



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

[این صفحه آگاهانه خالی گذاشته شده است.]

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.

[این صفحه آگاهانه خالی گذاشته شده است.]

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.

^۱ Behavior Driven Development

^۲ Technique

^۳ Agile Software Development

^۴ Framework

^۵ Test

^۶ Decoupling

[این صفحه آگاهانه خالی گذاشته شده است.]

سرخ‌طها

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

^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
۲۷	۳ طراحی و توسعه‌ی چارچوب
۲۷	۱.۳ معماری چارچوب
۲۸	۲.۳ فواید
۲۸	۱.۲.۳ کاهش حجم آزمون‌ها
۲۹	۲.۲.۳ کاهش هزینه‌ی نگهداری
۲۹	۳.۲.۳ افزایش احتمال یافتن خطا
۲۹	۳.۳ در سطح سیستمی
۳۰	۴.۳ خارج از محیط آزمون
۳۰	۵.۳ پیاده‌سازی بر مبنای چارچوب جنگو ^{۱۷}
۳۱	۱.۵.۳ نحوه‌ی استفاده
۳۱	۲.۵.۳ اجزاء پیاده‌سازی
۳۱	۳.۵.۳ کد منبع
۳۳	۴ پیاده‌سازی
۳۳	۱.۴ تألیف آزمون برای سامانه‌ی آموزش
۳۴	۱.۱.۴ موجودیت‌های سامانه
۳۵	۲.۱.۴ تألیف سناریو برای سامانه
۳۷	۳.۱.۴ رابط کاربری ^{۱۸} سامانه
۴۵	۵ جمع‌بندی و کارهای آتی
۴۵	۱.۵ جمع‌بندی
۴۵	۲.۵ کارهای آتی
۴۷	کتاب‌نامه

^{۱۴}Template

^{۱۵}User Stories

^{۱۶}Scenario

^{۱۷}Django

^{۱۸}User Interface

[این صفحه آگاهانه خالی گذاشته شده است.]

فهرست شکل‌ها

۱.۲	نمودار سطوحی آزمون در مدل چرخه عمر V- شکل ^{۱۹}	۹
۲.۲	نمودار سطوح آزمون در هرم آزمون	۱۰
۳.۲	رابطه‌ی میان آزمون واحد ^{۲۰} و واحدهای برنامه	۱۱
۴.۲	نمودار فعالیت مراحل توسعه‌ی اول-آزمون	۱۳
۵.۲	نمودار حالت وضعیت‌های مجموعه‌ی آزمون و نحوه‌ی گذار بین آنها	۱۴
۶.۲	نمودار فعالیت مراحل توسعه‌ی آزمون مبتنی بر پذیرش	۱۶
۷.۲	مدل مفهومی در تحلیل توسعه‌ی مبتنی بر رفتار	۱۹
۸.۲	فرآیند توسعه‌ی ویژگی ^{۲۱} ها در توسعه‌ی مبتنی بر رفتار	۲۰
۱.۴	نمودار موجودیت‌های سامانه‌ی ثبت نام آموزش	۳۴
۲.۴	کد معادل سناریو	۳۷
۳.۴	صفحه‌ی ورود سامانه‌ی ثبت نام	۳۸
۴.۴	صفحه‌ی کاربری کارمند در سامانه‌ی ثبت نام	۳۸
۵.۴	صفحه‌ی ویرایش مشخصات ارائه توسط کارمند	۳۹
۶.۴	تغییر ظرفیت درس توسط کارمند	۴۰
۷.۴	دسترسی مدیر سامانه به صفحه‌های کاربرها	۴۰
۸.۴	صفحه‌ی کاربری استاد در سامانه‌ی ثبت نام	۴۱
۹.۴	صفحه‌ی کاربری دانشجو در سامانه‌ی ثبت نام	۴۲
۱۰.۴	صفحه‌ی ثبت نام دانشجو در درس‌های ارائه شده توسط سامانه	۴۲
۱۱.۴	خطای پر بودن ظرفیت هنگام ثبت نام دانشجو در سامانه	۴۳

^{۱۹}V-Shaped Software Lifecycle Model

^{۲۰}Unit Test

^{۲۱}Feature

[این صفحه آگاهانه خالی گذاشته شده است.]

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].

^۱Building the product right

^۲Building the right product

در تکنیک توسعه‌ی مبتنی بر رفتار، فرآیند توصیف نیازمندی‌ها به صورت تکراری^۳ انجام می‌شود و با توجه به زبان مشترک توصیف نرم‌افزار 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 می‌شود. این امر باعث کاهش هزینه‌ی نگهداری، افزایش سرعت توسعه آزمون و انطباق‌پذیری با تغییرات با وجود حفظ کیفیت نرم‌افزار می‌شود. همچنین وجود چارچوبی کارا برای پیاده‌سازی این تکنیک، بر سرعت توسعه و به‌کارگیری صحیح اصول این تکنیک تأثیر مستقیم دارد.

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 تأثیرگذار 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 آزمون‌پذیرش^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 ثبت نام 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

^۳ Iterative

^۴ Software Development Process

^۵ Acceptance Test

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 سامانه^۶ی 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 جزئیات 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.

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

^۶System

^۷Refactoring

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 اشکال-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.

^۱Software Test

^۲Debugging-oriented

^۳Demonstration-oriented

^۴Destruction-oriented

^۵Evaluation-oriented

^۶Prevention-oriented

^۷Bug

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

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

۵. در سال ۱۹۹۰، آزمون نرم‌افزار به عنوان 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

^۸Charles L. Baker

^۹Debug

^{۱۰}Edsger Dijkstra

^{۱۱}Validation

^{۱۲}Verification

^{۱۳}Reliability

^{۱۴}Software Quality

^{۱۵}Extreme Programming

^{۱۶}Waterfall Model

^{۱۷}Analysis

^{۱۸}Design

^{۱۹}Implementation

و آزمون به صورت 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 به صورت 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 به عنوان «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 بر مدل^{۲۵}، از آزمون نرم‌افزار به عنوان «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

^{۲۰} Iteration

^{۲۱} Manifesto for Agile Software Development

^{۲۲} Quality Assistance

^{۲۳} Quality Assurance

^{۲۴} Kent Beck

^{۲۵} Agile Model Driven Development

^{۲۶} Specification

^{۲۷} Specification by Example

^{۲۸} Agile Acceptance Testing

^{۲۹} User Story Testing

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. برای معتبر بودن یک مورد آزمون، ممکن است لازم باشد ورودی دارای پیش-شرط^{۳۰}‌هایی و خروجی دارای پس-شرط^{۳۱}‌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^{۳۲} (موفقیت‌آمیز) به معنی سازگاری خروجی برنامه با خروجی مورد انتظار و قرمز^{۳۳} (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].

۲.۲.۲ دسته‌بندی مدل V

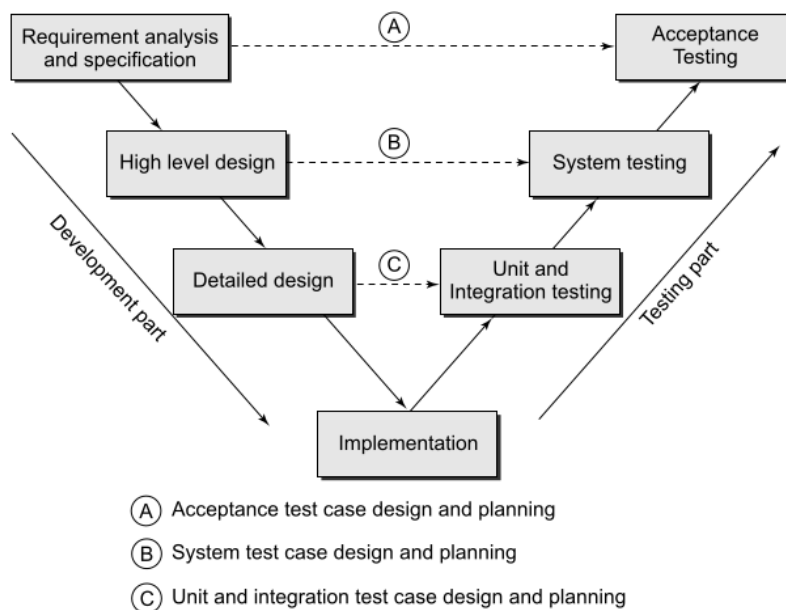
یکی از دسته‌بندی‌های مشهور برای سطوحی آزمون، دسته‌بندی مدل V می‌باشد. این دسته‌بندی، پیش از پیدایش متدولوژی‌های چابک، در متدولوژی «مدل چرخه عمر V-شکل»، که با کمی تغییر در متدولوژی مدل آبشاری و تمرکز بیشتر بر آزمون بوجود آمده، تعریف شده است [۱۷]. وجه تمایز سطوح در این دسته‌بندی، مراحل تولید نرم‌افزار و تقدم زمانی فازهای تحلیل، طراحی و پیاده‌سازی می‌باشد. همانطور که در شکل ۱.۲ نشان داده شده، پس از هر فاز،

^{۳۰} Pre-condition

^{۳۱} Post-condition

^{۳۲} Success

^{۳۳} Fail



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

آزمونی برای ساخته^{۳۴} آن فاز نوشته می‌شود [۱۷]. در این دسته‌بندی چهار سطح «آزمون پذیرش»، «آزمون سامانه^{۳۵}»، «آزمون یکپارچه‌سازی^{۳۶}» 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^{۳۷} 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].

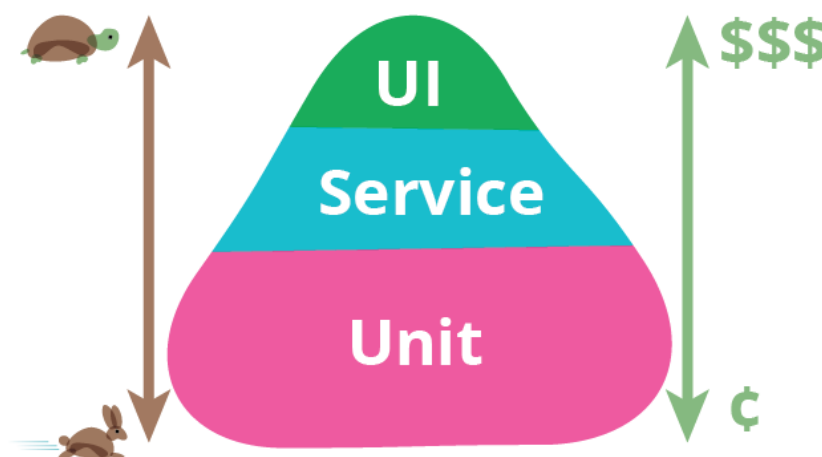
همانطور که در نمودار ۲.۲ نشان داده شده‌است، در این دسته‌بندی هرچه سطح آزمون بالاتر باشد، هزینه و زمان اجرای آزمون در آن سطح بیشتر، و تعداد آزمون‌های مورد نیاز آن سطح،

^{۳۴}Artifact

^{۳۵}System Test

^{۳۶}Integration Test

^{۳۷}Mike Cohn



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

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^{۳۸} با سایر واحدهای ماثول^{۳۹}، 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].

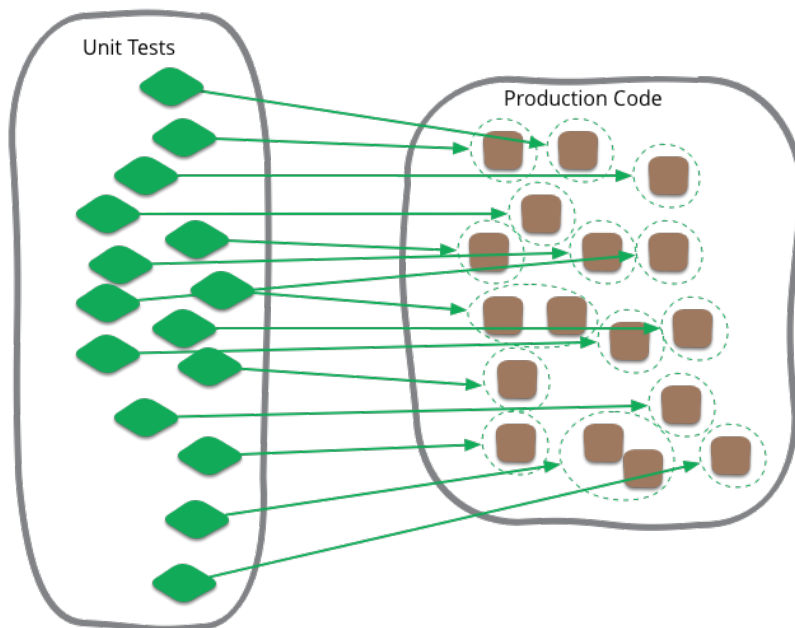
این‌که منظور از یک «واحد» چه باشد، به نیازمندی‌ها و شرایط پروژه بستگی دارد. معمولاً در برنامه‌نویسی شیء‌گرا، کلاس^{۴۰} به عنوان واحد انتخاب می‌شود [۲۱].

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

^{۳۸}Integration

^{۳۹}Module

^{۴۰}Class



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

اختیار رابط کاربری قرار می‌دهند. منطق برنامه توسط سرویس‌ها پیاده‌سازی می‌شود. این سطح آزمون، عملکرد صحیح سرویس‌های سامانه را می‌آزماید [۲۲].

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^{۴۱} چارچوبی را برای اعمال تجربه^{۴۲} های مهندسی نرم‌افزار با هدف «فراهم نمودن ابزارهای لازم برای توسعه‌ی سامانه‌های نرم‌افزار-متمرکز^{۴۳}» فراهم می‌آورد. متدولوژی شامل دو عنصر اصلی زیر است [۲۳]:

۱. مجموعه‌ای از عرف^{۴۴} ها که شامل زبان مدل‌سازی^{۴۵} (صرفی^{۴۶} و معنایی^{۴۷}) است.
 ۲. یک فرآیند، که ترتیب فعالیت^{۴۸} 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 دونفره^{۴۹} و یکپارچه‌سازی مداوم^{۵۰} 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.

^{۴۱} Software Development Methodology

^{۴۲} Practice

^{۴۳} Software-intensive

^{۴۴} Convention

^{۴۵} Modeling Language

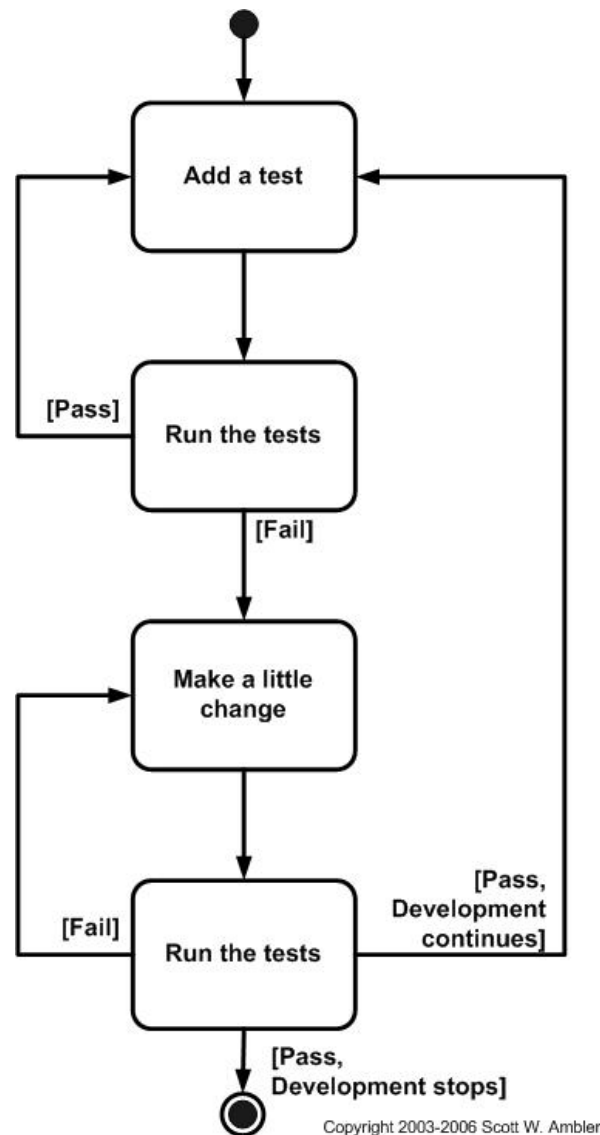
^{۴۶} Syntax

^{۴۷} Semantics

^{۴۸} Activity

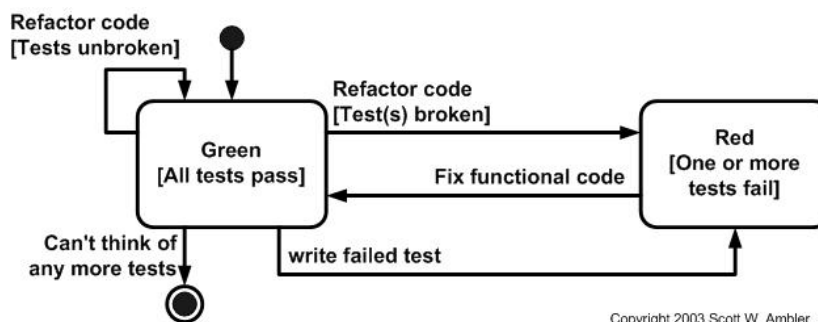
^{۴۹} Pair Programming

^{۵۰} Continuous Integration



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

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].



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]

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.

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 شده است، 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»^{۵۱} 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.

۳.۲ تکنیک توسعه‌ی مبتنی بر رفتار

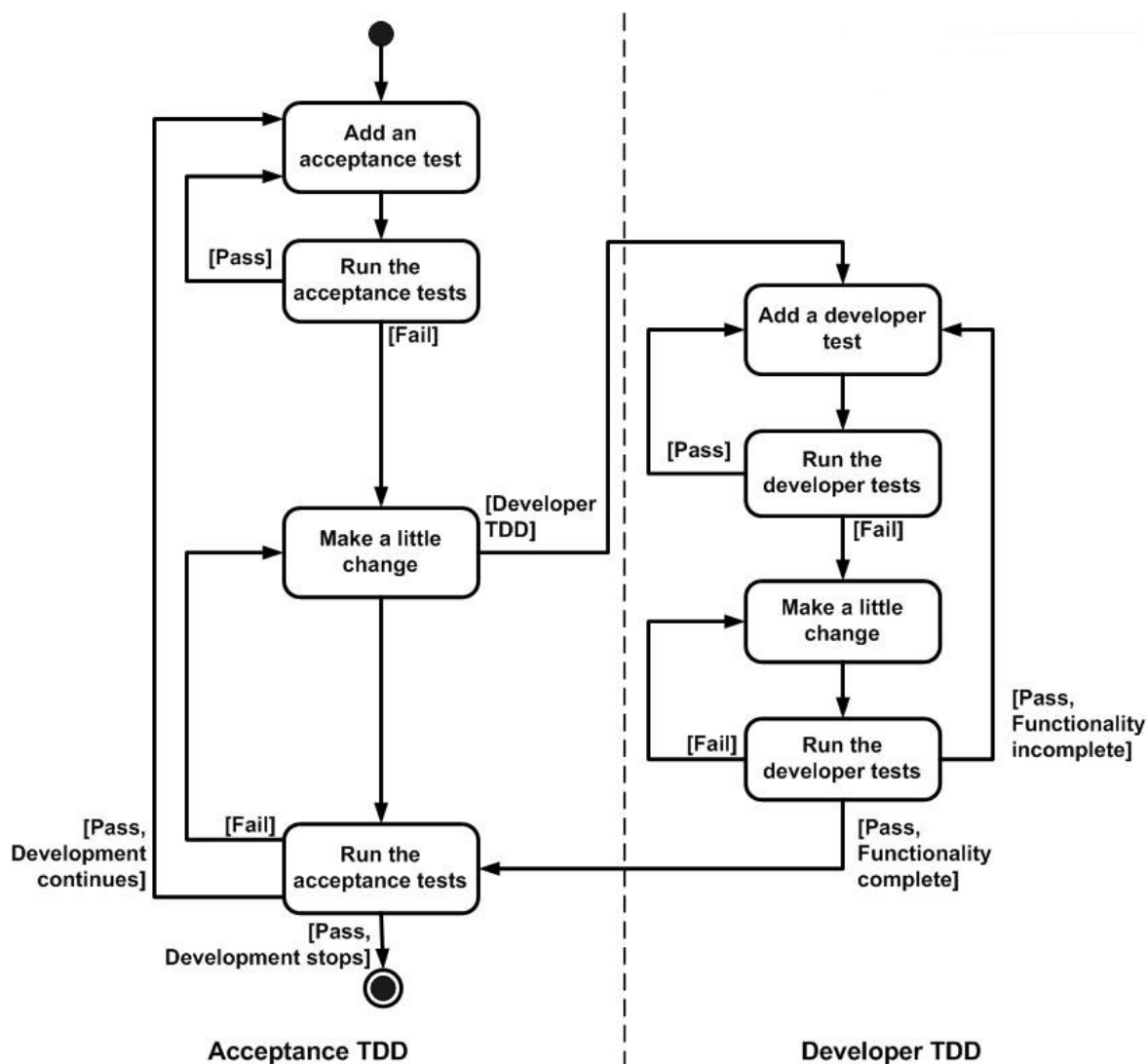
توسعه‌ی مبتنی بر رفتار یکی از تکنیک‌های توسعه‌ی چابک نرم‌افزار است که در سال ۲۰۰۶ توسط نورث^{۵۲} برای پاسخ‌گویی به ابهام‌ها و مشکل‌هایی که در توسعه‌ی مبتنی بر آزمون بوجود آمده بود، معرفی شد. [۲۵]. تأکید توسعه‌ی مبتنی بر رفتار بر بهینه‌سازی ارتباط میان نقش‌های برنامه‌نویس، آزمون‌گر و خبره‌ی حوزه^{۵۳} 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 معرفی‌شده در تکنیک‌های توسعه‌ی مبتنی بر آزمون و توسعه‌ی آزمون مبتنی بر پذیرش و علاوه بر آن در نظر گرفتن اصول زیر است [۲۷]:

^{۵۱}Done

^{۵۲}Dan North

^{۵۳}Domain Expert



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

- لازم است هدف هر داستان کاربر^{۵۴} 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 جزییاتی 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].

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}:

^{۵۴}User Story

^{۵۵}From the outside in

^{۵۶}Ubiquitous Language

«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 پروژه 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.

^{۵۷}Business Domain

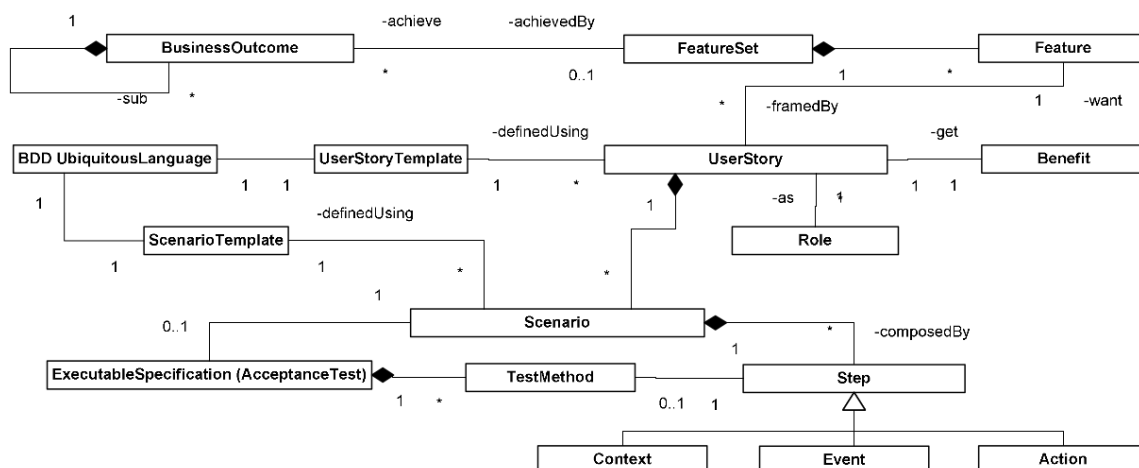
^{۵۸}Decomposition

^{۵۹}Business Value

^{۶۰}Behavior

^{۶۱}Business Outcome

^{۶۲}Feature Set



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

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:

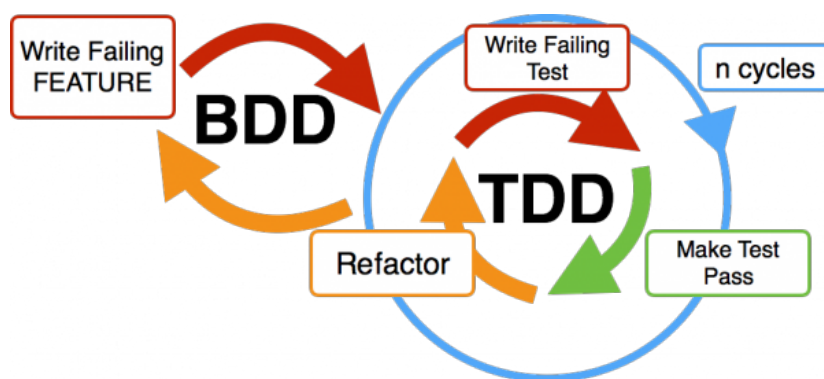
- نقش کاربر در داستان کاربر چیست؟
 - کاربر از این ویژگی چه می‌خواهد؟
 - اگر سامانه این ویژگی را داشته باشد، چه سودی به کاربر می‌رسد؟
- همچنین برای هر داستان کاربر، چند شرط پذیرش^{۶۴} به زبان مشتری تعیین می‌گردد که اگر برآورده شوند، آن داستان کاربر به درستی محقق شده است. در توسعه‌ی مبتنی بر رفتار، شرط 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

^{۶۳}Scope

^{۶۴}Acceptance Criteria



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

صورت می‌گیرد و دلخواه نیست. این قالب‌ها، قسمتی از زبان مشترک را تشکیل می‌دهند و ساختار متن ساده^{۶۵} 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 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^{۶۶}:

در توسعه‌ی مبتنی بر رفتار، با توجه به اینکه آزمون پذیرش مستقیماً از روی شرط پذیرش

^{۶۵} Plain Text

^{۶۶} Executable

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^{۶۷}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، ابزار موجود برای توسعه‌ی مبتنی بر رفتار، در تمام مراحل، برای آن قابلیت ندارند. برای مثال، در حال حاضر، هیچ کدام از ابزاری که به عنوان 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:

^{۶۷}Method

Story: [Story Title]

As a [Role]

I want to [Feature]

So that [Benefit]

این قالب، چهار قسمت برای پرکردن دارد که متن این قسمت‌ها باید مطابق با زبان مشترک تکمیل گردند:

۱. در سطر اول، در قسمت «عنوان داستان^{۶۸}» عنوانی یک خطی برای داستان کاربر نوشته می‌شود. عنوان داستان کاربر، در یک نگاه گویای کلیات آن داستان کاربر است.

۲. در قسمت «نقش^{۶۹}»، 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^{۷۰}»، فعالیتی که کاربر می‌تواند توسط سامانه انجام دهد را تعیین می‌کند.

۴. در قسمت «سود^{۷۱}»، هدف از اضافه نمودن این داستان کاربر به سامانه و سودی که به کاربر می‌رسد 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.

برای مثال، در سامانه‌ی آموزش یک دانشگاه، برای ثبت‌نام دانشجویان در دروسی که ارائه‌شده است، داستان کاربر زیر را می‌توان نوشت:

^{۶۸}Story Title

^{۶۹}Role

^{۷۰}Feature

^{۷۱}Benefit

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

از هر ویژگی، در شرایط متفاوت، رفتارهای متفاوتی انتظار داریم. برای مثال، 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»^{۷۲}، عنوانی را برای سناریو تعیین می‌کند که به صورت اجمالی محتوای سایر قسمت‌های آن سناریو را توصیف می‌کند.

^{۷۲}Scenario Title

۲. قسمت «زمینه^{۷۳}» که به قسمت «اگر^{۷۴}» 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^{۷۵}» که به قسمت «وقتی^{۷۶}» 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^{۷۷}» که به قسمت «آنگاه^{۷۸}» نیز معروف است، وضعیت مورد انتظار از سامانه پس از رویداد را تبیین می‌کند.

۵. قسمت «و^{۷۹}»، به هر کدام از بخش‌های زمینه یا پیامد می‌تواند اضافه شود تا شرایط آن قسمت را دقیق‌تر توصیف نماید.

برای مثال، برای داستان کاربر ثبت‌نام درس در دانشجو که در بالا ذکر شد، دو سناریوی زیر را می‌توان تعریف نمود:

^{۷۳}Context

^{۷۴}Given

^{۷۵}Event

^{۷۶}When

^{۷۷}Outcome

^{۷۸}Then

^{۷۹}And

Scenario 1: A successful enrollment should be consistent with offering's capacity **Given** an offering o1, and a student s1

When s1 successfully enrolls in o1

Then s1 should have had at least one empty seat before enrollment

And s1's available capacity should have been decreased by one

Scenario 2: An offering's capacity should stay consistent with actual enrollments **Given** an offering o1

When an enrollment in o1 happens

Then value of o1's used_capacity field should be equal to number of enrolled students in o1

[این صفحه آگاهانه خالی گذاشته شده است.]

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 مسأله‌ی اصلی را بیان می‌کند.

چارچوب پیشنهادی ما، حول ایده‌ی جداسازی 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 فراهم کند تا صحت آن‌ها سنجیده شود.

دو لایه‌ی سناریو و تعامل‌گر، ارکان اصلی معماری پیشنهادی جهت پیاده‌سازی چارچوب آزمون هستند.

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 بازبهره‌کارگیری کد^۳ وضعیت یک سامانه سناریوهای متفاوت وجود دارد که این موضوع منجر به کاهش حجم آزمون‌ها می‌شود.

^۱ Metric

^۲ Actor

^۳ Code Reuse

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^۴ به مؤلفه^۵های 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.

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 در سطح سیستمی

اگر چه معماری پیشنهادی ما مستقل از سطح آزمون است، با این حال برداشت مؤلفان این است که تأثیر مثبت آن در زمینه‌ی آزمون سامانه مشهودتر خواهد بود. در این لایه به دلیل سطح بالا بودن آزمون‌ها، حجم آماده‌سازی‌هایی که یک مورد آزمون می‌بایست انجام دهد

^۴Coupling

^۵Component

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 بازه‌کارگیری کد آن، به کاهش حجم و هزینه‌ی نگهداری این‌گونه آزمون‌ها کمک شایانی می‌کند.

۴.۳ 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^۶ نرم‌افزار، پیش از عملیاتی شدن آن است. اما بسیاری از نقص‌های نرم‌افزاری حتی پس از عملیاتی شدن نیز از دیده نمان 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^۷ در مقادیر محاسبه‌شدنی است.

۵.۳ پیاده‌سازی بر مبنای چارچوب جنگو

به عنوان بخشی از این پروژه، یک چارچوب آزمون با جنگو و مبتنی بر معماری ارائه‌شده در فصل ۱.۳ را پیاده‌سازی کردیم.

^۶Defect

^۷Inconsistency

۱.۵.۳ نحوه‌ی استفاده

این چارچوب در قالب یک افزونه برای جنگو نوشته شده. برای استفاده از این افزونه نیاز است تا:

۱. نام افزونه (insanity) به لیست افزونه‌های جنگو اضافه گردد.
۲. سناریوهای مطلوب در فایل scenarios.py مرتبط با هر مؤلفه جنگو قرار بگیرند.
۳. هر کدام از توابعی که به عنوان محرک^۸ سناریوها عمل می‌کنند، توسط دکوراتور^۹ action پوشیده شوند.

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.

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:

^۸Trigger

^۹Decorator

https://github.com/SeMeKh/bsc_project

فصل ۴

پیاده‌سازی

در این فصل، گزارشی از پیاده‌سازی یک سامانه‌ی کوچک ارائه می‌شود، که آزمون‌های آن با استفاده از چارچوب پیاده‌سازی‌شده در قسمت ۵.۳ تألیف شده است.

۱.۴ تألیف آزمون برای سامانه‌ی آموزش

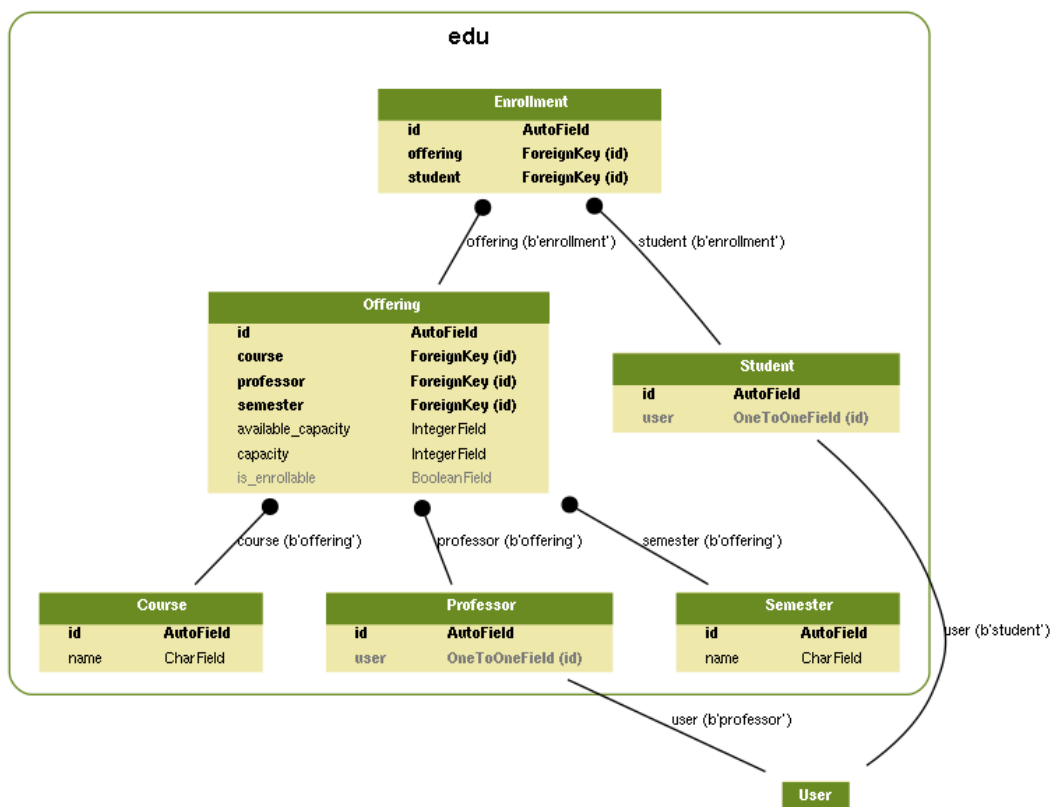
برای بررسی صحت عملکرد / اینسنیتی^۱، یک سامانه‌ی بسیار ساده را طراحی و در چارچوب جنگو پیاده‌سازی نمودیم تا بتوانیم در آن سامانه، آزمون‌هایی را در چارچوب اینسنیتی تألیف نماییم.

جهت ملموس‌تر بودن موجودیت^۲ها برای خواننده، سامانه‌ی انتخاب واحد را برای آزمون انتخاب نمودیم و سعی نمودیم جزییاتی که مرتبط با آزمون نیستند را از آن حذف نماییم. در این سامانه تنها عملیات سی.آر.یو.دی^۳ برای موجودیت‌ها و انتخاب واحد توسط دانشجو پیاده‌سازی شده است.

^۱Insanity

^۲Entity

^۳Create Read Update Delete



شکل ۱.۴: نمودار موجودیت‌های سامانه‌ی ثبت نام آموزش

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^۴: 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.
۲. دانشجو^۵ و استاد^۶: 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.

^۴User

^۵Student

^۶Professor

۳. نیمسال تحصیلی^۷ و درس^۸: برای این دو موجودیت، نگهداری صفت «نام» برای آن‌ها در سامانه کافیت.

۴. ارائه^۹: ارتباط چند به چند بین موجودیت‌های درس و نیمسال تحصیلی و استاد ارائه‌دهنده‌ی آن، در این موجودی 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^{۱۰}: این موجودیت ارتباط چند به چند ثبت‌نام میان موجودیت‌های دانشجو و ارائه را نگهداری می‌کند.

۲.۱.۴ 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:

^۷Semester

^۸Course

^۹Offering

^{۱۰}Enrollment

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: A successful enrollment should be consistent with offering's capacity **Given** an offering o1, and a student s1

When s1 successfully enrolls in o1

Then s1 should have had at least one empty seat before enrollment

And s1's available capacity should have been decreased by one

Scenario 2: An offering's capacity should stay consistent with actual enrollments **Given** an offering o1

When an enrollment in o1 happens

Then value of o1's used_capacity field should be equal to number of enrolled students in o1

گام بعدی انتقال سناریوها از زبان اگر-وقتی-آنگاه^{۱۱} 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 ۲.۴ آمده.

^{۱۱} Given-When-Then

```
class EnrollmentScenario1(Scenario):
    action_name = 'edu.models.Offering.enroll'

    def given(scenario, ol, s2, **payload):
        scenario.old_capacity = ol.available_capacity
        return True

    def when(scenario, commit, **payload):
        return commit

    def then(scenario, payload, return_value, exc_type, **kwargs):
        assert scenario.old_capacity > 0
        assert payload['self'].available_capacity == scenario.old_capacity - 1

class EnrollmentScenario2(Scenario):
    action_name = 'edu.models.Offering.enroll'

    def then(scenario, payload, **kwargs):
        ol = payload['self']
        used_capacity = ol.capacity - ol.available_capacity
        enrollment_count = Enrollment.objects.filter(offering=ol).count()
        assert used_capacity == enrollment_count
```

شکل ۲.۴: کد معادل سناریو

۳.۱.۴ رابط کاربری سامانه

برای شروع به کار سامانه، با اجرای دستور `runserver_plus` در `./manage.py` در پوشه‌ی اصلی برنامه، کارگزار^{۱۲} شروع به کار نموده و می‌توان از طریق نشانی `http://localhost:8000/` به رابط کاربری سامانه دسترسی پیدا نمود.

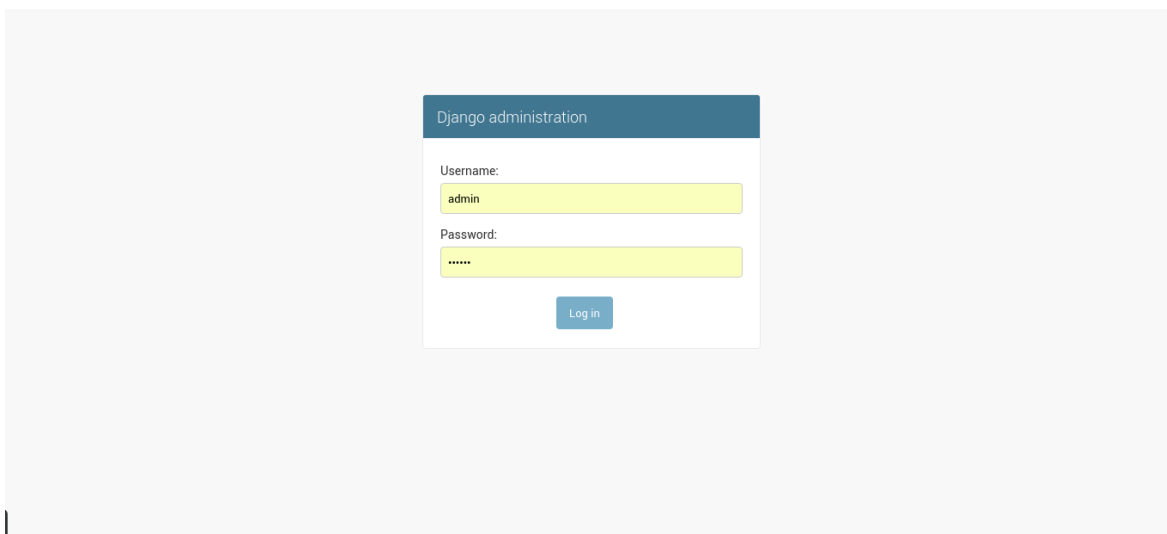
ابتدا نام کاربری و کلمه عبور را وارد می‌نماییم (شکل ۳.۴). سپس بسته به نقشی که کاربر در سامانه داشته باشد، وارد یکی از صفحه‌های کارمند، دانشجو یا استاد خواهیم شد. در صفحه‌ی کارمند (شکل ۴.۴)، می‌توان از منوی سمت چپ، هر کدام از گزینه‌های کاربر، دانشجو، استاد، درس، نیم‌سال تحصیلی و ارائه را انتخاب نمود؛ که در تصویر ۴.۴، گزینه‌ی ارائه انتخاب شده است.

با انتخاب هر گزینه، لیست موارد ثبت‌شده در سامانه برای آن گزینه قابل نمایش است. دو ستون ویرایش^{۱۳} و حذف^{۱۴} در انتهای هر ردیف از لیست هر گزینه قرار دارد که می‌توان

^{۱۲}Server

^{۱۳}Edit

^{۱۴}Delete



شکل ۳.۴: صفحه‌ی ورود سامانه‌ی ثبت نام

Education System Logout(admin) Django Admin Staff Panel

[Users](#)
[Students](#)
[Professors](#)
[Courses](#)
[Semesters](#)
[Offerings](#)

Offering List Add

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

« 1 »

شکل ۴.۴: صفحه‌ی کاربری کارمند در سامانه‌ی ثبت نام

شکل ۵.۴: صفحه‌ی ویرایش مشخصات ارائه توسط کارمند

آن ردیف را ویرایش یا حذف نمود. برای مثال، در شکل ۵.۴، 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^{۱۵} از جدول، می‌تواند ظرفیت آن درس را تغییر دهد. (شکل ۶.۴)

همچنین اگر کاربر دسترسی مدیر سیستم^{۱۶} نیز داشته باشد، می‌تواند با استفاده از منوی بالا-راست به قسمت مدیریت سیستم جنگو^{۱۷} نیز دسترسی پیدا کند و به تمام موجودیت‌های سامانه دسترسی پیدا کند. برای مثال، می‌تواند از طریق تنظیمات کاربران (شکل ۷.۴) 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

^{۱۵}Capacity

^{۱۶}Superuser

^{۱۷}Django Administration

Education System [Logout\(admin\)](#) [Django Admin](#) [Staff Panel](#)

- Users
- Students
- Professors
- Courses**
- Semesters
- Offerings

Capacity

شکل ۶.۴: تغییر ظرفیت درس توسط کارمند

Django administration WELCOME **ADMIN** [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

[Home](#) > [Authentication and Authorization](#) > [Users](#)

Select user to change [ADD USER](#) **+**

Action: 0 of 18 selected

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

FILTER

By staff status

☐ All

☐ Yes

☐ No

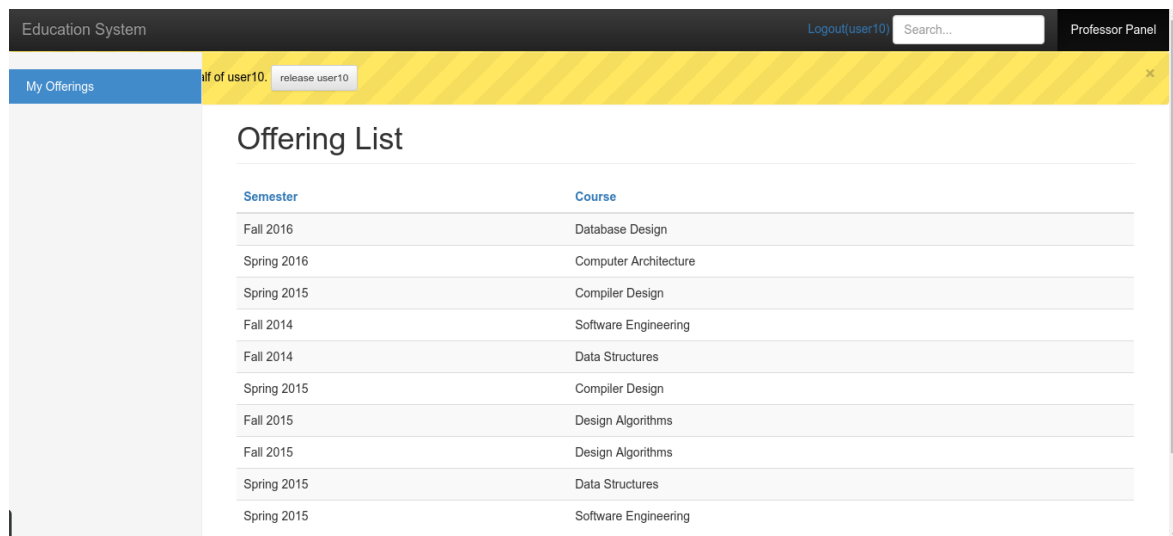
By superuser status

☐ All

☐ Yes

☐ No

شکل ۷.۴: دسترسی مدیر سامانه به صفحه‌های کاربرها



The screenshot shows a web application titled "Education System". At the top right, there are links for "Logout(user10)", a search bar, and a "Professor Panel". Below this is a yellow banner with the text "Offering List" and a "release user10" button. The main content area is titled "Offering List" and contains a table with two columns: "Semester" and "Course".

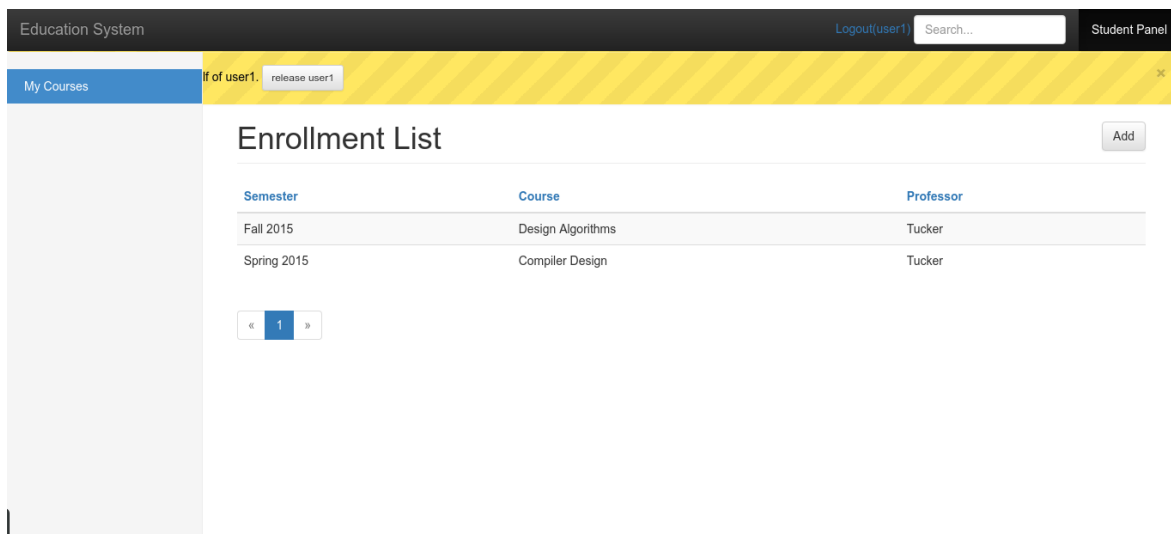
Semester	Course
Fall 2016	Database Design
Spring 2016	Computer Architecture
Spring 2015	Compiler Design
Fall 2014	Software Engineering
Fall 2014	Data Structures
Spring 2015	Compiler Design
Fall 2015	Design Algorithms
Fall 2015	Design Algorithms
Spring 2015	Data Structures
Spring 2015	Software Engineering

شکل ۸.۴: صفحه‌ی کاربری استاد در سامانه‌ی ثبت نام

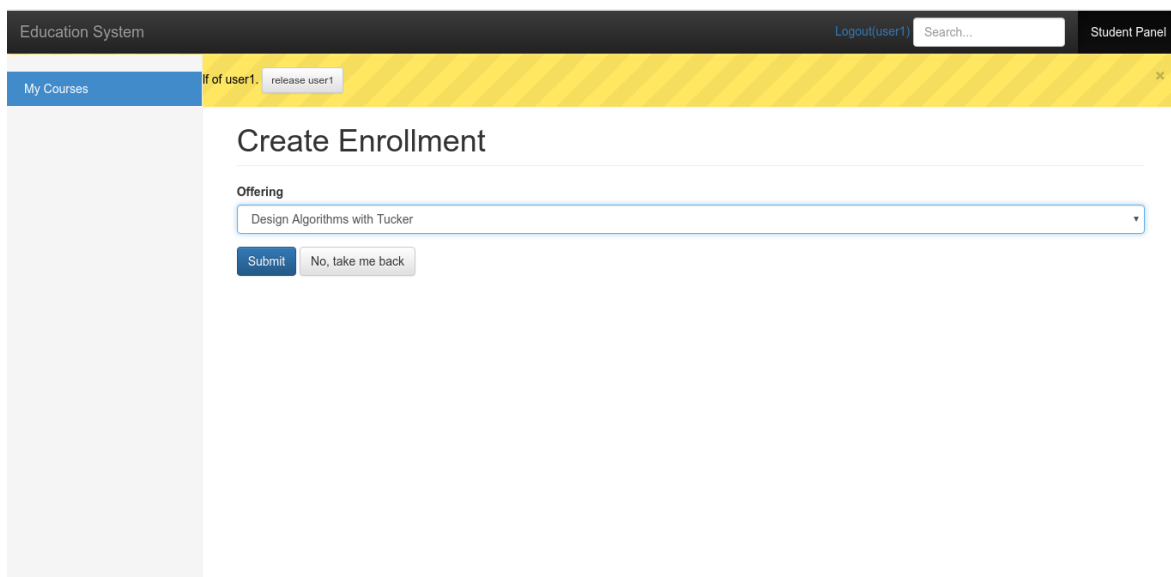
که توسط وی ارائه شده است را مشاهده کند. امکان حذف یا اضافه‌ی دروس ارائه شده‌ی هر استاد، برای نقش کارمند فعال می‌باشد. همچنین اگر دانشجو وارد سامانه شود، می‌تواند لیست درس‌هایی که در آن‌ها ثبت‌نام کرده را مشاهده نماید. (شکل ۹.۴).

برای ثبت‌نام در یک ارائه از یک درس، دانشجو می‌تواند بر روی کلید در سمت بالا-راست صفحه‌ی شکل ۹.۴ کلیک نماید. در این صورت به صفحه‌ی ثبت‌نام که در شکل ۱۰.۴ نمایش داده شده، منتقل می‌شود. در 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 نشان داده شده در شکل ۱۱.۴ 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.



شکل ۹.۴: صفحه‌ی کاربری دانشجو در سامانه‌ی ثبت نام



شکل ۱۰.۴: صفحه‌ی ثبت نام دانشجو در درس‌های ارائه شده توسط سامانه

Education System

Logout(user1) Search...

Student Panel

If of user1. release user1

Create Enrollment

There is no available capacity

Offering

Design Algorithms with Tucker

Submit No, take me back

شکل ۱۱.۴ : خطای پر بودن ظرفیت هنگام ثبت نام دانشجو در سامانه

[این صفحه آگاهانه خالی گذاشته شده است.]

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

کتاب نامه

- [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://sauceio.com/index.php/2016/03/a-two-minute-bdd-overview/> (visited on 08/10/2016).

[این صفحه آگاهانه خالی گذاشته شده است.]

واژه‌نامه‌ی فارسی به انگلیسی

آزمون..... Test	بیانیه‌ی توسعه‌ی چابک نرم افزار.....
آزمون پذیرش..... Acceptance Test	Manifesto for Agile Software
آزمون سامانه..... System Test	Development
آزمون نرم افزار..... Software Test	پسا-شرط..... Post-condition
آزمون واحد..... Unit Test	پایاده سازی..... Implementation
آزمون یکپارچه سازی .. Integration Test	پیش-شرط..... Pre-condition
آنگاه..... Then	تابع..... Method
اتکاپذیری..... Reliability	تجربه..... Practice
اجرایی..... Executable	تحلیل..... Analysis
ارزش تجاری..... Business Value	تعامل گر..... Actor
اعتبارسنجی..... Validation	تفکیک..... Decomposition
اکس پی..... Extreme Programming	تکرار..... Iteration
اگر..... Given	تکراری..... Iterative
انجام شده..... Done	تکنیک..... Technique
اینسیتی..... Insanity	توسعه‌ی آزمون مبتنی بر پذیرش.....
بازبه کارگیری کد..... Code Reuse	Acceptance Test Driven
باگ..... Bug	Development
باگ زادی..... Debug	توسعه‌ی اول-آزمون..... Test First
برنامه نویسی دونفره .. Pair Programming	Development

Scenario	سناریو	Agile Model	توسعه‌ی چابک مبتنی بر مدل
Create Read Update	سی.آر.یو.دی	Driven Development	
Delete		Agile Software	توسعه‌ی چابک نرم‌افزار
Acceptance Criteria	شرط پذیرش	Development	
Syntax	صرفی	Test Driven	توسعه‌ی مبتنی بر آزمون
Design	طراحی	Development	
Convention	عرف	Behavior Driven	توسعه‌ی مبتنی بر رفتار
Refactoring	فاکتوربندی مجدد	Development	
Software	فرآیند توسعه‌ی نرم‌افزار	Specification	توصیف
Development Process		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
Software Quality	کیفیت نرم‌افزار	User Stories	داستان‌های کاربر
Module	ماژول	Decorator	دکوراتور
Methodology	متدولوژی	User Interface	رابط کاربری
Software	متدولوژی توسعه‌ی نرم‌افزار	Behavior	رفتار
Development Methodology		Modeling Language	زبان مدل‌سازی
Metric	متریک	Ubiquitous Language	زبان مشترک
Plain Text	متن ساده	Artifact	ساخته
Test Suite	مجموعه‌ی آزمون	System	سامانه
Feature Set	مجموعه‌ی ویژگی	Success	سبز

Software-intensive ... نرم افزار-متمرکز	Scope محدوده
Defect نقص	Trigger محرک
Coupling وابستگی	Waterfall Model مدل آبشاری
Verification واریسی	V-Shaped مدل چرخه عمر V-شکل
When وقتی	Software Lifecycle Model
Feature ویژگی	Semantics معنایی
Test Pyramid هرم آزمون	Entity موجودیت
Integration یکپارچه سازی	Test Case مورد آزمون
Continuous یکپارچه سازی مداوم	Component مؤلفه
Integration	Inconsistency ناهمخوانی
	Business Outcome نتیجه ی تجاری

[این صفحه آگاهانه خالی گذاشته شده است.]

واژه‌نامه‌ی انگلیسی به فارسی

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

Plain Text.....	متن ساده	Executable	اجرایی
Post-condition.....	پسا-شرط	Extreme Programming	اکس.پی.
Practice	تجربه	Fail	قرمز
Pre-condition.....	پیش-شرط	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	پیاده‌سازی
Server	کارگزار	Inconsistency	ناهم‌خوانی
Software Development		Insanity	اینسنیتی
Methodology	متدولوژی نرم افزار توسعه‌ی	Integration.....	یکپارچه‌سازی
Software Development Process	فرآیند	Integration Test...	آزمون یکپارچه‌سازی
توسعه‌ی نرم افزار		Iteration.....	تکرار
Software Quality.....	کیفیت نرم افزار	Iterative	تکراری
Software Test.....	آزمون نرم افزار	Manifesto for Agile Software	
Software-intensive ...	نرم افزار-متمركز	Development	بیانیه‌ی توسعه‌ی چابک نرم
Specification.....	توصیف	افزار	
Success	سبز	Method	تابع
Syntax	صرفی	Methodology	متدولوژی
System.....	سامانه	Metric	متریک
System Test	آزمون سامانه	Modeling Language ...	زبان مدل‌سازی
Technique	تکنیک	Module	ماژول
Template.....	قالب	Pair Programming..	برنامه‌نویسی دونفره

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

[این صفحه آگاهانه خالی گذاشته شده است.]

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.

[این صفحه آگاهانه خالی گذاشته شده است.]



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

