علوم رایانش و فناوری اطلاعات
نشریه علمی انجمن کامپیوتر ایران
مجلد ۱۴، شماره ۱، ۱۳۹۵
صفحات ۱- ۱۵
مقاله عادی

توصیه‌گر راه‌حل استثنائات در محیط یکپارچه ایجاد نرم‌افزار

وحید امین‌تبار عباس حیدرنوری

دانشکده مهندسی کامپیوتر، دانشگاه صنعتی شریف، تهران، ایران

چکیده

در فرآیند ایجاد نرم‌افزار، استثناءها اجتناب‌ناپذیر هستند. ولیکن، استثناءها در بسیاری از موارد توسط برنامه‌نویسان دیگری نیز تجربه شده‌اند و به احتمال زیاد در فضای مجازی راه‌حل آن‌ها موجود است. با رخ دادن یک استثناء در زمان اجرا، معمولاً یک پیغام خطا و موارد بالای پشته فراخوانی[1]، نمایش داده می‌شوند. با گسترش استفاده از سایت‌های پرسش و پاسخ برنامه‌نویسی مانند Stack Overflow و یا سایت‌های حاوی کدهای متن‌باز مانند Source Forge، برنامه‌نویسان اغلب برای پیدا کردن راه‌حل یک استثناء، با استفاده از مرورگر و موتورهای جست‌وجو به دنبال مشکلات و راه‌حل‌های مشابه می‌گردند. در اینجا شکاف بین محیط ایجاد نرم‌افزار و مرورگر وجود دارد. همچنین جست‌وجوی دستی، تمام اطلاعات کد برنامه‌نویس را دارا نمی‌باشد و کاری زمان‌بر است. در این مقاله قصد آن داریم تا Exception Tracer را معرفی نماییم. در این روش با استفاده از پشته فراخوانی مربوط به استثناء و کد برنامه‌نویس ابتدا در پروژه‌های Source Forge فایل‌های مرتبط بازیابی می‌شوند و در ادامه با ساخت گراف GROUM از کد برنامه‌نویس و فایل‌های مرتبط، قطعه کدها رتبه‌بندی می‌شوند و بعد از تغییر نام متغیرهای آن‌ها به زمینه برنامه‌نویس، قطعه کدهایی که می‌توانند حاوی راه حل استثناء باشند، به او پیشنهاد داده می‌شوند. GROUM گرافی است که در آن توالی فراخوانی متدها و وابستگی داده‌ای بین آن‌ها را شامل می‌شود و نتیجه بهتری نسبت به جست‌وجوی متنی و یا درخت نحو می‌دهد. این روش،علاوه بر قطعه کد، مباحثه‌های مرتبط را نیز از Stack Overflow استخراج می‌کند و به برنامه‌نویس نمایش می‌دهد. روش پیشنهادی این مقاله، به صورت یک افزونه Eclipse پیاده‌سازی شده است. دقت این روش با استفاده از تعدادی از نمونه کدهای مربوط به کتابخانه‌های Apache جاوا اندازه‌گیری شده است. در ارزیابی‌های انجام شده، در ۷۵ درصد مواقع، راه‌حل استثناء رخ‌داده در ۵ نتیجه ابتدایی وجود دارد. علاوه بر محاسبه دقت، میزان صرفه‌جویی زمانی نسبت به استفاده ساده از مرورگر نیز از طریق مطالعه کاربران سنجیده شده است. طبق این ارزیابی، میزان صرفه‌جویی زمانی، در حدود ۳۸ درصد است.

کلمات کلیدی: محیط یکپارچه ایجاد نرم‌افزار، سیستم‌های توصیه‌گر، قطعه کد، Stack Overflow، افزونه Eclipse.

۱- مقدمه

مهندسی نرم‌افزار به سرعت در حال پیشرفت است. روزانه چارچوب‌ها[2] و کتابخانه‌های برنامه‌نویسی جدیدی ایجاد و یا به‌روزرسانی می‌شوند، و استفاده از آنها به دلیل افزایش سرعت و کیفیت برنامه‌نویسی به سرعت در حال گسترش است. ولیکن، فهم چگونگی استفاده از رابط کاربردی برنامه‌نویسی[3] این کتابخانه‌ها و چارچوب‌های برنامه‌نویسی سخت و زمان‌بر است. برنامه‌نویس برای استفاده از آن‌ها باید یک دنباله مشخصی از فراخوانی متدها را دنبال کند؛ عدم‌رعایت این قوانین ممکن است منجر به استثناء شود. به عنوان نمونه، برای نوشتن در فایل باید ابتدا فایل را باز کنیم؛ سپس داده موردنظر را در فایل بنویسیم و در انتها نیز فایل را ببندیم. عدم‌رعایت توالی ذکر شده، منجر به استثناء خواهد شد. به علت سرعت چشم‌گیر ایجاد و به‌روزرسانی چارچوب‌ها و کتابخانه‌ها، مستندات کافی و به‌روز برای آن‌ها وجود ندارد. به دلیل عدم‌وجود مستندات مناسب، ممکن است برنامه‌نویس به اشتباه از کتابخانه‌ها استفاده کند و اجرای کد به استثناء منجر شود.

وقتی که یک استثناء رخ می‌دهد، برنامه‌نویسان اغلب از محیط یکپارچه ایجاد نرم‌افزار به مرورگرهای وب می‌روند و با جست‌وجوی پیغام استثناء در موتورهای جست‌وجوی عمومی به دنبال راه‌حل مشکل خود می‌گردند. ولیکن، برنامه‌نویس معمولا در ایجاد پرس‌وجو فقط از پیغام خطا استفاده می‌کند و اطلاعات زمینه کد را دخیل نمی‌کند. علاوه بر آن، برنامه‌نویس زمان نسبتاً زیادی را برای یافتن پاسخ به‌وسیله مرورگر و دیدن صفحات وب سپری می‌کند. مطالعات نشان می‌دهد که ایجادکنندگان نرم‌افزار در طول فرآیند ایجاد و نگه‌داری نرم‌افزار، ۱۹٪ زمان خود را

به جست‌وجوی اطلاعات با استفاده از موتورهای جست‌وجوی رایج می‌گذرانند [۱]. همچنین تغییر محیط به مرورگر از محیط یکپارچه ایجاد نرم‌افزار به عدم‌تمرکز برنامه‌نویس منجر می‌شود. تاکنون روش‌های مختلفی برای حل این مسئله مطرح شده است [۱-۶]. روش‌های [۱-۲] فقط از پشته فراخوانی در جست‌وجو استفاده می‌کنند. هیچ‌کدام از این روش‌ها جست‌وجو را به روی درخت انتزاعی تجزیه انجام نمی‌دهند. همچنین تمامی این روش‌ها یک نوع خروجی می‌دهند. این خروجی‌ها یکی از موارد قطعه کد، مباحثه و یا صفحات وب است. روش‌های [۱-۲، ۴، ۶] فقط مباحثه‌های مرتبط را به عنوان خروجی به برنامه‌نویس ارائه می‌دهند؛ روش [۳] مباحثه و نمونه کد خروجی می‌دهد ولی نمونه کد آن در حد یک خط است. روش [۵] نیز صفحات وب را ارائه می‌دهد. هر کدام از این خروجی‌ها (کد و یا مباحثه) بخشی از استثناءها را پوشش می‌دهند. همچنین، در تمام این روش‌ها نمونه کدهای خروجی ابزارهای مشابه به زمینه کد برنامه‌نویس منتقل نشده‌اند. برنامه‌نویس به‌منظور استفاده از این نمونه کدها، مجبور است اسامی متغیرها را به صورت دستی به اسامی موجود در کد خود تغییر دهد. در این مقاله، منظور از انتقال به زمینه برنامه‌نویس، تغییر نام متغیرهاست به گونه‌ای که اسامی متغیرها با کدی که برنامه‌نویس در حال کارکردن بر روی آن است، سازگار باشد.

به منظور حل مشکلات فوق، در این مقاله، ابزاری با عنوان Exception Tracer ارائه شده است. این ابزار، به‌صورت خودکار، به حل استثناء در زبان جاوا به برنامه‌نویس کمک می‌کند. به عبارت مشخص‌تر، با رخ دادن یک استثناء، ابزار ما قادر است تا مباحثه‌های مرتبط به آن استثناء را از سایت پرسش و پاسخ Stack Overflow، و قطعه کدهایی که ممکن است راه‌حل استثناء باشد را از سایت Source Forge به برنامه‌نویس پیشنهاد دهد. برای یافتن مباحثه‌های مرتبط از سایت پرسش و پاسخ Stack Overflow، از اطلاعات موجود در پشته فراخوانی استفاده می‌شود. برای یافتن نمونه کدهای مرتبط، ابتدا با استفاده از پشته فراخوانی، رابط کاربردی برنامه‌نویسی مورد استفاده کاربر را شناسایی می‌کند؛ فایل‌های شامل این رابط کاربردی برنامه‌نویسی به وسیله زبان پرس‌وجوی BOA [۳] از پروژه‌های Source Forge بازیابی می‌شوند. BOA یک موتور جست‌وجو است که جست‌وجو را به صورت موازی به روی درخت نحو پروژه‌های Source Forge انجام می‌دهد. سپس یک دید سطح بالا (گراف) از این کدها و کد برنامه‌نویس ایجاد می‌شود. بعد از آن، با استفاده از گراف GROUM [۷]، قطعه کدها به زمینه کاربر انتقال داده می‌شوند. در نهایت، از این گراف‌ها کد تولید می‌شود. با توجه به میزان شباهت گراف قطعه کدهای تولید شده به گراف برنامه برنامه‌نویس، آن قطعه کدها رتبه‌بندی می‌شوند و به برنامه‌نویس نمایش داده می‌شوند. Exception Tracer با هدف ارائه پیشنهاد برای آسان‌سازی اصلاح کدهایی که قوانین استفاده از رابط کاربردی برنامه‌نویسی در آن‌ها رعایت نشده‌اند، ایجاد شده است. روش‌های مشابه، در جست‌وجوهایشان در نهایت با کد به صورت متنی برخورد می‌کنند. روش ارائه شده در این مقاله، برای جلوگیری از این امر، یک دید سطح بالا از کد نیز ایجاد می‌شود. این دید سطح بالا باید شامل قوانین استفاده از رابط کاربردی برنامه‌نویسی (معمولا به شکل دنباله‌ای از فراخوانی متدها) باشد. برای استخراج دنباله فراخوانی متدها از کد، از گرافی که از درخت انتزاعی نحو[۴] برنامه ساخته شده، استفاده می‌شود و الگوهای استفاده را شامل می‌شود. این گراف با توجه به وابستگی‌های داده‌ای موجود در کد ایجاد می‌شود. با روش ارائه شده، اولاً، کاربر برای یافتن راه‌حل نیازی به خروج از محیط یکپارچه ایجاد نرم‌افزار ندارد. ثانیاً، اطلاعات زمینه کد کاربر در پرس‌وجو درگیر است. ثالثاً، کاربر زمان کم‌تری را به جست‌وجو می‌پردازد. رابعاً، از قطعه کد پیشنهادی دستورات نامرتبط حذف می‌شوند و خروجی نهایی در زمینه یک در برنامه‌نویس است؛ به همین دلیل فهم و استفاده از آن آسان‌تر صورت می‌پذیرد.

به دلیل آن‌که روش‌های مشابه این مقاله با معیارهای گوناگونی ارزیابی شده‌اند و با توجه به آنکه باید Exception Tracer با آن‌ها و نیز روش معمول مورد استفاده برنامه‌نویسان، (یعنی، استفاده از مرورگر) مقایسه شود، روش پیشنهادی را به دو روش متفاوت ارزیابی نمودیم: (۱) ارزیابی کارایی و دقت؛ (۲) مطالعه کاربران. در روش اول ارزیابی که به روی ۱۴ کتابخانه از پروژه‌های Apache و کتابخانه‌های استاندارد Java انجام گرفت، خروجی روش پیشنهادی از دقت ۷۵ درصدی برخوردار بود، بدین معنی که برنامه‌نویس در ۷۵ درصد موارد در ۵ نتیجه اول جواب خود را می‌یابد. با توجه به نتایج ارزیابی دوم، برنامه‌نویسان با استفاده از Exception Tracer نسبت به حالتی که از مرورگر استفاده نمایند، ۳۸ درصد در زمان خود صرفه‌جویی می‌نمایند.

با توجه به بحث فوق، نوآوری‌های ما در این مقاله عبارتند از:

- ایجاد امکان یافتن راه‌حل استثناء در محیط یکپارچه ایجاد نرم‌افزار و بدون استفاده از مرورگر.
- پیاده‌سازی روش پیشنهادی به صورت یک افزونه برای نرم‌افزار Eclipse.
- ایجاد خودکار پرس‌وجو از زمینه کد برنامه‌نویس برای موتور جست‌وجوی BOA.
- رتبه‌بندی و تبدیل قطعه‌کدهای استخراجی از Source Forge به زمینه برنامه‌نویس.

خوانندگان علاقمند، برای مشاهده محیط این ابزار می‌توانند ویدئو موجود در آدرس زیر را مشاهده نمایند:

https://goo.gl/q7Fr56

در ادامه این مقاله، در بخش دوم کارهای مرتبط را بررسی می کنیم. در بخش سوم، مسئله‌ای که قصد حل کردن آن را داریم را به صورت دقیق بیان می‌کنیم. سپس در بخش چهارم، به بیان راه‌حل پیشنهادی برای حل این مسئله می‌پردازیم. در بخش پنجم، به ارزیابی راه‌حل پیشنهادی پرداخته خواهد شد. در نهایت، در بخش ششم، به بیان کارهای آتی، و در بخش هفتم، به نتیجه‌گیری می‌پردازیم.

۲- کارهای مرتبط

در این بخش کارهای مرتبط در دو دسته سیستم‌های توصیه‌گر و استخراج مشخصات تقسیم‌بندی شده‌اند. این دسته‌بندی‌ها هر کدام شامل حوزه‌های بسیار گسترده‌ای می‌شوند. در این مقاله فقط مقالاتی از این دسته‌ها که شباهتی با روش پیشنهادی داشته‌اند مورد بررسی قرار گرفته‌اند. روش پیشنهادی در این مقاله ترکیبی از روش‌های سیستم‌های توصیه‌گر و روش‌های استخراج مشخصات است.

۲-۱- سیستم‌های توصیه‌گر

موارد بررسی شده در این دسته به طور معمول به صورت افزونه Eclipse پیاده‌سازی شده‌اند و به شکل‌های مختلف به برنامه‌نویس پیشنهادهایی را برای انجام بهتر برنامه‌نویسی ارائه می‌دهند. به عنوان مثال، سیستم‌های توصیه‌گر نمونه کد، با توجه به اطلاعات استخراج شده از محیط برنامه‌نویسی و جست‌وجو در منابع کد به منظور بهبود فرآیند برنامه‌نویسی به برنامه‌نویس نمونه کدهایی را پیشنهاد می‌دهند. روش‌های [۸-۱۱] از این قبیل روش‌ها هستند. از دیگر سیستم‌های توصیه‌گر می‌توان [۱۲-۱۳] را نام برد که در این مقالات پیشنهاد روش‌های بازآرایی[۵] مدنظر است. به عنوان یک نمونه دیگر می‌توان [۱۴] را نام برد که نمونه کد خاص واحد آزمون پیشنهاد می‌دهد. گروهی دیگر از سیستم‌های توصیه‌گر هستند که به منظور اصلاح خطای موجود در کد برنامه‌نویس طراحی شده‌اند. روش پیشنهادی این مقاله نیز همین هدف را داراست. با توجه به ارتباط زیاد این دسته از کارها به این مقاله، در ادامه ۶ روش از این حوزه، به تفصیل توضیح داده شده‌اند.

Oscilloscope -۱-۱-۲

ابزار Oscilloscope [۲] برای شناسایی خطاهای مشابه از پشته فراخوانی استفاده کرده است. برای یک خطای جدید، با استفاده از پشته فراخوانی در پایگاه داده خود جست‌وجو می‌کند و خطاهای مشابه را پیدا می‌کند. سپس به برنامه‌نویس گزارش‌های آن خطاها را نمایش می‌دهد. این روش به صورت افزونه Eclipse پیاده‌سازی شده است. وقتی یک Junit منجر به خطا می‌شود، نتیجه آزمون به سرور بارگذاری می‌شود. با استفاده از اطلاعات بارگذاری شده، در پایگاه داده جست‌وجو انجام می‌گیرد و نتیجه (لیستی از گزارشات خطا) به برنامه‌نویس نمایش داده می‌شود. از آنجایی که پشته فراخوانی ممکن است بزرگ باشد و جست‌وجو زمان‌بر شود، برنامه‌نویس می‌تواند قسمتی از پشته فراخوانی را برای جست‌وجو انتخاب کند. در مقایسه با Exception Tracer این ابزار قطعه کد پیشنهاد نمی‌دهد و زمینه کدی که منجر به استثناء شده است را در جست‌وجو دخیل نمی‌کند.

Help Me Out -۲-۱-۲

ابزار Help Me Out [۳] از سه بخش اصلی تشکیل شده است: (۱) افزونه Eclipse (۲) پایگاه داده شامل خطاها و راه‌حل‌هایشان (۳) واسط کاربری برای اضافه کردن توضیح به راه‌حل‌ها. در این روش تغییرات کد برنامه‌نویس مورد بررسی قرار می‌گیرد. هنگامی که برنامه از حالت خطادار به حالت بدون خطا می‌رود، تغییرات کد به عنوان راه‌حل برای خطای رخ‌داده، در نظر گرفته می‌شود و در پایگاه داده ذخیره می‌شود. هنگامی‌که خطایی در برنامه رخ می‌دهد، در پایگاه داده براساس پیغام خطا، پشته فراخوانی، خطی از کد که خطا در آنجا رخ‌داده و رای کاربران به راه‌حل‌ها، در پایگاه داده جست‌وجو انجام می‌گیرد. یک مرحله مهم، تشخیص گذار از حالت خطادار به حالت بدون خطا در کد است. دو مورد خطا داریم: (۱) خطای زمان کامپایل (۲) خطای زمان اجرا. در مورد اول اگر در کامپایل بعدی خطا رخ ندهد گذار به‌راحتی تشخیص داده می‌شود. اما در مورد دوم عدم رخ دادن خطای زمان اجرا ممکن است به دلیل متفاوت بودن ورودی‌های برنامه باشد. برای کم‌تر شدن رخداد این اشتباه، تعداد دفعات اجرای خطی که در آن قبلا خطا رخ داده را نگه‌داری می‌کنیم. در صورتی که در اجرای بعدی در آن خط خطا رخ نداد و به‌علاوه تعداد دفعات اجرا شدن آن خط با اجرای قبلی یکسان بود، گذار تشخیص داده می‌شود. فقط در مورد اول (خطای زمان کامپایل) در جست‌وجو، کد برنامه‌نویس (فقط تک‌خطی که خطا در آن رخ داده با تعمیم) در نظر گرفته می‌شود. در مورد دوم فقط پشته فراخوانی برای جست‌وجو استفاده می‌شود. در نهایت راه‌حل به برنامه‌نویس نمایش داده می‌شود. برنامه‌نویس می‌تواند به راه‌حل رای دهد و از این اطلاعات برای جست‌وجوی بهتر بهره برده می‌شود. این ابزار در ابتدای استفاده از آن و وقتی که پایگاه داده هنوز پر نشده است، پیشنهادی نمی‌دهد، در حالی که Exception Tracer این گونه نیست.

Surf Clips -۳-۱-۲

روش Surf Clips [۵] نتایج جست‌وجو در چهار وب‌سایت Google، Bing، Yahoo و Stack Overflow رتبه‌بندی و به برنامه‌نویس نمایش می‌دهد. با رخ دادن استثناء، پرس‌وجو براساس کد و پشته فراخوانی ساخته می‌شود. پرس‌وجو از کلمات زیر ساخته می‌شود: (۱) پنج نام کلاس یا متد با درجه جذابیت بالاتر (۲) پنج پرتکرارترین نام متد فراخوانی شده یا کلاس (موجود در import) از کد برنامه‌نویس. سپس نتایج موتورهای جست‌وجو جمع‌آوری و براساس معیارهای زیر نتایج به‌دست آمده رتبه‌بندی می‌شوند:

۱. رتبه موتور جست‌وجو در Alexia.
۲. شباهت عنوان نتیجه یافته شده از موتورهای جست‌وجو با متن استثناء.
۳. وجود شباهت بین کد یا پشته فراخوانی در نتیجه یافته شده از موتورهای جست‌وجو و استثناء.
۴. وجود شباهت متنی بین نتیجه یافته شده از موتورهای جست‌وجو و استثناء. در نهایت نتایج رتبه‌بندی شده به برنامه‌نویس نمایش داده می‌شود.

در مقایسه با Exception Tracer این ابزار فقط مباحثه‌های مرتبط را پیشنهاد می‌دهد و هیچ‌گونه قطعه کدی به برنامه‌نویس پیشنهاد داده نمی‌شود.

ابزار ارائه شده توسط کوردیرو -۴-۱-۲

ابزار ارائه شده توسط کوردیرو [۱] دو بخش اصلی دارد: (۱) خدمتگزار؛ (۲) افزونه Eclipse. در خدمتگزار یک پایگاه داده از داده‌های Stack Overflow درست شده است. با رخ دادن استثناء اطلاعات پشته فراخوانی به خدمتگزار ارسال می‌شود و خدمتگزار مرتبط‌ترین مباحث به آن را برمی‌گرداند. هر مباحثه‌ی Stack Overflow علاوه بر متن، ممکن است دارای کد و یا پشته فراخوانی باشد؛ قطعه کدهای کوچک در نظر گرفته نمی‌شوند. قطعه کدهای بزرگ تجزیه می‌شوند و ارجاع‌های آن‌ها به دست می‌آید. ارجاع‌های پشته فراخوانی نیز با عبارت منظم از کد تمیز داده می‌شود. منظور از ارجاع، رخداد یک نوع است. این اطلاعات به وسیله موتور جست‌وجوی Apache Lucerne نمایه‌سازی می‌شوند.

با ارسال اطلاعات پشته فراخوانی، خدمتگزار یک پرس‌وجو شامل نام تمام استثناءهای موجود در پشته فراخوانی و متن پنج ارجاع بالای پشته فراخوانی را می‌سازد. از نتایج به‌دست آمده دویست موردی که در عنوان آن‌ها نام تمام استثناءها آمده باشد، برگردانده می‌شود و به برنامه‌نویس نمایش داده می‌شود. در مقایسه با Exception Tracer این ابزار قطعه کد ارائه نمی‌دهد و همچنین زمینه کد را در جست‌وجو دخیل نمی‌کند.

Seahawk -۵-۱-۲

ابزار Seahawk [۱۵] به صورت خودکار پرس‌وجو را می‌سازد و با جست‌وجو در داده‌های Stack Overflow نتایج به برنامه‌نویس نمایش داده می‌شود. برنامه‌نویس علاوه بر مشاهده مباحثه می‌تواند در کد به آن ارجاع دهد تا بقیه افراد گروه نیز بتوانند آن را مطالعه کنند.

داده‌ها در یک پایگاه داده رابطه‌ای ذخیره شده‌اند و جست‌وجو به وسیله Apache Solar انجام می‌شود. برای ساخت پرس‌وجو کدهای نزدیک اشاره‌گر mouse در نظر گرفته می‌شوند. اعمال بازیابی اطلاعات روی این کد انجام می‌شود و ۱۰ کلمه پرتکرار آن استخراج می‌شود. همچنین از بخش importهای فایل، کلمات استخراج می‌شوند. اجتماع این دو مجموعه پرس‌وجو را می‌سازد. نتیجه این پرس‌وجو به برنامه‌نویس نمایش داده می‌شود. این ابزار در جست‌وجو پشته فراخوانی را دخیل نمی‌کند و برای خطاهای مبتنی بر استثناء مناسب نیست.

Prompter -۶-۱-۲

ابزار Prompter [۴] تغییرات کد برنامه‌نویس را رصد می‌کند. زمینه‌ی کدهای جدید به سازنده پرس‌وجو فرستاده می‌شود. در صورتی‌که پرس‌وجوی جدید ساخته شود، به همراه زمینه‌ی کد به موتور جست‌وجو ارسال می‌شود.

جدول ۱- مقایسه سیستم‌های توصیه‌گر موجود برای اصلاح خطاهای برنامه‌نویسی

نام ابزار	انتقال به زمینه برنامه‌نویس	داده‌های ورودی	استفاده از کد برنامه‌نویس در جست‌وجو	پیش‌پردازش به روی کد داده‌های پایگاه داده	استفاده از پشته فراخوانی	جست‌وجوی تجزیه‌ای کد
Oscilloscope	-	منابع گزارش‌خطا	-	-	☑	-
Help Me Out	☑	پایگاه‌داده شخصی	☑	☑	☑	-
Surfclipse	-	صفحات وب	☑	-	☑	-
ابزار کوردیرو	-	Stack Overflow	-	☑	☑	-
Seahawk	-	Stack Overflow	☑	-	-	-
Prompter	-	Stack Overflow	☑	☑	-	-
Exception Tracer	☑	کدهای مخازن نرم‌افزاری	☑	-	☑	☑

موتور جست‌وجو با استفاده از موتورهای جست‌وجوی Google، Bing و Blekoo مباحثه‌های مرتبط از Stack Overflow را استخراج می‌کند. سپس این مباحثه‌ها رتبه‌بندی می‌شوند و به برنامه‌نویس نمایش داده می‌شود. در جست‌وجوها کلماتی که در اکثر سندها تکرار شده‌اند، ارزش جست‌وجوی کم‌تری دارند؛ زیرا این کلمات تمایز خوبی بین اسناد ایجاد نمی‌کنند. در نتیجه بهتر است این کلمات از پرس‌وجو حذف شوند. این مهم با معیاری به نام آنتروپی سنجیده می‌شود. هر چه آنتروپی یک کلمه بیش‌تر باشد، ارزش کم‌تری دارد. بعد از ساخته شدن پرس‌وجو و بازیابی مباحثه‌ها از موتورهای جست‌وجو، باید آن‌ها را رتبه‌بندی کنیم. رتبه‌بندی براساس ۸ ویژگی زیر انجام می‌شود:

۱. شباهت متنی بین کد برنامه‌نویس و متن (غیر کد) مباحثه.
۲. درصد خط‌هایی از کد برنامه‌نویس که با قسمتی از مباحثه شباهت دارد.
۳. نوع‌هایی که در کد برنامه‌نویس و مباحثه وجود دارند.
۴. متدهای فراخوانی شده در کد برنامه‌نویس که در مباحثه آمده است.
۵. امتیازی که کاربران Stack Overflow به مباحثه داده‌اند.
۶. مباحثه‌ای که جواب تایید شده داشته باشد، ارزش بیش‌تری دارد.
۷. تعداد پاسخ‌های مباحثه، در امتیازدهی تاثیرگذار است.
۸. تعداد برچسب‌های مباحثه که با کلمات به‌دست آمده از بخش Import کد برنامه‌نویس یکسان‌اند.

مباحثه‌ها براساس معیارهای بالا رتبه‌بندی می‌شوند و در صورتی‌که مباحثه‌ای وجود داشته باشد که از یک حدی امتیازش بیشتر باشد، به برنامه‌نویس نمایش داده می‌شود. این ابزار در جست‌وجو پشته فراخوانی را دخیل نمی‌کند و برای خطاهای مبتنی بر استثناء مناسب نیست.

۲-۱-۷- مقایسه سیستم‌های توصیه‌گر موجود برای اصلاح خطاهای برنامه‌نویسی

تاکنون روش‌های مرتبط به تفصیل بیان شدند. در این بخش این روش‌ها را با چند معیار مختلف با یکدیگر مقایسه می‌کنیم (جدول ۱). این معیارها با توجه به مراحل این ابزارها انتخاب شده‌اند. تمام این ابزارها شامل داده‌های ورودی، مرحله جست‌وجو و نمایش نتیجه هستند. ولیکن، هر کدام به نحوی این مراحل را داراهستند:

۱. انتقال به زمینه کد برنامه‌نویس: این معیار بیان می‌کند که آیا ابزار به روی خروجی خود پس پردازشی برای انتقال به زمینه کد برنامه‌نویس انجام می دهد یا خیر. از بین ابزارهای معرفی شده تنها Help Me Out مانند Exception Tracer به برنامه‌نویس راه‌حل را به زمینه کاربر منتقل می‌کند. این انتقال بسیار محدود است و در حد یک خط از کد، انجام می‌گیرد و لیکن هیچ کدام از این ابزارها پیشنهاداتشان را به زمینه کاربر به طور کامل منتقل نمی‌کنند.

۲. داده‌های ورودی: هر یک از ابزارهای معرفی شده با جست‌وجو به روی داده‌هایی به برنامه‌نویس پیشنهاد ارائه می‌دهد از این داده‌ها می‌توان صفحات وب، اطلاعات Stack Overflow را نام برد. ابزار ارائه شده توسط کوردیرو، Seahawk و Prompter تنها از Stack Overflow استفاده می‌کنند Oscilloscope. روش عمومی‌تری ارائه کرده است و جست‌وجو را به روی منابع گزارش خطا انجام می‌دهد؛ اما برای استفاده از اطلاعات وبگاه‌های مختلف باید آن‌ها را به استاندارد پایگاه داده آن تبدیل کرد. Help Me Out از هیچ داده آماده‌ای استفاده نمی‌کند و با استفاده از کاربران خودش، به‌روزرسانی می‌شود؛ در نتیجه تا وقتی ابزار توسط افراد زیادی استفاده نشود، پایگاه داده خالی خواهد بود و قابلیت اضافه کردن اطلاعات از وبگاه‌های دیگر را ندارد. Surfclipse نتایج جست‌وجوی موتور جست‌وجوی Google را رتبه‌بندی می‌کند. Exception Tracer از موتور جست‌وجوی BOA استفاده می‌کند. طبق گفته سازندگان این موتور جست‌وجو برای مخزن‌های نرم‌افزاری طراحی شده است و در ابتدا جست‌وجو به روی پروژه‌های Source Forge انجام می‌دهد. در نتیجه با بهبود BOA ابزار ارائه‌شده در این مقاله، دیگر محدود به Source Forge نخواهد بود.

۳. استفاده از کد برنامه‌نویس در جست‌وجو: در جست‌وجو می‌توان از پشته فراخوانی و کدبرنامه‌نویس بهره جست. این معیار استفاده از کدبرنامه‌نویس در جست‌وجو را بررسی می‌نماید. ابزار ارائه شده توسط کوردیرو و Oscilloscope برخلاف بقیه ابزارها در جست‌وجو، کد برنامه‌نویس را دخیل نمی‌کنند. داده‌هایی که جست‌وجو در آن‌ها صورت می‌گیرد، ممکن است شامل کد باشند؛ ابزارهای Oscilloscope، Surfclipse و Seahawk برای این بخش از اطلاعات تدبیر خاصی در نظر نگرفته‌اند و مانند بقیه متن‌ها با آن‌ها برخورد می‌کنند. اما بقیه ابزارها یک پیش‌پردازشی به روی این بخش از اطلاعات انجام می‌دهند تا جست‌وجو با کیفیت بهتری انجام شود. در کل همه ابزارها، جست‌وجوهایشان در انتها به جست‌وجوی متنی انجام می‌گیرد. Exception Tracer جست‌وجو را به صورت گرافی انجام می‌دهد و کد برنامه‌نویس در جست‌وجو تاثیر مستقیم دارد.

۴. پیش‌پردازش به روی کدهای پایگاه داده: برای جست‌وجوی دقیق‌تر و سریع‌تر برخی ابزارها به روی داده‌های ورودی خود پیش پردازش انجام می‌دهند. این معیار این مورد را مشخص می‌نماید. برخی از ابزارها در جست‌وجو با کد مانند متن برخورد می‌کنند (Oscilloscope، Surf Clips و Seahawk) و برخی دیگر برای افزایش دقت در جست‌وجو یک پیش‌پردازشی به روی کد انجام می‌دهند و در نهایت همان جست‌وجوی متنی را انجام می‌دهند (Help Me Out، ابزار کوردیرو و Seahawk). Exception Tracer

جست‌وجویش را به روی درخت تجزیه کد انجام می‌دهد و ساختار کد در جست‌وجو دخیل می‌شود.

۵. استفاده از پشته فراخوانی در جست‌وجو: این معیار استفاده از پشته فراخوانی در جست‌وجو را مورد بررسی قرار می‌دهد. تمامی ابزارها به غیر از Seahawk و Prompter در جست‌وجوهایشان از اطلاعات موجود در پشته فراخوانی استفاده می‌کنند. ابزارهای Oscilloscope، ابزار ارائه شده توسط کوردیرو و Seahawk پایگاه داده‌هایشان از محتویات دیگر وبگاه‌ها استفاده کرده‌اند. ابزار Help Me Out امکان ورود اطلاعات از دیگر وبگاه‌ها را ندارد، اما خودبه‌خود اطلاعاتش توسط کاربرانش به‌روزرسانی می‌شود. ابزارهای Surfclipse و Prompter جست‌وجو را به صورت برخط انجام می‌دهند و از این نظر اطلاعاتشان بدون نیاز به انجام کاری به‌روزرسانی می‌شوند. عملا پایگاه داده Exception Tracer در موتور جست‌وجوی BOA قرار دارد و برخط است. در نتیجه با به روزرسانی BOA پایگاه داده آن نیز ارتقا می‌یابد.

۶. جست‌وجوی تجزیه‌ای کد: عدم جست‌وجوی متنی هدف این معیار است. همان‌طور که در معیار قبلی توضیح داده شد، تمامی این ابزارها درنهایت کد را به‌صورت متن در نظر می‌گیرند و جست‌وجوی متنی انجام می‌دهند. اما Exception Tracer با مقایسه گراف‌های تولید شده از کد ساختار کد را به‌طور کامل در جست‌وجو دخیل می‌کند.

۲-۲- استخراج مشخصات

حوزه این روش‌ها بسیار گسترده می‌باشد و کاربردهای گوناگونی دارند. ولیکن هدف کلی این روش‌ها شناسایی پروتوکل‌هایی است که برنامه‌های کاربران بایستی رعایت کنند وقتی که از یک کتابخانه و یا چارچوب نرم‌افزاری استفاده می‌کنند. این روش‌ها به صورت عمومی به دو دسته پویا و ایستا تقسیم می‌شوند. روش‌های پویا به اجرای برنامه نیاز دارند و داده‌های خود را از داده‌های زمان اجرا استخراج می‌کنند. به عنوان نمونه می‌توان به روش‌های [۱۶-۱۸] اشاره نمود. روش [۱۹] ابتدا به‌صورت پویا قوانین را استخراج می‌کند، ولیکن بررسی عدم‌رعایت قوانین در کد برنامه‌نویس به‌صورت ایستا انجام می‌گیرد. روش‌های ایستا به اجرای برنامه‌ی برنامه‌نویس نیازی ندارند و با تحلیل کد او قوانین را استخراج می‌کنند. Exception Tracer به‌منظور ارائه پیشنهاد به برنامه‌نویس در جهت اصلاح استفاده غلط از یک رابط کاربردی برنامه‌نویسی ایجاد شده است. ضعف ابزارهای مشابه در جست‌وجوی متنی به روی کد است. به همین دلیل روش ارائه شده در این مقاله به یک دید سطح بالا از کد نیاز دارد تا جست‌وجو را با دقت بهتری انجام دهد. مهم‌ترین خصوصیت این دید سطح بالا این است که دنباله فراخوانی را نمایش دهد؛ زیرا دنباله فراخوانی شامل قوانین استفاده از رابط کاربردی برنامه‌نویسی است. ازآنجایی‌که دنباله فراخوانی برای ما اهمیت دارد، دیدهایی مثل نمودار کلاس برای ما مفید نخواهد بود. بهترین انتخاب گراف جهت‌دار است. علاوه بر دنباله فراخوانی، به‌منظور تبدیل کد به زمینه کاربر در دید سطح بالا نام و نوع اشیاء نیز باید نمایش داده شود. علت این امر در بخش ۳ آمده است. برای تولید یک دید سطح بالا از کد در Exception Tracer، از روش‌های موجود در این دسته کمک گرفته شده است. در ادامه، ابتدا اندکی در مورد GROUM [۷]، Jungloid [۲۰] و چند ابزار مشابه دیگر توضیحاتی ارائه می‌کنیم و درنهایت مزایا و معایب آن‌ها را بررسی می‌نماییم.

۲-۲-۱- GROUM

گراف GROUM مستقیم از کد تولید می‌شود. درخت AST کد به وسیله کتابخانه‌ای از Eclipse استخراج می‌شود. سپس با پیمایش آن، گراف ایجاد می‌شود. این گراف از نوع گراف جهت‌دار بدون دور (DAG) است و می‌توان به خاطر همین خاصیت، روند استفاده از یک API را به‌راحتی از آن استخراج نمود. راس‌ها در این گراف دو نوع هستند: (۱) رأس کنشی (فراخوانی متد، دسترسی به Field و فراخوانی Constructor) (۲) رأس کنترلی (بیان‌کننده ساختار کنترلی مانند while). یال‌ها در این گراف دو نوع هستند؛ برخی یال‌ها توالی خط‌های کد مربوط به هر رأس را مشخص می‌کند؛ برخی دیگر وابستگی داده‌ای را نشان می‌دهند. رأسی که یال وابستگی داده‌ای به آن وارد شده است، به رأس ابتدای یال وابستگی داده‌ای دارد و از داده‌ی مهیا شده توسط رأس ابتدای یال استفاده می‌کند.

۲-۲-۲- Jungloid

در این روش یک فرض وجود دارد. "برنامه‌نویس نوع شی نهایی موردنیاز خود را می‌داند اما نمی‌داند این شی را چگونه به دست آورد [۲۰]." در واقع ورودی این روش کد به همراه نوع خروجی است. در نهایت قطعه کد مربوط به این سؤال به برنامه‌نویس نمایش داده می‌شود. ابتدا از کد، یک گراف ساخته می‌شود. رأس‌های این گراف نوع‌های کلاس موجود در API هستند. یال‌ها ممکن است یکی از موارد زیر باشند:

۱. دسترسی به Field.
۲. فراخوانی Static متد و یا Constructor.
۳. فراخوانی متد به روی یک شی.
۴. ارجاع از شیء فرزند به پدر (Up cast).
۵. ارجاع از شیء پدر به فرزند (Downcast).

بعد از رسم گراف کد مربوط به API، مسیرهای منتهی به نوع موردنظر بررسی می‌شوند. مسیرهایی که دور ندارند، جواب ما هستند. در نهایت این گراف به کد تبدیل می‌شود و به برنامه‌نویس نمایش داده می‌شود. روش‌های دیگری نیز برای هدف Jungloid که در بالا ذکر شد، ارائه شده است. X Snippet [۲۱]، API Synth [۲۲] و PARSE Web [۲۳] سه روش دیگر از این دسته هستند. از آنجایی این دسته برای پیاده‌سازی انتخاب نشده است، وارد جزئیات تفاوت ابزارهای این دسته نشده‌ایم.

۲-۲-۳- روش انتخابی

این روش‌ها از کد، گراف بدون دور DAG ایجاد می‌کنند و پردازش‌هایی روی آن گراف‌ها انجام می‌دهند. نوع و نحوه کشیدن گراف در این روش‌ها برای ما اهمیت دارد. در روش‌های یاد شده، به‌جز GROUM فرض بر این است که می‌خواهیم از یک قطعه کد، روش ساختن شی از نوع داده‌ای موردنظر را پیدا کنیم. در این روش‌ها، ورودی یک قطعه کد به همراه یک نوع داده‌ای است و خروجی آن یک دنباله از فراخوانی متدهایی است که به شی از نوع موردنظر ختم می‌شود. اما GROUM این فرض را ندارد و یک گراف با جزئیات بیشتر از کد ترسیم می‌کند. GROUM برای جلوگیری از بزرگ شدن، گراف‌هایش را درون متد ایجاد می‌کند. به این معنی که برای هر متد یک گراف جدا کشیده می‌شود و وابستگی‌های فرامتد در نظر گرفته نمی‌شوند. همچنین در گراف GROUM ساختار شرطی نیز وجود دارد، اما در مابقی روش‌ها خیر. Jungloid برای پاسخ‌گویی در زمان مناسب، نیاز دارد تا نوع شی ورودی نیز تعیین شود. به عبارت دیگر، ورودی روش، کد، نوع شی ورودی و نوع شی نهایی خواهد بود و بین این دو رأس در گراف به دنبال کوتاه‌ترین مسیر می‌گردد. از طرفی در Jungloid کوتاه‌ترین مسیر به عنوان جواب

انتخاب می‌شود که ممکن است اشتباه باشد [۲۲].

۳- معرفی مسئله

برنامه‌نویسان هنگام استفاده از مرورگر برای یافتن راه‌حل استثناء خود معمولاً فقط از پیغام استثناء استفاده می‌کنند، در حالی که زمینه کد اطلاعاتی دارد که جست‌وجو را دقیق‌تر می‌کند. همچنین راه‌حلی که برنامه‌نویس به صورت دستی پیدا می‌کند، شامل کدهای اضافی است و برای استفاده باید نام متغیرهای موجود در آن را تغییر دهد؛ این دو عمل (حذف کدهای اضافی و تغییر نام متغیر) به صورت دستی ممکن است به‌درستی انجام نپذیرد و برنامه‌نویس مجبور شود زمان زیادی را صرف اصلاح آن کند. علاوه بر موارد ذکر شده، تغییر محیط برنامه‌نویسی از محیط یکپارچه ایجاد نرم‌افزار به مرورگر و بالعکس عدم‌تمرکز برنامه‌نویس را در پی خواهد داشت. برای حل این مشکل روش‌های مختلفی ارائه شده است که در بخش ۲ به تفصیل بیان شدند. ولیکن، این روش‌ها نواقصی دارند که عبارتند از: (۱) ابزارها فقط نمونه کد پیشنهاد می‌دهند و یا مباحثه، (۲) در جست‌وجو از اطاعات زمینه به‌خوبی استفاده نمی‌شود، (۳) نمونه کد پیشنهادی در زمینه کاربر نیست. این دلایل باعث شد تا Exception Tracer ایجاد گردد. با رخداد استثناء این ابزار فعالیت خود را آغاز می‌کند و به‌طور خودکار قطعه کدها و مباحثه‌های مرتبط را به برنامه‌نویس نمایش می‌دهد.

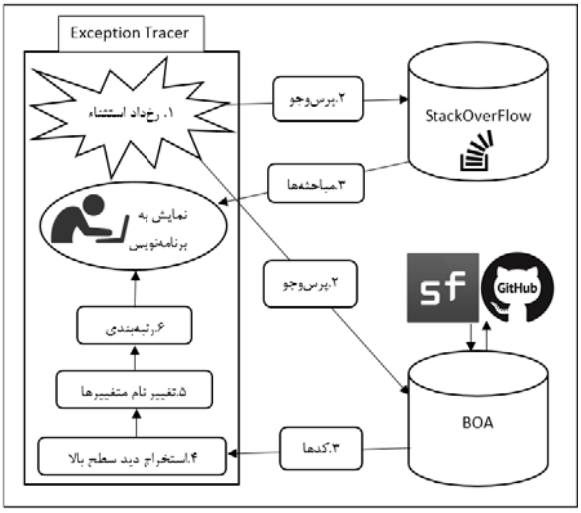
استثناءها انواع مختلفی دارند که به صورت کلی، آنها را به دو دسته تقسیم می‌کنیم: (۱) استثناءای که به علت عدم‌رعایت قوانین حاکم بر رابط کاربردی برنامه‌نویسی ایجاد شده است و (۲) بقیه استثنائات. برای استثناءهای دسته اول قطعه کد می‌تواند مفید باشد. زیرا برنامه‌نویس با دیدن توالی فراخوانی متدها و یا نحوه ساخته‌شدن اشیاء اشتباه خود را می‌یابد. اما کدهایی که استثناء نوع دوم را ایجاد کرده‌اند، قطعه کد نمی‌تواند مفید باشد؛ زیرا برنامه‌نویس دنباله فراخوانی مورد انتظار را رعایت کرده است ولی به عنوان مثال داده غلط به عنوان ورودی داده شده است.

برای درک بهتر این تفاوت به کد موجود در شکل ۱ توجه فرمایید. در این شکل کد ایجاد شی برای تعامل با یک فایل را مشاهده می‌نمایید. درصورتی‌که آدرس فایل غلط باشد، برنامه با استثناء File Not Found Exception متوقف می‌شود. در این مثال نمونه کد نمی‌تواند برای برنامه‌نویس مفید باشد زیرا الگوهای استفاده در تعامل با فایل رعایت شده‌اند؛ ولی پارامتر ورودی غلط است. برای این قبیل استثناءها که قوانین استفاده از کتابخانه رعایت شده است اما داده ورودی غلط است، مباحثه‌های مرتبط از Stack Overflow را پیشنهاد می‌دهیم. درصورتی‌که در همین مثال آدرس فایل غلط نباشد، به دلیل آن‌که در خط ۱۰ فایل بسته شده و نمی‌توان روی فایل بسته شده عملیات خواندن انجام داد، خط ۱۱ منجر به استثناء IO Exception می‌شود. در این حالت قوانین استفاده از کتابخانه رعایت نشده است و نمونه کد مفید خواهد بود.

```
1  String log="config loaded";
2  Properties config = new Properties();
3  InputStream in =
4      new FileInputStream("config.properties");
5  config.load(in);
6  System.out.println(log);
7  String path = config.getProperty("path");
8  FileInputStream reader =
9      new FileInputStream(new File(path));
10 reader.close();
11 reader.read();
```

شکل ۱- کد خطادار

همان‌طور که در مثال بالا بیان‌شده برای استثناءهای مربوط به عدم رعایت قوانین کتابخانه‌ها قطعه کد مفیدتر است و برای بقیه استثناءها قطعه کد مفید نخواهد بود و مباحثه‌های Stack Overflow این کمبود را جبران خواهند کرد.



شکل ۲- فرآیند Exception Tracer

۴- راه‌کار پیشنهادی

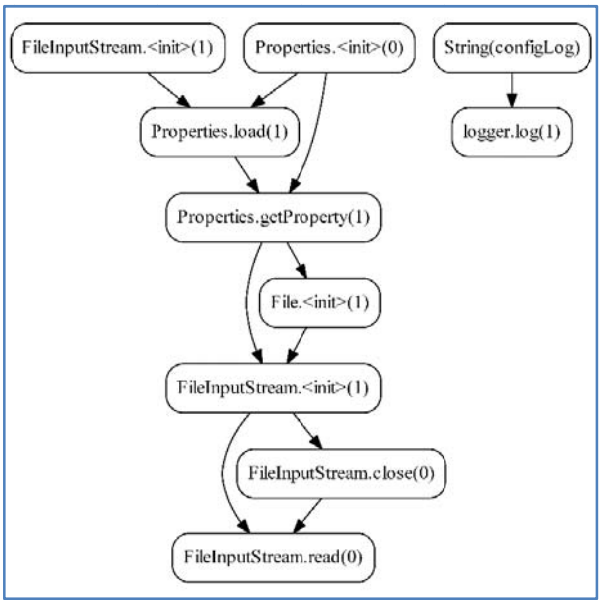
همان‌طور که در بخش‌های قبلی عنوان شد، عدم‌رعایت قوانین کتابخانه‌ها و چارچوب‌های نرم‌افزاری منجر به خطا می‌شود. دسته‌ی دیگری از خطاها نیز به دلیل دادن ورودی غلط است. در این دسته، با اینکه برنامه‌نویس قوانین چارچوب را رعایت کرده است، اما به دلیل عدم‌وجود ورودی درست استثناء رخ می‌دهد. روش ارائه شده در این مقاله دو نوع یاد شده را پوشش می‌دهد. فرآیند Exception Tracer برای ایجاد راه‌حل، در شکل ۲ نشان داده شده است. با رخ داد استثناء، پشته فراخوانی را خواهیم داشت. با استفاده از آن، متدی که فراخوانی آن باعث رخ داد استثناء شده است، را می‌یابیم. سپس یک جست‌وجو به وسیله موتور جست‌وجوی BOA [۲۴] به روی کدهای پروژه‌های Source Forge انجام می‌گردد تا فایل‌های شامل فراخوانی ذکر شده را بیابد. BOA یک موتور جست‌وجو است که به‌صورت موازی به روی کدهای پروژه‌های Source Frog جست‌وجو انجام می‌دهد. پرس‌وجوهای این موتور جست‌وجو، یک زبان خاص دارد که جست‌وجوی موازی را امکان‌پذیر می‌نماید. در مرحله بعد از کدهای به‌دست آمده گراف GROUM [۷] استخراج می‌شود. سپس از گراف‌های به‌دست آمده الگوهای استفاده استخراج می‌شوند. آنگاه گراف GROUM کد برنامه‌نویس استخراج می‌شود. سپس با مقایسه گراف‌ها، قطعه کدها رتبه‌بندی می‌شوند. در نهایت با استفاده از گراف‌ها، کدهای غیرمرتبط از قطعه کدها حذف می‌شوند و بعد از تبدیل آن‌ها به زمینه کاربر (تغییر نام متغیرها)، قطعه کدها به برنامه‌نویس نمایش داده می‌شود. در ادامه، این گام‌ها به تفصیل بیان می‌شوند.

```
Exception in thread "main" java.io.IOException: Stream Closed
    at java.io.FileInputStream.read0(Native Method)
    at java.io.FileInputStream.read(FileInputStream.java:210)
    at Test.fileInput(Test.java:20)
    at Test.main(Test.java:26)
```

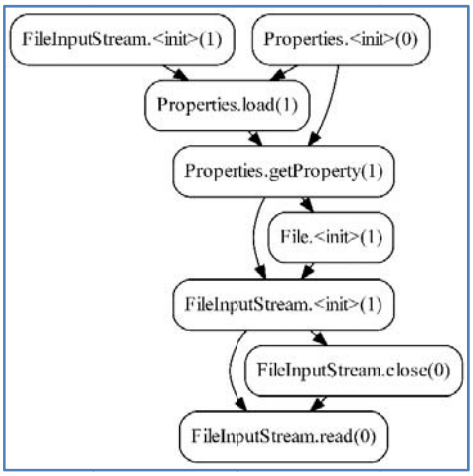
شکل ۳- استثناء مربوط به کد شکل ۱

۴-۱- ساخت پرس‌وجو

با استفاده از پشته فراخوانی استثناء، پرس‌وجو به زبان BOA تولید می‌شود. BOA یک موتور جست‌وجو است که به صورت موازی به روی کدهای پروژه‌های مخازن نرم‌افزاری (به صورت آزمایشی اکنون فقط به روی پروژه‌های Source Forge) جست‌وجو انجام می‌دهد. BOA مبتنی بر زبان موازی Sewall [۲۵] است و از چارچوب Map Reduce پشتیبانی می‌کند. با پیمایش خط به خط پشته فراخوانی نام متد و کلاس داخلی‌ترین فراخوانی از کد برنامه‌نویس که منجر به استثناء شده است را می‌یابیم. Exception Tracer خطوط پشته فراخوانی را از بالا به پایین می‌خواند. اولین مکانی از پشته فراخوانی که خط بعدی آن مربوط به کد برنامه‌نویس است، اطلاعات فراخوانی موردنظر را مشخص می‌کند. از این خط نام متد و نام کلاس شی‌ای که آن متد به روی آن فراخوانی شده است را می‌یابد. در ادامه این رأس را "فراخوانی اولیه" می‌نامیم.



شکل ۴- گراف GROUM مربوط به کد شکل ۱



شکل ۵- گراف GROUM بعد از حذف راس‌های غیرمرتبط

پرس‌وجو تنها از فراخوانی اولیه یادشده ساخته می‌شود. پرس‌وجو با استفاده از اسم کلاس شی‌ای که فراخوانی روی آن منجر به استثناء شده و نام متد ساخته می‌شود. با استفاده از BOA فایل‌های شامل این فراخوانی را می‌یابیم. این پرس‌وجو کران بالایی از مجموعه کدهای مرتبط با استثناء را برمی‌گرداند، زیرا فایل‌های خروجی فقط از این منظر محدود می‌شوند که شامل فراخوانی از کد برنامه‌نویس است که استثناء در آنجا رخ داده است. در شکل ۳ استثناء مربوط به کد شکل ۱ نمایش داده شده است. در این شکل فراخوانی موردنظر با یک اشاره‌گر نشان داده شده است.

۴-۲- استخراج کد

پرس‌وجوی ساخته شده در مرحله ۴-۴-۱ در BOA ثبت می‌شود. نتیجه این پرس‌وجو فایل‌هایی است که شامل این فراخوانی هستند. خروجی این جست‌وجو شامل لیستی از فایل‌های شامل فراخوانی موردنظر است. لیست نهایی به ازای هر فایل خروجی شامل موارد زیر است:

۱- نوع مخزن نرم‌افزاری (SVN، CVS و یا GIT).
۲- لینک مخزن نرم‌افزاری پروژه شامل فایل موردنظر.
۳- آدرس فایل در آن مخزن نرم‌افزاری.
۴- شماره نسخه فایل.

با اضافه کردن آدرس فایل به لینک مخزن نرم‌افزاری به‌تنهایی، لینک نهایی فایل ساخته نمی‌شود. برای تولید لینک فایل باید با توجه به نوع مخزن پیش‌پردازشی به روی آدرس فایل انجام بگیرد.

۴-۳- استخراج دید سطح بالا

یکی از برتری‌های Exception Tracer نسبت به ابزارهای مشابه ایجاد دید سطح بالا از کد و استفاده از آن در جست‌وجوها، حذف کدهای اضافی و تبدیل به زمینه کد برنامه‌نویس است. Exception Tracer برای ایجاد دید سطح بالا از GROUM کمک گرفته است. با ذکر یک مثال علت این انتخاب را بیان می‌کنیم. فرض کنید محلی از کد برنامه‌نویس که مربوط به فراخوانی یک متد است و استثناء در آنجا رخ داده است، مربوط به فراخوانی یک متد با نوع خروجی void باشد. در روش‌های به‌غیر از GROUM این فراخوانی‌ها در گراف ظاهر نمی‌شوند و روش ما با مشکل روبرو می‌شود. شکل ۴ گراف GROUM مربوط به کد شکل ۱را نمایش می‌دهد (این گراف به وسیله ابزار Graph Via ایجاد شده است). GROUM از سه نوع رأس داده‌ای (مستطیل)، فراخوانی (مستطیل با گوشه‌های نرم) و شرطی (لوزی) تشکیل شده است. یال‌های GROUM همه یکسان هستند و با خط تیره نمایش داده می‌شوند. این یال‌ها براساس وابستگی داده‌ای بین رأس‌ها کشیده می‌شوند. در بخش ۴-۴-۴ نحوه مقایسه گراف‌ها را بیان می‌کنیم.

۴-۴- رتبه‌بندی قطعه کدها

گراف شکل ۴ از دو بخش همبند تشکیل شده است. بخش‌های جدا بدین معنی است که بین رأس‌های این دو بخش هیچ‌گونه ارتباط داده‌ای وجود ندارد. در مرحله ساخت پرس‌وجو، آن فراخوانی که منجر به استثناء شده بود، شناسایی شد. از گراف فقط زیر بخش همبندی که شامل آن رأس است را نگه می‌داریم و بقیه گراف را حذف می‌کنیم. عملا با این کار بخش‌هایی از کد که نسبت به محل استثناء هیچ‌گونه رابطه داده‌ای (با هر عمقی) ندارند، حذف می‌شوند. در شکل ۵ گراف بعد از این عملیات قابل مشاهده است. در این شکل دو رأس مربوط به خطوط ۱ و ۶ به دلیل عدم‌وجود وابستگی داده‌ای مستقیم یا غیرمستقیم با رأس فراخوانی اولیه حذف شده‌اند.

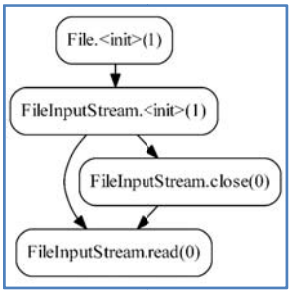
به منظور مقایسه دو گراف، از روش بیان شده در [۷] استفاده شده است. تمام مسیرهای منتهی به رأس فراخوانی اولیه را می‌یابیم. عدد شباهت دو گراف درصد مسیرهای مشابه آن دو گراف است. P مسیر و N رأس‌ها را نشان می‌دهند. تابع is Equal Path (P, P') درصورتی‌که این دو مسیر عیناً یکسان باشند، عدد یک و در غیر این صورت عدد صفر را برمی‌گرداند. دو مسیر در صورتی یکسان‌اند که ازنظر تعداد رأس و همچنین برچسب‌های درون هر رأس کاملاً مساوی باشند. برچسب‌های درون هر رأس از کلاس شی و متدی که فراخوانی به روی آن صورت گرفته، ساخته می‌شود. یعنی در جست‌وجو نام شی تاثیری ندارد. اگر تعداد رئوس دو مسیر یکسان نباشند و یا یک رأس از یک گراف با رأس متناظر خود در گراف دیگر متفاوت باشد، خروجی این تابع صفر خواهد بود. این تابع به شرح زیر تعریف می‌شود:

$$P =< N_1, N_2, \dots, N_n >$$
$$\text{isEqualPath}(P, P') = \begin{cases} 1 & |P| = |P'| \wedge \forall i \epsilon \{1, \dots, n\}: N_i = N'_i \\ 0 & \text{else} \end{cases}$$

از آنجایی‌که در محاسبه شباهت‌ها، همواره گراف کد برنامه‌نویس با یک گراف دیگر مقایسه می‌شود، تابع شباهت به صورت زیر محاسبه می‌شود. U گراف کد برنامه‌نویس و G گراف مربوط به نمونه کد است. رأس x همان رأس مربوط به کد محل وقوع استثناء (فراخوانی اولیه) در کد برنامه‌نویس است. همچنین، تابع paths (G, n) تمام مسیرهای منتهی به رأس n را نمایش می‌دهد.

$$\text{sim}(U, G) = \frac{\sum_{p \in athes(U,x)} \sum_{p' \in pathes(G,x)} \text{isEqualPath}(p, p')}{|pathes(U, x)|}$$

هر چه مسیرها طولانی‌تر باشند، هزینه محاسبه شباهت دو گراف نیز بیش‌تر می‌شود. در ادامه، راه‌حلی برای کاهش طول مسیرها بدون کم شدن دقت را با ذکر یک مثال بیان می‌کنیم. فرض کنید یک متد از کتابخانه مدنظر است که یک رشته به عنوان پارامتر ورودی می‌گیرد. یک کاربر ممکن است این رشته را مستقیماً به متد دهد (خط اول در نمونه کد شکل ۱) و یا آن را از یک فایل بخواند و به متد دهد (خط هفتم، در نمونه کد شکل ۱). در این مثال، در نظر گرفتن کل رأس‌های مسیر، نه‌تنها دقت جست‌وجو را افزایش نمی‌دهد، بلکه باعث کاهش دقت نیز می‌شود. در این مثال ما دو مسیر داریم که تابع sim آن‌ها را یکسان تشخیص نمی‌دهد، در حالی‌که این بخش از مسیر نباید تأثیری در نتیجه بگذارد؛ زیرا کاربر با توجه به نیازهای خود، ورودی متد را به‌گونه‌ای متفاوت تعیین می‌کند. در حالت کلی، این ورودی می‌تواند هر نوع شی دیگری نیز باشد. البته اگر کلاس این شی در کتابخانه کلاس مربوط به رأس فراخوانی اولیه باشد، دیگر نمی‌توان با قاطعیت، رای به حذف آن از مسیر داد. بنابراین به ازای هر مسیر از انتهای آن شروع می‌کنیم، رأس‌هایی که با رأس اولیه هم کتابخانه هستند را نگه می‌داریم. مسیر از اولین رأسی که کلاسش در بسته کتابخانه نبود، قطع می‌شود. همانگونه که در شکل ۶ مشاهده می‌شود، فقط چهار رأس پایینی گراف که مربوط به کتابخانه مورد استفاده هستند، باقی می‌مانند و بقیه رأس‌ها حذف می‌شوند.



شکل ۶- گراف نمونه کد بعد از اصلاح مسیرها

برای تشخیص هم کتابخانه بودن کلاس‌ها از اسم بسته آن‌ها کمک می‌گیریم. تابع is Same Library بدین‌صورت عمل می‌کند که بزرگ‌ترین زیررشته پیشوندی بسته‌های دو کلاس را به دست می‌آورد. سپس تعداد نقطه‌های این زیررشته را می‌شمارد. تعداد نقطه‌ها عمق بسته‌های مشترک دو کلاس را مشخص می‌کند. در نام‌گذاری بسته‌های کتابخانه‌ها نام چند بسته اول می‌تواند به عنوان تمایز بین کتابخانه‌ها درنظر گرفته شود. با بررسی کتابخانه‌های استاندارد Java و کتابخانه‌های Apache در نظر گرفتن نام دو بسته اولیه کتابخانه می‌تواند تمایز ایجاد نماید؛ در نتیجه اگر زیررشته مشترک بسته‌های دو کلاس بیش از ۲ نقطه داشته باشند، هم‌کتابخانه تشخیص داده می‌شوند.

$$\text{isSameLibrary}(C, C') = \begin{cases} 0 & \text{اگر C و C' در یک کتابخانه نباشند} \\ 1 & \text{اگر C و C' در یک کتابخانه باشند} \end{cases}$$

به عنوان مثال، دو کلاس java.io.File و java.io.FileInputStream هم‌کتابخانه تشخیص داده می‌شوند، زیرا بزرگ‌ترین زیررشته پیشوندی مشترکشان (یعنی، java.io)، ۲ نقطه دارد.

بعد از هرس کردن گراف، محاسبه شباهت کدهای دریافت شده از Source Forge و کد برنامه‌نویس با هزینه کم‌تری محاسبه می‌شود. به عنوان مثال مسیرهای منتهی به رأس محل وقوع استثناء در کد شکل ۱ به شرح زیر است:

File.<init>(1)
FileInputStream.<init>(1)
FileInputStream.close(1)
FileInputStream.read(0)

File.<init>(1)
FileInputStream.<init>(1)
FileInputStream.read(0)

شباهت گراف نمونه کد دریافتی درصورتی‌که عیناً هر دو مسیر را دارا باشند، عدد "۱" و اگر یک مسیر را دارا باشد، عدد "۰.۵" و درصورتی‌که هیچ کدام از مسیرها را دارا نباشد، عدد "۰" خواهد بود. گراف‌ها بعد از هرس شدن، به کد تبدیل می‌شوند. به دلیل وجود مرحله هرس قبل از تبدیل به کد، قسمت‌هایی از کد که به فراخوانی اولیه مرتبط نیستند، حذف می‌شوند. همچنین بخش‌های غیروابسته به کتابخانه فراخوانی اولیه نیز حذف می‌شوند. بعد از محاسبه شباهت هر نمونه کد با کد برنامه‌نویس، لیست قطعه کدهای پیشنهادی به او نمایش داده می‌شود.

```
1 File tmpFile =
2     File.createTempFile("b12-app-paths",".reg");
3 String[] cmdArgs =
4     new String[]{"regedit.exe","/E"};
5 tmpFile.delete();
6 FileInputStream fis =
7     new FileInputStream(tmpFile);
8 fis.read(magic);
9 fis.close();
```

(الف) قطعه کد نمونه قبل از انتقال به زمینه کد برنامه‌نویس

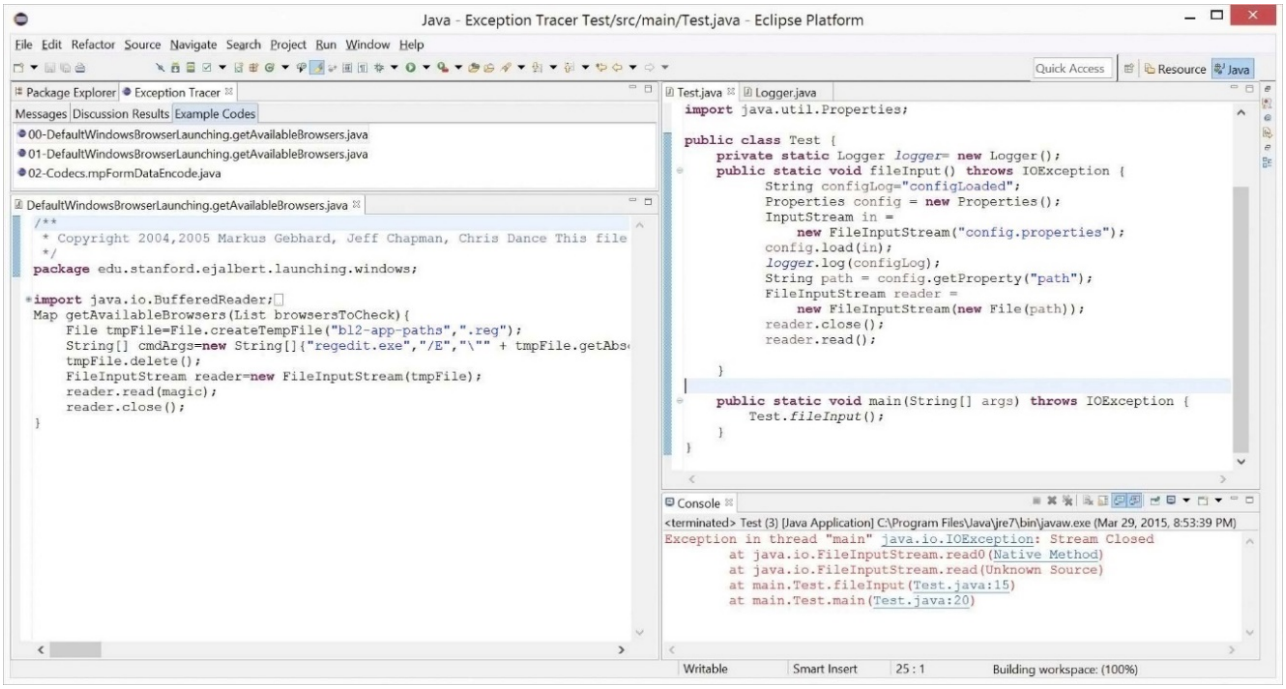
```
1 File tmpFile =
2     File.createTempFile("b12-app-paths",".reg");
3 String[] cmdArgs =
4     new String[]{"regedit.exe","/E"};
5 tmpFile.delete();
6 FileInputStream reader =
7     new FileInputStream(tmpFile);
8 reader.read(magic);
9 reader.close();
```

(ب) قطعه کد نمونه بعد از انتقال به زمینه برنامه‌نویس

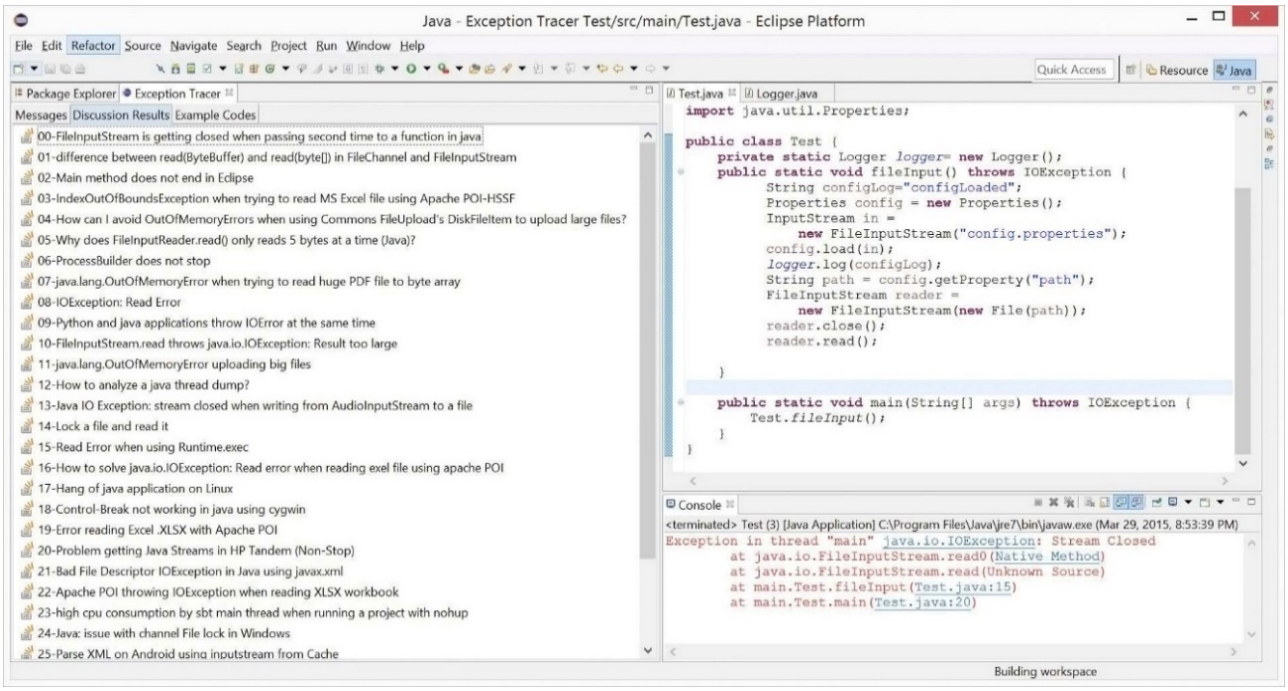
شکل ۷- قطعه کد نمونه قبل و بعد از انتقال به زمینه کد برنامه‌نویس

۴-۵- تبدیل به زمینه برنامه‌نویس

هر قطعه کد یک فضای نامی[۶] دارد که شامل اسم متغیرهای آن است. برنامه‌نویس هنگام استفاده از قطعه کد، باید نام متغیرها را به صورت دستی تغییر دهد که ممکن است این کار منجر به اشتباه و در نتیجه آن، اتلاف وقت شود. با توجه به روش ارائه شده در grappa [۲۶]، هر رأس گراف قطعه کد با هر رأس همسایه‌های گراف برنامه‌نویس مقایسه می‌شود. شباهت هر رأس از گراف برنامه‌نویس که بیشتر به رأس موردنظر از گراف اول بود، به عنوان رأس‌های متناظر مشخص می‌شوند. از رأس‌های متناظر برای تبدیل نام متغیرها استفاده می‌کنیم. در مثال نشان داده شده در شکل ۷، یک مثال از قطعه کد نهایی قبل و بعد از انتقال به زمینه برنامه‌نویس قابل مشاهده است. قبل از این تغییر، متغیر reader در قطعه کد اولیه نام دیگری داشت. همچنین، شکل ۸، محیط Exception Tracer در هنگام توصیه نمونه کد به کاربر را نمایش می‌دهد.



شکل ۸- محیط Exception Tracer در حالت پیشنهاد نمونه کدهای مرتبط از سایت Source Forge



شکل ۹- محیط Exception Tracer در حالت پیشنهاد مباحثه‌های مرتبط از سایت Stack Overflow

۴-۶- جست‌وجو در Stack Overflow و نمایش مباحثه‌های مرتبط

همان‌طور که در بخش‌های قبلی نیز بیان شد، درصورتی‌که برنامه‌نویس قوانین استفاده از کتابخانه را رعایت کرده باشد، قطعه کد لزوماً نمی‌تواند مفید باشد. همچنین، این امکان وجود دارد که قطعه کد مناسبی برای حل استثناء پیش آمده، در Source Forge موجود نباشد. در این مواقع، نمایش مباحثه‌های مرتبط از Stack Overflow می‌تواند مفید واقع شود. به همین دلیل، Exception Tracer همراه هر جست‌وجو، لیستی از مباحثه‌های مرتبط از Stack Overflow را نیز نمایش می‌دهد. برای این منظور، با استفاده از پیغام استثناء و API مورد استفاده کاربر یک پرس‌وجو ساخته می‌شود و به وسیله واسط REST مهیا شده توسط وب‌سایت Stack Overflow جست‌وجو انجام می‌گیرد، و نتایج جست‌وجو در یک لیست به کاربر نمایش داده می‌شود (شکل ۹).

روش پیشنهادی در این مقاله، در ابتدا فقط شامل پیشنهاد قطعه کدهای مرتبط بود. ولیکن، در عمل متوجه شدیم که قطعه کد مرتبط به تنهایی لزوماً نمی‌تواند در همه موارد مفید باشد و در برخی مواقع مباحثه‌ها مفیدترند. به همین دلیل، به روش پیشنهادی، نمایش مباحثه‌های مرتبط نیز اضافه شد. پیشنهاد همزمان قطعه کد و مباحثه از نوآوری‌های این روش است. ولیکن، تمرکز اصلی ما بر ارائه قطعه کدهای مرتبط بوده است و برای پیشنهاد مباحثه‌های مرتبط، از یک روش ساده یاری جسته‌ایم. با این حال، همانطور که در بخش ارزیابی بحث خواهد شد، همین روش ساده نیز نتایج را بهبود بخشیده است.

۵- ارزیابی

ابزار Exception Tracer به منظور تسهیل فرآیند یافتن راه‌حل یک استثناء معرفی شده است. برای ارزیابی این‌که چقدر این ابزار در رسیدن به این هدف موفق بوده است، در این بخش به سوالات تحقیق زیر پاسخ می‌دهیم:

۱- دقت ابزار برای چند نمونه کد پیشنهادی چقدر است؟

۲- نسبت به دیگر روش‌ها، چقدر در زمان برنامه‌نویس صرفه‌جویی می‌شود؟

۵-۱- ارزیابی کارایی و دقت روش

در ارزیابی کارایی و دقت روش معیار "رتبه اولین پیشنهاد مرتبط" محاسبه شده است. بدین معنی که به ازای هر لیست خروجی که به برنامه‌نویس داده می‌شود رتبه اولین پیشنهادی که به او در یافتن راه‌حل کمک کرده است، چند است. به عنوان مثال، اگر به برنامه‌نویس یک لیست شامل ۱۰ قطعه کد ارائه شود و قطعه کد سوم به او در یافتن راه‌حل استثناء کمک کند، مقدار این معیار برای این آزمایش ۳ می‌شود. این معیار برای ۱۶ استثناء مختلف محاسبه شده است. ازآنجایی‌که این ابزار به همراه قطعه کد، مباحثه نیز ارائه می‌کند، در هر آزمایش این معیار برای مباحثه‌ها و قطعه کدهای پیشنهادی به صورت جداگانه محاسبه شده است.

۵-۱-۱- برپایی ارزیابی

برای اندازه‌گیری دقت روش نیاز است تا چند نمونه کد که در زمان اجرا با استثناء متوقف می‌شوند، داشته باشیم و برای این نمونه کدها خروجی Exception Tracer را ثبت نماییم. در ادامه نحوه انتخاب نمونه کدهای دارای استثناء و نحوه اندازه‌گیری بیان شده است.

۵-۱-۲- انتخاب کتابخانه‌ها

ابزار Exception Tracer از فراخوانی محل استثناء و اطلاعات زمینه کد برنامه‌نویس به او پیشنهادهایی ارائه می‌کند. برای تولید استثناءهای مختلف از نمونه کد مربوط به کتابخانه‌های مختلفی کمک گرفته شده است. ۹ مورد از کتابخانه‌ها مربوط به کتابخانه‌های Apache هستند ۳ مورد هم مربوط به کلاس‌های پایه Java هستند. Apache در مجموع حدود ۲۷۰ پروژه (شامل کتابخانه، چارچوب و غیره) دارد و پروژه‌هایش به علت معروفیت توسط برنامه‌نویسان زیادی استفاده می‌شود. در سه مرحله از ۲۷۰ پروژه به ۹ کتابخانه رسیدیم. در مرحله اول با توجه به نام و حوزه استفاده آن پروژه، حدود ۱۰۰ کتابخانه انتخاب شدند. از بین ۱۰۰ کتابخانه با بررسی دقیق‌تر، ۲۰ کتابخانه‌ای که در اینترنت معروف‌تر بودند، انتخاب شدند. در حال حاضر و تا وقتی‌که BOA مخزن نرم‌افزاری‌اش را گسترش ندهد، پیش‌نیاز یافتن راه‌حل توسط Exception Tracer این است که برای آن کتابخانه در Source Forge نمونه کد وجود داشته باشد. برای اطمینان از وجود نمونه کد، کتابخانه‌های معروف‌تر انتخاب شدند. در مرحله سوم سعی شد برای این کتابخانه‌ها نمونه کد ایجاد شود و با تغییراتی باعث رخداد استثناء در زمان اجرا شویم.

۵-۱-۳- نحوه یافتن نمونه کد

به ازای هر کتابخانه یک یا چند نمونه کد صحیح از اینترنت استخراج کردیم. نمونه کدها اکثرا از وب‌سایت‌های معرفی کتابخانه‌ها و یا با جست‌وجو در Google استخراج شدند. سپس با حذف، تغییر یا اضافه کردن یک فراخوانی، به نمونه کد دارای استثناء رسیدیم.

۵-۱-۴- نحوه ایجاد استثناء

در بخش قبلی ما به ازای هر کتابخانه یک یا چند نمونه کد داریم که بدون مشکل اجرا می‌شوند. در این مرحله هدف این است تا با ایجاد تغییراتی در نمونه کد باعث رخ داد استثناء در زمان اجرای آن بشویم. با انجام یکی از تغییرات زیر در یک یا چند خط از کد به نمونه کد آماده برای ارزیابی می‌رسیم:

۱- حذف فراخوانی

۲- اضافه کردن فراخوانی

۳- تغییر فراخوانی

۵-۱-۵- اجرای ارزیابی

در این مرحله ابزار Exception Tracer را بر روی یک محیط یکپارچه ایجاد نرم‌افزار نصب نمودیم و هر یک از نمونه کدهای تولید شده در مرحله قبل را در آن اجرا کردیم. با رخ داد استثناء و ارائه پیشنهاد، اعداد زیر استخراج شدند:

۱- تعداد نمونه کدهای استخراج شده از BOA: برای افزایش در سرعت پاسخ‌دهی ابزار، محدودیت ۲۰تایی برای تعداد لینک‌هایی خروجی از BOA در نظر گرفته شده است. از آنجایی‌که دانلود فایل مربوط به هر لینک در حدود یک ثانیه زمان می‌برد، نمی‌توان این عدد را خیلی بزرگ در نظر گرفت. از طرفی ممکن است لینک خراب باشد، در نتیجه تعداد پیشنهادات ابزار در

نهایت در حدود ۵ تا ۱۰ قطعه کد است.

۲- رتبه قطعه کدی که شامل راه‌حل برطرف کردن استثناء است: برای یافتن این عدد از ابتدا تک‌تک قطعه کدها را بررسی کردیم و رتبه قطعه کدی که شامل معکوس تغییر ایجاد شده توسط ما بود را یادداشت نمودیم. به عنوان نمونه درصورتی‌که ما یک فراخوانی را حذف کرده باشیم، به دنبال رتبه قطعه کدی که شامل آن فراخوانی (با رعایت توالی موردنظر) بوده است، می‌گشتیم.

۳- تعداد مباحثه‌های بازگردانده شده: تعداد کل مباحثه‌هایی که ابزار به عنوان مباحثه مرتبط نمایش می‌دهد.

۴- رتبه اولین مباحثه مرتبط با راه‌حل استثناء: رتبه مباحثه‌ای که در پاسخ‌های آن، راه‌حل رفع استثناء موجود باشد را یادداشت کردیم.

۵-۱-۶- تحلیل نتایج

نتایج ارزیابی در جدول ۲ قابل مشاهده است. در ۶۲.۵ درصد موارد قطعه کد شامل راه‌حل استثناء در ۵ نتیجه اول وجود داشته است. در ۴۴ درصد موارد مباحثه مرتبط با استثناء در ۵ نتیجه اول نمایش داده شده، وجود داشته است. در روش ارائه شده در این مقاله تمرکز به روی تولید قطعه کد بوده است و برای جست‌وجو در Stack Overflow پرس‌وجوی بسیار ساده‌ای تولید می‌کند؛ به همین دلیل در ارزیابی انقدر ضعیف بوده است. در ۷۵ درصد موارد، قطعه کد و یا مباحثه مرتبط وجود داشته است و برنامه‌نویس با مشاهده هر دوی پیشنهادها می‌تواند به راه‌حل برسد. علاوه بر این معیار، می‌توان معیار MRR (Mean Reciprocal Rank) را نیز محاسبه نمود. این معیار میانگین معکوس رتبه اولین جواب است و هر چه بیش‌تر باشد، به معنی آن است که جواب در رتبه کم‌تری قراردارد و در نتیجه، نشان‌دهنده بهتر بودن ابزار است. MRR برای مباحثه‌های Exception Tracer برابر ۰.۲۶۷ است. این معیار برای قطعه کدها ۰.۴۳۲ است.

جدول ۲- نتایج ارزیابی کارایی و دقت روش

ردیف	نام کتابخانه	تعداد نتایج Stack Overflow	مباحثه مرتبط	تعداد قطعه کدهای نتیجه شده	قطعه کد مرتبط
۱	Email	۱۱	-	۱	۱
۲	Tar	۰	-	۱	-
۳	Velocity	۴	-	۵	۱
۴	DB	۳۰	۳	۶	۴
۵	HTTP	۳۰	۹	۶	۱
۶	CLI	۰	-	۵	-
۷	EXEC	۱۰	۱	۲	۱
۸	IO	۱	-	۰	-
۹	file	۳۰	۲	۲	۱
۱۰	net	۱۷	۳	۰	-
۱۱	Regular Expression	۳۰	۲	۴	۲
۱۲	Buffered Input Stream	۵	۲	۳	۳
۱۳	EMAIL-SSL	۱۲	-	۰	-
۱۴	PDF	۰	-	۴	۲
۱۵	POI	۳۰	۱	۰	-
۱۶	Velocity2	۴	-	۵	۳

جدول ۳- اطلاعات فردی افراد مورد مطالعه قرار گرفته شده

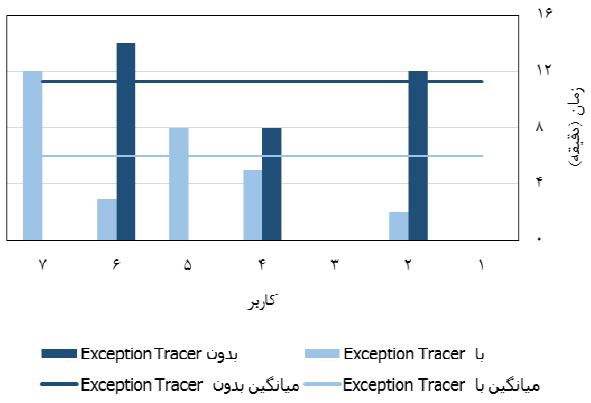
کاربر	مقطع	سابقه برنامه‌نویسی (سال)	میزان آشنایی با زبان جاوا	تجربه برنامه‌نویسی با زبان جاوا (سال)	میزان آشنایی با مفاهیم کتابخانه‌ها و چارچوب‌ها	میزان آشنایی با کتابخانه Velocity	میزان آشنایی با کتابخانه RegEx
۱	کارشناسی	۲	متوسط	۲	کم	نا آشنا	کم
۲	کارشناسی	۵	کم	۱	متوسط	کم	متوسط
۳	کارشناسی	۴	زیاد	۲	متوسط	نا آشنا	زیاد
۴	کارشناسی ارشد	۵	متوسط	۴	متوسط	نا آشنا	کم
۵	کارشناسی	۴	زیاد	۴	متوسط	کم	متوسط
۶	کارشناسی	۳	کم	۰.۵	کم	نا آشنا	متوسط
۷	کارشناسی	۲	کم	۱	کم	نا آشنا	نا آشنا
۸	کارشناسی	۳	متوسط	۳	متوسط	متوسط	متوسط
۹	کارشناسی	۵	زیاد	۴	متوسط	کم	متوسط
۱۰	غیردانشجو	۱	کم	۰.۵	کم	نا آشنا	کم
۱۱	کارشناسی ارشد	۵	زیاد	۴	متوسط	کم	متوسط
۱۲	کارشناسی	۳	کم	۲	کم	نا آشنا	کم
۱۳	کارشناسی	۴	متوسط	۳	متوسط	کم	کم
۱۴	غیردانشجو	۱	کم	۰.۵	کم	نا آشنا	نا آشنا

۵-۲- مطالعه کاربران

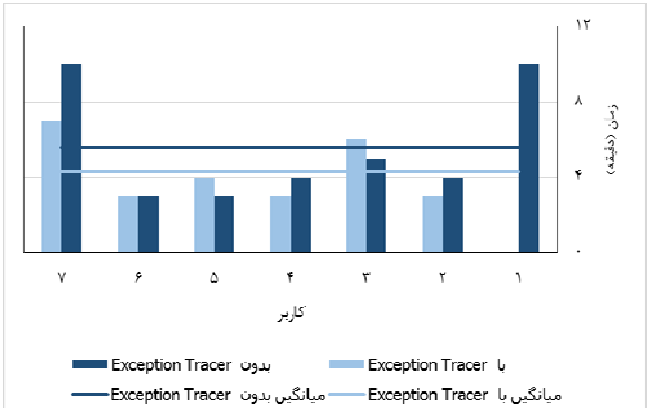
در بخش ۵-۱ دقت ابزار مورد بررسی قرار گرفت. در این بخش،از طریق مطالعه کاربران، میزان مفید بودن این ابزار در عمل مورد ارزیابی قرار گرفت. در این بخش به پرسش "آیا این ابزار نسبت به روش استفاده از مرورگر، برای برنامه‌نویس صرفه‌جویی زمانی دارد؟" پاسخ می‌دهیم.

۵-۲-۱- برپایی ارزیابی

از بین نمونه کدهایی که در زمان اجرا با استثناء متوقف شدند، دو نمونه کد انتخاب نمودیم و به ۱۴ نفر این نمونه کدها را دادیم. این افراد راه‌حل یکی از نمونه کدها را با استفاده از Exception Tracer یافتند و راه‌حل قطعه کد دیگر را با استفاده از مرورگر. در ادامه نحوه انتخاب افراد و نمونه کدها بیان می‌شود.



شکل ۱۰- نمودار مقایسه زمان‌های یافتن راه حل کد کتابخانه Velocity



شکل ۱۱- نمودار مقایسه زمان های یافتن راه حل کد کتابخانه RegEx

۵-۲-۲- انتخاب افراد

افرادی موردمطالعه قرار گرفتند که آشنایی نسبتا خوبی با زبان جاوا داشتند و در فهم و برنامه‌نویسی به این زبان مشکلی نداشتند. همچنین این افراد می‌بایست از دو کتابخانه نمونه کدها به‌صورت حرفه‌ای استفاده نکرده باشند و برای اصلاح کد نیاز به جست‌وجو داشته باشند. به‌طور میانگین، افراد انتخاب شده ۳.۵ سال تجربه برنامه‌نویسی و ۲ سال تجربه برنامه‌نویسی به زبان جاوا داشتند. همه این افراد یک دوره آموزش جاوا در دانشگاه و یا آموزشگاه گذرانده‌اند. ۸۵ درصد افراد انتخاب شده دانشجو بودند. در جدول ۳، اطلاعات افرادی که در این ارزیابی شرکت نمودند، نمایش داده شده‌اند.

۵-۲-۳- انتخاب نمونه کدها

نمونه کدها باید به‌گونه‌ای انتخاب می‌شدند که هم به‌راحتی با Exception Tracer بتوان به راه‌حل رسید و هم با استفاده از مرورگر. با این تفاسیر، نمونه کد مربوط به دو کتابخانه Velocity و RegEx انتخاب شدند. راه‌حل نمونه کد مربوط به RegEx در وب‌سایت Stack Overflow موجود است. اما کتابخانه دیگر به‌گونه‌ای انتخاب شده است که راه‌حل آن در Stack Overflow نیست و برنامه‌نویس مجبور است سایت‌های دیگر را جست‌وجو نماید. علاوه بر این مورد نمونه کدها باید کوتاه باشند. حجیم بودن نمونه کدها باعث افزایش زمان ارزیابی و همچنین دشوار شدن ارزیابی برای افراد را در پی دارد. بیش از ۹۰ درصد افراد آشنایی خود با کتابخانه Velocityرا کم و یا "تا حالا استفاده نکرده‌ام" بیان کردند. این عدد برای کتابخانه RegEx ۵۰ درصد بود.

۵-۲-۴- اجرای ارزیابی

برای مطالعه کاربران دو پرسش‌نامه طراحی شد. یک پرسش‌نامه از فرد مورد مطالعه اطلاعات اولیه را می‌خواهد و پرسش‌نامه دوم، بعد از انجام آزمایش به او داده می‌شود و جزئیات تجربه او در استفاده از ابزار را از او می‌پرسد.

۵-۲-۵- تحلیل نتایج

در این بخش تحلیل نتایج به‌دست آمده در دو دسته کمی و کیفی بیان می‌شود.

تحلیل کمی

هر یک از نمونه کدهای دو کتابخانه Velocity و RegEx توسط ۱۴ نفر آزمایش شدند. هر نمونه کد توسط ۷ نفر با استفاده از Exception Tracer و توسط ۷ نفر دیگر و با استفاده از مرورگر آزمایش شدند. در مورد کتابخانه RegEx استفاده از Exception Tracer در یافتن راه‌حل، ۲۲ درصد صرفه‌جویی در زمان داشت. این عدد برای کتابخانه Velocity، ۵۴ درصد بود. علت این امر نیز نبودن مباحثه مربوط به کتابخانه Velocity در وب‌سایت Stack Overflow است. نبودن مباحثه مرتبط در این وب‌سایت باعث شد تا از ۷ نفری که باید راه‌حل استثناء این کتابخانه را با مرورگر پیدا می‌کردند، فقط ۳ نفر راه‌حل را بیابند. این افراد با جست‌وجو در Google نمونه کدهای مرتبط را پیدا کردند و به وسیله آن کد را اصلاح نمودند. در مورد کتابخانه RegEx همه افرادی که باید با مرورگر جست‌وجو می‌کردند، روش جست‌وجو در Google و مشاهده مباحثه‌های Stack Overflow را در پیش گرفتند. در کل و به‌طور میانگین این ابزار در یافتن راه‌حل ۳۸ درصد در زمان صرفه‌جویی می‌کند. نکته قابل توجه در این ارزیابی این است که ۳۵ درصد افراد از Eclipse استفاده نمی‌کنند، در حالی‌که انتظار می‌رفت این عدد کم‌تر باشد. نمودارهای ترسیم شده در شکل ۱۰ و شکل ۱۱، به ترتیب زمان یافتن پاسخ کد مشکل‌دار در کتابخانه‌های Velocity و RegEx را در هنگامی که از Exception Tracer استفاده شده است را با هنگامی که از آن استفاده نشده است، مقایسه می‌کند.

جدول ۴- آماره‌های آزمون

ردیف	$t_{v,i,br-et} = t_{v,i,br} - \overline{T}_{v,et}$	$t_{r,i,br-et} = t_{r,i,br} - \overline{T}_{r,et}$
۱	-	5.66
۲	6.92	-0.33
۳	-	0.66
۴	2.92	-0.33
۵	-	-1.33
۶	8.92	-1.33
۷	-	5.66

حال با استفاده از اطلاعات به‌دست آمده و تحلیل آماری درستی فرض "ابزار ارائه شده در این مقاله باعث صرفه‌جویی در زمان برنامه‌نویس می‌شود" را بررسی می‌کنیم.

هر نمونه کد (مربوط به کتابخانه Velocity یا RegEx) توسط ۷ نفر با ابزار Exception Tracer و توسط ۷ نفر بدون Exception Tracer اشکال‌زدایی شده‌اند. $t_{v,i,et}$ زمانی که برنامه‌نویس iأم برای یافتن راه‌حل نمونه کد Velocity با استفاده از Exception Tracer صرف کرده است را نشان می‌دهد. به همین شکل $t_{r,j,br}$ زمانی که برنامه‌نویس jأم برای یافتن راه‌حل نمونه کد RegEx با استفاده از مرورگر صرف کرده است را نشان می‌دهد. میانگین زمان یافتن راه‌حل نمونه کد Velocity با استفاده از Exception Tracer به شرح زیر تعریف می‌شود.

$$\overline{T}_{v,et} = \frac{\sum_{i=1}^{7} t_{v,i,et}}{7}$$

برای یافتن درستی فرض به ازای هر زمان عبارت $t_{v,i,br} - \overline{T}_{v,et}$ را محاسبه می‌کنیم. مقادیر این متغیر تصادفی نمی‌تواند از آزمایش انجام گرفته بر روی یک نفر حاصل شود. زیرا فردی که با استفاده از ابزار ما راه حل را می‌یابد، دیگر جواب را می‌داند و همین مورد باعث می‌شود با استفاده از مرورگر سریع‌تر به جواب برسد. این اشکال در حالت برعکس نیز وجود دارد. به همین دلیل متغیر تصادفی حاصل زمان یافتن راه‌حل یک نفر با استفاده از ابزار منهای میانگین زمان افرادی که از مرورگر استفاده کرده‌اند، تعریف شده است. مقادیر متغیر تصادفی جدید و آماره آزمون در جدول ۴ آمده است.

از آنجایی‌که انتخاب افراد کاملا تصادفی بود، با توزیع t-student فرض را بررسی نمودیم. در مورد کتابخانه Velocity از آنجایی‌که سه عدد بی‌نهایت داریم (به جواب نرسیده‌اند) این تحلیل را انجام نمی‌دهیم، زیرا در صورت محاسبه، به احتمال ۱۰۰ درصد فرضیه صحیح خواهد بود. فقط برای کتابخانه RegEx محاسبه می‌کنیم. میانگین و واریانس به ترتیب μ = ۱.۲۴ و S_n = ۳.۱ هستند.

$$H_1: \mu > 0$$

$$t = \frac{\overline{T}_{r,et} - 0}{\frac{S_n}{\sqrt{n}}} = 1.06$$

$$t \sim t_{1-0.16,6}$$

با توجه به توزیع t-student با درجه آزادی ۶ و α = ۰.۱۶، با احتمال ۸۴ درصد فرضیه "آیا این ابزار نسبت به روش استفاده از مرورگر، برای برنامه‌نویس صرفه‌جویی زمانی دارد؟" درست است.

تحلیل کیفی

سؤال پایانی در پرسش‌نامه از افراد نظرشان در مورد ابزار را می‌پرسد. با توجه به این پاسخ‌ها مشکلاتی در ابزار کشف شدند که به عنوان کارهای آتی می‌توان به آن‌ها توجه کرد. برنامه‌نویس دوم، هشتم و دوازدهم ایراد ابزار را واسط کاربری نامناسب مباحثه‌های Stack Overflow (فونت ریز، عملکرد نامناسب اسکرول) بیان کردند. برنامه‌نویس چهارم در مورد نمونه کدها، سرعت پایین آماده شدن قطعه کدها را مطرح کرده است. دلیل این مورد هم پاسخ‌گویی کند BOA است. در صورت بهبود سخت‌افزاری BOA این مشکل رفع خواهد شد. مورد دیگر که برنامه‌نویس پنجم به آن اشاره کرده، تنوع نداشتن قطعه کدهای پیشنهادی است. این مشکل زمانی پیش می‌آمد که از یک پروژه نرم‌افزاری چند قطعه کد یافته می‌شود. ازآنجایی‌که این مثال‌ها در یک پروژه قرار دارند، سبک تمام این قطعه کدها یکسان است و برای برنامه‌نویس اطلاعات جدیدی ندارد.

مباحثه‌های پیشنهادی Stack Overflow در Exception Tracer توسط افراد، خیلی مورد استفاده قرار نگرفت؛ علت این موضوع را می‌توان در این موارد دید: (۱) افراد در مباحثه‌ها هم به دنبال نمونه کد می‌گردند، وقتی ابزار مستقیما نمونه کد در اختیارشان قرار می‌دهد، اول به دنبال آن می‌روند. (۲) جست‌وجوی مباحثه‌ها خیلی دقیق نیست و با خواندن عناوین علاقه به کلیک کردن و خواندن متن مباحثه پیدا نمی‌کنند.

یکی از دو قطعه کد طوری انتخاب شده بود که راه‌حل آن در Stack Overflow نباشد. برای این قطعه کد، افراد کم‌تری بدون استفاده از ابزار به راه حل رسیدند. این مورد نشان می‌دهد که این ابزار علاوه بر پوشش استثناءهایی که جوابشان در Stack Overflow هست، برای سایر استثناءها هم می‌تواند مفید واقع شود.

ارزیابی ابزارهای مشابه با یک روش انجام نشده است، به همین دلیل ارزیابی Exception Tracer با معیارهای گوناگون انجام شده است. ابزار Help Me Out با برگزاری چند کارسوق به ارزیابی ابزار خود پرداخته است. سازندگان این ابزار، دقت ابزار خود را ۵۰ درصد بیان کرده‌اند، درحالی‌که Exception Tracer از دقت ۷۵ درصد برخوردار است. ابزار Surf Clips معیار MRR را محاسبه کرده است که به معنی میانگین معکوس رتبه اولین جواب است. این معیار برای Surf Clips مقدار ۰.۴۶ را داراست که در مقایسه با قطعه کدهای Exception Tracer، ۰.۰۴ کم‌تر است و ازآنجایی‌که هر چه MRR بیشتر باشد، بهتر است، ابزار Exception Tracer عملکرد بهتری دارد. البته این معیار برای مباحثه‌های Exception Tracer برابر ۰.۲۶۷ است که از Surf Clips عملکرد ضعیف‌تری دارد. علت این ضعف هم آن است که Exception Tracer تمرکز بیشتری روی نمونه کد داشته است و مباحثه‌هایش را با یک جست‌وجوی ساده از سایت Stack Overflow مهیا می‌کند؛ درحالی‌که ابزارهای مشابه Surf Clips به‌طور تخصصی بر روی جست‌وجوی مباحثه‌های Stack Overflow کار کرده‌اند.

۶- کارهای آتی

در بخش‌های مختلف ابزار کارهای فراوانی می‌تواند برای بهبود آن انجام شود. در زیر لیست کارهای آتی را بیان می‌کنیم:

۱- واسط کاربری مباحثه‌ها: با توجه به نظرات کیفی برنامه‌نویسان در ارزیابی انجام شده، واسط کاربری بخش مربوط به مباحثه‌ها نیاز به بهبود دارد. این بهبود به‌اندازه فونت، اسکرول و نمایش جواب تایید شده (جوابی که توسط سازنده مباحثه تایید شده است، مشخص باشد) مربوط می‌شود.

۲- تنوع در نمونه کدها: برخی اوقات پیش می‌آید که چند قطعه کد از یک پروژه نرم‌افزاری استخراج می‌شود. این قطعه کدها معمولا یک دنباله فراخوانی یکسانی را طی می‌کنند و عملا برای برنامه‌نویس تفاوت ملموسی ندارد. زیرا نام متغیرها نیز به فضای نام متغیرهای برنامه‌نویس منتقل شده است. از این‌رو

باید از هر پروژه فقط یک قطعه کد در لیست نتایج نهایی وجود داشته باشد.

۳- کوتاه کردن قطعه کدها: برخی متدها مانند put ممکن است وجود داشته باشند که در قطعه کد ۵ الی ۱۰ بار پشت سر هم تکرار شده باشند. این تکرر باعث افت کیفیت قطعه کد می‌شود.

۴- بهبود روش یافتن متدی که باعث استثناء شده است: برای یافتن این متد از پشته فراخوانی استفاده می‌کنیم. در هنگام استفاده از فریموورک‌های وب و پیچیده، اطلاعات پشته فراخوانی که هنگام رخداد استثناء نمایش داده می‌شود، خیلی پیچیده است و به همان سادگی که در بخش ۲ مطرح شد، نمی‌توان متد موردنظر را یافت. بهبود این بخش از ابزار، به فراگیرتر شدن آن کمک خواهد کرد.

۵- بهبود پیشنهاد مباحثه: روش پیشنهادی در این مقاله از یک روش ساده برای پیشنهاد مباحثه‌های مرتبط از Stack Overflow استفاده نموده است. صرف پیشنهاد تواما مباحثه و قطعه کد باعث افزایش دقت کار شده است. اما در آینده می توان پیشنهاد مباحثه را بهبود بخشید.

۷- نتیجه‌گیری

در استفاده از کتابخانه‌ها و چارچوب‌های برنامه‌نویسی، برنامه‌نویسان قوانینی را بایستی رعایت نمایند. عدم‌رعایت این قوانین ممکن است منجر به استثناء در هنگام اجرای برنامه شود. برنامه‌نویسان اغلب برای یافتن راه‌حل استثناءها، به جستجو در سایت‌های پرسش و پاسخ برنامه‌نویسی و یا کدهای متن‌باز متوسل می‌شوند. ولیکن، این روش‌ها دارای مشکلاتی مانند عدم‌دقت در ساخت پرس‌وجو و یا زمان‌بر بودن می‌باشند. برای کمک به حل این مشکلات، در این مقاله، ابزار Exception Tracer که به‌صورتیک افزونه Eclipse پیاده‌سازی شده است، ارائه گردید. این ابزار، با استفاده از اطلاعات پشته فراخوانی و زمینه کد برنامه‌نویس، قطعه کدهایی از سایت Source Forge را که می‌توانند به حل استثناء کمک کنند، به برنامه‌نویس پیشنهاد می‌دهد. همچنین، برای کامل‌تر شدن روش پیشنهادی و کمک به برنامه‌نویسان به درک بهتر، قطعه کدهای پیشنهادی، مباحثه‌های مرتبط را نیز از سایت پرسش و پاسخ Stack Overflow نمایش می‌دهد. دقت این ابزار، با استفاده از تعدادی از نمونه کدهای مربوط به کتابخانه‌های Apache اندازه‌گیری گردید و ملاحظه شد که در ۷۵ درصد مواقع، راه‌حل استثناء در ۵ نتیجه ابتدایی وجود دارد. علاوه بر محاسبه دقت، میزان صرفه‌جویی زمانی نسبت به جستجوی ساده در اینترنت نیز بامطالعه کاربران اندازه‌گیری شد. طبق این ارزیابی، میزان صرفه‌جویی زمانی در صورت استفاده از Exception Tracer در حدود ۳۸ درصد است.

مراجع

[1] J. Cordeiro, B. Antunes, and P. Gomes, "Context-based recommendation to support problem solving in software development," *Proc. 3rd Intl Workshop on Recommendation Systems for Soft. Eng.*, pp. 85-89, 2012.

[2] Z. Gu, E. T. Barr, D. Schleck, and Z. Su, "Reusing debugging knowledge via trace-based bug search," *Proc. 27th IntlConf. on Object Oriented Programming*, pp. 927-942, 2012.

[3] B. Hartmann, D. MacDougall, J. Brandt, and S. R. Klemmer, "What would other programmers do: suggesting solutions to error messages," *Proc. 28th Conf. on Human Factors in Computing Systems*, pp. 1019-1028, 2010.

[4] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto , and M. Lanza, "Mining StackOverflow to turn the IDE into a self-confident programming prompter," *Proc. 11th Working Conf. on Mining Software Repositories*, pp. 102-111, 2014.

[5] M. M. Rahman, S. Yeasmin, and C. K. Roy, "Towardsa context-aware IDE-based meta search engine for recommendation about programming errors and exceptions," *Proc. 21th Software Evolution Week-IEEE Conf. on Reengineering and Reverse Engineering*, pp. 194-203, 2014.

[6] P. Vekris, R. Jhala, S. Lerner, and Y. Agarwal, "Towards verifying android apps for the absence of no-sleep energy bugs," *Proc. USENIX Conf. on Power-Aware Computing and Systems,* p. 3, 2012.

[7] T. T. Nguyen, H. A. Nguyen, N. H. Pham, J. M. Al-Kofahi, and T. N. Nguyen, "Graph-based mining of multiple object usage patterns," *Proc. 7th joint meeting of the European Soft.Eng.Conf. and the ACM SIGSOFT Symp. on the Foundations of Soft.Eng.*, pp. 383-392, 2009.

[8] R. Holmes, and G. C. Murphy, "Using structural contextto recommend source code examples," *Proc. 27th IntlConf. on Soft. Eng.*, pp. 117-125, 2005.

[9] R. Holmes, R. J. Walker, and G. C. Murphy, "Strathcona example recommendation tool," *ACM SIGSOFT Soft. Eng. Notes,* vol. 30, no. 5, pp. 237-240, 2005.

[10] C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu, "Portfolio: finding relevant functions and their usage," *Proc. 33th IntlConf. on Soft. Eng.,* pp. 111-120, 2011.

[11] S. Thummalapenta, and T. Xie, "SpotWeb: Detecting framework hotspots and coldspots via mining open source code on the web," *Proc. 23th IntlConf. on Automated Soft. Eng.*, pp. 327-336, 2008.

[12] H. Cai, and R. Santelices, "Diver: precise dynamic impact analysis using dependence-based trace pruning," *Proc. 29th IntlConf. on Automated Soft. Eng.,* pp. 342-348, 2014.

[13] G. Bavota, S. Panichella, N. Tsantalis, M. D. Penta, R. Oliveto, and G. Canfora, "Recommending refactorings based on team co-maintenance patterns," *Proc. 29th IntlConf. on Automated Soft. Eng.,* pp. 337-342, 2014.

[14] M. Ghafari, C. Ghezzi, A. Mocci, and G. Tamburrelli, "Mining unit tests for code recommendation," *Proc. 22th IntlConf. on Program Comprehension*, pp. 142-145, 2014.

[15] L. Ponzanelli, A. Bacchelli, and M. Lanza, "Seahawk: stack overflow in the ide," *Proc. 35th IntlConf. on Soft. Eng.*, pp. 1295-1298, 2013.

[16] J. Yang, D. Evans, D. Bhardwaj, T. Bhat, and M. Das, "Perracotta: mining temporal API rules from imperfect traces," *Proc. 28th IntlConf. on Soft. Eng.*, pp. 282-291,

2006.

[17] M. Gabel, and Z. Su, "Javert: fully automatic mining of general temporal properties from dynamic traces," *Proc. 16th Intl Symposiumon Foundations of Soft. Eng.*, pp. 339-349, 2008.

[18] S. Sankaranarayanan, F. Ivancic, and A. Gupta, "Mining library specifications using inductive logic programming," *Proc. 30th IntlConf. on Soft. Eng.*, pp. 131-140, 2008.

[19] M. Pradel, C. Jaspan, J. Aldrich, and T. R. Gross, "Statically checking API protocol conformance with mined multi-object specifications," *Proc. 34th IntlConf. on Soft. Eng.*, pp. 925-935, 2012.

[20] D. Mandelin, L. Xu, R. Bodík, and D. Kimelman, "Jungloid mining: helping to navigate the API jungle," *ACM SIGPLAN Notices,* vol. 40, no. 6, pp. 48-61, 2005.

[21] N. Sahavechaphan, and K. Claypool, "XSnippet: mining for sample code," *ACM SIGPLAN Notices,* vol. 41, no. 10, pp. 413-430, 2006.

[22] C. Lv, W. Jiang, Y. Liu, and S. Hu, "APISynth: a new graph-based API recommender system," *Proc. 36th IntlConf. on Soft. Eng.*, pp. 596-597, 2014.

[23] S. Thummalapenta, and T. Xie, "Parseweb: a programmer assistant for reusing open source code on the web," *Proc. 22th IntlConf. on Automated Soft. Eng.,* pp. 204-213, 2007.

[24] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen, "BOA: A language and infrastructure for analyzing ultra-large-scale software repositories," *Proc. 35th IntlConf. on Soft. Eng.*, pp. 422-431, 2013.

[25] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan, "Interpreting the data: Parallel analysis with Sawzall," *Scientific Programming,* vol. 13, no. 4, pp. 277-298, 2005.

[26] A. T. Nguyen, H. A. Nguyen, T. T. Nguyen, and T. N. Nguyen, "GraPacc: a graph-based pattern-oriented, context-sensitive code completion tool," *Proc. 34th IntlConf. on Soft. Eng.*, pp. 1407-1410, 2012.

وحید امین‌تبار مدارک کارشناسی و کارشناسی‌ارشد مهندسی نرم‌افزار خود را به ترتیب در سال‌های ۱۳۹۲ و ۱۳۹۴ از دانشگاه صنعتی شریف دریافت نموده است. زمینه‌های پژوهشی مورد علاقه ایشان عبارتند از مهندسی نرم‌افزار، داده‌کاوی در مخازن نرم‌افزاری، و سیستم‌های توصیه‌گر در مهندسی نرم‌افزار.

آدرس پست‌الکترونیکی ایشان عبارت است از:

amintabar@ce.sharif.edu

عباس حیدرنوری مدرک دکترای خود در علوم کامپیوتر را در سال ۱۳۸۸ از دانشگاه واترلو کانادا و مدارک کارشناسی و کارشناسی‌ارشد در مهندسی نرم‌افزار خود را به ترتیب در سال‌های ۱۳۷۸ و ۱۳۸۰ از دانشگاه صنعتی شریف دریافت نموده است. بعد از اتمام دوره دکترا، ایشان به مدت یک سال به عنوان پژوهشگر پسادکترا در دانشگاه لوگانو سوییس مشغول به انجام پژوهش بوده است. ایشان سپس به مدت یک سال به عنوان مهندس ارشد نرم‌افزار در زمینه تولید نرم‌افزارهای هوشمند همراه در شرکت Extreme Labs Inc. در تورنتو کانادا مشغول به کار بوده است. دکتر حیدرنوری، از سال ۱۳۹۱ به عنوان عضو هیأت علمی در دانشکده مهندسی کامپیوتر دانشگاه صنعتی شریف مشغول به فعالیت می‌باشد. زمینه‌های پژوهشی ایشان عبارتند از مهندسی نرم‌افزار، مهندسی معکوس و بازمهندسی سیستم‌های نرم‌افزاری، و سیستم‌های توصیه‌گر در مهندسی نرم‌افزار.

آدرس پست‌الکترونیکی ایشان عبارت است از:

heydarnoori@sharif.edu

اطلاعات بررسی مقاله:

تاریخ ارسال: ۱۳۹۵/۰۲/۰۸

تاریخ اصلاح: ۱۳۹۵/۰۴/۲۷

تاریخ قبول شدن: ۱۳۹۵/۰۵/۲۰

نویسنده مرتبط: دکتر عباس حیدرنوری، دانشکده مهندسی کامپیوتر، دانشگاه صنعتی شریف، تهران، ایران.

[1] Call Stack
[2] Application Frame Works
[3] Application Programming Interface (API)
[4] Abstract Syntax Tree (AST)
[5] Refactoring
[6] Name space