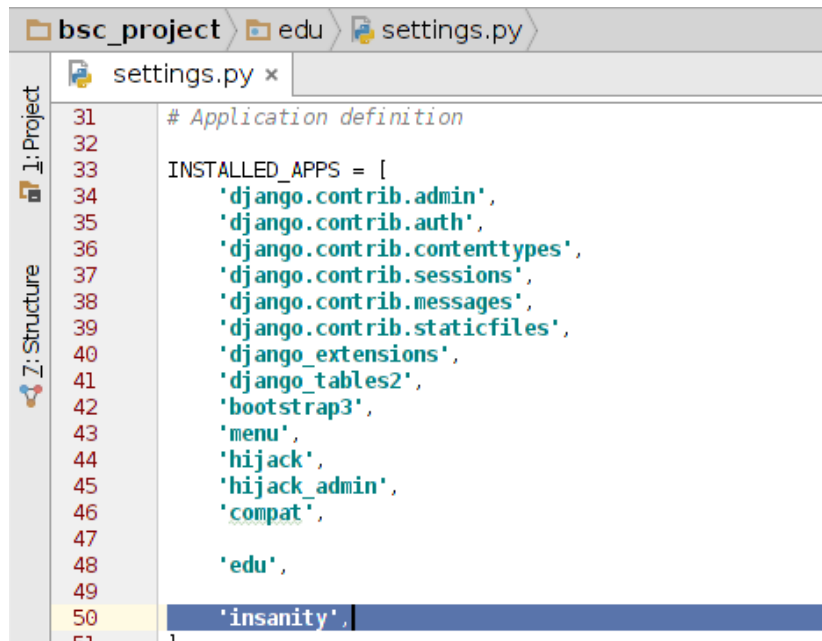
انجام آزمون تنها یکی از روش‌های افزایش کیفیت نرم‌افزار است. ادعا می‌کنیم که معماری پیشنهادی خارج از محیط آزمون نیز می‌تواند به افزایش کیفیت نرم‌افزار کمک کند.

در هنگام آزمون، هدف یافتن نقص^{۱۶} نرم‌افزار، پیش از عملیاتی شدن آن است. اما بسیاری از نقص‌های نرم‌افزاری حتی پس از عملیاتی شدن نیز از دیده‌نمان می‌مانند.

در چارچوب ارائه‌شده، می‌توان به جای لایه‌ی اکتور، کاربران حقیقی سامانه را قرار داد تا از نرم‌افزار استفاده کنند. به این ترتیب، سامانه می‌تواند عملکرد خودش را (حتی در حالی که عملیاتی است) ارزیابی کند و برخی از این نقص‌ها را یافته، و جهت رسیدگی توسعه‌دهندگان گزارش کند. از جمله نقص‌هایی که به خوبی به این روش پیدا می‌شوند، بروز ناهم‌خوانی^{۱۷} در مقادیر محاسبه‌شده‌ی است.

^{۱۶}Defect

^{۱۷}Inconsistency



شکل ۲.۳: افزودن چارچوب به جنگو

۵.۳. پیاده‌سازی بر مبنای چارچوب جنگو

به عنوان بخشی از پروژه کارشناسی، یک چارچوب برای آزمون توسعه‌ی مبتنی بر رفتار با جنگو و مبتنی بر معماری ارائه‌شده در فصل ۱.۳ را پیاده‌سازی نمودیم.

۱.۵.۳. نحوه‌ی استفاده

این چارچوب در قالب یک افزونه برای جنگو نوشته شده. برای استفاده از این افزونه مراحل زیر را انجام دهید:

۱. با اجرای دستور

```
pip install https://github.com/SeMeKh/bsc_project/archive/master.zip
```

چارچوب اینسیتی را نصب نمایید.

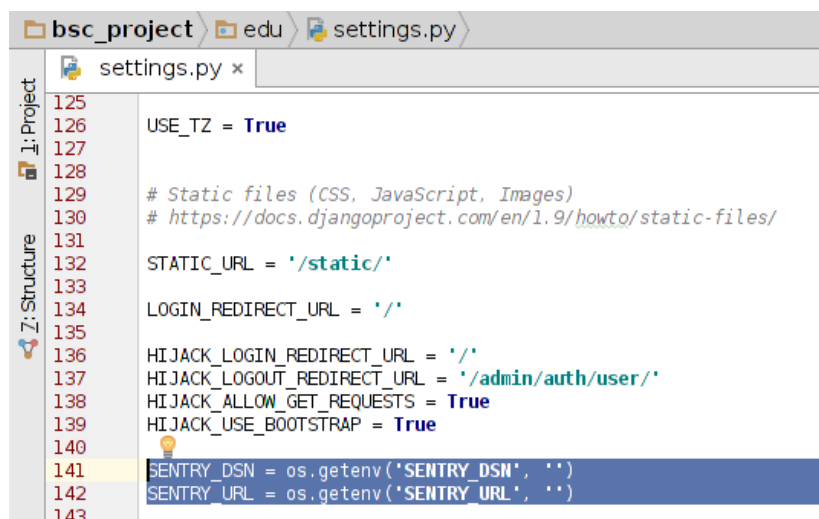
۲. نام افزونه (insanity) را مطابق شکل ۲.۳ به لیست افزونه‌ها در تنظیمات جنگو اضافه نمایید.

۳. تنظیمات ستری را جهت ثبت جزییات خطاها مطابق شکل ۳.۳ در فایل settings.py وارد نمایید.

۴. سناریوهای مطلوب را در فایل scenarios.py مرتبط با هر مؤلفه جنگو قرار دهید. هر سناریو، با یک

کلاس که از کلاس پایه‌ی Scenario ارث می‌برد، نمایش داده می‌شود که موظف است در تابع «اگر»،

پیش‌شرایط سناریو را در حالت سامانه بررسی کند و در صورتی که پیش‌شرایط برآورده می‌شوند، مقدار True



شکل ۵.۳.۳: ورود تنظیمات سنتری

را باز گرداند. صفت «وقتی» نام کنشی که این سناریو را فعال می‌کند را تعیین می‌کند. در متد «آنگاه» نیز پس-شرط‌های سناریو را بررسی نمایید.

۵. هر کدام از توابعی که به عنوان محرک^{۱۸} سناریوها عمل می‌کنند، توسط دکوراتور action پوشیده شوند.

۶. به طریق دلخواه، با سامانه تعامل ایجاد نمایید به طوریکه کنش‌های تعیین شده توسط سناریوها اعمال شوند.

۷. از نشانی `http://localhost:8000/insanity` نتیجه‌ی بررسی سناریوها را مشاهده کنید. همانطور

که در شکل ۵.۳ مشاهده می‌نمایید، می‌توان برای هر سناریو تعداد اجراهای موفق و ناموفق نمایش داده می‌شود. همچنین مطابق شکل ۴.۳، با کلیک بر روی علامت زنجیر در کنار سناریوهایی که ناموفق بوده‌اند، به صفحه‌ی سنتری مرتبط به آن سناریو منتقل می‌شوید و می‌توانید از حالت برنامه هنگام شکست سناریو مطلع شوید.

۲.۵.۳. اجزاء پیاده‌سازی

۱.۲.۵.۳. دکوراتور action

این دکوراتور به توابع مختلف اعمال می‌شود و آن‌ها را به عنوان یک محرک سناریو ثبت می‌کند. به این ترتیب، در صورت اجرای این توابع، صحت سناریوهای مرتبط با آن‌ها بررسی می‌شود.

همانطور که در نمودار توالی شکل ۶.۳ نمایش داده شده است، اگر یک تابع (مانند متد `enroll` از

کلاس Offering در شکل) توسط دکوراتور action نشانه‌گذاری شده باشد، هنگام فراخوانی آن تابع، پیش

^{۱۸}Trigger

AssertionError

insanity/action.py in _assert_scenarios at line 35

```
return_value=self.return_value, payload=self.payload)
```

edu/scenarios.py in then at line 50

```
45.     def then(scenario, payload, **kwargs):
46.         available_capacity = scenario.ol.available_capacity
47.         total_capacity = scenario.ol.capacity
48.         enrollment_count = scenario.ol.enrollment_set.count()
49.         assert available_capacity == total_capacity - enrollment_count
50.         assert total_capacity >= enrollment_count
51.
52.
53. class EnrollmentShouldFailForOfferingWithZeroCapacity(Scenario):
54.     """
55.     Scenario: Enrollment should fail for offering with zero capacity
```

available_capacity

-1

enrollment_count

1

kwargs

```
{
  'exc_tb': None,
  'exc_type': None,
  'exc_val': None,
  'return_value': None
}
```

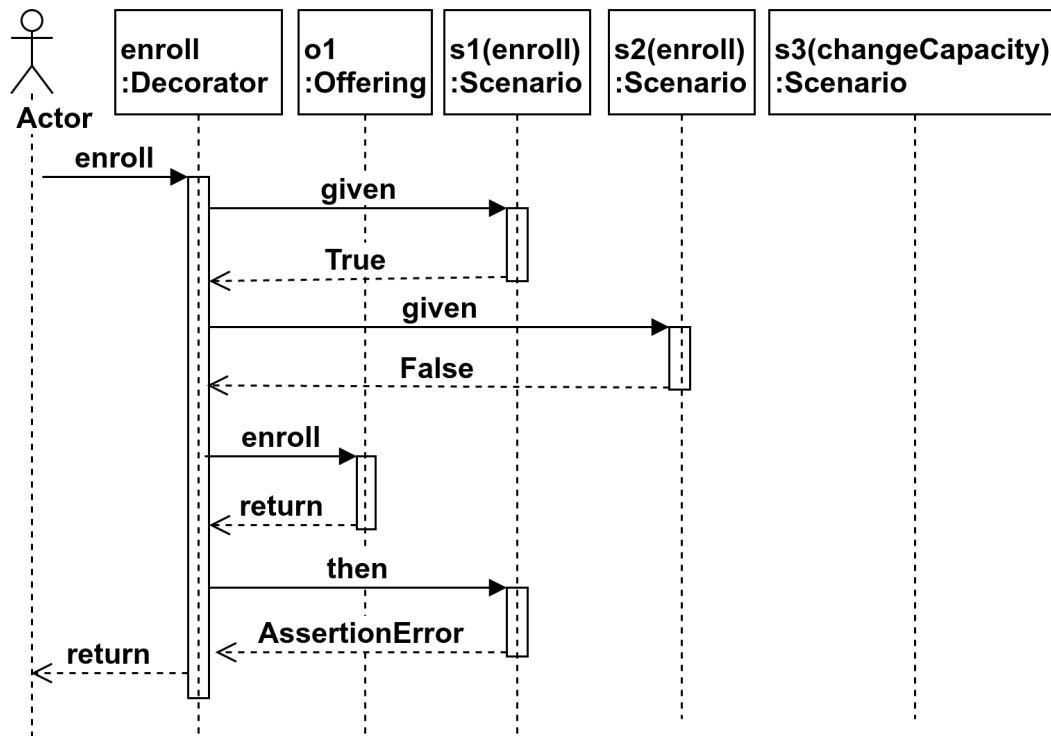
شکل ۴.۳: نمایش جزئیات شکست یک سناریو در سنتری



Insanity :: Scenario Validation Report

Scenario	Triggered (Pass/Fail)
edu.scenarios.AvailableCapacityRemainsConsistentWhenCapacityChanges	8 (6/2) 🔗
edu.scenarios.EnrollmentShouldFailForOfferingWithZeroCapacity	0 (0/0)
edu.scenarios.CapacityDecreasesByEnroll	1 (1/0)

شکل ۵.۳: گزارش پوشش سناریوهای تعریف شده



شکل ۶.۳: نمودار توالی عملکرد decorator

از رسیدن کنترل برنامه به تابع enroll، کنترل به دست دکوراتور می‌رسد و این دکوراتور پیش از فراخوانی تابع اصلی، سناریوهایی که when آن‌ها برابر enroll باشد را انتخاب نموده، پیش از فراخوانی تابع اصلی given آن‌ها را اجرا نموده و اگر مقدار خروجی قسمت given برابر True باشد، پس از فراخوانی تابع اصلی enroll، تابع then آن سناریوها را اجرا می‌نماید.

۲.۲.۵.۳. کلاس Scenario

هر سناریو به صورت یک کلاس پیاده‌سازی می‌شود که از Scenario ارث می‌برد. این کلاس دارای توابع given، when و then است که معادل مفاهیم متناظر در توسعه‌ی مبتنی بر رفتار هستند.

۳.۵.۳. کد منبع

کد منبع چارچوب پیاده‌سازی شده از طریق آدرس زیر در دسترس است:

https://github.com/SeMeKh/bsc_project

فصل ۴

مطالعه‌ی موردی

در این فصل، گزارشی از استفاده از چارچوب پیاده‌سازی شده در قسمت ۵.۳، در یک سامانه‌ی کوچک ارائه می‌شود.

در بخش ۱.۴، این سامانه معرفی شده و در بخش ۲.۴، داستان کاربر ثبت‌نام به روش توسعه‌ی مبتنی بر رفتار و با استفاده از چارچوب اینسیتی توصیف و پیاده‌سازی شده است.

۱.۴. معرفی سامانه ثبت‌نام

برای بررسی صحت عملکرد اینسیتی، یک سامانه‌ی بسیار ساده را طراحی و در چارچوب جنگو پیاده‌سازی نمودیم تا بتوانیم در آن سامانه، آزمون‌هایی را در چارچوب اینسیتی تألیف نماییم.

جهت ملموس‌تر بودن موجودیت^۱ها برای خواننده، سامانه‌ی انتخاب واحد را انتخاب نمودیم و سعی نمودیم جزییاتی که مرتبط با توصیف توسعه‌ی مبتنی بر رفتار و آزمون آن نیستند را از سامانه حذف نماییم. در این سامانه تنها عملیات سی.آر.یو.دی.^۲ برای موجودیت‌ها و انتخاب واحد توسط دانشجو پیاده‌سازی شده است.

۱.۱.۴. موجودیت‌های سامانه

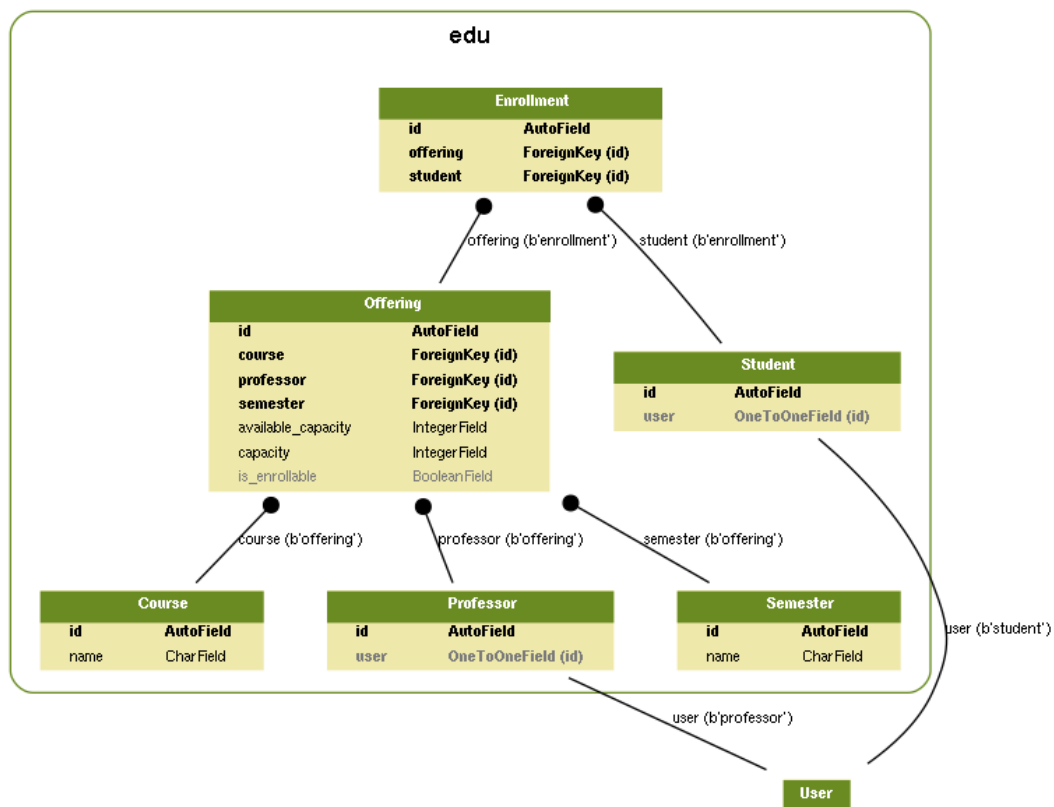
همانطور که در شکل ۱.۴ نشان داده شده، در حالت ساده این سامانه دارای موجودیت‌های زیر می‌باشد:

۱. کاربر^۳: این موجودیت اطلاعات کاربری یک نفر را در سامانه نگه می‌دارد. نام، نام خانوادگی و مشخصات احراز هویت از جمله صفات این موجودیت می‌باشند. این موجودیت توسط چارچوب جنگو ارائه می‌شود.

^۱Entity

^۲Create Read Update Delete

^۳User



شکل ۱.۴: نمودار موجودیت‌های سامانه‌ی ثبت نام آموزش

۲. دانشجو^۴ و استاد^۵: در این سامانه به اطلاعاتی جز مشخصات فردی و دسترسی‌ها برای دانشجو و استاد نیاز نداریم. این موجودیت‌ها ارتباط یک‌به‌یک با موجودیت کاربر دارند که مشخصات فردی و دسترسی‌ها در موجودیت کاربر ذخیره می‌شوند.

۳. نیمسال تحصیلی^۶ و درس^۷: برای این دو موجودیت، نگهداری صفت «نام» برای آن‌ها در سامانه کافیت.

۴. ارائه^۸: ارتباط چند به چند بین موجودیت‌های درس و نیمسال تحصیلی و استاد ارائه‌دهنده‌ی آن، در این موجودیت نگهداری می‌شود.

صفت capacity، ظرفیت ارائه را نگه می‌دارد. صفت available_capacity، یک صفت محاسبه‌پذیر است که برای کارایی بالاتر، در پایگاه داده ذخیره می‌شود. این صفت حاصل تفریق ظرفیت درس از تعداد ثبت‌نام‌های آن است. صفت is_enrollable فعال یا غیرفعال بودن قابلیت ثبت نام دانشجویان در آن ارائه را مشخص می‌نماید.

۵. ثبت نام^۹: این موجودیت ارتباط چند به چند ثبت نام میان موجودیت‌های دانشجو و ارائه را نگهداری می‌کند.

۲.۱.۴. رابط کاربری سامانه

برای شروع به کار سامانه، با اجرای دستور `runserver_plus /manage.py` در پوشه‌ی اصلی برنامه، کارگزار^{۱۰} شروع به کار نموده و می‌توان از طریق نشانی `http://localhost:8000/` به رابط کاربری سامانه دسترسی پیدا نمود.

ابتدا نام کاربری و کلمه عبور را وارد می‌نماییم (شکل ۲.۴). سپس بسته به نقشی که کاربر در سامانه داشته باشد، وارد یکی از صفحه‌های کارمند، دانشجو یا استاد خواهیم شد.

در صفحه‌ی کارمند (شکل ۳.۴)، می‌توان از منوی سمت چپ، هر کدام از گزینه‌های کاربر، دانشجو، استاد، درس، نیمسال تحصیلی و ارائه را انتخاب نمود؛ که در تصویر ۳.۴، گزینه‌ی ارائه انتخاب شده است.

با انتخاب هر گزینه، لیست موارد ثبت شده در سامانه برای آن گزینه قابل نمایش است. دو ستون ویرایش^{۱۱} و حذف^{۱۲} در انتهای هر ردیف از لیست هر گزینه قرار دارد که می‌توان آن ردیف را ویرایش یا حذف نمود.

^۴ Student

^۵ Professor

^۶ Semester

^۷ Course

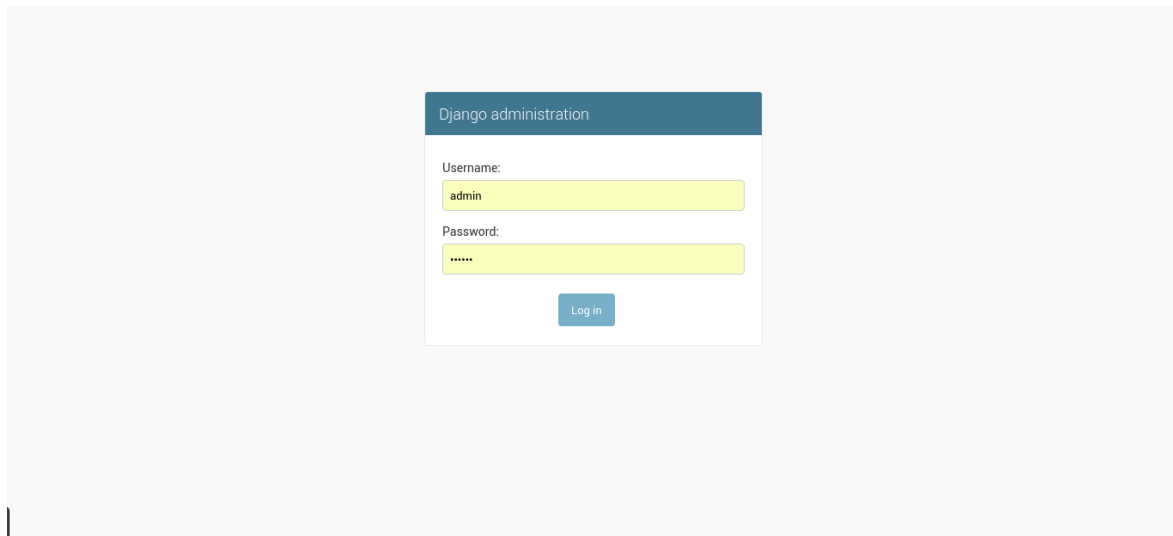
^۸ Offering

^۹ Enrollment

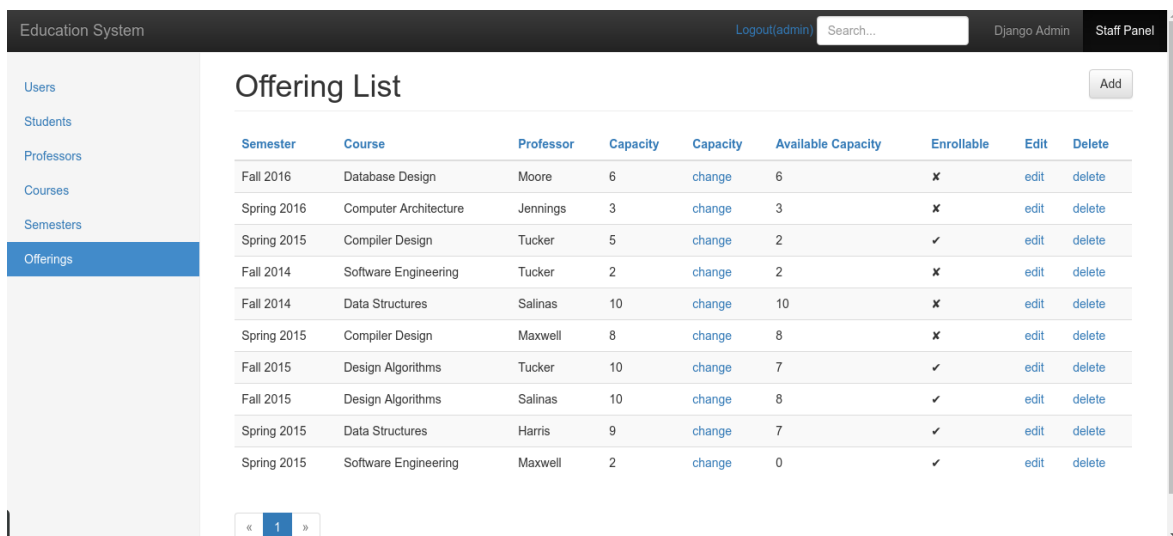
^{۱۰} Server

^{۱۱} Edit

^{۱۲} Delete



شکل ۲.۴: صفحه‌ی ورود سامانه‌ی ثبت نام



Semester	Course	Professor	Capacity	Capacity	Available Capacity	Enrollable	Edit	Delete
Fall 2016	Database Design	Moore	6	change	6	✗	edit	delete
Spring 2016	Computer Architecture	Jennings	3	change	3	✗	edit	delete
Spring 2015	Compiler Design	Tucker	5	change	2	✓	edit	delete
Fall 2014	Software Engineering	Tucker	2	change	2	✗	edit	delete
Fall 2014	Data Structures	Salinas	10	change	10	✗	edit	delete
Spring 2015	Compiler Design	Maxwell	8	change	8	✗	edit	delete
Fall 2015	Design Algorithms	Tucker	10	change	7	✓	edit	delete
Fall 2015	Design Algorithms	Salinas	10	change	8	✓	edit	delete
Spring 2015	Data Structures	Harris	9	change	7	✓	edit	delete
Spring 2015	Software Engineering	Maxwell	2	change	0	✓	edit	delete

شکل ۳.۴: صفحه‌ی کاربری کارمند در سامانه‌ی ثبت نام

شکل ۴.۴: صفحه‌ی ویرایش مشخصات ارائه توسط کارمند

برای مثال، در شکل ۴.۴، صفحه‌ی مربوط به ویرایش مشخصات یک ارائه نمایش داده شده است. علاوه بر عملیات ویرایش و حذف، ظرفیت ارائه توسط کارمند قابل تغییر است. برای این کار، با کلیک بر روی کلید تغییر در ستون ظرفیت^{۱۳} از جدول، می‌تواند ظرفیت آن درس را تغییر دهد. (شکل ۵.۴) همچنین اگر کاربر دسترسی مدیر سیستم^{۱۴} نیز داشته باشد، می‌تواند با استفاده از منوی بالا-راست به قسمت مدیریت سیستم جنگو^{۱۵} نیز دسترسی پیدا کند و به تمام موجودیت‌های سامانه دسترسی پیدا کند. برای مثال، می‌تواند از طریق تنظیمات کاربران (شکل ۶.۴) بدون نیاز به اطلاع از کلمه‌ی عبور سایر کاربرها، از طرف آن‌ها وارد سامانه بشود و صفحه‌های قابل مشاهده توسط آن‌ها را ببیند. نوار زردرنگی که در بالای برخی صفحات در شکل‌های بعد مشاهده می‌شود، به این خاطر است.

در صورتی که کاربر وارد شده به سامانه نقش استاد داشته باشد، می‌تواند لیست درس‌هایی که توسط وی ارائه شده است را مشاهده کند. امکان حذف یا اضافه‌ی دروس ارائه شده‌ی هر استاد، برای نقش کارمند فعال می‌باشد.

همچنین اگر دانشجو وارد سامانه شود، می‌تواند لیست درس‌هایی که در آن‌ها ثبت‌نام کرده را مشاهده نماید. (شکل ۸.۴).

برای ثبت‌نام در یک ارائه از یک درس، دانشجو می‌تواند بر روی کلید در سمت بالا-راست صفحه‌ی شکل ۸.۴ کلیک نماید. در این صورت به صفحه‌ی ثبت‌نام که در شکل ۹.۴ نمایش داده شده، منتقل می‌شود. در

^{۱۳}Capacity

^{۱۴}Superuser

^{۱۵}Django Administration

The screenshot shows a web application titled "Education System". On the left is a sidebar menu with links: Users, Students, Professors, Courses (highlighted), Semesters, and Offerings. The main content area has a header with "Logout(admin)", a search bar, and links for "Django Admin" and "Staff Panel". Below the header, there is a form titled "Capacity" with a text input field containing the number "5". At the bottom of the form are two buttons: "Submit" and "No, take me back".

شکل ۵.۴: تغییر ظرفیت درس توسط کارمند

The screenshot shows the Django administration interface for the "Authentication and Authorization" app, specifically the "Users" section. The page title is "Select user to change". There is a search bar at the top. Below it, there is an "Action:" dropdown menu and a "Go" button. The main part of the page is a table listing users. To the right of the table is a "FILTER" sidebar with two sections: "By staff status" and "By superuser status".

	USERNAME	EMAIL ADDRESS	FIRST NAME	LAST NAME	STUDENT	PROFESSOR	HIJACK USER
<input type="checkbox"/>	admin	admin@admin.com			-	-	Hijack admin
<input type="checkbox"/>	user0		Daniel	Maddox	Student413	-	Hijack user0
<input type="checkbox"/>	user1		Charles	Sellers	Student414	-	Hijack user1
<input type="checkbox"/>	user10		Ashley	Tucker	-	Ashley Tucker	Hijack user10
<input type="checkbox"/>	user11		Jason	Murphy	-	Jason Murphy	Hijack user11
<input type="checkbox"/>	user12		Erika	Garcia	-	Erika Garcia	Hijack user12
<input type="checkbox"/>	user13		Monica	Griffin	-	Monica Griffin	Hijack user13
<input type="checkbox"/>	user14		Valerie	Salinas	-	Valerie Salinas	Hijack user14
<input type="checkbox"/>	user15		Alison	Moore	-	Alison Moore	Hijack user15
<input type="checkbox"/>	user16		Brandon	Mitchell	-	Brandon Mitchell	Hijack user16
<input type="checkbox"/>	user17		Isa	Barie	-	Isa Barie	Hijack user17

شکل ۶.۴: دسترسی مدیر سامانه به صفحه‌های کاربرها

Education System Logout(user10) Search... Professor Panel

My Offerings

Offering List

Semester	Course
Fall 2016	Database Design
Spring 2016	Computer Architecture
Spring 2015	Compiler Design
Fall 2014	Software Engineering
Fall 2014	Data Structures
Spring 2015	Compiler Design
Fall 2015	Design Algorithms
Fall 2015	Design Algorithms
Spring 2015	Data Structures
Spring 2015	Software Engineering

شکل ۷.۴: صفحه‌ی کاربری استاد در سامانه‌ی ثبت نام

Education System Logout(user1) Search... Student Panel

My Courses

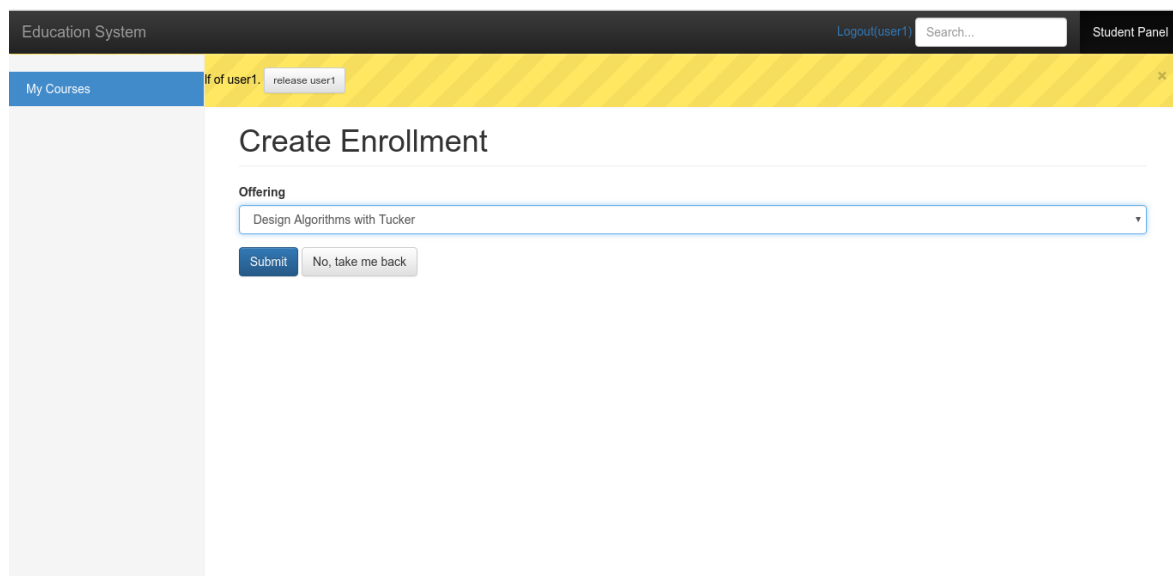
Enrollment List

Semester	Course	Professor
Fall 2015	Design Algorithms	Tucker
Spring 2015	Compiler Design	Tucker

Add

« 1 »

شکل ۸.۴: صفحه‌ی کاربری دانشجو در سامانه‌ی ثبت نام



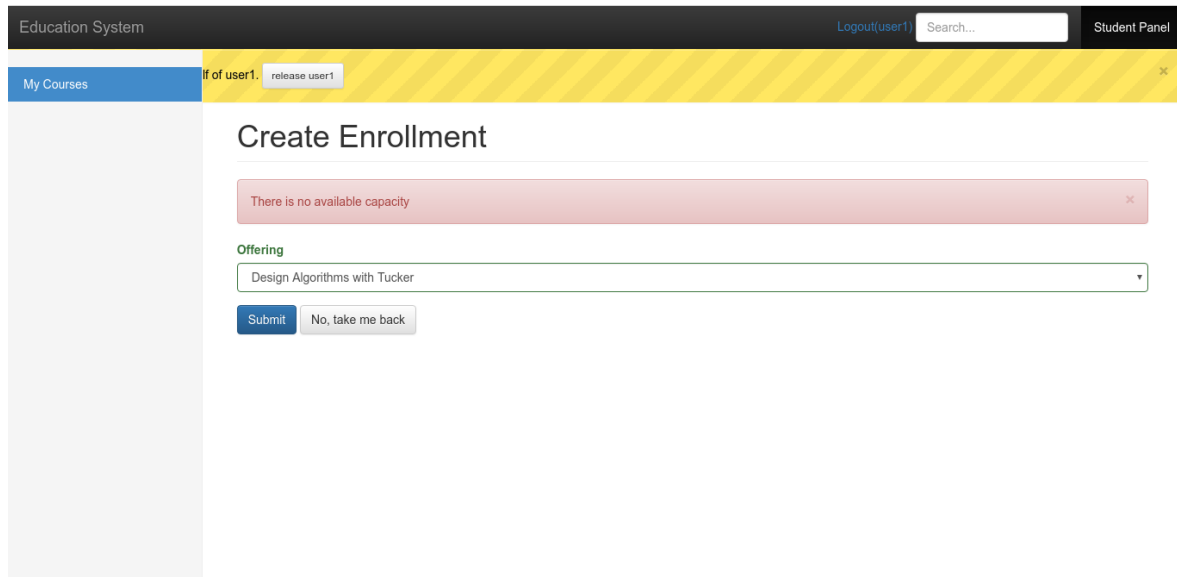
شکل ۹.۴: صفحه‌ی ثبت‌نام دانشجو در درس‌های ارائه‌شده توسط سامانه

این صفحه، لیستی از ارائه‌هایی که صفت `is_enrollable` آن‌ها فعال باشد، در اختیار دانشجو برای انتخاب قرار می‌گیرد.

پس از تأیید فرم ثبت‌نام، در صورتی که ظرفیت درس تکمیل شده باشد، با خطای نشان‌داده‌شده در شکل ۱۰.۴ مواجه می‌شود و دانشجو می‌تواند درسی دیگر را برای ثبت‌نام انتخاب نماید. در غیر این صورت، ثبت‌نام انجام شده و ردیف مربوط به درس جدید در لیست درس‌های دانشجو مشاهده می‌شود.

۲.۴. تألیف سناریو برای سامانه

جهت تألیف آزمون، ابتدا داستان‌های کاربری مورد نظر نوشته شده و از آن‌ها سناریوهایی استخراج می‌شود. نتیجه‌ی این کار برای دو سناریوی نمونه در زیر آمده:



شکل ۱۰.۴: خطای پر بودن ظرفیت هنگام ثبت نام دانشجو در سامانه

Story: Enrollment of student in offering

As a student

I want to enroll in an offering

So that I am allowed to participate in an offering's classes

Scenario 1: Capacity should decrease by enroll

Given student s1, offering o1 with available capacity c0

When s1 commits enrollment in o1

Then available capacity of o1 should become c0-1

Scenario 2: Enrollment should fail for offering with zero capacity

Given offering o1 with zero capacity

When someone enrolls in it

Then it should fail with error

```

class CapacityDecreasesByEnroll(Scenario):
    """
    Scenario: Capacity should decrease by enroll
    Given student s1, offering o1 with available capacity c0
    When s1 commits enrollment in o1
    Then available capacity of o1 should become c0-1
    """
    def given(scenario, self, student, **payload):
        scenario.s1 = student
        scenario.o1 = self
        scenario.c0 = self.available_capacity
        return True

    when = 'edu.models.Offering.enroll'

    def when_params(scenario, commit, **payload):
        return commit == True

    def then(scenario, payload, return_value, exc_type, **kwargs):
        assert scenario.o1.available_capacity == scenario.c0 - 1

class EnrollmentShouldFailForOfferingWithZeroCapacity(Scenario):
    """
    Scenario: Enrollment should fail for offering with zero capacity
    Given offering o1 with zero capacity
    When someone enrolls in it
    Then it should fail with error
    """
    def given(scenario, self, **payload):
        return self.available_capacity == 0

    when = 'edu.models.Offering.enroll'

    def then(scenario, exc_type, **kwargs):
        assert issubclass(exc_type, EnrollmentError)

```

شکل ۱۱.۴: کد معادل سناریو

گام بعدی انتقال سناریوها از زبان اگر-وقتی-آنگاه^{۱۶} به نرم‌افزار است. باید توجه کرد که نباید در این مسیر از کلیت سناریوها کاسته شود، که این موضوع در چارچوب پیشنهادی، با توجه به جدا شدن لایه‌ی سناریو از لایه‌ی تعامل‌گر به سادگی محقق می‌شود و تقریباً ترجمه‌ی عبارات به زبان برنامه‌نویسی مقصد کفایت می‌کند. نتیجه‌ی انجام این کار برای دو سناریوی مذکور در شکل‌های ۱۱.۴ آمده.

^{۱۶}Given-When-Then

فصل ۵

جمع‌بندی و کارهای آتی

۱.۵. جمع‌بندی

در طول این پروژه مجموعاً فعالیت‌های زیر صورت گرفت:

۱. بررسی کارهای پیشین در زمینه‌ی توسعه‌ی آزمون
 ۲. ارائه‌ی چارچوب آزمون پیشنهادی با ایده گرفتن از نقاط قوت و ضعف مشهود در روش‌های پیشین
 ۳. پیاده‌سازی یک نرم‌افزار نمونه و تألیف آزمون برای آن توسط چارچوب پیشنهادی
- نهایتاً برآورد مؤلفان این بود که استفاده از چارچوب پیشنهادی، همان‌طور که انتظار می‌رفت، کمک شایانی به ساده‌سازی انجام آزمون نرم‌افزار نمونه کرد. انتظار می‌رود که استفاده از این معماری در آزمون نرم‌افزارهای بزرگتر، مؤثرتر نیز واقع شود.

۲.۵. کارهای آتی

در طول انجام پروژه، ایده‌های فراوانی مرتبط با ایده‌ی چارچوب پیشنهادی به ذهنمان رسید که در راستای خارج نشدن از حوزه‌ی این پروژه وارد آن‌ها نشدیم، اما بررسی آن‌ها خالی از لطف نخواهد بود:

- حذف و هرس سناریوها پس از بررسی صحت آن‌ها به دفعات کافی
- تعریف متریک‌های جدید سنجش کیفیت نرم‌افزار بر حسب سناریوها
- استفاده از چارچوب آزمون پیاده‌شده در یک پروژه‌ی متن‌باز و بزرگ‌تر، جهت ارزیابی دقیق‌تر این روش

- پیاده‌سازی تعامل‌گر به شیوه‌های مختلف و بررسی میزان اثرگذاری آنها

پیوست‌ها

پیوست آ

پیاده‌سازی بخش جمع‌آوری سناریوها در اینسنیتی

```
import logging
import importlib
import inspect
from collections import defaultdict
from django.apps import AppConfig, apps
from django.conf import settings

from insanity.scenario import Scenario

logger = logging.getLogger('insanity')

all_scenarios = defaultdict(list)
all_stats = dict()

STATS_FAIL = 'fail'
STATS_COUNT = 'count'
STATS_PASS = 'pass'

from raven import Client

client = Client(settings.SENTRY_DSN)

def fqn(cls):
    return '%s.%s' % (cls.__module__, cls.__name__)

def report_exec(scenario, fail):
    name = fqn(scenario.__class__)
```

```

stats = all_stats[name]
if fail:
    client.captureException(tags={
        'scenario': name,
    })
    stats[STATS_FAIL] += 1
else:
    stats[STATS_PASS] += 1
stats[STATS_COUNT] += 1

class InsanityConfig(AppConfig):
    name = 'insanity'

    def ready(self):
        logging.info("Harvesting")
        for _, app in apps.app_configs.items():
            module = app.module
            scenario_name = module.__name__ + '.scenarios'
            try:
                scenarios = importlib.import_module(scenario_name)

                for name, obj in inspect.getmembers(scenarios, inspect.isclass):
                    if obj.__module__ == scenario_name and issubclass(obj, Scenario):
                        _name = fqn(obj)
                        all_stats[_name] = {
                            STATS_FAIL: 0,
                            STATS_COUNT: 0,
                            STATS_PASS: 0,
                        }
                        all_scenarios[obj.when].append(obj)
            except:
                pass

```


پیوست ب

پیاده‌سازی دکوراتورهای مربوط به تشخیص و اجرای سناریوها در اینسنیتی

```
import functools
import inspect
from insanity.apps import all_scenarios, report_exec

import logging

logger = logging.getLogger(__name__)

class Action(object):
    return_value = None
    exc_type = None
    exc_val = None
    exc_tb = None

    def __init__(self, name, payload):
        self.name = name
        self.payload = payload
        self._scenarios = []

    def __getitem__(self, item):
        return self.payload[item]

    def _collect_scenarios(self):
        for scenario_class in all_scenarios[self.name]:
            scenario = scenario_class(self)
            if scenario.when_params(**self.payload) and scenario.given(**self.payload):
                self._scenarios.append(scenario)
            print('~~~ Scenario %s detected ~~~' % scenario_class.__name__)
```

```

def _assert_scenarios(self):
    for scenario in self._scenarios:
        try:
            scenario.then(exc_type=self.exc_type, exc_val=self.exc_val, exc_tb=self.exc_tb,
                          return_value=self.return_value, payload=self.payload)
            print('~~~ Scenario %s succeeded ~~~' % scenario.__class__.__name__)
            report_exec(scenario, fail=False)
        except:
            print('~~~ Scenario %s failed ~~~' % scenario.__class__.__name__)
            report_exec(scenario, fail=True)

class action(object):
    """
    ContextManager and Decorator for creating Action instances
    """

    def __init__(__insanity_self__, __insanity_when__=None, **payload):
        __insanity_self__.name = __insanity_when__
        __insanity_self__.payload = payload
        __insanity_self__.stack = []
        __insanity_self__._self_name = None

    def __enter__(self):
        if self.name is None:
            raise ValueError('Action should be decorator or have explicit name')
        action_instance = Action(self.name, self._payload)
        action_instance._collect_scenarios()
        self._stack.append(action_instance)
        return action_instance

    def __exit__(self, exc_type, exc_val, exc_tb):
        action_instance = self._stack.pop()
        action_instance.exc_type = exc_type
        action_instance.exc_val = exc_val
        action_instance.exc_tb = exc_tb
        action_instance._assert_scenarios()
        return False # Do not suppress exception

    def rename_self(self, name):
        self._self_name = name
        return self

    def _build_context_manager(self, signature, name, args, kwargs):
        bound_signature = signature.bind(*args, **kwargs)
        bound_signature.apply_defaults()

```

```
payload = dict(**self._payload, **bound_signature.arguments)
return action(name, **payload)

def _get_signature(self, func):
    s = inspect.signature(func)
    return s

def __call__(self, func):
    name = self.name if self.name is not None else '%s.%s' % (func.__module__, func.__name__)
    signature = self._get_signature(func)

    @functools.wraps(func)
    def inner(*args, **kwargs):
        with self._build_context_manager(signature, name, args, kwargs) as action_instance:
            return_value = func(*args, **kwargs)
            action_instance.return_value = return_value
        return return_value

    return inner
```

پیوست پ

متن برنامه‌ی منطق تجاری سامانه‌ی آموزش

```
from django.db import models
from django.apps import apps

from insanity.action import action

class Course(models.Model):
    name = models.CharField(max_length=32)

    def __str__(self):
        return self.name

class Student(models.Model):
    user = models.OneToOneField('auth.User', blank=True, null=True)

    def __str__(self):
        return 'Student%d' % (self.id or -1)

class Professor(models.Model):
    user = models.OneToOneField('auth.User', blank=True, null=True)

    def __str__(self):
        return self.user.get_full_name()

class EnrollmentError(Exception):
    pass

class ChangeCapacityError(Exception):
    pass
```

```

class Offering(models.Model):
    course = models.ForeignKey('edu.Course')
    semester = models.ForeignKey('edu.Semester')
    professor = models.ForeignKey('edu.Professor')
    capacity = models.IntegerField()
    available_capacity = models.IntegerField()
    is_enrollable = models.BooleanField(default=False)

    @action()
    def enroll(self, student, commit=True):
        if self.available_capacity == 0:
            raise EnrollmentError("There is no available capacity")
        if not self.is_enrollable:
            raise EnrollmentError("This offering is not enabled for enrollment")

        Enrollment = apps.get_model('edu.Enrollment')
        enrollment = Enrollment(offering=self, student=student)
        if commit:
            enrollment.save()
            self.available_capacity = self.available_capacity - 1
            self.save()
        return enrollment

    def get_students(self):
        return Student.objects.filter(id__in=self.enrollment_set.values_list('student', flat=True))

    def change_capacity(self, new_capacity, commit=True):
        enrollment_count = self.capacity - self.available_capacity
        if new_capacity < enrollment_count:
            raise ChangeCapacityError(
                'There are already %d enrollments which is less than %d' % (enrollment_count, new_capacity)
            )
        self.capacity = new_capacity
        self.available_capacity = new_capacity - enrollment_count
        if commit:
            self.save()

class Enrollment(models.Model):
    offering = models.ForeignKey('edu.Offering')
    student = models.ForeignKey('edu.Student')

class Semester(models.Model):
    name = models.CharField(max_length=16)

```

```
def __str__(self):  
    return self.name
```

پیوست

متن برنامه‌ی تعامل‌گر سامانه‌ی ثبت‌نام

```
from django.contrib.auth.models import User
from django.contrib.auth import get_user_model

from edu.models import Course, Student, Semester, Professor, Offering, EnrollmentError
import random
import factory
from factory import fuzzy

class UserFactory(factory.Factory):
    class Meta:
        model = User

    first_name = factory.Faker('first_name')
    last_name = factory.Faker('last_name')
    username = factory.Sequence(lambda n: 'user%d' % n)
    password = factory.Sequence(lambda n: 'pass%d' % n)

    @classmethod
    def _create(cls, model_class, *args, **kwargs):
        """Override the default ``_create`` with our custom call."""
        manager = model_class.objects
        # The default would use ``manager.create(*args, **kwargs)``
        return manager.create_user(*args, **kwargs)

class StudentFactory(factory.Factory):
    class Meta:
        model = Student

    user = factory.SubFactory(UserFactory)

class ProfessorFactory(factory.Factory):
```

```

class Meta:
    model = Professor

user = factory.SubFactory(UserFactory)

class CourseFactory(factory.Factory):
    class Meta:
        model = Course

    name = fuzzy.FuzzyText(length=8, prefix='Crs_')

class SemesterFactory(factory.Factory):
    class Meta:
        model = Semester

    name = fuzzy.FuzzyText(length=4, prefix='Sms_')

class OfferingFactory(factory.Factory):
    class Meta:
        model = Offering

    course = fuzzy.FuzzyChoice(Course.objects.all())
    semester = fuzzy.FuzzyChoice(Semester.objects.all())
    professor = fuzzy.FuzzyChoice(Professor.objects.all())
    capacity = fuzzy.FuzzyInteger(0, 10)
    available_capacity = factory.LazyAttribute(lambda self: self.capacity)
    is_enrollable = fuzzy.FuzzyChoice([True, False])

def run():
    User = get_user_model()
    User.objects.all().delete()
    User.objects.create_superuser('admin', 'admin@admin.com', '321321')

    Course.objects.all().delete()
    courses = CourseFactory.create_batch(10)
    for c in courses:
        c.save()

    Student.objects.all().delete()
    students = StudentFactory.create_batch(10)
    for s in students:
        s.user.save()
        s.save()

```



```
Semester.objects.all().delete()
semesters = SemesterFactory.create_batch(10)
for s in semesters:
    s.save()

Professor.objects.all().delete()
professors = ProfessorFactory.create_batch(10)
for p in professors:
    p.user.save()
    p.save()

offerings = OfferingFactory.create_batch(20)
for o in offerings:
    o.save()

for i in range(10):
    offering = random.choice(list(Offering.objects.filter(available_capacity__gt=0)))
    student = random.choice(students)
    offering.enroll(student)

try:
    offering = random.choice(list(Offering.objects.filter(available_capacity=0)))
    student = random.choice(students)
    offering.enroll(student, commit=False)
except EnrollmentError:
    "That's OK"
    pass
```

کتابنامه

- [1] C. Andres and K. Beck, "Extreme programming explained: Embrace change," *Reading: Addison-Wesley Professional*, 2004.
- [2] G. Adzik, *Specification by example*. Manning Publications Co., 2011.
- [3] (2015). Projects using cucumber, [Online]. Available: <https://github.com/cucumber/cucumber/wiki/Projects-Using-Cucumber> (visited on 08/10/2016).
- [4] G. T. Laycock, "The theory and practice of specification based software testing," PhD thesis, Citeseer, 1993.
- [5] C. L. Baker, "Review of dd mc-cracken, digital computer programming," *Math. Comput*, vol. 11, no. 60, pp. 298–305, 1957.
- [6] J. Meerts. (2015). The history of software testing, [Online]. Available: <http://www.testingreferences.com/testinghistory.php> (visited on 08/10/2016).
- [7] W. R. Adrion, M. A. Branstad, and J. C. Cherniavsky, "Validation, verification, and testing of computer software," *ACM Computing Surveys (CSUR)*, vol. 14, no. 2, pp. 159–192, 1982.
- [8] L. Luo, "Software testing techniques," *Institute for software research international Carnegie mellon university Pittsburgh, PA*, vol. 15232, no. 1-19, p. 19, 2001.
- [9] J. D. Musa, A. Iannino, and K. Okumoto, *Software reliability: Measurement, prediction, application*. McGraw-Hill, Inc., 1987.
- [10] B. Beizer, "Software testing techniques," *New York, ISBN: 0-442-20672-0*, 1990.
- [11] J. Rasmusson. (2016). Agile vs waterfall, [Online]. Available: http://www.agilenutshell.com/agile_vs_waterfall (visited on 08/10/2016).
- [12] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, *et al.* (2001). Manifesto for agile software development, [Online]. Available: <http://agilemanifesto.org/> (visited on 08/10/2016).
- [13] A. Ghahrai. (2016). 12 principles of agile testing, [Online]. Available: <https://www.linkedin.com/pulse/12-principles-agile-testing-amir-ghahrai?trk=prof-post> (visited on 08/10/2016).
- [14] K. Beck, *Test-driven development: By example*. Addison-Wesley Professional, 2003.
- [15] S. W. Ambler. (2015). Agile best practice: Executable specifications, [Online]. Available: <http://agilemodeling.com/essays/executableSpecifications.htm> (visited on 08/10/2016).
- [16] M. Gärtner, *Atdd by example: A practical guide to acceptance test-driven development*. Addison-Wesley, 2012.

- [17] Y. Singh, *Software testing*. Cambridge University Press, 2011.
- [18] (2015). Unit testing, [Online]. Available: <https://www.agilealliance.org/glossary/unit-test/> (visited on 08/10/2016).
- [19] M. Fowler. (2012). Test pyramid, [Online]. Available: <http://martinfowler.com/bliki/TestPyramid.html> (visited on 08/10/2016).
- [20] (2003). Unit test, [Online]. Available: [https://msdn.microsoft.com/en-us/library/aa292197\(v=vs.71\).aspx](https://msdn.microsoft.com/en-us/library/aa292197(v=vs.71).aspx) (visited on 08/10/2016).
- [21] —, (2014). Unit test, [Online]. Available: <http://martinfowler.com/bliki/UnitTest.html> (visited on 08/10/2016).
- [22] M. Cohn, *Succeeding with agile: Software development using scrum*. Pearson Education, 2010.
- [23] R. Ramsin, *The engineering of an object-oriented software development methodology*. University of York, 2006.
- [24] S. W. Ambler. (2013). Introduction to test driven development (tdd), [Online]. Available: <http://agiledata.org/essays/tdd.html> (visited on 08/10/2016).
- [25] D. North *et al.*, “Introducing bdd,” *Better Software*, March, 2006.
- [26] D. North. (2012). Bdd is like tdd if, [Online]. Available: <https://dannorth.net/2012/05/31/bdd-is-like-tdd-if/> (visited on 08/10/2016).
- [27] (2015). Bdd, [Online]. Available: <https://www.agilealliance.org/glossary/bdd/> (visited on 08/10/2016).
- [28] C. Solis and X. Wang, “A study of the characteristics of behaviour driven development,” pp. 383–387, 2011.
- [29] A. Hunsberger. (2016). A two-minute bdd overview, [Online]. Available: <http://saucio.com/index.php/2016/03/a-two-minute-bdd-overview/> (visited on 08/10/2016).
- [30] L. Kuzynski. (2012). Bdd in php, [Online]. Available: <http://www.slideshare.net/wookieb/bdd-11756856> (visited on 08/10/2016).

واژه‌نامه‌ی فارسی به انگلیسی

آزمون..... Test	بیانیه‌ی توسعه‌ی چابک نرم افزار .. Manifesto for
آزمون پذیرش	Acceptance Test Agile Software Development
آزمون سامانه	System Test پساً-شرط
آزمون نرم‌افزار	Software Test پیاده‌سازی
آزمون واحد	Unit Test پیش-شرط
آزمون یکپارچه‌سازی	Integration Test تابع
آنگاه	Then تجربه
انتکاپذیری	Reliability تحلیل
اجرایی	Executable تعامل‌گر
ارزش تجاری	Business Value تفکیک
اس.یو.تی.	System Under Test تکرار
اعتبارسنجی	Validation تکراری
اکس.پی.	Extreme Programming تکنیک
اگر	Given توسعه‌ی آزمون مبتنی بر پذیرش . Acceptance Test
انجام‌شده	Done Driven Development
اینسیتی	Insanity توسعه‌ی اول-آزمون ... Test First Development
بازبه‌کارگیری کد	Code Reuse توسعه‌ی چابک مبتنی بر مدل Agile Model Driven
باگ	Bug Development
باگ‌زادی	Debug توسعه‌ی چابک نرم‌افزار
برنامه‌نویسی دونفره	Pair Programming Development

Syntax	صرفی	Test Driven Development	توسعه‌ی مبتنی بر آزمون
Design	طراحی		
Convention	عرف	Behavior Driven Development	توسعه‌ی مبتنی بر رفتار
Refactoring	فاکتوربندی مجدد		
Software Development Process	فرآیند توسعه‌ی نرم‌افزار	Specification	توصیف
		Django	جنگو
Activity	فعالیت	Framework	چارچوب
Template	قالب	Given-When-Then	اگر-وقتی-آن‌گاه
Fail	قرمز	Business Domain	حوزه تجاری
Server	کارگزار	Domain Expert	خبره‌ی حوزه
Class	کلاس	User Story	داستان کاربر
Action	کنش	User Stories	داستان‌های کاربر
Software Quality	کیفیت نرم‌افزار	Decorator	دکوراتور
Module	ماژول	User Interface	رابط کاربری
Methodology	متدولوژی	Behavior	رفتار
Software Development Methodology	متدولوژی توسعه‌ی نرم‌افزار	Modeling Language	زبان مدل‌سازی
		Ubiquitous Language	زبان مشترک
Metric	متریک	Artifact	ساخته
Plain Text	متن ساده	System	سامانه
Test Suite	مجموعه‌ی آزمون	Success	سبز
Feature Set	مجموعه‌ی ویژگی	Scenario	سناریو
Scope	محدوده	Sentry	سنتری
Trigger	محرک	Create Read Update Delete	سی.آر.یو.دی
Production	محصول نهایی	Acceptance Criteria	شرط پذیرش

Business Outcome	نتیجه‌ی تجاری	Waterfall Model	مدل آبشاری
Software-intensive	نرم‌افزار-متمرکز	V-Shaped Software ...	مدل چرخه عمر V-شکل
Defect	نقص	Lifecycle Model	
Coupling	وابستگی	Decoupling	مستقل‌سازی
Verification	وارسی	Semantics	معنایی
When	وقتی	Business Logic	منطق تجاری
Feature	ویژگی	Entity	موجودیت
Test Pyramid	هرم آزمون	Test Case	مورد آزمون
Integration	یکپارچه‌سازی	Component	مؤلفه
Continuous Integration ..	یکپارچه‌سازی مداوم	Inconsistency	ناهمخوانی

واژه‌نامه‌ی انگلیسی به فارسی

Business Value	ارزش تجاری	Acceptance Criteria	شرط پذیرش
Class	کلاس	Acceptance Test	آزمون پذیرش
Code Reuse	بازیه‌کارگیری کد	Acceptance Test Driven Development	
Component	مؤلفه		توسعه‌ی آزمون مبتنی بر پذیرش
Continuous Integration ..	یکپارچه‌سازی مداوم	Action	کنش
Convention	عرف	Activity	فعالیت
Coupling	وابستگی	Actor	تعامل‌گر
Create Read Update Delete	سی.آر.یو.دی.	Agile Model Driven Development ..	توسعه‌ی ..
Debug	باگ‌زادی		چابک مبتنی بر مدل
Decomposition	تفکیک	Agile Software Development .	توسعه‌ی چابک .
Decorator	دکوراتور		نرم‌افزار
Decoupling	مستقل‌سازی	Analysis	تحلیل
Defect	نقص	Artifact	ساخته
Design	طراحی	Behavior	رفتار
Django	جنگو	Behavior Driven Development .	توسعه‌ی مبتنی .
Domain Expert	خبره‌ی حوزه		بر رفتار
Done	انجام‌شده	Bug	باگ
Entity	موجودیت	Business Domain	حوزه تجاری
Executable	اجرایی	Business Logic	منطق تجاری
Extreme Programming	اکس.پی.	Business Outcome	نتیجه‌ی تجاری

Pre-condition	پیش-شرط	Fail	قمرز
Production	محصول نهایی	Feature	ویژگی
Refactoring	فاکتوربندی مجدد	Feature Set	مجموعه‌ی ویژگی
Reliability	اتکاپذیری	Framework	چارچوب
Scenario	سناریو	Given	اگر
Scope	محدوده	Given-When-Then	اگر-وقتی-آنگاه
Semantics	معنایی	Implementation	پیاپی‌سازی
Sentry	سنتری	Inconsistency	ناهمخوانی
Server	کارگزار	Insanity	اینسیتی
Software Development Methodology		Integration	یکپارچه‌سازی
	متدولوژی توسعه‌ی نرم افزار	Integration Test	آزمون یکپارچه‌سازی
Software Development Process	فرآیند توسعه‌ی	Iteration	تکرار
	نرم افزار	Iterative	تکراری
Software Quality	کیفیت نرم افزار	Manifesto for Agile Software Development	
Software Test	آزمون نرم افزار		بیانیه‌ی توسعه‌ی چابک نرم افزار
Software-intensive	نرم افزار-متمركز	Method	تابع
Specification	توصیف	Methodology	متدولوژی
Success	سبز	Metric	متریک
Syntax	صرفی	Modeling Language	زبان مدل‌سازی
System	سامانه	Module	ماژول
System Test	آزمون سامانه	Pair Programming	برنامه‌نویسی دونفره
System Under Test	اس.یو.تی.	Plain Text	متن ساده
Technique	تکنیک	Post-condition	پسا-شرط
Template	قالب	Practice	تجربه

Unit Test	آزمون واحد	Test.....	آزمون
User Interface.....	رابط کاربری	Test Case	مورد آزمون
User Stories	داستان‌های کاربر	Test Driven Development ...	توسعه‌ی مبتنی بر
User Story	داستان کاربر		آزمون
Validation	اعتبارسنجی	Test First Development ...	توسعه‌ی اول-آزمون
Verification	وارسی	Test Pyramid.....	هرم آزمون
V-Shaped Software Lifecycle Model....	مدل	Test Suite	مجموعه‌ی آزمون
	چرخه عمر V-شکل	Then	آن‌گاه
Waterfall Model.....	مدل آبشاری	Trigger.....	محرک
When.....	وقتی	Ubiquitous Language	زبان مشترک

Decoupling Scenarios from Behavior-Driven Tests

Abstract

Testing is one of the main means of achieving higher quality of software. Behavior-Driven Development is an industry-wide popular techniques in effectively integrating testing with development process.

In this article, we will first examine BDD and other similar techniques in software testing.

Next, we will propose and implement a BDD testing framework, designed based on the idea of decoupling scenarios from tests. This framework should supposedly help the software developer in writing higher quality tests, as well as decreasing the maintenance cost of such test suites.

And at last, we will implement a sample system, and use our testing framework toward writing tests for our sample application.

Keywords: Behavior-Driven Development, Decoupling, Software Test Quality, Test Framework.



Sharif University of Technology
Computer Engineering Department

B. Sc. Thesis
Computer Software Engineering

Decoupling Scenarios from Behavior-Driven Tests

By:

Seyed Mehran Kholdi, Mohammad Hossein Sekhavat

Supervisor:

Dr. Seyed Hassan Mirian Hosseinabadi

August 2016

