یک روش آزمون مدل‌رانده برای تولید خودکار موارد آزمون براساس نمودار توالی

امیر سواری[۱]، بهمن زمانی[۲]

[۱]دانشجوی کارشناسی ارشد دانشکده مهندسی کامپیوتر، دانشگاه اصفهان ، amir.savari@eng.ui.ac.ir

[۲]استادیار دانشکده مهندسی کامپیوتر، دانشگاه اصفهان، zamani@eng.ui.ac.ir

چکیده

یکی از فازهای اصلی و پرهزینه چرخه توسعه نرم‌افزار آزمون است. خودکارسازی آزمون یک روش مهم برای کاهش هزینه توسعه نرم‌افزار است و تولید خودکار موارد آزمون نیز یکی از مهم‌ترین بخش‌ها در خودکارسازی آزمون می‌باشد. این تحقیق، یک رویکرد آزمون مبتنی بر مدل با استفاده از مفاهیم توسعه مدل‌رانده برای تولید خودکار موارد آزمون ارائه می‌کند. در این رویکرد با استفاده از چارچوب مدل‌رانده و با به‌کارگیری نمودارهای توالی برای تشریح رفتار سیستم به عنوان ورودی، مدل آزمون و کد آزمون تولید می‌شوند. در این پژوهش با استفاده از ترکیب نمودارهای توالی موارد آزمون پیچیده‌تری تولید می‌شود. نتایج ارزیابی روش پیشنهادی برای یک مطالعه‌ی موردی حاکی از مفید بودن این روش و صرفه‌جویی در هزینه آزمون می‌شود.

واژه‌های کلیدی

آزمون مبتنی بر مدل، آزمون مدل‌رانده، تولید خودکار موارد آزمون، نمودار توالی، پروفایل آزمون یوام‌ال

۱- مقدمه

آزمون نرم‌افزار یکی از فعالیت‌های اصلی و مهم در فرایند تولید نرم‌افزار و نیز اصلی‌ترین روش جهت ارزیابی کیفیت نرم‌افزار است که حدود ۵۰٪ از تلاش‌ها و بودجه‌ی پروژه صرف انجام این عمل می‌گردد [۱]. در عمل، آزمون نرم‌افزار به جای استفاده از یک رویه سامان‌مند و خودکار، اغلب به صورت دستی انجام می‌شود که پرهزینه، وقت‌گیر و مستعد خطاست. بنابراین خودکارسازی آزمون روشی است تا کیفیت سیستم را تضمین کند و نیز هزینه توسعه را کاهش دهد.

برای غلبه بر پیچیدگی و هزینه آزمون، نیاز است تکنیک‌های جدید آزمون ابداع شوند که یکی از این تکنیک‌ها آزمون مدل‌رانده[۱] است. امروزه زبان مدل‌سازی یکنواخت[۲] (UML) یک زبان مدل‌سازی مورد قبول است و مهندسی مدل‌رانده[۳] کمک می‌کند تا سطح انتزاع در فاز اولیه برای توسعه نرم‌افزار افزایش یابد. در سوی دیگر، آزمون مبتنی بر مدل[۴] (MBT) تکنیکی برای تولید خودکار موارد آزمون با استفاده از مدل‌های استخراج شده از مصنوعات نرم‌افزاری، مهیا می‌کند [۲]. در MBT، مدلی از رفتار مورد انتظار سیستم یا مدل محیطی سیستم، ایجاد می‌شود، سپس با استفاده از این مدل، موارد آزمون می‌توانند تولید شوند [۳]. این روش باعث می‌شود طرح آزمون در مرحله اولیه چرخه توسعه نرم‌افزار موجود باشد.

آزمون مدل‌رانده به آزمون مبتنی بر مدلی اشاره دارد که مفاهیم مدل-رانده در آن به‌کار برده شود. در مهندسی مدل‌رانده، مدل‌ها به عنوان موجودیت اصلی برای تولید خودکار کد استفاده می‌شوند و برای تحقق فرایند توسعه مدل‌رانده نیاز به تبدیل خودکار مدل است. در این تحقیق از دو نوع تبدیل[۵] استفاده می‌شود: ۱) تبدیل مدل به مدل[۶] برای تولید مدل‌های آزمون از مدل‌های طراحی. این تبدیل، مدل‌های UML را به عنوان ورودی دریافت و مدل‌های پروفایل آزمون یوام‌ال[۷] (UTP) تولید می‌کند. ۲) تبدیل مدل به کد[۸] برای تولید کد آزمون از مدل‌های آزمون. مدل‌های آزمون به عنوان ورودی این تبدیل در نظر گرفته و کد Junit تولید می‌شود.

---

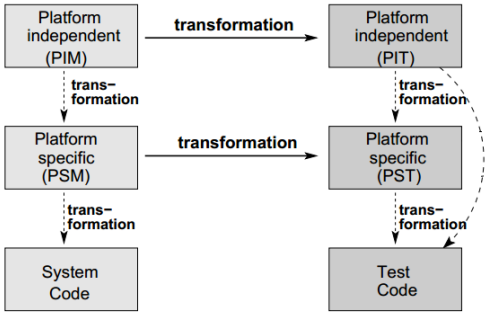
[۱] Model-Driven Testing
[۲] Unified Modeling Language
[۳] Model-Driven Engineering
[۴] Model-Based Testing
[۵] Transformation
[۶] Model-to-model transformation
[۷] UML Testing Profile
[۸] Model-to-Text Transformation

در این تحقیق با به‌کارگیری مفاهیم مدل‌رانده در آزمون مبتنی بر مدل یک روش آزمون مدل‌رانده ارائه می‌شود. هدف اصلی این تحقیق کاهش هزینه فرایند توسعه نرم‌افزار با تولید خودکار موارد آزمون است.

ساختار این مقاله این چنین است: در بخش ۲ آزمون مدل‌رانده و مفاهیم آن شرح داده می‌شود. در این بخش ابتدا آزمون مدل‌رانده معرفی می‌شود؛ در بخش ۲-۱ و بخش ۲-۲ مدل‌های توالی برای نمایش رفتار سیستم و پروفایل آزمون UML برای تشریح مدل آزمون معرفی می‌شوند، سپس در بخش ۲-۳ زبان اپسیلون برای خودکارسازی روش مدل‌رانده معرفی می‌شود. در بخش ۳ روش آزمون مدل‌رانده پیشنهادی شرح داده می‌شود. برای روش پیشنهادی و ارزیابی آن از یک مثال کتابخانه استفاده می‌شود که در بخش ۴ به آن پرداخته می‌شود. در بخش ۵ مزایای روش پیشنهادی مورد ارزیابی قرار می‌گیرد. در بخش ۶ پژوهش‌های مرتبط با این پژوهش بررسی می‌شوند و در نهایت در بخش ۷ نتیجه‌گیری و پیشنهادات ارئه می‌شوند.

۲- آزمون مدل‌رانده

فلسفه معماری مدل‌رانده[۱] (MDA) می‌تواند برای مدل‌سازی سیستم و مدل‌سازی آزمون بکار برده شود. همان‌طور که در شکل ۱ مشخص است، مدل طراحی سیستم مستقل از سکو[۲] (PIM) می‌تواند به مدل‌های طراحی سیستم خاص سکو[۳] (PSM) تبدیل شود. PIM ها بر روی توصیف عملکرد خالص یک سیستم که برای تحقق و اجرای سیستم استفاده می‌شوند، تمرکز دارد و PSM های مرتبط با آن حاوی مقدار زیادی از اطلاعات سکوی زیرین می‌باشند. در تبدیل دیگر، کد سیستم ممکن است از PSM نتیجه شود. بدیهی است کامل بودن کد سیستم به کامل بودن مدل طراحی سیستم بستگی دارد.



شکل ۱: مدل‌های طراحی سیستم (سمت چپ) در مقابل مدل‌های طراحی آزمون [۴]

این اصطلاحات (مدل‌سازی مستقل از سکو، خاص سکو و تولید کد سیستم) می‌توانند برای مدل‌های طراحی آزمون بکار برده شوند. علاوه بر این، مدل‌های طراحی سیستم ممکن است به طور مستقیم به مدل‌های طراحی آزمون تبدیل شوند. هنگامی‌که مدل طراحی سیستم در سطح PIM تعریف می‌شود یک مدل طراحی مستقل از سکوی آزمون می‌تواند از آن نتیجه شود. این مدل‌ها همچنین می‌توانند به طور مستقیم به مدل طراحی آزمون یا کد آزمون خاص سکو تبدیل شوند [۴]. در شکل ۱ تبدیلات بین مدل‌ها نشان داده شده است که یکی از اهداف پژوهش حاضر این است که تبدیلات بین مدل‌ها به صورت خودکار انجام شوند.

۲-۱- نمودار توالی

نمودار توالی رایج‌ترین نوع نمودار تعاملی است که بر روی تبادل پیام‌ها میان تعدادی Lifeline تمرکز دارد. نمودار توالی می‌تواند برای نمایش رفتار سیستم و همچنین برای نمایش رفتار یک مورد آزمون در مدل آزمون استفاده شود. در این پژوهش تولید موارد آزمون از نمودارهای توالی گسترش می‌یابد و از قابلیت استفاده مجدد نمودارهای توالی برای تولید خودکار موارد آزمون استفاده می‌شود.

برای گسترش تولید موارد آزمون از نمودار توالی از قابلیت استفاده مجدد نرم‌افزار استفاده می‌شود. استفاده مجدد نرم‌افزار[۴] به عنوان فرایند ایجاد سیستم‌های نرم‌افزاری از نرم‌افزار موجود تعریف می‌شود که بهره‌وری در توسعه نرم‌افزار و همچنین کیفیت سیستم‌های نرم‌افزاری را به دلیل استفاده از مصنوعات تایید شده افزایش می‌دهد [۵]. استفاده مجدد نرم‌افزار امروزه در همه سسیستم‌های بزرگ و پیچیده نرم‌افزاری بکار برده می‌شود، این مکانیزم می‌تواند در مدل‌سازی نرم‌افزار نیز استفاده شود. مکانیزمی در نمودارهای توالی وجود دارد که می‌توان نمودارهای توالی پیچیده و بزرگ را به وسیله Interaction Use ساده‌سازی کند. Interaction Use یک Interaction Fragment است که اجازه می‌دهد تعامل‌های دیگر را فراخوانی کرد. این یک راه حل عمومی برای استفاده مجدد برخی تعاملات میان چندین تعامل دیگر است. Interaction Use به عنوان یک Combined Fragment با عمل‌گر ref نشان داده می‌شود [۶].

۲-۲- پروفایل آزمون UML (UTP)

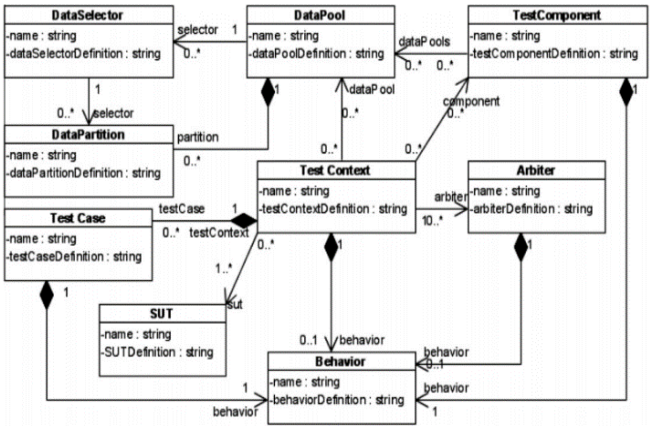
پروفایل UML یک مکانیزم گسترش عمومی برای ساخت مدل‌های UML در دامنه‌های خاص را مهیا می‌کند. پروفایل آزمون UML یک گسترش از UML است و برای دامنه آزمون، توسعه داده شده است که در چهارگروه معماری آزمون، داده آزمون، رفتار آزمون و زمان آزمون تقسیم بندی می‌شود. این پروفایل به وسیله استفاده از UML برای مدل‌سازی سیستم و مشخصات آزمون، پلی میان طراح و آزمون‌گرها ایجاد می‌کند [۴]. UTP یک زبان استاندارد شده مبتنی‌بر زبان مدل‌سازی یکپارچه OMG است. مدل مبتنی

[۱] Model-Driven Architecture
[۲] Platform Independent Model
[۳] Platform Specific Model
[۴] Software Reuse

بر مشخصات آزمون ارائه شده با UTP مستقل از هر متدولوژی، دامنه یا نوع سیستم است [۷].

پروفایل، مفاهیمی برای توسعه مدل‌های آزمون برای آزمون جعبه سیاه می‌باشد. شکل ۲ بخشی از متامدل UTP را نشان می‌دهد. معماری آزمون در UTP، مجموعه‌ای از مفاهیمی برای تعیین جنبه‌های ساختاری آزمون می‌باشد که شامل TestContext است و موارد آزمون در آن قرار می‌گیرند. رفتار آزمون مشخص کننده عملیات و ارزیابی اهداف آزمون است که تعیین می‌کند چه چیزی باید آزموده شود. TestCase یک حالت برای آزمون سیستم مشخص می‌کند که شامل ورودی مورد نیاز، خروجی و شرایط اولیه است. TestCase تعیین کننده خصوصیاتی است که تعامل بین TestComponent با سیستم تحت آزمون[۱] (SUT) را مشخص می‌کند.

تعیین موارد آزمون نیاز به مشخص کردن داده‌های آزمون دارد. داده-های آزمون بخشی از مشخصات آزمون مبتنی بر مدل است که در یک مورد آزمون یا به SUT ارسال و یا به عنوان یک پاسخ مورد انتظار از سیستم بازیابی می‌شوند. در متامدل، داده‌های آزمون در DataPool قرار می‌گیرند و داده‌های آزمون می‌توانند در قالب فایل (به طور مثال فایل XML) مشخص شوند.



شکل ۲: متامدل UTP [۷]

۲-۳- زبان تبدیل اپسیلون

روش آزمون مدل‌رانده شکل ۱ از تبدیلات مدل به مدل و مدل به کد تشکیل می‌شود. برای تحقق تبدیلات نیاز به یک زبان تبدیل برای اجرایی کردن آن می‌باشد که اپسیلون یک زبان تبدیل استاندارد و مناسب در این زمینه می-باشد. اپسلیون سکویی برای ساخت زبان‌های خاص-وظیفه تعاملی و سازگار برای مدیریت مدل؛ مانند تبدیل مدل، تولید کد، مقایسه مدل، ادغام و اعتبارسنجی فراهم می‌کند. برای هر زبان، اپسیلون ابزارهای توسعه مبتنی بر ایکلیپس و یک مترجم ارائه می‌کند که می‌تواند برنامه‌های نوشته شده در این زبان را اجرا کند [۸].

در هسته اپسیلون زبان شی اپسیلون[۲] (EOL) قرار دارد. این زبان یک زبان دستوری بر اساس OCL است که ویژگی‌هایی از قبیل اصلاح مدل، دسترسی به مدل‌های مختلف، ساختار برنامه‌نویسی مرسوم (متغیرها، حلقه-ها، شاخه‌ها و غیره) و تعامل با کاربر را فراهم می‌کند. هرچند EOL می‌تواند به عنوان زبان مدیریت مدل همه منظوره استفاده شود، اما هدف اصلی آن استفاده در زبان‌های خاص-وظیفه است. بنابراین، تعدادی زبان خاص-وظیفه در بالای EOL پیاده‌سازی شده‌اند که در این پژوهش از زبان تبدیل اپسیلون[۳] (ETL) و زبان تولید اپسیلون[۴] (EGL) استفاده می‌شود.

۳- روش آزمون مدل‌رانده پیشنهادی

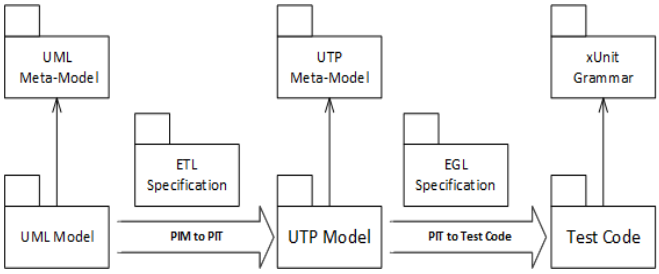
برای خودکارسازی چارچوب ارائه شده در شکل ۱ نیاز است تبدیلات این چارچوب به صورت خودکار انجام شوند. مدل‌های طراحی مستقل از سکو در این پژوهش با نمودارهای کلاس و توالی و مدل‌های آزمون مستقل از سکو با UTP ارائه می‌شوند و از این مدل‌ها می‌توان کد آزمون را به صورت مستقیم تولید کرد. بنابراین هدف اصلی این پژوهش خودکارسازی تبدیل PIM به PIT و PIT به Test Code (شکل ۱) است.

با توجه به آزمون مدل‌رانده و تبدیلات بین مدل‌ها، یک روش آزمون مدل‌رانده پیشنهادی در شکل ۳ ارئه شده است. در این روش نمودارهای UML به عنوان ورودی دریافت و با استفاده از تبدیل‌های نوشته شده به زبان ETL به مدل آزمون تبدیل می‌شوند. مدل‌های آزمون با پروفایل آزمون UML مطابقت دارند. مدل آزمون نیز با استفاده از تبدیل‌های نوشته شده به زبان EGL به کد آزمون JUnit که با نحو خانواده زبان‌های xUnit مطابقت دارد، تبدیل می‌شود.

در این پژوهش از نمودار توالی برای نمایش رفتار سیستم در سطح مدل استفاده می‌شود. با توجه به مطالب گفته شده در بخش نمودار توالی، از قابلیت استفاده مجدد نمودارهای توالی برای تولید خودکار موارد آزمون، پشتیبانی می‌شود. در صورتی که نمودار توالی به نمودار دیگری ارجاع دهد، نمودار مربوطه در نمودار اصلی ترکیب و مورد آزمون از ترکیب آن‌ها ایجاد می‌شود. ورودی این روش یک نمودار توالی و خروجی کد آزمون است. شکل ۴ مراحل تولید خودکار موارد آزمون را نشان می‌دهد. این نمودار در دو محور عمودی و افقی مشخص است که در محور عمودی تبدیل PIM به PIT و محور افقی تبدیل PIT به کد آزمون انجام می‌شود (جهت یادآوری این تبدیل‌ها به شکل ۱ رجوع کنید).

[۱] System Under Test

[۲] Epsilon Object Language
[۳] Epsilon Transformation Language
[۴] Epsilon Generation Language



شکل ۳: روش آزمون مدل‌رانده پیشنهادی

روش پیشنهادی به این صورت است که ابتدا بررسی می‌کند آیا نمودار توالی به تعاملات دیگر ارجاع می‌دهد یا خیر. در صورتی که هیچ ارجاعی در نمودار توالی نباشد؛ به عنوان ورودی تولید مدل آزمون و کد آزمون استفاده می‌شود در غیر این‌صورت، همه تعاملاتی که به آن‌ها ارجاع می‌دهد را می‌خواند. در این مرحله نمودارهای توالی با نمودار توالی اصلی ترکیب می‌شوند و یک نمودار توالی واحد برای تولید کد آزمون به وجود می‌آید.

پس از تولید مدل آزمون براساس مدل UTP، این مدل به قالب کد آزمون تبدیل می‌شود، همزمان داده‌های آزمون از فایل XML خوانده می‌شوند و به ازای هر سطر از فایل XML یک مورد آزمون ایجاد می‌شود.

برای تولید موارد آزمون از نمودار توالی، سیستم به عنوان یک جعبه سیاه در نظر گرفته می‌شود. براساس چارچوب ارائه شده در شکل ۴ ابتدا نمودارهای ارجاع شده در نمودار اصلی به آن ترکیب می‌شوند. این تبدیل با زبان ETL انجام می‌شود. خروجی این تبدیل یک نمودار توالی با ترکیب تمام نمودارهای ارجاع شده است و از این نمودار، موارد آزمون به صورت خودکار ایجاد می‌شوند. برای تولید خودکار موارد آزمون از دو تبدیل استفاده می‌شود:

۱- تبدیل مدل به مدل (PIM به PIT) برای تولید خودکار مدل‌های آزمون. مدل طراحی مستقل از سکوی سیستم به مدل آزمون مسقل از سکو تبدیل می‌شود (شکل ۳). این تبدیل، مدل‌های UML را به عنوان ورودی دریافت می‌کند و مدل‌های UTP ایجاد می‌کند.

برای اعمال این تبدیل از قوانین زیر استفاده می‌شود:

قانون ۱: به ازای هر نمودار توالی:

- یک Lifeline با استریوتایپ << TestComponent >> ایجاد می‌شود.
- یک Lifeline با استریوتایپ <<SUT>> به اسم Manager ایجاد می‌شود.
- یک Lifeline با استریوتایپ <<SUT>> به اسم SUT ایجاد می‌شود.

قانون ۲: به ازای هر پیام از Actor به سیستم:

- یک پیام از TestComponent به Manager با همان نام ایجاد می‌شود.
- یک پیام با قالب "Class Name : Function Name" از Manager به SUT ایجاد می‌شود. نام پیام به عنوان Function Name قرار می‌گیرد و از Lifeline مقصد نام Class Name استخراج می‌شود.

قانون ۳: به ازای هر پیام از سیستم به سیستم، پیامی با قالب قانون ۲ از Manager به SUT ایجاد می‌شود.

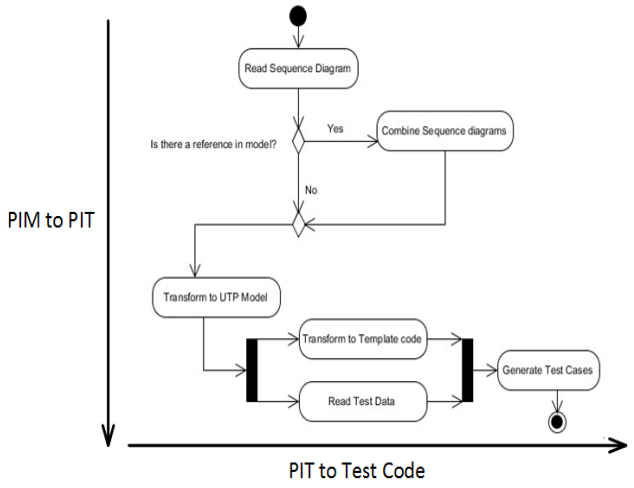
قانون ۴: به ازای هر پیام از LifeLine غیر از سیستم به Actor:

- همان پیام از Manager به TestComponent ایجاد می‌شود.

قانون ۵: به ازای هر پیام از سیستم به Actor همان پیام از Manager به TestComponent ایجاد می‌شود.

قانون ۶: به ازای هر پیام از سیستم به سیستم، همان پیام از Manager به SUT ایجاد می‌شود.

قانون ۷: به ازای هر Combine Fragment بر اساس نوع آن، در مدل آزمون یک Combine Fragment ایجاد می‌شود و پیام‌های درون آن همانند قانون‌های بالا ایجاد می‌شوند.



شکل ۴: مراحل تولید خودکار موارد آزمون

۲- تبدیل مدل به کد (PIT به Test Code) برای تولید خودکار کد آزمون. مدل آزمون مستقل از سکو به کد آزمون تبدیل می‌شود. ورودی این تبدیل مدل آزمون تولید شده در مرحله قبل است و خروجی، کد آزمون است. در این مرحله ابتدا قالب آزمون بدون داده آزمون تولید می‌شود، سپس به ازای هر سطر داده‌ی آزمون یک مورد آزمون ایجاد می‌شود. برای این تبدیل به صورت زیر عمل می‌کنیم:

قانون ۱: به ازای هر پیام از TestComponent به Manager یک تابع با خصوصیت Test@ ایجاد می‌شود.

قانون ۲: به ازای هر پیام از Manager به SUT:

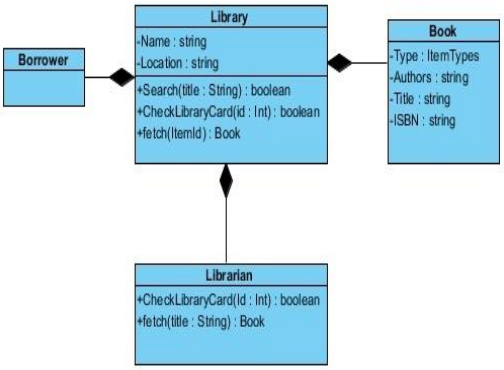
- چون در پیام نام کلاس مشخص است، یک شی از نوع کلاس ایجاد می‌شود.
- نوع خروجی تابع از کلاس دیاگرام خوانده می‌شود و دو متغیر با نوع خروجی تابع و نام پیام بازگشتی برای مقدار واقعی و مقدار مورد انتظار سیستم ایجاد می‌شود.
- یک Assertion برای این دو متغییر ایجاد می‌شود.

قانون ۳: به ازای هر Combine Fragment بر اساس نوع آن کد مربوط به آن را ایجاد می‌شود. شرط‌های این قسمت خوانده می‌شوند و در قالب کد نوشته می‌شوند و درون آن با استفاده از قانون‌های بالا ایجاد می‌شوند.

قانون ۴: به ازای هر سطر از داده‌های آزمون یک مورد آزمون تولید می‌شود.

۴- مطالعه موردی : سیستم کتابخانه

برای نمایش روش ارائه شده از یک سیستم کتابخانه استفاده می‌شود. برای پیاده‌سازی روش ارائه شده بخشی از این سیستم به‌کارگرفته می‌شود. شکل ۵ بخشی از نمودار کلاس را به عنوان معماری سیستم نشان می‌دهد و برای نمایش رفتار سیستم از نمودار توالی استفاده می‌شود که شکل ۶ سناریو امانت گرفتن کتاب را نشان می‌دهد.



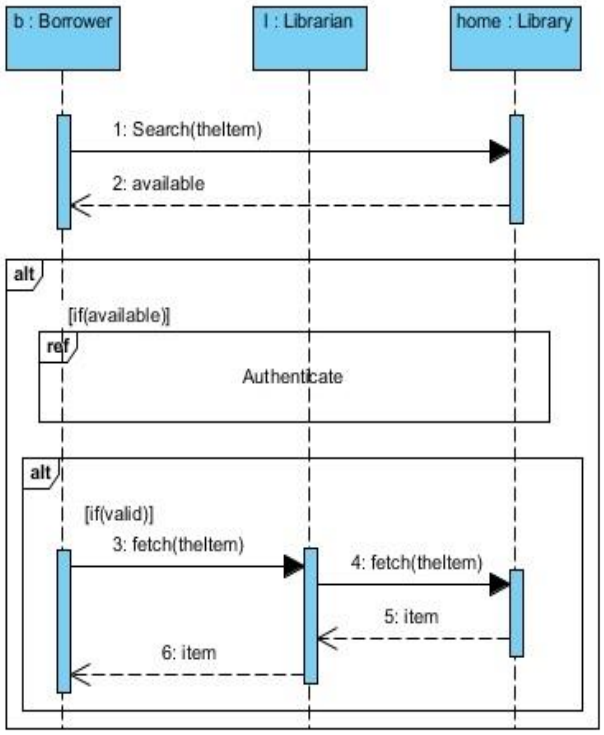
شکل ۵: نمودار کلاس کتابخانه

در نمودار توالی ابتدا فرد متقاضی درخواست خود را به کتابخانه ارسال می‌کند و سیستم بررسی می‌کند که آیتم ارسال شده موجود است یا خیر. در صورت موجود بودن و درخواست برای کتاب مورد نظر، صلاحیت فرد متقاضی بررسی می‌شود، که صحت فرد با ارجاع به یک سناریو دیگر (Authentication) انجام می‌شود (شکل ۷). پس احراز هویت فرد، درخواست به Librarian ارسال می‌شود و Librarian درخواست آن را انجام می‌دهد و آیتم مورد نظر را به فرد تحویل می‌دهد.

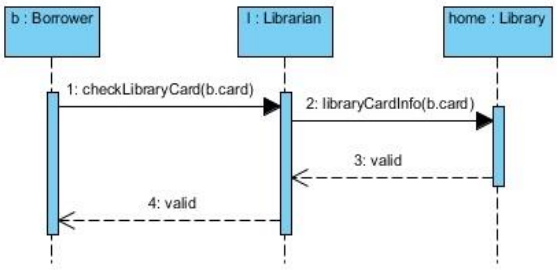
براساس روش ارائه شده در بخش ۳ ابتدا دو نمودار شکل ۶ و ۷ ترکیب می‌شوند که نمودار شکل ۸ حاصل تبدیل می‌باشد. حال یک نمودار واحد برای تولید مدل آزمون و کد آزمون وجود دارد و مطابق با روش آزمون مدل‌رانده ارائه شده در شکل ۳ عمل می‌شود. با توجه به قوانین ذکر شده در بخش تبدیل مدل به مدل (قوانین ۱تا ۷) نمودار توالی به مدل آزمون شکل ۹ تبدیل می‌شود. مدل آزمون تولید شده شکل ۹ به عنوان ورودی تبدیل مدل به کد استفاده می‌شود. مدل آزمون با قوانین بیان شده به قالب کد آزمون با نحو زبان Junit تبدیل می‌شود (شکل ۱۰).

در شکل ۱۰ خطوط ۴ تا ۷ مربوط به پیام Search است که باید جواب واقعی که از سیستم دریافت شده با جواب مورد انتظار که در داده‌های آزمون که در فایل XML ذخیره شده است، بررسی شود. یک نمونه از کلاس Library ایجاد می‌شود و تابع Search فراخوانی می‌شود و به عنوان مقدار واقعی که از سیستم دریافت می‌شود، تفسیر می‌گردد. همچنین یک متغیر برای جواب مورد انتظار تعریف می‌شود و مقدار این دو باهم مقایسه می‌شود (خطوط ۴ تا ۷). در صورتی که آیتم مورد نظر موجود باشد (خط ۹) درخواستی برای احراز هویت صورت می‌گیرد و تابع libraryCardInfo بررسی می‌شود، که این عملیات در خطوط ۱۱ تا ۱۳ انجام می‌شود. در صورتی که اطلاعات فرد معتبر باشد (خط ۱۵) باید آیتم دریافت شده با آیتم مورد انتظار برابر باشند (خطوط ۱۷ تا ۱۹).

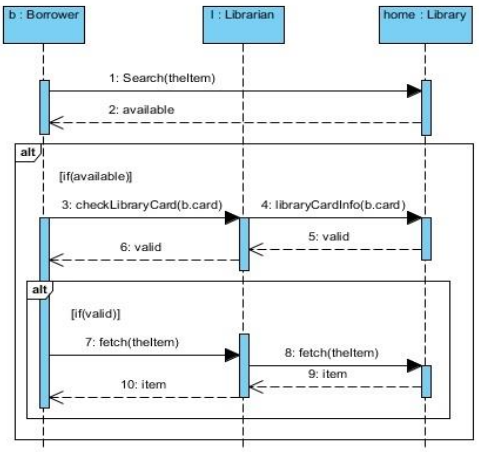
پس از ایجاد قالب آزمون از روی مدل آزمون، این قالب به موارد آزمون قابل اجرا تبدیل می‌شود. حال باید سطرهای آزمون داده در قالب ایجاد شده تکرار شود. به ازای هر سطر آزمون یک مورد آزمون از قالب موجود ایجاد می‌شود. شکل ۱۱ یک نمونه داده آزمون با سه مورد آزمون را نشان می‌دهد. که با استفاده از این فایل سه مورد آزمون ایجاد می‌شود که شکل ۱۲ مورد آزمونی را نشان می‌دهد که از قرار دادن سطر اول داده‌ی آزمون شکل ۱۱ ایجاد شده است.



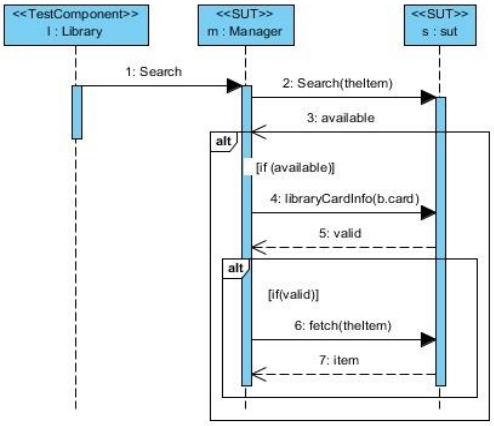
شکل ۶: نمودار توالی برای امانت یک آیتم از کتابخانه



شکل ۷: نمودار توالی برای احراز هویت فرد متقاضی



شکل ۸: نمودار توالی واحد بعد از ترکیب



شکل ۹: مدل آزمون UTP

```
@Test
public void TestCase[#]
{
    Library Library1 = new Library();
    boolean availableActual = Library1.Search([item]);
    boolean availableExpect = [Search];
    assertEquals(true,availableActual,availableExpect);

    if(availableActual)
    {
        boolean validActual = Library1.libraryCardInfo([b.card]);
        boolean validExpect=[valid];
        assertEquals(true,validActual,validExpect);

        if(validActual)
        {
            Item itemActual = Library1.fetch([item]);
            Item itemExpect=[fetch];
            assertEquals(true,itemActual,itemExpect);
        }

    }
}
```

شکل ۱۰: قالب مورد آزمون

۵- مزایای ترکیب نمودارهای توالی

بیشتر تحقیقات صورت گرفته در زمینه تولید خودکار موارد آزمون با استفاده از UTP بر روی آزمون سطح مولفه تمرکز دارند [۱۰،۹،۴]. در این پژوهش با استفاده از ترکیب نمودارهای توالی، موارد آزمون پیچیده‌تری تولید شده است که می‌توانند برای آزمون سطح ادغام استفاده شوند. در آزمون سطح ادغام هدف این است که ادغام مولفه‌ها به درستی اجرا شوند. علاوه بر به-کارگیری در آزمون سطح ادغام مزایای این پژوهش به صورت زیر می‌باشند:

۱- برخی از نمودارهای توالی نمی‌توانند از حالت اولیه اجرا شوند و باید درون نمودارهای توالی دیگری اجرا شوند. بنابراین با ترکیب این نمودارها می‌توان آن‌ها را آزمود.

۲- دومین مزیت این روش این است که ترکیب نمودارهای توالی منجر به طولانی‌تر شدن و کاهش موارد آزمون می‌شود. در برخی محیط‌ها مانند سیستم‌های نهفته، مقداردهی اولیه به موارد آزمون باعث افزایش هزینه‌ها می‌شود. بنابراین با ترکیب چندین مورد آزمون به یک مورد آزمون، هزینه آزمون را کاهش داد.

۳- ترکیب نمودارهای توالی منجر به کشف خطاهایی می‌شود که با اجرای یک نمودار توالی نمی‌توان آن‌ها را پیدا کرد. با این رویکرد خطاهای واسط قابل تشخیص می‌باشند.

```
<DataPool>
  <row>
    <column Name="theitem"> book1 </column>
    <column Name="available"> true </column>
    <column Name="b.card"> idNum1 </column>
    <column Name="valid"> true </column>
    <column Name="item"> book1 </column>
  <row>
  <row>
    <column Name="theitem"> book1 </column>
    <column Name="available"> true </column>
    <column Name="b.card"> idNum2 </column>
    <column Name="valid"> false </column>
    <column Name="item"> book2 </column>
  <row>
  <row>
    <column Name="theitem"> book4 </column>
    <column Name="available"> false </column>
    <column Name="b.card"> idNum1 </column>
    <column Name="valid"> true </column>
    <column Name="item"> book4 </column>
  <row>
</DataPool>
```

شکل ۱۱: دادهای آزمون

```
@Test
public void TestCase1
{
    Library Library1 = new Library();
    boolean availableActual = Library1.Search("book1");
    boolean availableExpect = true;
    assertEquals(true,availableActual,availableExpect);

    if(availableActual)
    {
        boolean validActual = Library1.libraryCardInfo("idNum1");
        boolean validExpect=true;
        assertEquals(true,validActual,validExpect);

        if(validActual)
        {
            Item itemActual = Library1.fetch("book1");
            Item itemExpect="book1";
            assertEquals(true,itemActual,itemExpect);
        }

    }
}
```

شکل ۱۲: یک مورد آزمون

۶- پژوهش‌های مرتبط

در این بخش به تکنیک‌های MBT با استفاده از UTP پرداخته می‌شود. دای [۴] دنباله‌ای از ایده‌ها و مفاهیمی برای استخراج مدل‌های UTP از مدل‌های UML که پایه و اساس متدلوژی آزمون مدل‌رانده است را تشریح می‌کند. مدل‌ها می‌توانند مستقیم به کد آزمون یا مدل آزمون خاص سکو تبدیل شوند. بعد از هر تبدیل مدل طراحی آزمون با خصوصیات خاص آزمون غنی می‌شوند. با این حال هیچ ابزار و پیاده‌سازی برای آن ارائه نشده است.

بوش و همکاران در [۹] یک روش MBT برای تولید مدل‌ها از مدل‌های طراحی ارائه کردند. PIM ها به مدل‌های مستقل سکوی آزمون (PIT) تبدیل می‌شوند. مدل‌های خاص سکوی آزمون (PST) از مدل‌های خاص سکو (PSM) و یا از PIT ها تولید می‌شوند. هم PIT و هم PST براساس UTP می‌باشند.

لامانچا و همکاران در [۱۰] یک رویکرد MDA برای تولید مدل‌های UTP از مدل‌های طراحی و تبدیل مدل‌های UTP به کد آزمون ارائه کرده اند. ایشان براساس چارچوب آزمون مدل‌رانده در [۱۰] موارد آزمون را برای آزمون سطح مولفه ایجاد کرده اند. این چارچوب مدل‌های UML را به عنوان ورودی دریافت می‌کند و با استفاده از زبان تبدیل QVT مدل‌های UTP به صورت خودکار ایجاد می‌شوند. مدل‌های ایجاد شده، مدل‌های آزمون مستقل از سکو می‌باشند که می‌توانند به مدل‌های خاص سکو و کد خاص سکو تبدیل شوند. سپس مدل‌های آزمون با استفاده از زبان تبدیل MOFScript به کد آزمون تبدیل می‌شوند. کد آزمون به دو زبان JUnit (برای برنامه‌های نوشته شده به زبان جاوا) و NUnit (برای برنامه‌های نوشته شده به زبان C#) ایجاد می‌شوند.

۷- نتیجه گیری و پیشنهادات

در این مقاله رویکردی برای تولید خودکار موارد آزمون ارائه شده است که دو تبدیل مدل به مدل و مدل به کد را به صورت خودکار انجام می‌دهد. این تبدیلات با استفاده از زبان اپسیلون انجام می‌شوند که الگوریتم این تبدیلات در قالب قانون‌های تبدیل مشخص شده‌اند. رویکرد، مدل‌هایی که سیستم را تشریح می‌کنند (این مدل‌ها با استفاده نمودارهای UML نشان داده می‌شوند) به عنوان ورودی دریافت می‌کند و مدل‌های آزمون و سپس موارد آزمون تولید می‌کند. موارد آزمون به کد Junit تبدیل و برای برنامه‌های نوشته شده به زبان جاوا اجرا می‌شوند.

هدف اصلی این تحقیق کاهش هزینه توسعه نرم‌افزار است که مزایای این روش نشان می‌دهد با استفاده از این روش هزینه‌های توسعه محصول کاهش می‌یابد و همچنین احتمال خطا برای بازنویسی دوباره موارد آزمون کمتر می‌شود.

در این روش موارد آزمون برای آزمون سطح مولفه ایجاد می‌شوند که در آینده هدف اصلی گسترش این رویکرد برای آزمون ادغام می‌باشد.

مراجع

[1] B. Korel, "Automated software test data generation," in IEEE Transactions on Software Engineering, vol. 16, no.8, pp. 870-879, Aug 1990.

[2] S.R. Dalal, A. Jain, N. Karunanithi, J.M. Leaton, C.M. Lott, G.C. Patton, and B.M. Horowitz, "Model-Based Testing in Practice," Proc. Int'l Conf. Software Eng.,1999.

[3] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing approaches," Softw. Test. Verif. Reliab., vol. 22, no. 5, pp. 297–312, Aug. 2012.

[4] Z. Dai, "Model-Driven Testing with UML 2.0," in Proc. of the 2nd European Workshop on Model Driven Architecture, 2004.

[5] W. N. Robinson and H. G. Woo, "Finding reusable UML sequence diagrams automatically," in IEEE Software, vol.21, no. 5, pp. 60-67. 2004.

[6] Object Management Group (OMG), "Unified Modeling Language (UML): Superstructure, v2.4.1," OMG document: formal/2011-08-06, 2011.

[7] Object Management Group (OMG), "UML Testing Profile v1.2," Tech. Rep. formal/05-07-07, 2013.

[8] D. Kolovos, L. Rose, A. Garcia-Dominguez, and R. Paige, "The Epsilon book," available at http://www.eclipse.org/epsilon/doc/book/, 2015.

[9] M. Busch, R. Chaparadza, Z. R. Dai, A. Hoffmann, L. Lacmene, T. Ngwangwen, G. Ndem, H. Ogawa, D. Serbanescu, I. Schieferdecker, and J. Zander-Nowicka, "Model transformers for test generation from system models," in Conquest 2006. Hanser Verlag, Berlin, 2006.

[10] B. P. Lamancha, M. Polo, D. Caivano, M. Piattini, and G. Visaggio, "Automated generation of test oracles using a model-driven approach," Information and Software Technology, vol. 55, no. 2, pp. 301-319, 2013.