کارگروه مهندسی نرم افزار

راه‌کارهای آزمون خودکار نرم‌افزار

مستند فنی

سنجش و تضمین کیفیت نرم‌افزار

نگارنده:

فاطمه بخشی

ویراستاران:

—

نگارش: ۰/۱/۰

۳ تیر ۱۴۰۳

فهرست مطالب

۱ مقدمه ۱

۲ آزمون و آزمون‌پذیری ۱

۱.۲ تصمیم‌ناپذیری آزمون‌نرم‌افزار ۱

۲.۲ معیارهای کفایت آزمون ۲

۳.۲ آزمون‌پذیری ۲

۴.۲ توسعه آزمون‌پذیر رانه ۴

۳ فنون مؤثر آزمون خودکار پیاده‌سازی (کد) ۴

۱.۳ خودکارسازی آزمون واحد و آزمون یکپارچگی ۴

۲.۳ خودکارسازی آزمون سیستم ۵

۱.۲.۳ فازر ۶

۳.۳ خودکارسازی آزمون کارایی ۷

۴ خودکارسازی آزمون سایر مصنوعات ۷

۱.۴ آزمون نیازمندی‌ها ۷

۲.۴ آزمون طراحی و معماری ۷

۵ نتیجه‌گیری ۷

مراجع ۸

۱ مقدمه

«آزمون نرم‌افزار می‌تواند وجود خطاها را نشان دهد، اما هرگز نبود آنها را تضمین نمی‌کند!»

✠ ادسخر دایکسترا

نرم‌افزار به مثابه هر هر محصول دیگری، نیازمند آزمون و راستی‌آزمایی (درستی‌یابی) است. ماهیت غیرقابل لمس و پیچیدگی ذاتی نرم‌افزار سبب شده تا فرایند آزمون آن، در قیاس با دیگر محصولات مهندسی متفاوت، پیچیده، زمان‌بر و پرهزینه باشد؛ این دشواری‌ها اما از اهمیت موضوع آزمون نمی‌کاهد. از نقطه نظر محاسباتی، آزمون نرم‌افزار یک مسئله تصمیم‌ناپذیر است به همین سبب، خودکارسازی فرآیند آزمون امری اجتناب ناپذیر است. در این سند، ابتدا در بخش ۲ بیان می‌شود که چگونه آزمون‌پذیری و توجه به آن می‌تواند منجر به تسهیل فرانید آزمون و یافتن حداکثر تعداد خطا و دستیابی به کیفیت آزمون بالا شود. سپس، رویکرد جدید **توسعه آزمون‌پذیر رانه** معرفی می‌شود. در بخش ۳ به بررسی مهم‌ترین فنون خودکارسازی فرایند آزمون پرداخته می‌شود.

۲ آزمون و آزمون‌پذیری

۱.۲ تصمیم‌ناپذیری آزمون‌نرم‌افزار

متأسفانه آزمون تنها قادر است وجود خرابی را نشان دهد؛ اما، نبود آن را تضمین نمی‌کند. این مهم را نخستین بار دانشمند هلندی، ادسخر دایکسترا بیان کرد [۱]. به‌بیان صوری، مسئله یافتن تمامی خرابی‌های software under test (SUT)، تصمیم‌ناپذیر[۱] است [۲]. این موضوع، سبب می‌شود به‌دنبال راهکارهایی برای اندازه‌گیری آزمون و تعیین اثربخشی آن باشیم. معیارهای کفایت آزمون، برای این منظور و با هدف کمّی‌سازی و اندازه‌گیری مقدار آزمون انجام شده مطرح شده‌اند؛ هرچند متقابلاً در تولید داده آزمون هم کاربرد دارند. چون با مسئله‌ای تصمیم‌ناپذیر مواجه هستیم، بایستی معیاری وجود داشته باشد که مشخص کند چه زمانی

[۱] undecidable

می‌توانیم آزمون را خاتمه دهیم و در این زمان آزمون تا چه‌حد خوب انجام شده است.

۲.۲ معیارهای کفایت آزمون

Ammann و Offutt [۲] معیارهای کفایت آزمون را تحت عنوان معیارهای پوشش و به‌صورت چهار معیار افراز فضای ورودی[۱]، پوشش گراف[۲]، پوشش منطق[۳] و پوشش ساختار نحوی[۴] مطرح کرده‌اند، که از این میان معیارهای پوشش گراف، استفاده گسترده‌تری دارند.

پوشش گراف در سطح کد اجرایی که به آن پوشش کد هم گفته می‌شود، شامل پوشش گراف جریان کنترل (control flow graph (CFG)) و گراف جریان داده (data flow graph (DFG)) برنامه می‌شود. پوشش منطق، مقداردهی و تعیین ارزش عبارات منطقی ظاهر شده در کد منبع SUT است. افراز فضای ورودی، یعنی انتخاب از بین حالت‌های مختلف ترکیب ورودی‌ها. در نهایت، پوشش ساختار نحوی، استفاده از قوانین گرامر برای اعتبارسنجی[۵] داده‌های ورودی یا تولید داده‌های جدید آزمون را شامل می‌شود.

۳.۲ آزمون‌پذیری

آزمون‌پذیری[۶] یا قابلیت آزمون‌نرم‌افزار، یکی از صفات کیفی نرم‌افزار و بیانگر درجه سهولت آزمون یک مصنوع نرم‌افزاری داده شده، یعنی آسان بودن یافتن خطا در آن است [۳]. چنانچه، مسئله آزمون نرم‌افزار تصمیم‌پذیر بود، نیازی به مفهوم آزمون‌پذیری وجود نداشت؛ زیرا، در هر حال تمامی خطاهای نرم‌افزار توسط یک الگوریتم شناسایی می‌شد. بنابراین آزمون‌پذیری در سیستم‌های نرم‌افزاری، یک مبنای وجودی نظری دارد که آن را از سایر صفات کیفی مجزا می‌کند. در مهندسی سیستم و در حوزه کنترل کیفیت، تعداد قابل توجهی از صفات کیفی تعریف شده است. در حقیقت، این صفات، میزان تحقق نیازمندی‌های غیرعملیاتی سیستم

[۱] input space partitioning
[۲] graph coverage
[۳] logic coverage
[۴] syntax-based coverage
[۵] validation
[۶] testability

را مشخص می‌کنند. صفات کیفی، به دو رده داخلی و خارجی تقسیم می‌شوند [۴]؛ صفات کیفیت داخلی[۱] مشخص می‌کنند که سیستم، چگونه و تا چه‌میزان نیازهای غیر عملیاتی توسعه‌دهندگان خود را ملاقات می‌کند. صفات کیفیت خارجی[۲] مشخص می‌کنند که سیستم، تا چه میزان نیازهای غیر عملیاتی کاربران خود را ملاقات می‌کند. در این دیدگاه، صفاتی مانند آزمون‌پذیری، قابلیت درک[۳]، قابلیت استفاده مجدد[۴]، و قابلیت تغییر[۵] جزو صفات کیفیت داخلی هستند و به‌عنوان مثال، قابلیت اطمینان[۶] را می‌توان، یک صفت کیفیت خارجی به شمار آورد.

در برخی مراجع، صفت کیفیت داخلی به صفاتی اطلاق شده است که به‌صورت مستقیم توسط متریک‌های نرم‌افزار قابل اندازه‌گیری هستند؛ مانند اندازه نرم‌افزار[۷]. در مقابل، صفت کیفیت خارجی، به صفاتی اطلاق شده که به‌صورت مستقیم قابل اندازه‌گیری نیستند [۵]؛ مانند آزمون‌پذیری. در هر دو رده‌بندی، سنجش صفات کیفیت نرم‌افزار، کماکان مسئله‌ای باز محسوب می‌گردد. تمرکز ما در این رساله، بر روی آزمون‌پذیری به‌عنوان یک صفت کیفیت داخلی است و در سطح مصنوعات مختلف کد منبع، طراحی و نیازمندی‌های نرم‌افزار است. به دلیل اهمیت موضوع، تعاریف متعددی از آزمون‌پذیری نرم‌افزار در استانداردهای جداگانه و نیز مشترکِ IEEE، ISO و IET آمده است [۳، ۶–۸]. رویکرد ما در این رساله، بر مبنای تعاریف ذکر شده در استانداردهای *ISO/IEC* 25010:2011 [۶] و *ISO/IEC/IEEE* 24765:2017 [۳] است که به ترتیب بر آزمون‌پذیری کد منبع و طراحی [۶] و نیازمندی‌ها [۳] متمرکز بوده و بروزترین تعاریف در این حوزه هستند. این تعاریف به تفکیک مصنوعات ذکر شده در ذیل آمده‌اند:

تعریف ۱.۲. *آزمون‌پذیری کد منبع و طراحی (مطابق ISO/IEC 25010:2011 [۶]):* «درجه‌ای از اثربخشی[۸]

[۱] internal quality attribute
[۲] external quality attribute
[۳] understandability
[۴] reusability
[۵] changeability
[۶] reliability
[۷] software sizing
[۸] effectiveness

و کارایی[1] که معیارهای آزمون می‌تواند بر اساس آن، برای یک سیستم، محصول یا مؤلفه، بنا گردد و آزمون‌ها می‌توانند برای تعیین اینکه این معیارها برآورده (ملاقات) شده‌اند یا نه، صورت پذیرند».

تعریف ۲.۲. *آزمون‌پذیری نیازمندی‌ها* (*مطابق ISO/IEC/IEEE 24765:2017* [۳]): *«درجه‌ای که می‌توان یک آزمون عینی و شدنی را برای یک نیازمندی داده شده، برای تعیین اینکه آیا یک نیاز برآورده (ملاقات) شده است یا خیر، طراحی کرد».*

در اینجا روش‌هایی برای سنجش و بهبود آزمون‌پذیری نرم‌افزار در سیستم‌های Legacy و یا سیستم‌هایی که با رویکرد TFD توسعه می‌یابند، پیش از هرگونه آزمون نرم‌افزار ارائه می‌گردد. سنجش و بهبود آزمون‌پذیری کد منبع نرم افزار در [۹] بحث شده است. سنجش و بهبود آزمون‌پذیری طراحی نرم‌افزار در [۱۰] بحث شده است. سنجش و بهبود آزمون‌پذیری نیازمندی‌های نرم‌افزار در [۱۱] بحث شده است.

۴.۲ توسعه آزمون‌پذیر رانه

توسعه آزمون‌پذیر رانه راهکار جایگزینی توسعه آزمون‌رانه، برای تولید خودکار نرم‌افزار آزمون شده و باکیفیت است. توسعه آزمون‌پذیر رانه در [۱۲] بحث شده است.

۳ فنون مؤثر آزمون خودکار پیاده‌سازی (کد)

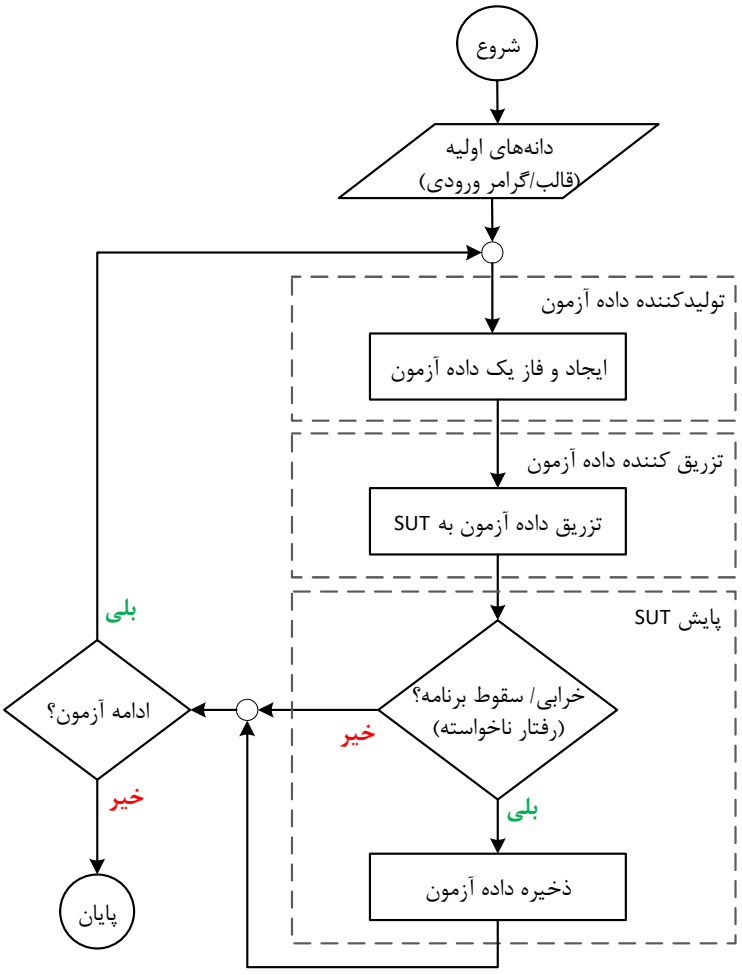
۱.۳ خودکارسازی آزمون واحد و آزمون یکپارچگی

توضیحات ابزارهایی مثل EvoSuite و Pynguin

[1] efficiency

۲.۳ خودکارسازی آزمون سیستم

در این بخش بر روی آزمون خودکار برنامه در سطح سیستم متمرکز می‌شویم. یکی از کارآمدترین فنون برای آزمون خودکار در سطح سیستم، آزمون فازی است. آزمون فازی فرایند ساده تولید و سپس تزریق یک ورودی ناخواسته (بدشکل شده یا نامتعارف) به SUT است. چنان‌چه برنامه بر اثر پردازش این ورودی ناخواسته دچار خرابی شود، حافظه برنامه مورد تحلیل قرار گرفته و خطای احتمالی موجود در کد آشکار می‌گردد. چون SUT با تعداد ورودی‌های بسیار زیادی مورد آزمون قرار می‌گیرد، آزمون فازی را می‌توان نوعی آزمون فشار هم به‌شمار آورد. فرایند معمول آزمون فازی در شکل ۱ نشان داده شده است.



شکل ۱: روندنمای فرایند آزمون فازی در حالت ساده. پیمانه‌های مورد نیاز برای خودکارسازی فرایند با مستطیل خطچین مشخص شده‌اند. شرایط ادامه آزمون می‌تواند بر عهده فرد آزمون‌گر قرار داده شود یا توسط خود فازر تعیین گردد.

۱.۲.۳ فازر

فازر ابزاری است که فرایند آزمون فازی را خودکار می‌کند. پیاده‌سازی هریک از پیمانه‌های شکل ؟؟ و تجمیع آنها در کنار یکدیگر یک فازر را ایجاد می‌کند. تمرکز اصلی در یک فازر، نحوه تولید داده آزمون است به‌گونه‌ای که می‌توان آن را وجه تمایز اصلی فازرهای مختلف دانست. فازری که برای آزمون برنامه‌های با ورودی فایل توسعه داده می‌شود، فازر قالب فایل هم نامیده می‌شود [۱۳]. روش‌های تولید داده در فازرها قابل تفکیک به دو دسته کلی روش‌های مبتنی بر جابه‌جایی یا جهش و روش‌های مبتنی بر تولید هستند [۱۴].

راه‌کارهای مطرح در فن آزمون فازی [۱۵–۱۸]، برای شناسایی خطاها و آسیب‌پذیری‌ها نیازمند تولید تعداد زیادی ورودی یا همان داده آزمون[۱] هستند. برای نرم‌افزارهایی با ساختار ورودی ساده، تولید داده آزمون نیز ساده است. به‌عنوان مثال می‌توان با روش تصادفی این کار را انجام داد؛ اما، در نرم‌افزارهایی با ساختار ورودی پیچیده، مانند فایل با قالب مشخص تولید داده آزمون متنوع که بتواند مسیرهای اجرایی بیشتری را پوشش دهد، کار آسانی نیست.

آزمون نرم‌افزارهایی که ورودیشان فایل است با مشکل اساسی تشخیص ساختار فایل مواجه هستند. بایستی بتوان ابزاری تولید نمود که با توجه به ساختار و قالب‌بندی فایل، قادر به تولید داده‌های آزمون باشد. داده‌های آزمونی که هم محتوا و هم قالب‌های مورد استفاده را چنان تغییر دهند که بتوان کلیه مسیرهای اصلی اجرایی برنامه را بررسی و پوشش داد. در این راستا دو برنامه گیج کننده یا در اصطلاح فازر به نام‌های AFL [۱۹] و learn&fuzz [۲۰] ارایه شده‌اند که نسبت به فازرهای قبلی از هوشمندی بیشتری برای تولید داده‌های آزمون، برخوردار هستند.

AFL [۱۹] با استفاده از یک فرایند تکاملی، جمعیت تصادفی از دانه‌های اولیه[۲]، را آن‌چنان اصلاح می‌کند که فایل‌های حاصل هنگامی که به برنامه مورد نظر به عنوان ورودی داده شوند در مجموع تعداد بیشتری از دستورالعمل‌های برنامه پوشش داده شود. چالش عمده این روش این است که چون بدون دانشِ ساختار فایل، برای مثال فایل PDF، مبادرت به ایجاد فایل جدید از ترکیب قبلی‌ها می‌نماید؛ در عمل فایل حاصل

[۱] data Test
[۲] seeds Initial

ممکن است برچسب‌ها و اشیای مورد نیاز برای ساختار مورد نظر را نداشته باشد و در نتیجه غیر قابل استفاده باشد.

۳.۳ خودکارسازی آزمون کارایی

۴ خودکارسازی آزمون سایر مصنوعات

۱.۴ آزمون نیازمندی‌ها

۲.۴ آزمون طراحی و معماری

مصنوعات نرم‌افزاری در مراحل مختلف توسط گراف قابل توصیف هستند. در نتیجه می‌توان از فنون تحلیل شبکه در مهندسی نرم‌افزار استفاده کرد. هدف آشنایی با کاربرد دانش شبکه و تحلیل گراف، در مهندسی نرم‌افزار است.

۵ نتیجه‌گیری

مراجع

[1] E. Dijkstra, “Notes on structured programming,” *APIC studies in data processing*, vol.8, 1972.

[2] P. Ammann and J. Offutt. *Introduction to software testing*. Cambridge University Press, 2016.

[3] ISO/IEC/IEEE, “ISO/IEC/IEEE 24765:2017 Systems and software engineering — Vocabulary,” [Online]. Available: https://www.iso.org/standard/71952.html, 2017.

[4] S. Freeman and N. Pryce. *Growing object-oriented software, guided by tests*. Addison-Wesley, 2010.

[5] K. O. Elish and M. Alshayeb, “Using software quality attributes to classify refactoring to patterns,” *Journal of Software*, vol.7, feb 2012.

[6] ISO/IEC, “ISO/IEC 25010:2011 systems and software engineering — systems and software quality requirements and evaluation (SQuaRE) — system and software quality models,” [Online]. Available: https://www.iso.org/standard/35733.html, 2011.

[7] ISO/IEC/IEEE, “ISO/IEC/IEEE 29148:2018(en) Systems and software engineering — Life cycle processes — Requirements engineering,” [Online]. Available: https://www.iso.org/obp/ui#iso:std:iso-iec-ieee:29148:ed-2:v1:en, 2018.

[8] ISO/IEC/IEEE, “ISO/IEC/IEEE 12207:2017(en) Systems and software engineering — Software life cycle processes,” [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:12207:ed-1:v1:en, 2017.

[9] M. Zakeri-Nasrabadi and S. Parsa, “An ensemble meta-estimator to predict source code testability,” *Applied Soft Computing*, vol.129, p.109562, 11 2022.

[10] M. Zakeri-Nasrabadi, S. Parsa, and S. Jafari, “Measuring and improving software testability at the design level,” *Information and Software Technology*, vol.174, p.107511, 10 2024.

[11] M. Zakeri-Nasrabadi and S. Parsa, “Natural language requirements testability measurement based on requirement smells,” *Neural Computing and Applications*, 4 2024.

[12] S. Parsa, M. Zakeri-Nasrabadi, and B. Turhan, “Testability-driven development: An improvement to the tdd efficiency,” *Computer Standards & Interfaces*, vol.91, p.103877, 1 2025.

[13] M. Sutton, A. Greene, and P. Amini. *Fuzzing: brute force vulnerability discovery*. Addison-Wesley Professional, 2007.

[14] C. Chen, B. Cui, J. Ma, R. Wu, J. Guo, and W. Liu, “A systematic review of fuzzing techniques,” *Computers and Security*, vol.75, pp.118–137, 2018.

[15] B. P. Miller, L. Fredriksen, and B. So, “An empirical study of the reliability of Unix utilities,” *Commun. ACM*, vol.33, pp.32–44, Dec. 1990.

[16] B. Miller, D. Koski, C. Lee, V. Maganty, R. Murthy, A. Natarajan, and J. Steidl, “Fuzz revisited - a re-examination of the reliability of Unix utilities and services,” *October*, vol.1525, no.October 1995, pp.1–23, 1995.

[17] J. E. Forrester and B. P. Miller, “An empirical study of the robustness of Windows NT applications using random testing,” in *Proceedings of the 4th Conference on USENIX Windows Systems Symposium - Volume 4*, WSS’00, (Berkeley, CA, USA), pp.6–6, USENIX Association, 2000.

[18] B. P. Miller, G. Cooksey, and F. Moore, “An empirical study of the robustness of MacOS applications using random testing,” in *Proceedings of the 1st International Workshop on Random Testing*, RT ’06, (New York, NY, USA), pp.46–54, ACM, 2006.

[19] M. Zalewsky, “American fuzzy lop,” [Online]. Available: http://lcamtuf.coredump.cx/afl/, [Accessed: 2020-06-29].

[20] P. Godefroid, H. Peleg, and R. Singh, “Learn&fuzz: machine learning for input fuzzing,” in *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*, ASE 2017, (Piscataway, NJ, USA), pp.50–59, IEEE Press, 2017.