رویکردی جدید مبتنی بر سنجه‌های نرم‌افزاری جهت افزایش سودمندی آزمون بازگشت

مجتبی وحیدی اصل[1]، استادیار؛ محمدرضا دهقانی تفتی[2]، کارشناسی ارشد نرم‌افزار؛ علیرضا خلیلیان[3]، مربی و دانشجوی دکتری نرم‌افزار
۱- دانشکده مهندسی و علوم کامپیوتر- دانشگاه شهید بهشتی-تهران- ایران - mo.dehghani@mail.sbu.ac.ir
۲- دانشکده مهندسی و علوم کامپیوتر- دانشگاه شهید بهشتی-تهران- ایران - mo_vahidi@sbu.ac.ir
۳- گروه مهندسی نرم‌افزار - دانشکده مهندسی کامپیوتر- دانشگاه اصفهان- اصفهان- ایران - khalilian@eng.ui.ac.ir

چکیده: اولویت‌دهی آزمایه فنی است که اغلب برای کاهش هزینه‌های آزمون بازگشت نرم‌افزار استفاده شده است. فنون فعلی سعی کرده‌اند با کمک اطلاعات مختلف پوشش کد، قدرت آشکارسازی خطای هر آزمایه را تخمین بزنند و سپس با روشی ابتکاری آن‌ها را رتبه‌بندی نمایند. اما مطالعه‌ها نشان داده‌اند که پوشش همبستگی قوی با سودمندی آزمایه‌ها و قدرت آن‌ها در آشکارسازی خطا ندارد. با تکیه بر مطالعه‌هایی که اثربخشی سنجه‌های کد را در پیش‌بینی خطاها نشان داده‌اند، حدس می‌زنیم که می‌توان از اطلاعات حاصل از سنجه‌های کد برای طراحی فن موثری جهت اولویت‌دهی آزمایه‌ها بهره‌برداری نمود. بر مبنای این فرضیه، در این مقاله فن جدیدی برای اولویت‌دهی پیشنهاد می‌شود که بر اساس امتزاج داده روی اطلاعات سنجه‌های پیچیدگی کد کار می‌کند. نوآوری این تحقیق اینست که قدرت آشکارکنندگی خطای آزمایه‌ها را در اولویت-دهی با نگاه جدیدی تخمین می‌زند. برای ارزیابی فن پیشنهادی، آزمایش‌هایی روی نسخه‌های خطادار هفت برنامه محک جاوا انجام دادیم. در آزمایش‌ها کارایی اولویت‌دهی اغلب حداقل ۷۰٪ بر حسب متوسط درصد آشکارسازی خطا مشاهده شد که این نتیجه فرضیه ما را معتبر می‌نماید.

واژه‌های کلیدی: آزمون نرم‌افزار، آزمون بازگشت، اولویت‌دهی آزمایه، سنجه‌های کد.

A New Approach Based on Software Metrics to Improve the Effectiveness of Regression Testing

Mohammad-Reza Dehghani-Tafti, Master of Software Engineering[1], Mojtaba Vahidi-Asl, Assistant Professor[2], Alireza Khalilian, Instructor and PhD candidate in Software Engineering[3]

1- Faculty of Computer Science and Engineering, Shahid Beheshti University G.C., Tehran, Iran, mo.dehghani@mail.sbu.ac.ir
2- Faculty of Computer Science and Engineering, Shahid Beheshti University G.C., Tehran, Iran, mo_vahidi@sbu.ac.ir
3- Software Engineering Department, Faculty of Computer Engineering, University of Isfahan, Isfahan, Iran, khalilian@eng.ui.ac.ir

Abstract: Test case prioritization has been often used to alleviate the costs associated with software regression testing. Current techniques have attempted to estimate the fault exposing potential of test cases using code coverage information and rank them using a heuristic. However, studies show that coverage does not strongly correlate with the effectiveness and fault exposing potential of test cases. Relying on the results of studies that demonstrated the effectiveness of code metrics in fault prediction, we speculate that code metric information can be leveraged to design a new effective technique for test case prioritization. Based on our hypothesis, in this paper, a new prioritization technique is proposed that works based on data fusion on code complexity metrics. The novelty of our technique lies in its original viewpoint to estimate fault exposing potential of test cases in prioritization. To evaluate the proposed technique, we have conducted experiments on faulty versions of seven Java benchmarks. In the experiments, we often observed at least 70% performance in prioritization measured in terms of average percentage of fault detection, which validates our hypothesis.

Keywords: Software Testing, Regression Testing, Test Case Prioritization, Software Metrics.

تاریخ ارسال مقاله:
تاریخ اصلاح مقاله:
تاریخ پذیرش مقاله:
نام نویسنده مسئول: مجتبی وحیدی اصل
نشانی نویسنده مسئول: ایران – تهران – دانشگاه شهید بهشتی – دانشکده مهندسی و علوم کامپیوتر

۱- مقدمه

آزمون بازگشت[۱] یکی از فعالیت‌های مهم در مرحله نگهداری نرم‌افزار است. همزمان با بزرگ‌تر شدن پروژه‌ها و افزایش تعداد آزمایه‌ها[۲]، آزمون بازگشت پرهزینه‌تر شده و انجام آزمون با اجرای همه‌ی آزمایه‌ها هم غیرعملی می‌گردد [۱۴]. طبق گزارش‌ها [۱۷-۱۵]، آزمون بازگشت ۸۰٪ از بودجه‌ی آزمون نرم‌افزار را به خود اختصاص می‌دهد و اجرای برخی از مجموعه‌های آزمون بیش از هفت هفته به طول می‌انجامد. برای مثال، محصول نرم‌افزاری یک شرکت مجموعه آزمونی با بیش از ۳۰ هزار آزمایه داشته است. این آزمایه‌ها به بیش از ۱۰۰۰ ماشین-ساعت زمان برای اجرا و صدها ساعت برای نظارت مهندسان بر فرآیند آزمون بازگشت نیاز داشته‌اند [۱۸].

برای کاهش هزینه‌های آزمون بازگشت فنون متعددی پیشنهاد شده است. یکی از فنون اثربخش، اولویت‌دهی آزمایه[۳] است. اولویت‌دهی سعی می‌کند به شیوه‌ای قدرت آشکارکنندگی خطای آزمایه‌ها را تخمین بزند و آن‌ها را بر این اساس رتبه‌بندی نماید. هدف اینست که آزمایه‌های مؤثرتر زودتر اجرا شوند و خطاها سریعتر آشکار گردند.

شمار بسیاری از فنون موجود قدرت آشکارکنندگی خطا را بر حسب معیارهای مختلف پوشش[۴] تخمین می‌زنند. متأسفانه تکیه بر پوشش مشکلاتی دارد. اغلب خطاهای پنهان در نقاطی بسیار خاص از کد اتفاق می‌افتند که پوشش بالای کد آزمایه، لزوماً منجر به یافتن آن‌ها نمی‌شود. ضمن آن‌که اطلاعات پوشش اولیه‌ی یک آزمایه با حرکت به سوی نسخه‌های بعدی نرم‌افزار، کم اثرتر می‌شوند و در جایی ممکن است گمراه‌کننده شوند. در نتیجه، بهبود آزمون بازگشت با در نظر گرفتن اطلاعات پوشش کد همیشه اثربخش نیست و نمی‌تواند به خوبی خطاهای ناشی از تغییرات را آشکار سازد [۱۹]. اختصاصاً نشان داده شده [۲۰] که پوشش همبستگی قوی با سودمندی مجموعه آزمون و قدرت آشکارکنندگی خطای آن ندارد.

مشکل دیگر این است که استفاده از اطلاعات پوشش از نسخه‌های قبلی، اثربخشی آزمون بازگشت برای نسخه‌های بعدی را کاهش می‌دهد. نشان داده شده است [۳] که افزایش فاصله بین نسخه‌ای از نرم‌افزار که برای جمع‌آوری اطلاعات پوشش استفاده می‌شود با نسخه‌ای که می‌خواهیم آزمایه‌ها را برای آن نسخه اولویت‌دهی کنیم، باعث کاهش قابل ملاحظه سودمندی فنون اولویت‌دهی می‌گردد. نتایج تحقیق لو[۵] و همکاران [۳]، اهمیت استفاده از اطلاعات پوشش به‌روز و جدید برای آزمون بازگشت را نشان می‌دهد.

انگیزه تحقیق حاضر این است که چگونه می‌توان با درنظرگرفتن اطلاعات دیگر در کنار اطلاعات پوشش، اثربخشی فن اولویت‌دهی را افزایش دهیم. با تکیه بر مطالعه‌هایی [۲۴-۲۱] که اثربخشی سنجه‌های کد را در پیش‌بینی خطاها و خرابی‌ها نشان داده‌اند، حدس می‌زنیم که می‌توان روش ابتکاری طراحی کرد که با اطلاعات حاصل از سنجه‌های کد، اولویت‌دهی مؤثری انجام دهد.

بر مبنای این فرضیه، در این مقاله فن جدیدی برای اولویت‌دهی پیشنهاد می‌شود. این فن با استفاده از امتزاج داده[۶] [۵] روی اطلاعات سنجه‌های پیچیدگی کد طراحی شده است. هدف ما این‌است که فن اولویت‌دهی طراحی کنیم که با کمک اطلاعات حاصل از سنجه‌های پیچیدگی و کیفیت کد، تخمین بهتری از قدرت آشکارکنندگی خطای آزمایه‌ها داشته باشد. نوآوری این تحقیق اینست که قدرت آشکارکنندگی خطای آزمایه‌ها را در اولویت‌دهی با نگاه جدیدی تخمین می‌زند.

برای ارزیابی فن پیشنهادی، از هفت برنامه محک[۷] جاوا استفاده شده است. این برنامه‌ها از آخرین تحقیقات معتبر [۴] در آزمون بازگشت گرفته شده‌اند. فن پیشنهادی روی نسخه‌های خطادار برنامه‌های محک اجرا شده و کارایی اولویت‌دهی بر اساس معیار متوسط درصد آشکارسازی خطا (APFD[۸]) محاسبه شده است. این معیار به‌طور گسترده‌ای در ادبیات آزمون بازگشت مورد استفاده قرار گرفته است. در آزمایش‌ها کارایی اولویت‌دهی اغلب حداقل ۷۰٪ بر حسب متوسط درصد آشکارسازی خطا مشاهده شد که این نتیجه فرضیه ما را معتبر می‌نماید.

به‌طور خلاصه، نوآوری‌های این مقاله عبارتند از:

- پیشنهاد استفاده از سنجه‌های پیچیدگی کد برای اولویت‌دهی آزمایه‌ها در آزمون بازگشت
- ترکیب مجموعه آزمون‌های اولویت‌دهی شده توسط سنجه‌های برتر به‌وسیله‌ی یکی از روش‌های امتزاج داده
- آزمایش‌های تجربی برای اولویت‌دهی روی هفت برنامه محک جاوا و مقایسه با روش پوشش کامل
- بحثی در نتایج به‌دست آمده و عوامل زمینه‌ای در موفقیت روش پیشنهادی

ساختار ادامه مقاله به این شرح است: در بخش دوم مفاهیم لازم برای درک مقاله و بیان مسئله ارائه می‌شود. روش پیشنهادی در بخش سوم تشریح می‌گردد. بخش چهارم به شرح ارزیابی، آزمایش‌ها، نتایج به‌دست آمده و بحث در نتایج می‌پردازد. در بخش پنجم برخی از مهمترین کارهای مرتبط را مرور می‌کنیم و مقاله با نتیجه‌گیری در بخش ششم به پایان می‌رسد.

۲- مفاهیم لازم و بیان مسئله

آزمون بازگشت یکی از فعالیت‌های رایج برای تضمین کیفیت نرم‌افزار است. این آزمون پس از هر بار تغییر کد انجام می‌شود تا اطمینان حاصل کنیم تغییرات خطایی ایجاد نکرده باشند. همچنین بررسی می‌کند قسمت‌هایی از نرم‌افزار که درست کار می‌کردند تحت تأثیر منفی قرار نگرفته باشند. برای این منظور آزمایه‌هایی روی برنامه اجرا می‌شود که آن‌ها را مجموعه آزمون بازگشت[۹] می‌نامند. به‌طور ساده، آزمایه عبارتست از مقادیر ورودی مورد نیاز برای تکمیل اجرای نرم‌افزار تحت آزمون، خروجی‌های مورد انتظار و سازوکاری برای ارزیابی خروجی برنامه تحت آزمون (اوراکل آزمون) [۳۴]. از آنجائیکه آزمون بازگشت

بارها تکرار می‌شود، هزینه اجرایی مجموعه آزمون بازگشت زیاد می‌شود. یکی از روش‌های کاراتر کردن آزمون بازگشت اولویت‌دهی آزمایه‌هاست؛ یعنی آزمایه‌ها را طوری مرتب کنیم که آن‌هایی که شانس آشکارسازی خطا دارند، زودتر اجرا شوند. تعریف دقیق اولویت‌دهی چنین است [۱۵]:

ورودی: یک مجموعه آزمون T، مجموعه‌ای از جایگشت‌های T موسوم به PT، و تابعی از PT به اعداد حقیقی، $f = PT \to \mathbb{R}$.

مسئله: یافتن $T' \in PT$ به طوری که:

$(\forall T'')(T'' \in PT)(T'' \neq T')[f(T') \geq f(T'')]$.

تابع f بیانگر کمیتی است که به دنبال بیشینه کردن آن هستیم.

از آن‌جا که مسئله‌ی اولویت‌دهی، NP سخت است، از روش‌های ابتکاری برای حل آن استفاده می‌شود. برای اولویت‌دهی معمولاً اطلاعاتی از برنامه استخراج می‌شود تا به‌کمک آن‌ها قدرت آشکارکنندگی خطای هر آزمایه را تخمین بزنند. یکی از اطلاعات رایج، پوشش کد بوده است. پوشش کد معمولاً بر اساس معیارهای مختلفی مانند شرط صورت می‌گیرد. در این صورت، پوشش شرط آزمایه t یعنی دستورهای شرطی که پس از اجرای t روی برنامه اجرا می‌شوند.

در مسئله اولویت‌دهی، f تعداد خطاهایی است که هرچه زودتر توسط آزمایه‌ها آشکار شده‌اند. برای تخمین f معمولاً از معیار APFD استفاده می‌شود. این معیار میانگین وزن‌داری از درصد خطاهایی که در طول عمر مجموعه آزمون کشف شده‌اند را اندازه‌گیری می‌کند. محدوده مقادیر APFD در بازه صفر تا ۱۰۰ قرار دارد. مقادیر بزرگ‌تر نشان‌دهنده‌ی نرخ تشخیص خطای بالاتر (بهتر) است. اگر بخواهیم به طور رسمی‌تر APFD را تعریف نماییم، فرض کنید T مجموعه آزمونی باشد که شامل n آزمایه است و F مجموعه‌ای از m خطا است که توسط T آشکار می‌شوند. فرض کنید TF_i، اولین آزمایه‌ای باشد که در ترتیب T' از T، خطای i را آشکار می‌سازد. مقدار APFD برای مجموعه آزمون T' طبق رابطه‌ی (۱) محاسبه می‌شود [۱۵]:

$$APFD = 1 - \frac{TF1 + TF2 + \ldots + TFm}{nm} + \frac{1}{2n} \qquad (۱)$$

۳- روش پیشنهادی

در این بخش فرضیه و رویکرد پیشنهادی ارائه می‌گردد. فرضیه اصلی این مقاله این است که تغییرات برنامه منجر به تغییر مقادیر سنجه‌ها می‌شوند و چنین تغییراتی باعث می‌شود امتیاز و اولویت آزمایه‌ها نیز تغییر کند. در نتیجه، آزمون بازگشت به تغییرات حساسیت نشان می‌دهد. به‌عبارت دیگر، محل رخداد تغییرات باعث می‌شود مقادیر بعضی از سنجه‌ها برای آزمایه‌های پوشاننده آن محل‌ها تغییر کند و این تغییر بر امتیاز و اولویت آن آزمایه‌ها اثر بگذارد. در نتیجه رویکرد پیشنهادی علاوه بر نسخه اصلی برنامه، روی نسخه‌های تغییرکرده برنامه قابل استفاده است.

ما تصدیق می‌کنیم که روش پیشنهادی در این مقاله ارتباط معنایی بین معیارها و سنجه‌های پیچیدگی ساختار یک برنامه (در هر بن‌انگاره[۱۰] برنامه‌سازی همچون شی‌گرایی) و آزمون بازگشت را مورد بررسی و ملاحظه قرار نداده است. در حقیقت روش پیشنهادی این مقاله اولین اکتشاف ما در این زمینه است. قطعاً مطالعه‌های نظری و تجربی بیشتر لازم است که با تعیین این ارتباط‌های معنایی، روابط ریاضی مؤثرتری برای بهره‌برداری و ترکیب سنجه‌ها طراحی نماید.

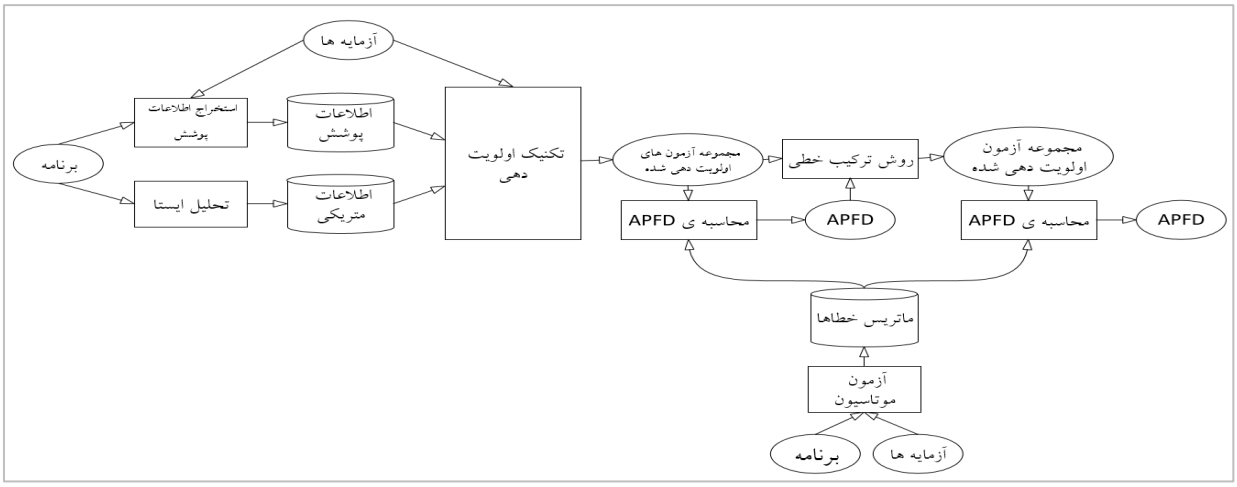
چهارچوب روش پیشنهادی در شکل (۱) نشان داده شده است. مرحله اول روش پیشنهادی نیازمند جمع‌آوری اطلاعاتی از کد است. برای این کار آزمایه‌ها بر روی برنامه‌ها اجرا می‌شوند. خروجی این عمل، میزان پوشش عناصر مختلف به ازای آزمایه‌های برنامه می‌باشد؛ یعنی مشخص می‌شود که هر آزمایه چه عناصری از برنامه را پوشش داده است. سپس اطلاعات سنجه‌های کد برنامه در سطح کلاس و متد استخراج می‌شوند. محدوده روش پیشنهادی برنامه‌های زبان‌های شی‌گرا است هر چند برای سایر زبان‌ها هم قابل استفاده است.

در ادامه این دو نوع اطلاع مبتنی بر پوشش و مبتنی بر سنجه با یکدیگر ترکیب می‌شوند. به‌عبارتی مشخص می‌شود که آزمایه‌ها، چه عناصری از برنامه را پوشش داده‌اند و مقدار سنجه کد برای آن عناصرِ پوشش داده شده چقدر است. سپس امتیاز هر آزمایه به ازای هر سنجه قابل محاسبه است. به ازای هر سنجه، جمع مقدار سنجهِ عناصر پوشش داده شده توسط هر آزمایه، امتیاز آن آزمایه محسوب می‌شود. اکنون به ازای هر سنجه، به آزمایه‌ها امتیازی داده شده است. پس از مرتب‌سازیِ امتیازهای داده شده، به ازای هر سنجه، مجموعه آزمون مرتب شده‌ای داریم. ارزیابی مجموعه آزمون‌های مرتب شده توسط سنجه‌ها معمولاً توسط معیار APFD صورت می‌گیرد.

برای ارزیابی روش پیشنهادی نیازمند نسخه‌های خطادار هستیم. این خطاها توسط ابزارهای آزمون جهش در برنامه کاشته می‌شوند. کاشت این خطاها باعث ایجاد نسخ تغییریافته می‌شود. عمل کاشت خطا و رسیدن به نسخ خطادار در قسمت آزمون موتاسیون در شکل (۱) انجام می‌شود. پس از ارزیابی، سنجه‌هایی که از لحاظ APFD برتر بوده‌اند را مشخص می‌کنیم. سپس برای بهبودِ مقدار APFD به ازای یک برنامه، از مجموعه آزمون‌های سنجه‌های برتر استفاده کرده و آن‌ها را با روش ترکیب خطی ترکیب می‌کنیم تا مجموعه آزمون مرتب شده‌ی جدید و بهبودیافته‌ای حاصل شود. در بخش‌های بعدی جزئیات بیشتری از اجزای روش پیشنهادی ارائه می‌شود.

۳-۱- جمع‌آوری اطلاعات پوشش آزمون

در این مرحله اطلاعات پوشش آزمایه‌های برنامه محاسبه می‌شود. این کار توسط ابزار جمع‌آوری پوشش کد انجام می‌گردد. این ابزار آزمایه‌ها را بر روی کد برنامه اجرا می‌کند و در خروجی، به ازای آزمایه‌های مختلف، میزان پوشش عناصر مختلف برنامه را مشخص می‌کند. بدین معنا که به ازای هر آزمایه مشخص می‌شود که چه عناصری از برنامه را در سطوح مختلف کلاس و متد پوشش داده است.



شکل ۱: نمای کلی راهکار پیشنهادی

۳-۲- جمع‌آوری اطلاعات سنجه‌های کد

شواهد موجود در مطالعه‌های قبل [۲۷-۲۲] نشان می‌دهد که می‌توان از سنجه‌های کد برای پیش‌بینی خطاها و خرابی‌ها و نیز خطاخیزی[11] کد استفاده کرد. به‌عبارتی می‌توان اطلاعات مفیدی را از مسیر توسعه‌ی نرم‌افزار استخراج کرد و با استفاده از سنجه‌های کد، اولویت‌دهی آزمایه‌ها را بهبود داد. مزیت این اطلاعات این است که بدون نیاز به اجرای آزمایه‌ها و فراکدگذاری[12] به دست می‌آید. سنجه‌ها رابطه‌ی مستقیم با تعداد خطاها در عناصر برنامه دارند. به‌عبارتی عنصری از برنامه که مقدار هر یک از سنجه‌هایش از مقدار سنجه متناظر عنصر دیگری از برنامه بیشتر باشد، احتمال خطاخیز بودن آن عنصر بیشتر است. با این اوصاف می‌توان از اطلاعات سنجه‌ها برای اولویت‌دهی آزمایه‌ها استفاده کرد.

در این مرحله اطلاعات سنجه‌های کد را از برنامه استخراج می‌کنیم. برای این کار از ابزارهای تحلیل ایستا استفاده می‌کنیم که اطلاعات سنجه‌های کد را به صورت ایستا از برنامه استخراج می‌کنند. ابزارهای مختلف برای استخراج این اطلاعات، کتاب‌خانه‌ها و واسط‌هایی را در اختیار برنامه نویسان قرار می‌دهند. بدین صورت که می‌توان اطلاعات سنجه‌های مختلف عناصر کد را از طریق ارسال درخواست به آن ابزارها، به دست آورد.

سنجه‌های کد را برای زبان‌های شی‌گرا می‌توان به دو دسته‌ی سنجه‌های سطح متد و سطح کلاس تقسیم‌بندی کرد [۲۳، ۲۴]. پس می‌توان به ازای هر برنامه دو نوع بردار در نظر گرفت. یک بردار بیان‌گر اطلاعات سنجه‌های سطح کلاس برنامه و دیگری بیان‌گر اطلاعات سنجه‌های سطح متد برنامه است. ما از ۱۱ سنجه در سطح متد و ۱۶ سنجه در سطح کلاس استفاده می‌کنیم. این سنجه‌ها به‌ترتیب در جدول‌های (۱) و (۲) خلاصه شده‌اند.

در انتخاب زیرمجموعه‌ای از سنجه‌ها، ما تصمیم گرفتیم سنجه‌هایی را انتخاب کنیم که در مطالعه‌های خطاپروری [۲۳، ۲۴] مورد استفاده قرار گرفته‌اند. در حقیقت این مقاله‌ها برای پیش‌بینی خطاها و خرابی‌های نرم‌افزار در زمینه مورد بررسی‌شان از این سنجه‌ها استفاده کرده‌اند. ما تصدیق می‌کنیم که این تصمیم بسیار ابتدایی است و ممکن است بعضی از سنجه‌ها با هم هم‌پوشانی داشته باشند یا بعضی از سنجه‌های مهم‌تر نادیده گرفته شده باشند. با این حال، هدف اصلی پیشنهاد این مقاله و آزمایش‌های آن، معتبرکردن این ایده است که سنجه‌های کد در زمینه اولویت‌دهی آزمایه‌ها قابل استفاده‌اند. مطالعه‌های تکمیلی مورد نیاز است که سنجه‌های بیشتری را مورد بررسی قرار دهد، همبستگی سنجه‌ها را به‌دست آورد و زیرمجموعه موثرتری بخصوص برای هر دامنه برنامه‌سازی انتخاب نماید.

جدول ۱: سنجه‌های پیچیدگی کد در سطح متد

نام	توضیح
(FANIN) CountInput	تعداد ورودی‌های یک متد
(FANOUT) CountOutput	تعداد خروجی‌های یک متد
(NPATH) CountPath	تعداد مسیرهای منحصر به فرد داخل بدنه‌ی یک متد
CountPathLog	لگاریتم در مبنای ۱۰ سنجه NPATH
CountStmt	تعداد دستورالعمل‌های اعلانی و اجرایی
Knots	تعداد تقاطع‌های جریان کنترلی
MinEssentialKnots	تعداد knot های کد کاهش یافته‌ی برنامه
MaxNesting	حداکثر تعداد سطوح تودرتویی ساختارهای کنترلی در یک متد
Cyclomatic Complexity	تعداد نقاط تصمیم در کد به علاوه‌ی یک
CyclomaticStrict	سنجه قبلی به علاوه‌ی تعداد رخدادهای and و or در کد
Essential Complexity	Cyclomatic complexity کد کاهش یافته برنامه

۳-۳- ترکیب اطلاعات پوشش و اطلاعات سنجه

در این مرحله اطلاعات جمع‌آوری شده در مرحله‌ی قبل را با یکدیگر ترکیب می‌کنیم، یعنی ترکیب ماتریس پوشش آزمون با بردار سنجه‌های کد. برای این کار باید سنجه‌هایی که در سطح کلاس هستند را با ماتریس پوشش سطح کلاس ترکیب کنیم و سنجه‌هایی که در سطح متد هستند را با ماتریس پوشش سطح متد ترکیب نماییم. این ماتریس

نشان می‌دهد که هر آزمایه چه عناصری از برنامه را پوشش داده است و مقدار سنجه مورد نظر برای آن عنصر برنامه چقدر است.

جدول ۲: سنجه‌های پیچیدگی کد در سطح کلاس

نام	توضیح
PercentLackOfCohesion (LCOM/LOCM)	یک منهای (انسجام بین داده‌های یک کلاس و متدهای آن)
(DIT) MaxInheritanceTree	عمق یک کلاس در سلسله مراتب وراثت
(NOC) CountClassDerived	تعداد زیرکلاس‌های مستقیم یک کلاس
(CBO) CountClassCoupled	تعداد کلاس‌های وابسته به یک کلاس
(IFANIN) CountClassBase	تعداد کلاس‌های پدر مستقیم یک کلاس
CountDeclClassMethod	تعداد متدهای یک کلاس
(NV) CountDeclClassVariable	تعداد متغیرهای یک کلاس
AvgCyclomaticStrict	میانگین CyclomaticStrict از تمامی متدهای یک کلاس
(AMS) AvgLineCode	میانگین تعداد خطوط متدهای یک کلاس
(NM) CountDeclMethodAll	تعداد متدهای یک کلاس شامل متدهایی هم که به ارث برده
CountDeclMethodDefault	تعداد متدهای محلی عمومی
(NPRM) CountDeclMethodPrivate	تعداد متدهای خصوصی که به ارث برده نشده‌اند
MaxCyclomatic	بیشینه‌ی پیچیدگی سیکلوماتیک متدهای یک کلاس
MaxEssential	بیشینه‌ی Essential Complexity متدهای یک کلاس
(WCM) SumCyclomatic	مجموع پیچیدگی سیکلوماتیک تمامی متدهای کلاس
SumEssential	مجموع Essential Complexity متدهای یک کلاس

به‌عنوان نمونه، فرض کنید سه آزمایه داشته باشیم و برنامه ما دو کلاس و دو متد داشته باشد. آزمایه‌ها را در سطرهای ماتریس و نام کلاس‌ها و متدها را در ستون‌های ماتریس قرار می‌دهیم. در این‌صورت درایه سطر i و ستون j نشان می‌دهد آزمایه i اُم آیا متد یا کلاس j را در اجرا پوشش داده است یا خیر (۱ یا ۰ به‌ترتیب). سپس به ازای درایه‌های ۱ (یعنی پوشش داده شده) مشخص می‌کنیم که مقدار انواع سنجه‌های کلاسی/متدی به‌ازای آن عنصر j و آزمایه i چقدر است.

فرض کنید آزمایه‌ی T_i عناصری از برنامه را پوشش می‌دهد که مقدار سنجه بیشتری نسبت به آزمایه‌ی T_j دارد که عناصر تحت پوشش آن، مقدار سنجه متناظر کمتری دارند. با توجه به ارتباط بین سنجه‌ها و خطاخیزی، آزمایه‌ی T_i پتانسیل بیشتری در تشخیص خطاها نسبت به آزمایه‌ی T_j دارد. پس می‌توان از اطلاعات سنجه‌ها برای اولویت‌دهی آزمایه‌ها استفاده کرد.

برای این کار، به ازای هر آزمایه، مقدار سنجه‌های عناصر برنامه را در صورتی که آن عنصر توسط آن آزمایه پوشش داده شده باشد، با هم جمع می‌زنیم. مقدار حاصل برابر است با امتیازی که آن سنجه به آن آزمایه داده است؛ به عبارتی این عدد قدرت تشخیص خطای آن آزمایه از دید آن سنجه تخمین می‌زند. اکنون می‌توان با توجه به امتیازی که هر سنجه برای هر آزمایه داده است، اولویت‌دهی را انجام داد.

۳-۴- ساخت ماتریس خطاها

برای ارزیابی مجموعه آزمون‌های مرتب شده توسط سنجه‌ها، نیازمند خطاهای کاشته شده در برنامه هستیم. این خطاها را توسط ابزارهای آزمون جهش به دست می‌آوریم. ابزارهای آزمون جهش نیاز به برنامه و مجموعه آزمون آن دارند. این ابزارها این دو قلم اطلاعات را دریافت می‌کنند و با استفاده از عملگرهای جهشی اقدام به کاشتن خطا در برنامه می‌نمایند. یعنی با استفاده از این عملگرها، در برنامه تغییراتی ایجاد می‌شود و به عنوان خطا در نظر گرفته می‌شوند. سپس به ازای هر خطای کاشته شده، آزمایه‌ها اجرا می‌گردند و آن آزمایه‌هایی که قادر به تشخیص آن خطا باشند، مشخص می‌شوند. بنابراین با استفاده از این ابزار، خطاهای مختلفی در برنامه کاشته می‌شوند و به ازای هر خطای کاشته شده مشخص می‌شود که کدام آزمایه‌ها توانسته‌اند آن خطا را تشخیص دهند. در نتیجه ماتریسی داریم که نشان می‌دهد هر آزمایه کدام‌یک از خطاهای کاشته شده در برنامه را آشکار می‌کند.

۳-۵- ارزیابی مجموعه آزمون‌های اولویت‌دهی شده

برای ارزیابی مجموعه آزمون‌های مرتب شده توسط سنجه‌ها، از معیار APFD مشابه مقالات قبلی [۳، ۴] استفاده می‌کنیم. این تحقیقات در مرحله‌ی ارزیابی، برای شبیه‌سازی نسخه‌های بعدی خطادار، از خطاهای کاشته شده با استفاده از ابزار جهش استفاده کرده‌اند. بدین منظور، به تعداد خاصی در آخرین نسخه از نرم‌افزار، خطا کاشته می‌شود. سپس این خطاهای کاشته شده در تعداد دسته‌های ثابتی به نام گروه خطا تقسیم‌بندی می‌شوند. پس هر گروه خطا شامل تعداد خاصی خطای کاشته شده است. همچنین تضمین می‌شود که خطاهای انتخابی می‌توانند حداقل توسط یک آزمایه تشخیص داده شوند و گروه‌های مختلف خطا، خطاهای یکسان ندارند. در نتیجه به تعداد گروه‌های خطا، نسخه‌های خطادار داریم. حال توانایی تشخیص خطای مجموعه آزمون‌های مرتب شده به ازای سنجه‌ها را با استفاده از معیار APFD بر روی این نسخه‌های خطادار، اندازه‌گیری می‌کنیم.

همچنین برای ارزیابی نیاز به مقایسه کردن مجموعه آزمون‌های تولید شده با فنون کنترلی (مبنای مقایسه) داریم. در این مقاله اولویت‌دهی پوشش کامل به‌عنوان فن کنترلی در نظر گرفته می‌شود [۱، ۲]. اولویت‌دهی پوشش کامل به آزمایه‌ای که متدهای بیشتری از برنامه را پوشش دهد، اولویت بیشتری تخصیص می‌دهد. گاهی فنون نتایج خود را با اولویت‌دهی تصادفی نیز مقایسه می‌کنند: روشی است که آزمایه‌ها را به صورت تصادفی در یک ترتیب قرار می‌دهد.

اکنون APFD مجموعه آزمون‌های مرتب شده توسط سنجه‌ها و مجموعه آزمون‌های مرتب شده توسط فنون کنترلی را بر روی نسخه‌های خطادار مذکور، محاسبه می‌کنیم. سپس مقادیر APFD حاصل از مجموعه آزمون‌های مرتب شده توسط سنجه‌ها را با استفاده از آزمون

فرض آماری با مقادیر APFD فن کنترلی مقایسه می‌کنیم. این کار را با استفاده از الگوریتمی که در شکل (۲) آمده است انجام می‌دهیم.

```
Algorithm: Statistical Testing
     Input:  list of APFD for metric technique and control technique
     Output: recognize which one is better
1:   procedure STATISTICALTESTING (metricAPFD, controlAPFD)
2:       if Kolmogrov-Smirnov-test(metricAPFD, controlAPFD) is normal
3:           if p-value of t-test (metricAPFD, controlAPFD) < 0.05
4:               reject the null hypothesis
5:               return p-value and the better one
6:           else
7:               return the means are equal
8:       else
9:           if p-value of Wilcoxon-test (metricAPFD, controlAPFD) < 0.05
10:              reject the null hypothesis
11:              return p-value and the better one
12:          else
13:              return the means are equal
14:      end if
15:  end procedure
```

شکل ۲: الگوریتم آزمون فرض آماری

برای انجام آزمون فرض آماری، ابتدا لازم است تا سطح معنی‌داری آماری تعیین شود. مطابق با عرف ادبیات، سطح معنی داری آماری را ۰/۰۵ در نظر گرفته‌ایم. سپس مطابق شکل ۲، الگوریتم آزمون فرض، دو لیست از مقادیر APFD فن سنجه و فن کنترلی را دریافت می‌کند. ابتدا با استفاده از آزمون کولموگروف- اسمیرنوف [۳۷] مشخص می‌کنیم که آیا هر دو از توزیع نرمال برخوردار هستند یا خیر. در واقع فرض صفر برای این آزمون، توزیع نرمال داشتن مقادیر دو لیست است. در صورت تأیید فرض صفر، با استفاده از آزمون t به بررسی مقایسه‌ی میانگین دو لیست می‌پردازیم. در این آزمون، فرض صفر برابر بودن میانگین‌های دو لیست است. در صورت کوچک‌تر بودن *p-value* از سطح معنی‌داری ۰/۰۵، فرض صفر رد می‌شود و مشخص می‌کنیم که کدام‌یک از دو لیست میانگین بهتری دارند. در غیر اینصورت، تفاوت معناداری بین میانگین‌ها وجود ندارد و فرض صفر پذیرفته می‌شود.

اگر فرض صفر برای آزمون کولموگروف- اسمیرنوف رد شود، معنایش این است که داده‌ها توزیع نرمال ندارند و باید از روش‌های آزمون ناپارامتری مثل آزمون ویلکاکسون [۳۷] برای مقایسه‌ی میانگین‌ها استفاده کنیم. در اینجا نیز فرض صفر برابر بودن میانگین‌های دو لیست است. در صورت کوچک‌تر بودن *p-value* از سطح معنی داری ۰/۰۵، فرض صفر رد می‌شود و مشخص می‌کنیم که کدام‌یک از دو لیست میانگین بهتری دارند. در غیر این‌صورت، تفاوت معناداری بین میانگین‌ها وجود ندارد و فرض صفر پذیرفته می‌شود. بنابراین مشخص می‌شود که کدام یک از سنجه‌ها توانسته‌اند مجموعه آزمون مرتب شده‌ی بهتری را نسبت به فن کنترلی تولید کنند.

۳-۶- ترکیب سنجه‌ها

اکنون از بین سنجه‌هایی که در قسمت قبل نسبت به فن کنترلی نتیجه‌ی بهتری داشته‌اند، k تا از بهترین‌هایشان را انتخاب می‌کنیم. ایده‌ی اصلی این است که هر کدام از سنجه‌ها از جنبه‌ای به عناصر ساختاری برنامه (کلاس یا متد) نگاه می‌کنند و از آن جنبه به لحاظ خطاخیزی به آن‌ها امتیاز می‌دهند. ما با استفاده از یکی از فنون امتزاج داده، به ترکیب اطلاعات این سنجه‌های برتر می‌پردازیم تا اولویت‌دهی آزمایه‌ها به صورت بهتری انجام گیرد. در سال‌های اخیر فنون امتزاج داده در حوزه‌های مختلفی برای بهبود اثربخشی استفاده شده‌اند [۵].

ما در این تحقیق از روش ترکیب خطی که یکی از روش‌های امتزاج داده است، استفاده کرده‌ایم [۵]. در واقع می‌خواهیم اولویت هر آزمایه را با توجه به اولویت‌دهی‌هایی که سنجه‌های برتر به آزمایه‌ها داده‌اند، به دست بیاوریم. برای این کار باید میزان اهمیت هر سنجه را بدانیم. بنابراین هدف، محاسبه‌ی میزان اهمیت هر سنجه در امتیازی که به آزمایه‌ها می‌دهد، می‌باشد. در این روش امتیاز هر آزمایه به صورت رابطه‌ی (۲) محاسبه می‌شود.

$$Score(tc_i) = \sum_{j=1}^{k} w_j \times s_{ij} \qquad (۲)$$

که در رابطه‌ی بالا، tc_i آزمایه‌ی i ام است. مقدار k برابر تعداد سنجه‌های برتر می‌باشد. w_j برابر وزن تخصیص یافته به سنجه j ام با استفاده از روش ترکیب خطی است. سادگی و هزینه کم محاسباتی یکی از دلایل انتخاب ترکیب خطی است. به‌علاوه چون در این تحقیق میزان اهمیت و تاثیر سنجه‌ها مورد بررسی قرار نگرفته است، استفاده از ترکیب خطی اولین انتخاب منطقی به‌نظر می‌رسد. s_{ij} نیز برابر با مقدار نرمال شده‌ی سنجه j ام برای آزمایه‌ی i ام است. نرمال‌سازی محدوده مقادیر را در بازه بین صفر و یک قرار می‌دهد. اکنون w_j ها مجهول هستند.

برای یافتن w_j ها لازم است معیاری به نام همپوشانی بین دو مجموعه‌ی مرتب شده معرفی کنیم. میزان همپوشانی دو مجموعه‌ی مرتب شده می‌تواند معیاری از شباهت یا همبستگی آن دو مجموعه باشد. یکی از روش‌ها برای یافتن میزان همپوشانی بین دو مجموعه که توسط وو[۱۳] [۵] معرفی شده است، به صورت رابطه‌ی (۳) می‌باشد:

$$o\_rate_{ij} = \frac{\left| L_i \cap L_j \right|}{n} \qquad (۳)$$

در این رابطه o_rate_{ij} نرخ همپوشانی بین دو مجموعه است و L_i نیز p٪ اول از مجموعه‌ی مرتب شده‌ی i ام و L_j برابر است با p٪ اول از مجموعه‌ی مرتب شده‌ی j ام. صورت کسر برابر است با تعداد اعضای اشتراک این دو مجموعه. مخرج کسر نیز برابر است با تعداد آزمایه‌های متفاوتی که در p٪ اول از کل k مجموعه‌ی مرتب شده وجود دارند.

در ادامه لازم است به ازای هر سنجه بدانیم که این سنجه با سایر سنجه‌ها به صورت میانگین چقدر همپوشانی یا همبستگی دارد. بدین منظور میزان همبستگی هر سنجه نسبت به سایر سنجه‌ها طبق رابطه‌ی (۴) محاسبه می‌شود:

$$o\_rate_{avg}(j) = \frac{1}{k-1} \sum_{i=1,2,\ldots,k, i \neq j} o\_rate_{ij} \qquad (۴)$$

حال سنجه‌ای که مقدار o_rate_{avg} آن به عدد یک نزدیک‌تر باشد، در واقع اشتراکی از سایر سنجه‌ها است و رأی مستقلی در تصمیم‌گیری نهایی ندارد. وزن هر سنجه طبق رابطه‌ی (۵) به دست می‌آید:

$$w_j = 1 - o\_rate_{avg}(j) \quad (۵)$$

طبق رابطه‌ی بالا آن سنجه‌ای که همبستگی کمتری با بقیه دارد، در اولویت‌دهی نهایی تعیین کننده‌تر است و وزن بیشتری می‌گیرد. اکنون وزن هر کدام از سنجه‌ها به دست آمده است. حال طبق رابطه، امتیازی که روش ترکیب خطی به آزمایه‌ها می‌دهد قابل محاسبه است. اما قبل از آن لازم است تا امتیازی که توسط سنجه‌ها به آزمایه‌ها داده شده‌اند، نرمال‌سازی شوند. برای نرمال‌سازی از روش خطی صفر-یک که در رابطه‌ی (۶) آمده است، استفاده می‌کنیم [۵]:

$$s_{ij} = \frac{MetricScore(i,j) - \min(MetricScore(j))}{\max(MetricScore(j)) - \min(MetricScore(j))} \quad (۶)$$

که MetricScore(i, j) امتیازی است که سنجه jام به آزمایه‌ی i ام داده و min(MetricScore(j)) برابر است با حداقل امتیازی که سنجه j ام به آزمایه‌ها داده است. همچنین max(MetricScore(j)) برابر است با حداکثر امتیازی که سنجه j ام به آزمایه‌ها داده است و S_{ij} نیز برابر است با امتیاز نرمال شده‌ای که سنجه jام به آزمایه‌ی i ام داده است.

حال با استفاده از رابطه‌ی (۲)، امتیاز آزمایه‌ها توسط روش ترکیب خطی به دست می‌آید. اکنون آزمایه‌ها را با استفاده از امتیازهای به دست آمده اولویت‌دهی می‌کنیم و مجموعه آزمون مرتب شده‌ی را با k مجموعه‌ی دیگری که مربوط به k سنجه برتر بودند مقایسه می‌کنیم.

۴- ارزیابی

به منظور ارزیابی فن پیشنهادی، ابتدا الگوریتم‌های معرفی شده پیاده-سازی می‌شوند و آزمایش‌هایی انجام می‌شوند. در آزمایش‌ها به دنبال پاسخ به سؤال‌های تحقیقی زیر هستیم:

سؤل اول: اولویت‌دهی با سنجه‌ها در مقایسه با روش‌های کنترلی (مبنای مقایسه) چگونه عمل می‌کند؟

سؤال دوم: سنجه‌های نرم‌افزاری بر اساس ریزدانگی‌شان چه تأثیری در اولویت‌دهی دارند؟

سؤال سوم: چگونه می‌توان اطلاعات این سنجه‌ها را به منظور بهبود اثربخشی، ترکیب کرد و این ترکیب چگونه عمل می‌کند؟

سؤال چهارم: استفاده از سنجه‌های پیشنهادی در اولویت‌دهی آزمایه‌ها در چه شرایطی خوب جواب می‌دهد و در چه شرایطی گمراه کننده است؟

گام‌های ارزیابی فن پیشنهادی عبارت‌اند از:

جمع‌آوری اطلاعات پوشش آزمون: این کار را با استفاده از ابزار جمع‌آوری پوشش کد انجام می‌دهیم. برای جمع‌آوری اطلاعات پوشش یک برنامه باید تغییراتی را در فایل تنظیمات برنامه اعمال کنیم تا ابزار بتواند اطلاعات پوشش را از برنامه جمع‌آوری کند. ابزار، آزمایه‌ها را بر روی برنامه اجرا می‌کند و سپس اطلاعات پوشش در سطح کلاس و متد به ازای هر برنامه ذخیره می‌گردد. ابزار و تنظیمات مورد نیاز برای انجام این کار در ادامه ذکر می‌شود.

جمع‌آوری اطلاعات سنجه‌های کد: این کار را با استفاده از ابزار تحلیل ایستا انجام می‌دهیم. بدین منظور از طریق برنامه‌سازی به آن ابزار درخواست می‌دهیم که اطلاعات سنجه‌ای برنامه را در اختیارمان قرار دهد. سپس این اطلاعات به ازای هر برنامه در سطح متد و کلاس ذخیره می‌گردد. ابزار مورد نیاز برای انجام این کار در بخش بعدی ذکر می‌شود.

ترکیب اطلاعات پوشش و اطلاعات سنجه‌ای: به همان روشی که در بخش روش پیشنهادی بیان شد، اطلاعات به دست آمده در دو مرحله‌ی قبل با یکدیگر ترکیب می‌شوند و نتیجه ذخیره می‌گردد.

ساخت ماتریس خطاها: این کار با استفاده از ابزار جهش انجام می‌گیرد. ابزار جهش با دریافت برنامه و مجموعه آزمون متناظر با آن، اقدام به کاشت خطا در برنامه می‌کند. سپس با اجرا کردن آزمایه‌ها بر روی کد خطادار، مشخص می‌کند که هر آزمایه کدام خطاهای کاشته شده را توانسته تشخیص دهد. این اطلاعات را در قالب ماتریسی که در بخش روش پیشنهادی بیان شد، ذخیره می‌کنیم.

اولویت‌دهی آزمایه‌ها بر اساس سنجه‌ها: اکنون بر اساس اطلاعاتی که در گام قبل به دست آوردیم و طبق روشی که در روش پیشنهادی بیان شد، به ازای هر کدام از سنجه‌ها یک مجموعه آزمون مرتب شده داریم.

مقایسه عملکرد مجموعه آزمون‌های مرتب شده بر اساس سنجه‌ها با فن کنترلی (مبنای مقایسه): در این گام به ازای هر برنامه، ۵۰۰ خطا با استفاده از ابزار جهش کاشته می‌شود. سپس نسخه‌های خطادار از آخرین نسخه‌ی هر برنامه را از طریق گروه‌بندی کردن هر پنج خطا با یکدیگر، ایجاد می‌کنیم؛ یعنی برای هر برنامه به طور تصادفی این ۵۰۰ خطا را در ۱۰۰ گروه خطا که هر کدام شامل پنج خطای تصادفی است، تقسیم‌بندی می‌کنیم. سپس مجموعه آزمون‌های مرتب شده از فنون مختلف را بر روی این ۱۰۰ نسخه‌ی خطادار از هر برنامه اعمال می‌کنیم و معیار APFD را محاسبه می-کنیم. اکنون با انجام آزمون فرض آماری و با استفاده از الگوریتم آزمون فرض ، به مقایسه‌ی نتایج این دو فن می‌پردازیم.

ترکیب سنجه‌های برتر از طریق روش ترکیب خطی: در این مرحله ابتدا لازم است تا سنجه‌های برتر به ازای هر برنامه را محاسبه کنیم؛ سنجه‌هایی که مجموعه آزمون‌هایی را تولید کرده‌اند که از لحاظ APFD برتر بوده‌اند. تعداد این سنجه‌ها به ازای هر برنامه به صورت پارامتریک تعیین می‌شود. سپس از طریق روش ترکیب خطی، به ترکیب این سنجه‌ها به منظور افزایش مقدار APFD می‌پردازیم. مقدار پارامتر p برای رابطه‌ی (۳)، برای تمامی برنامه‌ها برابر ۵۰٪ در نظر

گرفته شده است. در پایان مجموعه آزمون مرتب شده‌ی حاصل از ترکیب سنجه‌ها را با سنجه‌های برتر مقایسه می‌کنیم.

۴-۱- ابزارهای مورد استفاده

در این تحقیق برای جمع‌آوری اطلاعات پوشش، از ابزار پوشش کد Jacoco[۱۴] استفاده کرده‌ایم که کتابخانه‌ی استخراج پوشش کد برای جاوا می‌باشد. این ابزار از طریق فراکدگذاری بر روی کد میانی برنامه‌ی جاوا، اطلاعات پوشش برنامه را استخراج می‌نماید. همانطور که گفته شد برای جمع‌آوری اطلاعات پوشش یک برنامه باید تغییراتی را در فایل تنظیمات برنامه اعمال کنیم تا ابزار بتواند اطلاعات پوشش را از برنامه جمع‌آوری کند. از آنجا که برنامه‌های محک مورد استفاده در این تحقیق از Maven[۱۵] به عنوان ابزار ساخت و مدیریت پروژه استفاده می‌کنند، برای اینکه Jacoco بتواند راه‌اندازی شود، باید افزونه‌ی Jacoco را در فایل pom.xml هر پروژه پیکربندی کنیم.

برای جمع‌آوری اطلاعات سنجه‌ای از ابزار Understand[۱۶] استفاده شده است. این ابزار برای محاسبه‌ی سنجه‌های کد مناسب است که در مقالات متعددی از جمله [۹-۶] برای محاسبه‌ی سنجه‌ها در حوزه‌های مختلف استفاده شده است.

برای ایجاد خطا در برنامه‌ها و ساختن ماتریس خطاها، از ابزار PIT[۱۷] استفاده شده است. این ابزار در تحقیقات زیادی به منظور انجام آزمون جهش به کار گرفته شده است. مزیت این ابزار این است که برای یافتن سریع‌تر آزمایه‌هایی که خطاهای کاشته شده را تشخیص می‌دهند، از ابزار پوشش کد Jacoco استفاده می‌کند؛ بدان معنا که ابتدا با استفاده از Jacoco مشخص می‌کند که هر آزمایه چه خطوطی از برنامه را پوشش داده است. سپس به ازای هر خطای کاشته شده، تنها آن آزمایه‌هایی را اجرا می‌کند که خط متناظری که آن خطا در آن خط کاشته شده را پوشش می‌دهند. این کار باعث تسریع در عملکرد آزمون جهش می‌شود. ما از خطاهای کاشته شده‌ی آماده در تحقیق لو و همکاران [۳] با استفاده ازابزارPIT تولید شده‌اند، استفاده کرده‌ایم.

برای جمع‌آوری داده‌های مورد نیاز برای الگوریتم آزمون فرض از نرم‌افزار IBM SPSS Statistics 23[۱۸] استفاده شده است. نرم‌افزار IBM SPSS از جمله نرم‌افزارهایی است که برای تحلیل‌های آماری به صورت بسیار گسترده‌ای استفاده می‌شود. علت استفاده از این نرم‌افزار توانایی آن در انجام تحلیل‌های آماری و خروجی‌های مناسبی است که در اختیار کاربر قرار می‌دهد. همچنین برای پیاده‌سازی الگوریتم آزمون فرض از زبان برنامه‌سازی VBA و نرم‌افزار اِکسل نسخه‌ی ۲۰۱۳ استفاده شده است. برای یکپارچه کردن اطلاعات به دست آمده از ابزارها و پیاده‌سازی روش پیشنهادی از جاوا در محیط Intellij IDEA نسخه‌ی ۲۰۱۶ استفاده شده است. پیاده‌سازی و اجرای برنامه‌های این ارزیابی همگی توسط ماشینی با مشخصات زیر انجام شده است: سیستم عامل ویندوز ۱۰، پردازنده چهار هسته Intel Core i7-4710HQ 2.50GHz و 2.49GHz و حافظه ۱۲ گیگابایت.

۴-۲- برنامه‌های محک و ویژگی‌های آن‌ها

ما ارزیابی‌ها بر روی هفت برنامه‌ی جاوا که به طور گسترده در تحقیقات آزمون بازگشت مورد استفاده قرار گرفته‌اند [۳، ۴، ۱۲-۱۰]، انجام داده‌ایم. برای هر برنامه، مجموعه آزمون JUnit متناظر، اوراکل‌ها، راه‌اندازها و اسکریپت‌های لازم توسط محققان فراهم شده و در مخزن گیت‌هاب[۱۹] موجود هستند. تعداد آزمایه‌ها و تعداد خطوط مربوط به هر کدام از برنامه‌ها در جدول (۳) آمده است:

جدول ۳: ویژگی برنامه‌های محک

برنامه	apns	assertj	jasmine	jopt	la4j	metrics	scribe
آزمایه‌ها	۸۵	۲۴۶۲	۱۰۲	۷۲۷	۶۲۵	۱۴۹	۹۸
خطوط	۱۳۶۲	۵۵۴۴۳	۱۶۴۰	۶۶۳۶	۸۰۹۴	۱۱۴۷۷	۲۴۹۷

۴-۳- نتایج آزمایش‌ها

نتایج حاصل از اجرای الگوریتم آزمون فرض روی مقادیر APFD برنامه‌های محک در جدول (۴) آمده است. هر سلول این جدول حاصل مقایسه اولویت‌دهی با سنجه در سطر متناظر و فن پوشش کامل را نشان می‌دهد. در این جدول سطرهای ۱ تا ۱۱ مربوط به سنجه‌های سطح متد، از شماره‌ی ۱۲ تا ۲۷ مربوط به سنجه‌های سطح کلاس و سطر ۲۸ مربوط به فن اولویت‌دهی تصادفی است. هر سلول از این دو جدول حاوی چهار نوع اطلاعات است.در صورتی که فن مورد نظر از فن کنترلی بهتر باشد، با علامت ✔، در صورتی که بهتر نباشد با علامت ✘ و در صورتی که تفاوت قابل توجهی بین این دو فن وجود نداشته باشد با علامت O مشخص شده است. در صورتی که آزمون فرض آماری با ویلکاکسون باشد، علامت W و در صورتی که آزمون t باشد با حرف T مشخص شده است. مقدار *p-value* نیز در هر سلول آمده است. همچنین اگر مقدار *p-value* از سطح معنی‌داری ۰/۰۵ کمتر باشد، سلول به رنگ تیره در آمده است.

۴-۴- ترکیب سنجه‌های برتر

در این قسمت، سنجه‌های برتری که در مرحله‌ی قبل شناسایی شدند را با روش ترکیب خطی، ترکیب می‌کنیم. نتایج حاصل به ازای ۷ برنامه در قسمت‌های ۱ تا ۷ شکل (۳) در قالب نمودار جعبه‌ای آمده‌اند. به ازای هر برنامه، اسامی سنجه‌های برتر به ازای آن برنامه در زیر شکل از چپ به راست آمده‌اند. سنجه‌های سطح کلاس با (C) مشخص شده‌اند. آخرین جعبه در هر نمودار که در سمت راست قرار دارد و با نام LC مشخص شده است، مربوط به نتیجه‌ی ترکیب سنجه‌های برتر می‌باشد. هر جعبه در هر نمودار پراکندگی ۱۰۰ مقدار APFD حاصل از اولویت‌دهی بر اساس یک سنجه را در چارک‌ها نشان می‌دهد. چارک دوم و سوم در مربع/مستطیل وسط هر جعبه نشان داده می‌شود و خط درون آن نشان‌دهنده میانه داده‌هاست یا همان انتهای چارک دوم. دم پایین مربوط است به چارک اول داده‌ها و دوم بالا مربوط است به

چارک چهارم داده‌ها. کوچکتر بودن عرض مربع/مستطیل میانی و در کل کوچکتر بودن فاصله دو دم دلالت بر مرکزیت بیشتر داده‌ها دارد.

همچنین، قسمت ۸ در شکل (۳)، نمودار Q-Q را به‌ازای دو سنجه LC و MaxCyclomatic(C) در برنامه scribe نشان می‌دهد. محور عمودی در این نمودار به مقادیر APFD های حاصل از ترکیب سنجه‌های برتر و محور افقی به مقادیر APFD های سنجه MaxCyclomatic(C) اختصاص دارد. آن‌چنان که دیده می‌شود، در محدوده ۰/۷ تا ۰/۹ عملکرد ترکیب سنجه‌ها و سنجه منفرد تقریباً یکسان بوده است. اما در محدوده ۰/۵ تا ۰/۷ گاهی عملکرد یکی از دو حالت بهتر یا بدتر می‌گردد. برای نمونه، در قسمت ۷ از شکل (۳) و در مورد دو جعبه سمت راست، مشاهده می‌شود که پراکندگی مقادیر APFDهای سمت راست‌ترین جعبه اندکی کمتر است و میانه آن نیز اندکی بالاتر از جعبه سمت چپ آن است.

۴-۵- بحث در نتایج

جدول (۴) نشان می‌دهد که برای بیشتر برنامه‌ها، سنجه‌ها به نسبت فن پوشش کامل، بهتر عمل می‌کنند. ولی نحوه‌ی عملکرد سنجه‌ها نسبت به فن تصادفی در مقایسه با فن پوشش کامل بهتر می‌باشد. برای پی بردن به علت عدم عملکرد بهتر سنجه‌ها نسبت به فن کنترلی، امتیازهایی که سنجه‌ها به آزمایه‌ها داده‌اند، بررسی گردید.

در مواردی که سنجه‌ها نسبت به فن کنترلی عملکرد بهتری نداشتند، مشاهده شد که تعداد حالت‌های تساوی خیلی زیاد است؛ به عبارت بهتر، تعداد زیادی آزمایه هستند که سنجه‌ها، امتیاز یکسانی به آن‌ها داده‌اند. در واقع چنین سنجه‌هایی با ساختار و ویژگی‌های این برنامه تناسب ندارند. برای مثال سنجه عمق وراثت برای برنامه‌ای که ویژگی وراثتی زیادی ندارد، مناسب نیست. چنین سنجه‌هایی به خوبی قادر نیستند تا این آزمایه‌ها را از هم متمایز کنند. در نتیجه چنین آزمایه‌هایی به صورت تصادفی مرتب می‌شوند و باعث کاهش اثربخشی مجموعه آزمون می‌گردند. این مسئله در کار اقبالی و تحویلداری [۱۳] نیز وجود داشت که محققان به این نتیجه رسیده بودند که در صورت افزایش تعداد حالت تساوی، اثربخشی فنون اولویت‌دهی کاهش می‌یابد.

بنابراین **پاسخ سوال اول** تحقیق این است که اولویت‌دهی با سنجه‌ها عملکرد بهتر و پایدارتری نسبت به فن کنترلی (مبنای مقایسه) دارد.

در کلیه‌ی برنامه‌ها، سنجه‌های سطح متد در مجموع عملکرد بهتری نسبت به سنجه‌های سطح کلاس داشته‌اند. این مشاهده نشان می‌دهد که سنجه‌های سطح متد در تشخیص مکان‌هایی از برنامه که خطاخیزتر هستند، موفق‌تر عمل می‌کنند. این نتیجه حاکی از این است که هر چه ریزدانگی سنجه‌ها بیشتر باشد، سنجه‌ها بیشتر می‌توانند آزمایه‌ها را از یکدیگر متمایز کنند.

بدین ترتیب **پاسخ سوال دوم** تحقیق این است که با توجه به وجود این عملکرد در کلیه‌ی برنامه‌های مورد مطالعه، عملکرد سنجه‌ها در سطح متد پایدارتر است و نتایج بهتری را به همراه دارد.

با توجه به نتایج شکل (۳) از ترکیب سنجه‌های برتر، به تحلیل‌هایی که در ادامه آمده است می‌پردازیم. برای برنامه‌های apns، assertj، jasmine، la4j و metrics مقدار APFD برای ترکیب سنجه‌ها به طور میانگین به ترتیب ۳٪، ۲٪، ۲٪، ۲٪ و ۴٪ به نسبت برترین سنجه، بهبود داشته است. برای دو برنامه‌ی jopt و scribe، ترکیب سنجه‌ها باعث بهبود در مقدار میانگین APFD نمی‌شود.

بنابراین **پاسخ سوال سوم** تحقیق این است که سنجه‌های برتر را با استفاده از روش ترکیب خطی با یکدیگر ترکیب می‌کنیم.

طبق آزمایش‌های انجام شده، مقدار APFD مجموعه آزمون اولویت‌دهی شده‌ی حاصل در بیشتر موارد نسبت به مجموعه آزمون‌هایی که توسط سنجه‌های برتر اولویت‌دهی شده‌اند، بهبود داشته است. عدم بهبود مقدار APFD در برخی موارد به علت عدم قطعیت فنون امتزاج داده در بهبود اثربخشی می‌باشد. عدم قطعیت یکی از ویژگی‌های عمومی روش‌های امتزاج داده است [۵]؛ یعنی در برخی موارد اثربخشی داده‌های امتزاج شده توسط یک روش امتزاج داده بهتر از بهترینِ آن‌هاست. در برخی موارد نیز اثربخشی داده‌های امتزاج شده توسط همان روش امتزاج داده، از بدترین داده‌ها نیز بهتر نیست.

۴-۶- شناخت عوامل زمینه‌ای در موفقیت روش

در این بخش، بررسی می‌شود که استفاده از سنجه‌های کد در اولویت‌دهی آزمایه‌ها در چه شرایطی خوب جواب می‌دهد یا گمراه کننده می‌شود. بدین منظور، نحوه‌ی امتیازدهی سنجه‌ها به آزمایه‌ها را بررسی می‌کنیم. برای مثال سنجه MaxInheritanceTree (عمق وراثت) را در نظر بگیرید. این سنجه در برنامه‌ی apns عملکرد قابل قبولی نداشته است. با بررسی امتیازهایی که این سنجه به آزمایه‌ها داده است، متوجه شدیم که کلاس‌های این برنامه عمق وراثت چندانی ندارند و امتیازهایی که این سنجه به آزمایه‌ها داده است صفر، یک یا دو است. بنابراین امتیازهای آزمایه‌ها تقریباً یکسان است و تعداد گره‌ها افزایش می‌یابد. در نتیجه اولویت‌دهی با استفاده از این سنجه خیلی شبیه به اولویت‌دهی تصادفی آزمایه‌ها می‌باشد.

همین سنجه را در برنامه‌ی la4j مورد بررسی قرار می‌دهیم. این سنجه توانسته عملکرد قابل قبولی را در اولویت‌دهی آزمایه‌ها برای این برنامه داشته باشد. با بررسی امتیازهایی که این سنجه به آزمایه‌ها داده است متوجه شدیم که این برنامه عمق وراثت بیشتری دارد و کلاس‌ها مقادیر بیشتر و متفاوت‌تری از این سنجه دارند. بنابراین این سنجه

توانسته به خوبی آزمایه‌ها را از یکدیگر تفکیک کند و تعداد حالت‌های تساوی را کاهش داده است. به همین دلیل اثربخشی این سنجه در اولویت‌دهی آزمایه‌ها قابل قبول بوده است.

در واقع هر سنجه از یک جنبه‌ای به نرم‌افزار نگاه می‌کند و از آن جنبه به آزمایه‌ها امتیاز می‌دهد. حال اگر آن جنبه به آزمایه‌ها امتیاز یکسانی بدهد و نتواند بین آزمایه‌ها تفاوت قائل بشود، باعث ایجاد حالت تساوی گمراه‌کنندگی در فرآیند اولویت‌دهی می‌شود.

بنابراین **پاسخ سوال چهارم** این است که در شرایطی سنجه‌ها خوب جواب می‌دهند که بتوانند به خوبی آزمایه‌ها را از یکدیگر متمایز کنند. در صورتی که امتیاز یکسانی به آزمایه‌ها بدهند، عملکرد اولویت‌دهی احتمالاً شبیه فن تصادفی می‌شود. در نتیجه اطلاعات چنین سنجه-هایی برای اولویت‌دهی گمراه‌کننده است.

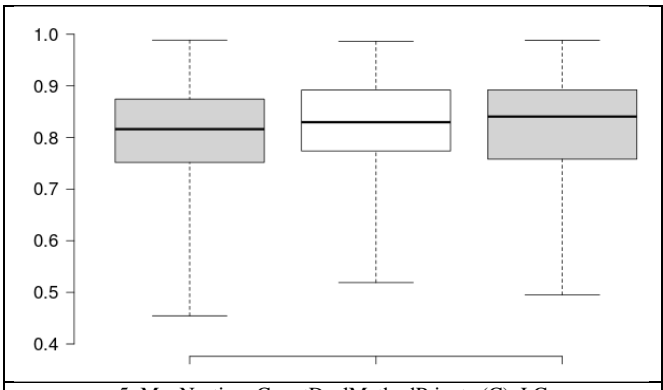
همچنین در مقیاس صنعتی برای شناسایی سنجه‌هایی که بتوانند تمایز خوبی بین آزمایه‌ها قائل شوند، می‌توان از نظرات توسعه‌دهندگان و خبرگان سیستم کمک گرفت. در واقع می‌توان از نظرات آنان جویا شد که پیچیدگی سیستم از چه جنبه‌ای بیشتر است. در نتیجه می-توان فرآیند اولویت‌دهی را به سمتی سوق داد که مجموعه آزمون مرتب شده‌ای تولید شود که هر چه سریع‌تر خطاها را تشخیص دهد.

۴-۷- تهدیدهای اعتبار نتایج

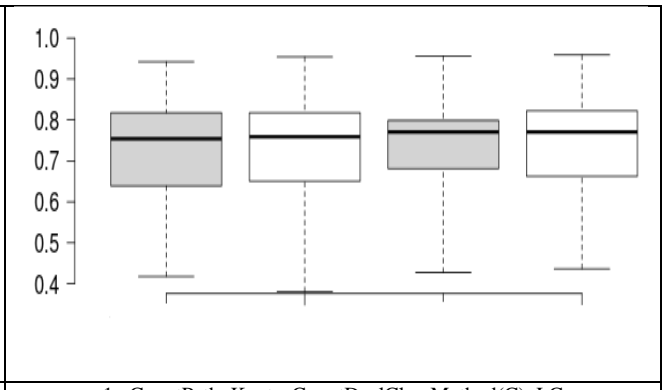
ما برای انجام آزمایش‌های این تحقیق از خطاهای کاشته شده‌ای که توسط ابزار PIT تولید شده‌اند، استفاده کردیم. یک تهدید احتمالی بر آزمایش‌ها این است که خطاهای کاشته شده بیانگر خصوصیات طبیعیِ خطاهای واقعی نباشند. برای کاهش این تهدید، طبق راهبردهایی که تحقیقاتِ در این حوزه پیشنهاد کرده بودند، از ۵۰۰ خطا به طور تصادفی، ۱۰۰ نسخه‌ی خطادار ساختیم.

جدول ۴: نتایج حاصل از انجام آزمون فرض آماری برای مقایسه فن پیشنهادی با فن پوشش کامل

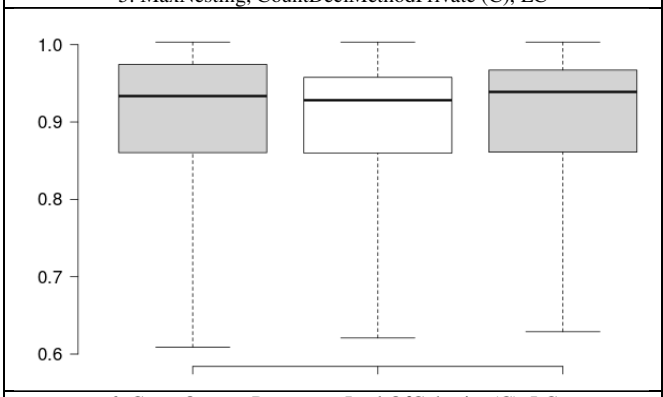
	سنجه (فن) / برنامه محک	apns	assertj	Jasmine	jopt	La4j	metrics	scribe
۱	**CountInput**	✔T0	OW0.428	✔T0	✘W0	✔W0	✔W0.012	✘T0.002
۲	**CountOutput**	✔W0	✔W0	✔T0	✔W0.001	✔T0	✘W0	✔T0
۳	**CountPath**	✔W0	✔W0.001	✔T0.001	✘W0	✔T0	✔W0.004	OW0.208
۴	**CountPathLog**	✔T0.025	OW0.07	✘T0	✘W0.011	✔T0	OW0.62	OW0.45
۵	**CountStmt (LOC)**	✔W0	✔W0	✔T0	✔W0.012	✔T0	✘W0.029	✔W0.005
۶	**Essential**	✔W0	✔W0.006	✔T0	OW0.119	✔T0	OW0.418	✔T0.032
۷	**Knots**	✔W0	✔W0.001	✘T0	✘W0	✔T0	✘W0	✔W0.012
۸	**MinEssentialKnots**	✘W0.044	✔W0.005	✘T0	✘W0	✔T0	✘W0.004	OT0.595
۹	**MaxNesting**	✔T0.013	✔W0	✘T0	✘W0.033	✔T0	✘W0.001	OW0.24
۱۰	**Cyclomatic**	✔W0	✔W0.002	✔T0.001	OW0.077	✔T0	OW0.49	OW0.198
۱۱	**CyclomaticStrict**	✔T0	✔W0.002	✔T0.002	✘W0	✔T0	OW0.476	OW0.08
۱۲	**PercentLackOfCohesion**	✘T0	OW0.665	✘T0	✘W0.033	✔T0	OW0.247	✘T0
۱۳	**MaxInheritanceTree**	✘T0	✘T0	✘T0	✘W0	✔T0	OW0.576	✘T0
۱۴	**CountClassDerived**	✘T0	OW0.282	✘T0	✘W0.001	✔T0	OW0.345	✘T0.025
۱۵	**CountClassCoupled**	OT0.675	OW0.536	✘T0	✘W0	✔T0	OW0.283	✘T0.023
۱۶	**CountClassBase**	OT0.074	OT0.286	✘T0	✘W0	✔T0	OW0.969	✘T0.012
۱۷	**CountDeclClassMethod**	✔W0	✔W0	✘T0	✘W0	✔T0	✘W0.002	✘T0.002
۱۸	**CountDeclClassVariable**	✘T0	OW0.643	✘T0	OW0.391	✔T0	OW0.976	✘W0
۱۹	**AvgCyclomaticStrict**	OT0.115	✔W0.037	✘T0.003	✘W0	✔T0	OW0.827	OT0.068
۲۰	**AvgLineCode**	OT0.57	✔W0.049	✘T0	✘W0	✔T0	OW0.253	OW0.393
۲۱	**CountDeclMethodAll**	✘T0	OT0.128	✘T0	✘W0	✔T0	OW0.244	✘W0
۲۲	**CountDeclMethodDefault**	✘T0	✘T0	✘T0	✘W0	✔W0	OW0.211	✘T0
۲۳	**CountDeclMethodPrivate**	OW0.391	OW0.606	OT0.274	✘W0	✔T0	OW0.236	✘W0.004
۲۴	**MaxCyclomatic**	OT0.157	✔W0.005	✘T0	✘W0	✔T0	OW0.819	OW0.204
۲۵	**MaxEssential**	OT0.735	✔W0.005	✘T0	✘W0	✔W0	OW0.558	OT0.126
۲۶	**SumCyclomatic**	✔T0.024	OW0.124	✘T0	✘W0	✔T0	✘W0.041	✘W0.001
۲۷	**SumEssential**	OT0.211	OW0.213	✘T0	✘W0	✔T0	✘W0.01	✘W0
	تصادفی	OT0.705	✘W0	✘T0	OW0.093	✔T0	✘W0	✘T0



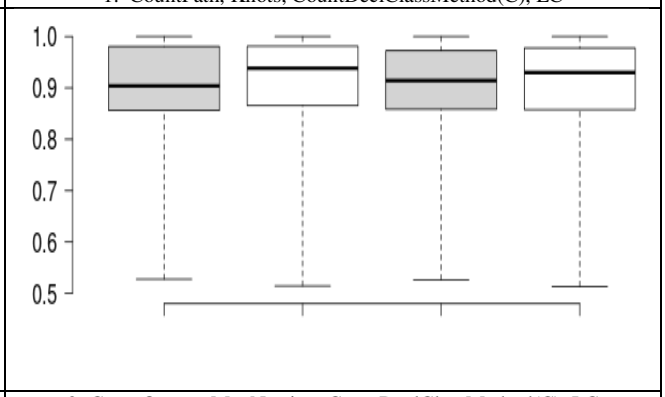
5: MaxNesting, CountDeclMethodPrivate (C), LC



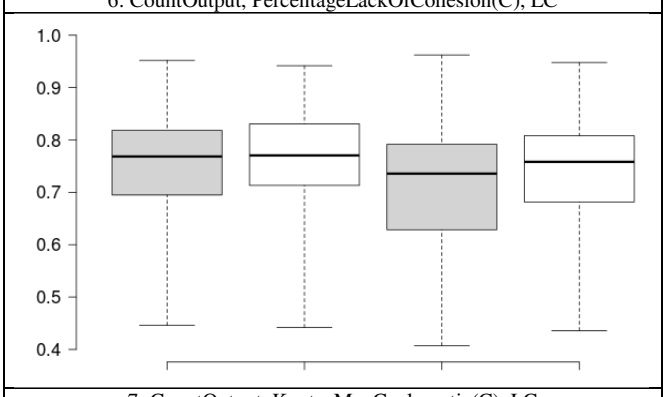
1: CountPath, Knots, CountDeclClassMethod(C), LC



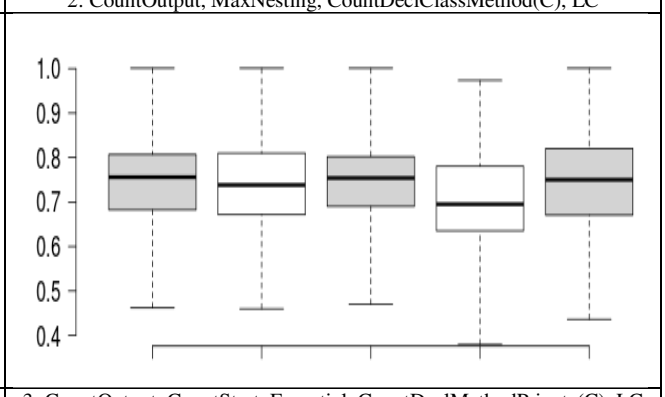
6: CountOutput, PercentageLackOfCohesion(C), LC



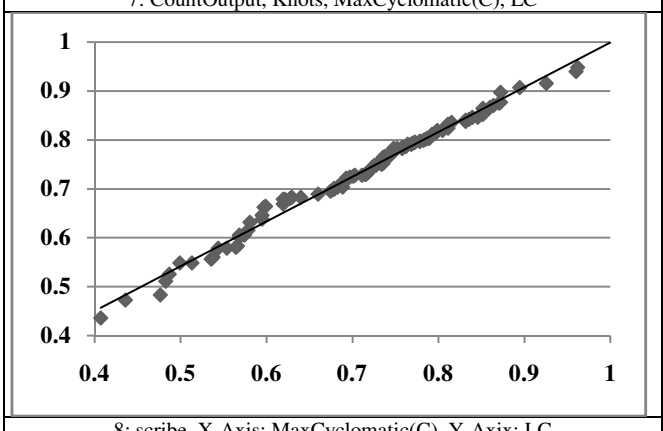
2: CountOutput, MaxNesting, CountDeclClassMethod(C), LC



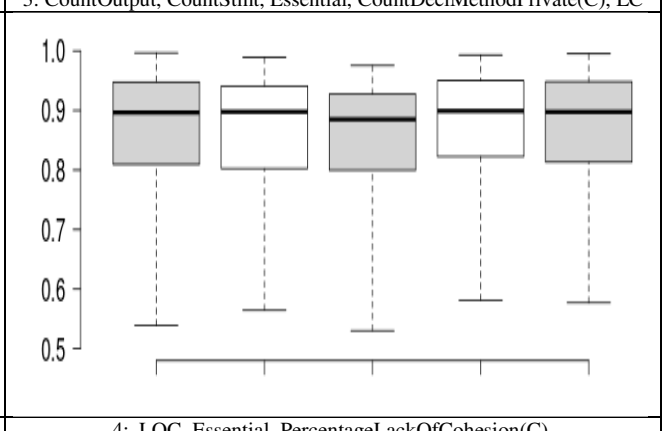
7: CountOutput, Knots, MaxCyclomatic(C), LC



3: CountOutput, CountStmt, Essential, CountDeclMethodPrivate(C), LC



8: scribe. X-Axis: MaxCyclomatic(C), Y-Axix: LC



4: LOC, Essential, PercentageLackOfCohesion(C), CountDeclClassVariable(C), LC

شکل ۳: نتایج APFD به ازای سنجه‌های برتر و ترکیب آن‌ها. شماره‌های ۱ تا ۷ به‌ترتیب برنامه‌های apns، assertj، jasmine، jopt، la4j، metrics و scribe هستند. نمودار مربوط به هر برنامه جعبه‌هایی از چپ به راست دارد. هر جعبه مربوط می‌شود به اولویت‌دهی با کمک اطلاعات یک سنجه که زیر هر شکل به‌ترتیب از چپ به راست مشخص شده‌اند. شکل ۸ نیز نمودار Q-Q از مقایسه دو سنجه را برای برنامه scribe نشان می‌دهد.

۵- کارهای مرتبط

عرفین و همکاران [۲۸]، از اطلاعات نیازمندی‌ها برای بهبود اولویت‌دهی آزمایه استفاده کرده‌اند. ریکانت[۲۰] و همکاران [۲۹] گزارش کرده‌اند که آزمایه‌های اولویت‌دهی‌شده بر اساس اهمیت و خطاخیزی نیازمندی‌ها، قادر بوده خطاهای دشوار را زودتر کشف نماید.

توماس و همکاران [۳۰]، اولویت‌دهی آزمایه‌ها را به صورت ایستا و با استفاده از مدل‌های موضوعی انجام داده‌اند. با وجودی که این روش سریع و ساده است، اما خطر نادیده گرفتن جنبه‌های مهم آزمایه‌ها و خودکارسازی کم روش از مشکلات آن هستند.

بنابر تحقیقات رودِرمِل و همکاران [۳۱]، اکثر فنون اولویت‌دهی موجود، فرضشان این بوده است که آزمایه‌ای که پوشش کد بیشتری دارد، احتمالاً خطاهای بیشتری را در برنامه تحت آزمون پیدا می‌کند. متأسفانه در بعضی موارد، پوشش کد پیش‌بینی کننده‌ی خوبی از قابلیت تشخیص خطای آزمایه نیست. همچنین تحقیقات کمی بر روی کدهایی که کمتر مورد آزمون قرار می‌گیرند انجام شده است. چنین کدهایی اغلب خطاخیزی بالایی دارند و آزمودنشان برای صحت برنامه حیاتی است. بنابر اظهار اقبالی و همکاران [۱۳]، روش‌های رایج برای اولویت‌دهی، از اطلاعات آزمایه‌های قبلاً اجرا شده، مانند اطلاعات پوشش استفاده می‌کنند. اثربخشی این روش‌ها در صورت افزایش تعداد گره‌ها در گام‌های اولویت‌دهی، کاهش می‌یابد.

ساها و همکاران [۳۲]، روشی مبتنی بر بازیابی اطلاعات برای اولویت‌دهی ارائه کرده‌اند. پاندا و همکاران [۳۳] پیشنهاد کردند که تنها قسمت‌های تغییر کرده از برنامه را دوباره بیازماییم. بر این مبنا آن‌ها روشی ایستا برای اولویت‌دهی آزمایه‌ها در برنامه‌های شی‌گرا پیشنهاد کردند. بنابر گزارش محققان ماژولی که مقدار اتصال بالایی دارد، امکان وجود خطای بیشتری نسبت به ماژول‌های دیگر دارد. در نتیجه آزمایه‌ای که ماژولی با مقدار اتصال بیشتری را اجرا می‌کند، خطاهای بیشتری را نسبت به دیگر آزمایه‌ها آشکار می‌نماید.

چودهاری[۲۱] و همکاران [۳۵] مطالعه‌ای انجام دادند تا اثر سنجه-های مختلف را در بهبود کارایی مدل‌های پیش‌گویی خطا مورد بررسی قرار دهند. محققان با استفاده از اطلاعات سنجه‌های تغییر و سنجه‌های کد منبع، دسته‌بندهایی به‌عنوان مدل پیش‌گوی خطا ساختند. نتایج آزمایش‌ها نشان داد که سنجه‌های تغییر اثر مثبتی در عملکرد مدل پیش‌گوی خطا دارد.

چِن[۲۲] و همکاران [۳۶] روشی برای بهبود اثربخشی آزمون بازگشت در نرم‌افزارهای شی‌گرایی ارائه کردند. این روش از فنون خوشه‌بندی روی اطلاعات جعبه سیاه استفاده می‌کند. محققان از تعداد اشیاء و متدها و رشته فراخوانی اشیاء و متدها به‌عنوان اطلاعات مورد استفاده در خوشه‌بندی استفاده کرده‌اند. در نتیجه آزمایه‌ها طوری اولویت‌دهی می‌شوند که آزمایه‌های همسایه آن‌ها تا جای ممکن تنوع بیشتری داشته باشند.

علیقارداشی و زارع [۳۸] با استفاده از انتخاب پیش‌روی ویژگی از یادگیری ماشینی، روشی برای پیش‌بینی خطای نرم‌افزار ارائه کردند. مشابه روش این مقاله، هدف مقاله مذکور نیز شناسایی خطاهاست با این تفاوت که محققان سعی می‌کنند ماژول‌های مستعد خطا را شناسایی نمایند. در رابطه با کمینه‌سازی مجموعه آزمون پس‌نمایی، رافع و اسفندیاری [۳۹] روشی با ترکیب الگوریتم جستجوی تپه‌نوردی و جستجوی خفاش ارائه کردند. کمینه‌سازی روشی است که در کنار اولویت‌دهی آزمایه‌ها (که هدف این مقاله بوده است) برای بهبود اثربخشی آزمون نرم‌افزار مورد استفاده قرار می‌گیرد.

۶- نتیجه‌گیری

یکی از چالش‌های فنون اولویت‌دهی آزمایه‌ها حین آزمون بازگشت، تخمین درست قدرت آشکارسازی خطای آزمایه‌هاست. با توجه به اثربخشی سنجه‌های کد در پیش‌بینی خطاها، پیشنهاد کردیم که از این اطلاعات برای تخمین میزان اثربخشی آزمایه‌ها در کشف خطا استفاده شود. بر این اساس فن اولویت‌دهی جدیدی بر اساس امتزاج داده روی اطلاعات سنجه‌های کد ارائه شد و برای اعتبارسنجی آن، آزمایش‌هایی ترتیب دادیم. نتایج آزمایش‌ها نشان داد: (۱) سنجه‌های نرم‌افزار در اولویت‌دهی آزمایه‌ها اثربخشند؛ (۲) برای گرفتن نتایج بهتر، باید سنجه‌های برتر را با هم ترکیب کنیم؛ (۳) استفاده از سنجه‌ها وقتی اثربخش‌تر می‌شوند که امتیازهایشان بتوانند تک‌تک آزمایه‌ها را از هم متمایز نمایند. در آینده قصد داریم فن پیشنهادی را روی برنامه-های بزرگتر و زبان‌های دیگر رایج نیز مورد ارزیابی قرار دهیم.

مراجع

[1] Z. Li, M. Harman, and R. M. Hierons, "Search algorithms for regression test case prioritization," *IEEE Trans. Softw. Eng.*, vol. 33, no. 4, 2007.

[2] L. Zhang, D. Hao, L. Zhang, G. Rothermel, and H. Mei, "Bridging the gap between the total and additional test-case prioritization strategies," *Proc. - Int. Conf. Softw. Eng.*, pp. 192–201, 2013.

[3] Y. Lu, Y. Lou, S. Cheng, L. Zhang, D. Hao, Y. Zhou, and L. Zhang, "How does regression test prioritization perform in real-world software evolution?," *Proc. 38th Int. Conf. Softw. Eng. - ICSE '16*, pp. 535–546, 2016.

[4] Q. Luo, K. Moran, and D. Poshyvanyk, "A large-scale empirical comparison of static and dynamic test case prioritization techniques," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 559–570, 2016.

[5] S. Wu, *Data Fusion in Information Retrieval*. Springer Berlin Heidelberg, 2012.

[6] T. Lee, J. Nam, D. Han, S. Kim, and H. Peter In, "Developer Micro Interaction Metrics for Software Defect Prediction," *IEEE Trans. Softw. Eng.*, vol. 42, no. 11, pp. 1015–1035, 2016.

[7] F. Rahman and P. Devanbu, "How, and why, process metrics are better," in *Proceedings of the 2013 International Conference on Software Engineering*, 2013, pp. 432–441.

[8] R. Premraj and K. Herzig, "Network Versus Code Metrics to Predict Defects: A Replication Study," *2011 Int. Symp. Empir. Softw. Eng. Meas.*, pp. 215–224, 2011.

[9] Y. Zhou, B. Xu, and H. Leung, "On the ability of complexity metrics to predict fault-prone classes in object-oriented systems," *J. Syst. Softw.*, vol. 83, no. 4, pp. 660–674, 2010.

[10] A. Shi, A. Gyori, M. Gligoric, A. Zaytsev, and D. Marinov, "Balancing trade-offs in test-suite reduction," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 246–256.

[11] A. Shi, T. Yung, A. Gyori, and D. Marinov, "Comparing and Combining Test-suite Reduction and Regression Test Selection," *Proc. 2015 10th Jt. Meet. Found. Softw. Eng.*, pp. 237–247, 2015.

[12] A. Gyori, A. Shi, F. Hariri, and D. Marinov, "Reliable testing: Detecting state-polluting tests to prevent test dependency," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, 2015, pp. 223–233

[13] S. Eghbali and L. Tahvildari, "Test Case Prioritization Using Lexicographical Ordering," *IEEE Trans. Softw. Eng.*, vol. 5589, no. January, pp. 1–1, 2016.

[14] H. Do, "Recent Advances in Regression Testing Techniques," in *Advances in Computers*, Elsevier, 2016, pp. 1–25.

[15] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Test case prioritization: An empirical study," in *Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on*, 1999, pp. 179–188.

[16] C. Kaner, "Improving the maintainability of automated test suites," *Softw. QA*, vol. 4, no. 4, 1997.

[17] P. K. Chittimalli and M. J. Harrold, "Recomputing coverage information to assist regression testing," *IEEE Trans. Softw. Eng.*, vol. 35, no. 4, pp. 452–469, 2009.

[18] H. Do, S. Mirarab, L. Tahvildari, and G. Rothermel, "The effects of time constraints on test case prioritization: A series of controlled experiments," *IEEE Trans. Softw. Eng.*, vol. 36, no. 5, pp. 593–617, 2010.

[19] D. Hao, L. Zhang, L. Zang, Y. Wang, X. Wu, and T. Xie, "To Be Optimal Or Not in Test-Case Prioritization," *IEEE Trans. Softw. Eng.*, vol. 6, no. 1, pp. 1–20, 2015.

[20] Inozemtseva, Laura, and Reid Holmes. "Coverage is not strongly correlated with test suite effectiveness." In *Proceedings of the 36th International Conference on Software Engineering*, pp. 435-445. ACM, 2014.

[21] Le Goues, Claire, and Westley Weimer. "Measuring code quality to improve specification mining." *IEEE Transactions on Software Engineering* 38, no. 1 (2012): 175-190.

[22] N. Nagappan and T. Ball, "Using software dependencies and churn metrics to predict field failures: An empirical case study," in *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, 2007, pp. 364–373.

[23] R. Subramanyam and M. S. Krishnan, "Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects," *IEEE Trans. Softw. Eng.*, vol. 29, no. 4, pp. 297–310, 2003.

[24] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Trans. Softw. Eng.*, vol. 31, no. 10, pp. 897–910, 2005.

[25] R. P. L. Buse and W. Weimer, "The road not taken: Estimating path execution frequency statically," in *Proceedings of the 31st International Conference on Software Engineering*, 2009, pp. 144–154.

[26] J. C. Sanchez, L. Williams, and E. M. Maximilien, "On the sustained use of a test-driven development practice at ibm," in *Agile Conference (AGILE), 2007*, 2007, pp. 5–14.

[27] R. P. L. Buse and W. R. Weimer, "x," in *Proceedings of the 2008 international symposium on Software testing and analysis*, 2008, pp. 121–130.

[28] M. J. Arafeen and H. Do, "Test Case Prioritization Using Requirements-Based Clustering," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, 2013, pp. 312–321.

[29] H. Srikanth, L. Williams, and J. Osborne, "System test case prioritization of new and regression test cases," in *2005 International Symposium on Empirical Software Engineering, 2005.*, 2005, p. 10–pp.

[30] S. W. Thomas, H. Hemmati, A. E. Hassan, and D. Blostein, "Static test case prioritization using topic models," *Empir. Softw. Eng.*, vol. 19, no. 1, pp. 182–212, 2014.

[31] J. H. Kwon, I. Y. Ko, G. Rothermel, and M. Staats, "Test case prioritization based on information retrieval concepts," *Proc. - Asia-Pacific Softw. Eng. Conf. APSEC*, vol. 1, pp. 19–26, 2014.

[32] R. K. Saha, L. Zhang, S. Khurshid, and D. E. Perry, "REPiR: An Information Retrieval based Approach for Regression Test Prioritization," in *37th International Conference on Software Engineering, Florence Italy, Mary*, 2015.

[33] S. Panda, D. Munjal, and D. P. Mohapatra, "A Slice-Based Change Impact Analysis for Regression Test Case Prioritization of Object-Oriented Programs," vol. 2016, 2016.

[34] Ammann, Paul, and Jeff Offutt. *Introduction to software testing*. Cambridge University Press, 2016.

[35] Choudhary, Garvit Rajesh, Sandeep Kumar, Kuldeep Kumar, Alok Mishra, and Cagatay Catal. "Empirical analysis of change metrics for software fault prediction." *Computers & Electrical Engineering* 67 (2018): 15-24.

[36] Chen, Jinfu, Lili Zhu, Tsong Yueh Chen, Dave Towey, Fei-Ching Kuo, Rubing Huang, and Yuchi Guo. "Test case prioritization for object-oriented software: An adaptive random sequence approach based on clustering." *Journal of Systems and Software* 135 (2018): 107-125.

[37] Wohlin, Claes, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén.*Experimentation in software engineering*. Springer Science & Business Media, 2012.

[۳۸] فاطمه علیقارداشی، محمدعلی زارع چاهوکی، «تأثیر ترکیب روش‌های انتخاب ویژگی فیلتر و بسته‌بندی در بهبود پیش‌بینی اشکال نرم‌افزار»، مجله مهندسی برق، دوره ۴۷، شماره ۱، ۱۹۵-۱۸۳، دانشگاه تبریز، بهار ۱۳۹۶.

[۳۹] وحید رافع، سجاد اسفندیاری، «راهکاری نوین جهت تولید دنباله آزمون کمینه در فرآیند آزمون نرم افزار با ترکیب الگوریتم های جستجوی تپه نوردی و جستجوی خفاش»، مجله مهندسی برق، دوره ۴۶، شماره ۳، ۳۵-۲۵، دانشگاه تبریز، پاییز ۱۳۹۵.

زیرنویس‌ها

[1] Regression testing
[2] Test case
[3] Test case prioritization
[4] Coverage criteria
[5] Lu
[6] Data fusion
[7] Benchmark
[8] Average Percentage of Fault Detection (APFD)
[9] Regression test suite
[10] Paradigm
[11] Fault proneness
[12] Instrumentation
[13] Wu
[14] http://www.eclemma.org/jacoco/
[15] https://maven.apache.org/index.html
[16] https://scitools.com/features/
[17] http://pitest.org/
[18] https://www.ibm.com/products/spss-statistics
[19] https://github.com/
[20] Srikanth
[21] Choudhary
[22] Chen