



دانشگاه صنعتی شریف

دانشکده‌ی مهندسی کامپیوتر

پایان نامه به عنوان تحقق بخشی از شرایط دریافت درجه‌ی کارشناسی ارشد
گرایش مهندسی نرم افزار

مدل پیش بینی خطا مبتنی بر معیارهای جهش

نگارش

علی محبی

استاد راهنما

دکتر حسن میریان

شهریور ۱۳۹۷

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

[این صفحه آگاهانه خالی گذاشته شده است.]

تصویب نامه

به نام خدا
دانشگاه صنعتی شریف
دانشکده‌ی مهندسی کامپیوتر

پایان نامه‌ی کارشناسی ارشد

عنوان: مدل پیش بینی خطا مبتنی بر معیارهای جهش
نگارش: علی محبی

کمیته‌ی ممتحنین:

امضاء.....	دکتر حسن میریان	استاد راهنما:
امضاء.....	دکتر <نام استاد مدعو ۱>	استاد مدعو:
امضاء.....	دکتر <نام استاد مدعو ۲>	استاد مدعو:
تاریخ:		



اظهاری نامه (اصالت متن و محتوای رساله‌ی دکتری)

عنوان رساله: _____

نام استاد راهنما: _____ نام استاد راهنمای هم‌کار: _____ نام استاد مشاور: _____
این جانب _____ اظهار می‌دارم:

۱. متن و نتایج علمی ارایه‌شده در این رساله اصیل بوده و منحصرأً توسط این جانب و زیر نظر استادان (راهنما، هم‌کار و مشاور) نام‌برده‌شده در بالا تهیه شده است.

۲. متن رساله به این صورت در هیچ جای دیگری منتشر نشده است.

۳. متن و نتایج مندرج در این رساله، حاصل تحقیقات این جانب به عنوان دانشجوی دکتری دانشگاه صنعتی شریف است.

۴. کلیه‌ی مطالبی که از منابع دیگر در این رساله مورد استفاده قرار گرفته، با ذکر مرجع مشخص شده است.

نام دانشجو: _____

تاریخ:

امضاء:

نتایج تحقیقات مندرج در این رساله و دستاوردهای مادی و معنوی ناشی از آن (شامل فرمول‌ها، نرم‌افزارها، سخت‌افزارها و مواردی که قابلیت ثبت اختراع دارد) متعلق به دانشگاه صنعتی شریف است. هیچ شخصیت حقیقی یا حقوقی بدون کسب اجازه از دانشگاه صنعتی شریف حق فروش و ادعای مالکیت مادی یا معنوی بر آن یا ثبت اختراع از آن را ندارد. همچنین کلیه‌ی حقوق مربوط به چاپ، تکثیر، نسخه‌برداری، ترجمه، اقتباس و نظایر آن در محیط‌های مختلف اعم از الکترونیکی، مجازی یا فیزیکی برای دانشگاه صنعتی شریف محفوظ است. نقل مطالب با ذکر ماخذ بلامانع است.

نام استادان راهنما: _____ نام دانشجو: _____

تاریخ:

امضاء:

تاریخ:

امضاء:

تقديم به ...؛ صفحه‌ی تقديم اختياري است.

[این صفحه آگاهانه خالی گذاشته شده است.]

قدردانی

صفحه‌ی قدردانی. این صفحه اختیاری بوده و می‌توانید آن را حذف کنید. برای این کار کافی است محیط قدردانی در پرونده‌ی تِک را حذف کنید. متداول است که در این صفحه از خانواده، استادها و همکارهای خود قدردانی نمایید.

[این صفحه آگاهانه خالی گذاشته شده است.]

مدل پیش بینی خطا مبتنی بر معیارهای جهش

چکیده

چکیده‌ی پایان‌نامه به زبان پارسی را پس از نگارش کامل پایان‌نامه آماده کنید. چکیده از ۳۰۰ واژه (یا کمتر) تشکیل شده و در ادامه‌ی آن ۴ تا ۷ واژه‌ی کلیدی بیان می‌شود. واژه‌های کلیدی در پرونده‌ی اصلی (به زبان پارسی و انگلیسی) نوشته می‌شوند و چکیده بسته به زبان در دو پرونده‌ی جداگانه در پوشه‌ی عمومی نوشته می‌شود.

کلیدواژه‌ها: واژه‌ی کلیدی نخست، واژه‌ی کلیدی دوم، واژه‌ی کلیدی پایانی.

[این صفحه آگاهانه خالی گذاشته شده است.]

سرخ‌ها

۱	۱	سرآغاز
۳	۲	مرور مطالعات پیشین
۳	۱.۲	پیش بینی خطا
۳	۱.۱.۲	فرآیند پیش‌بینی خطا
۴	۲.۱.۲	معیارهای ارزیابی
۷	۳.۱.۲	معیارهای پیش‌بینی خطا
۱۰	۴.۱.۲	مدل‌های پیش‌بینی خطا
۱۱	۲.۲	آزمون جهش و کاربردهای آن
۱۳	۱.۲.۲	مکان‌یابی خطا
۱۵	۲.۲.۲	مدل‌های یادگیری و جهش‌یافته‌ها
۱۶	۳.۲	جمع‌بندی مطالعات پیشین
۱۹	۳	معیارهای جهش و فرآیند
۱۹	۱.۳	معیارهای جهش و فرآیند
۲۲	۲.۳	معیارهای جهش مبتنی بر فرآیند
۲۳	۳.۳	معیارهای ترکیبی جهش-فرآیند
۲۷	۴	مورد مطالعاتی
۲۷	۱.۴	طراحی آزمایش
۲۷	۲.۴	آشنایی با ابزارها و مجموعه داده
۲۷	۱.۲.۴	مجموعه داده defect4j
۳۰	۲.۲.۴	ابزار Major
۳۴	۳.۲.۴	کتابخانه‌ی Jgit

۳۴ Hibernate چهارچوب	۴.۲.۴
۳۴ نکات پیاده‌سازی پروژه	۳.۴
۳۶ رویکرد اول : معیارهای فرآیند در کنار جهش	۴.۴
۳۶ استخراج اطلاعات مربوط به ثبت‌های حاوی خطا	۱.۴.۴
۳۸ استخراج معیارهای فرآیند	۲.۴.۴
۴۳ استخراج معیارهای جهش	۳.۴.۴
۴۶ رویکرد دوم: معیارهای فرآیند مبتنی بر جهش	۵.۴
۴۷ ارزیابی	۵
۴۹ نتیجه‌گیری و کارهای آتی	۶
۵۱ کتاب‌نامه	
۵۴ واژه نامه انگلیسی به فارسی	
۵۷ واژه نامه فارسی به انگلیسی	

فهرست جدول‌ها

۱۰۲	فرمول‌های محاسبه‌ی معیارهای ارزیابی	۵
۲۰۲	جدول مشخصات پژوهش‌های مرور شده در حوزه‌ی پیش‌بینی خطا	۱۷
۱۰۳	معیارهای فرآیند [۱۸]	۲۰
۲۰۳	معیارهای جهش [۲۷]	۲۰
۱۰۴	عملیات‌های موجود در defects4j	۲۸
۲۰۴	پروژه‌های موجود در defects4j	۲۹

[این صفحه آگاهانه خالی گذاشته شده است.]

فهرست شکل‌ها

۴	فرآیند پیش‌بینی خطا [۱]	۱.۲
۶	نمونه‌ای از نمودار ROC [۲]	۲.۲
۶	نمودار موثر بودن از نظر هزینه [۴]	۳.۲
۱۳	نمونه‌ای از جهش‌یافته‌های یک برنامه [۲۴]	۴.۲
۳۰	اجرای دستور info در defects4j	۱.۴
۳۱	نمونه کد MML در Major	۲.۴
۳۲	اجرای عملیات جهش برای یک پرونده	۳.۴
۳۳	نمونه‌ای از پرونده‌ی mutants.log	۴.۴
۳۳	اجرای تحلیل جهش	۵.۴
۳۴	نتایج خروجی تحلیل جهش	۶.۴
۳۶	نمایی از مخزن نرم‌افزاری	۷.۴
۳۷	نمایی از جدول محتوای انتشارها	۸.۴
۳۸	نمایی از جدول محتوای اطلاعات پرونده‌های حاوی خطا	۹.۴
۳۹	نمایی از جدول اطلاعات ثبت‌ها	۱۰.۴
۳۹	نمایی از جدول تغییرات پرونده‌ها در ثبت‌ها	۱۱.۴
۴۱	نمایی از مخزن نرم‌افزاری	۱۲.۴
۴۳	نمایی از جدول معیارهای فرآیند	۱۳.۴
۴۵	پرونده‌ی mml ساخته شده جهت تولید جهش‌یافته‌ها	۱۴.۴
۴۶	نمایی از جدول نتایج تحلیل جهش	۱۵.۴

[این صفحه آگاهانه خالی گذاشته شده است.]

فصل ۱

سرآغاز

[این صفحه آگاهانه خالی گذاشته شده است.]

فصل ۲

مرور مطالعات پیشین

۱.۲ پیش بینی خطا

۱.۱.۲ فرآیند پیش بینی خطا

اکثریت پژوهش‌های پیش‌بینی خطا از روش‌های یادگیری ماشین استفاده کرده‌اند. اولین گام در ساخت مدل پیش‌بینی تولید داده‌هایی با استفاده از آرشیوهای نرم‌افزاری همانند سیستم‌های کنترل نسخه^۱ مانند گیت^۲، سیستم‌های ردگیری مشکلات مانند جیرا و آرشیو ایمیل‌ها است. هر یک از این داده‌ها بر اساس درشت‌دانی پیش‌بینی می‌توانند نمایانگر یک سیستم، یک قطعه‌ی^۳ نرم‌افزاری، بسته^۴، فایل کد منبع، کلاس و یا تابع باشد. مقصود از داده یک بردار ویژگی حاوی چندین معیار (یا ویژگی) می‌باشد که از آرشیوهای نرم‌افزاری استخراج شده و دارای برچسب سالم و خطا/ار و یا تعداد خطاها است. پس از تولید داده‌ها با استفاده از معیارها و برچسب‌ها می‌توان به پیش پردازش داده‌ها پرداخت (مانند انتخاب معیار) که البته این امر اختیاری می‌باشد. پس از بدست آوردن مجموعه‌ی نهایی داده‌ها یک مدل پیش‌بینی را آموزش می‌دهیم که می‌تواند پیش‌بینی کند یک داده‌ی جدید حاوی خطا است یا خیر. تشخیص خطا/خیز^۵ بودن داده معادل دسته بندی دودویی است و پیش‌بینی تعداد خطاها معادل رگرسیون می‌باشد. در شکل ۱.۲ فرآیند پیش‌بینی خطا نشان داده شده است. داده‌ها نمونه‌هایی هستند که می‌توانند خطا/دار و بدون خطا بودن (B = buggy یا C = clean) و یا تعداد خطا را نشان دهند. لازم به ذکر است که در یک مدل پیش‌بینی تنها از یک نوع از این داده‌ها استفاده می‌شود.

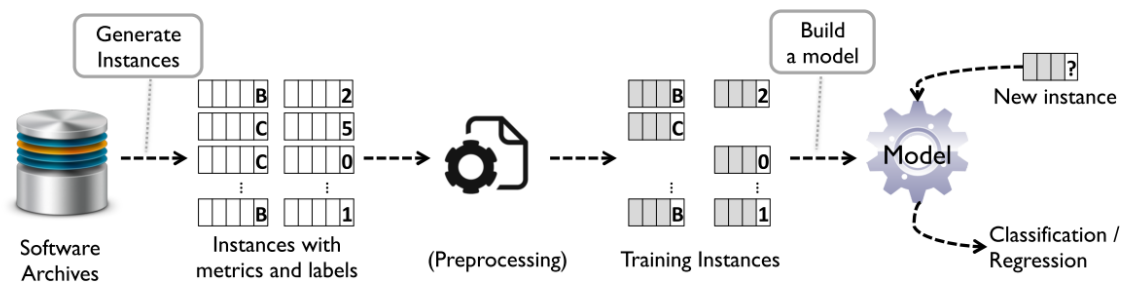
¹Version Control System

²Git

³Component

⁴Package

⁵Bug-proneness



شکل ۱.۲: فرآیند پیش‌بینی خطا [۱]

۲.۱.۲ معیارهای ارزیابی

معیارهای ارزیابی را می‌توان به دسته‌ی کلی معیارهای دسته‌بندی و رگرسیون تقسیم کرد. معیارهای دسته‌بندی را می‌توان با استفاده از ماتریس درهم‌ریختگی^۶ محاسبه نمود. در ماتریس درهم‌ریختگی پیش‌بینی خطا، عناصر به صورت زیر تعریف می‌شوند. همچنین نحوه‌ی محاسبه‌ی معیارها در جدول ۱.۲ آمده است.

• TP : تعداد داده‌های حاوی خطا که به درستی تشخیص داده شدند

• FP: تعداد داده‌های سالم که به عنوان خطا دار پیش‌بینی شدند

• TN: تعداد داده‌های سالم که به درستی تشخیص داده شدند

• FN: تعداد داده‌های حاوی خطا که به عنوان داده‌ی سالم پیش‌بینی شدند

^۶Confusion Matrix

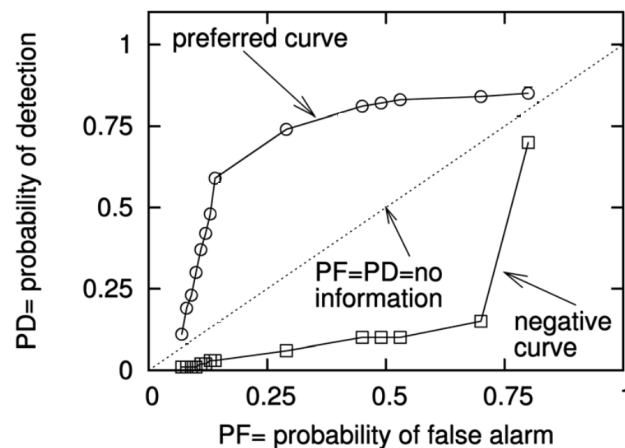
جدول ۱.۲: فرمول‌های محاسبه‌ی معیارهای ارزیابی

نام معیار	نام لاتین	نحوه‌ی محاسبه	توضیح
نرخ مثبت کاذب	False Positive Rate (PF)	$\frac{FP}{TN + FP}$	نسبت تعداد داده‌هایی که به اشتباه خطادار پیش‌بینی شده‌اند به تعداد کل داده‌های بدون خطا
صحت	Accuracy	$\frac{TP + TN}{TP + FP + TN + FN}$	نسبت تعداد پیش‌بینی‌های درست به تعداد کل پیش‌بینی‌ها
دقت	Precision	$\frac{TP}{TP + FP}$	نسبت تعداد داده‌هایی که به درستی خطادار پیش‌بینی شده‌اند به تعداد کل داده‌هایی که خطادار پیش‌بینی شده‌اند
بازخوانی	Recall (PD)	$\frac{TP}{TP + FN}$	نسبت تعداد داده‌هایی که به درستی خطادار پیش‌بینی شده‌اند به تعداد کل داده‌های خطادار
معیار اف	F-Measure	$\frac{2 \times Precision \times Recall}{Precision + Recall}$	از آنجا که در بین معیارهای دقت و بازخوانی مصالحه وجود دارد معیار اف ترکیبی از آن دو را در نظر می‌گیرد

دو معیار دیگر نیز که در پژوهش‌ها کاربرد دارند عبارتند از AUC ^۷ و $AUCEC$ ^۸ که هر دو به مساحت زیر یک منحنی اشاره می‌کنند. AUC مساحت زیر نمودار ROC ^۹ را اندازه‌گیری می‌کند. در نمودار ROC ، محورهای عمودی و افقی را به ترتیب بازخوانی و نرخ مثبت کاذب تشکیل می‌دهد. با تغییر آستانه پیش‌بینی برای یک مدل می‌توان میزان بازخوانی و نرخ مثبت کاذب را تغییر داده و بدین ترتیب منحنی ROC را رسم نمود. یک مدل بی‌نقص دارای مساحت زیر نمودار ۱ است. برای یک مدل تصادفی منحنی از مبدا به نقطه‌ی (۱،۱) رسم خواهد شد. یک نمونه از منحنی ROC در شکل ۲.۲ آمده است.

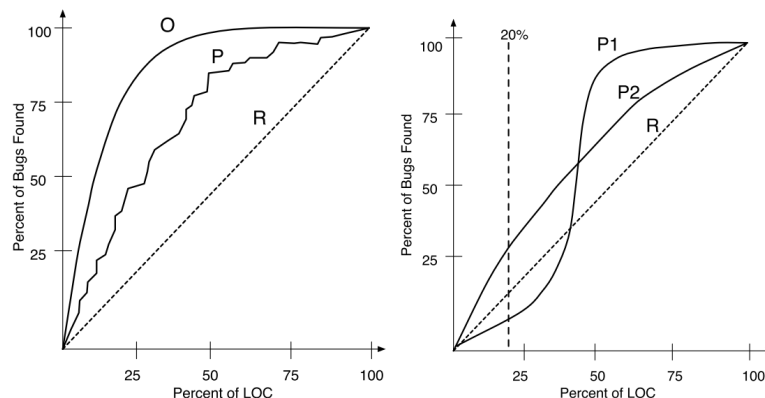
معیار $AUCEC$ معیاری است که تعداد خطوطی از برنامه که توسط تیم تضمین کیفیت و یا توسعه دهندگان نیاز است بررسی و آزموده شود را در نظر می‌گیرد. ایده‌ی موثر بودن از نظر هزینه^{۱۰} برای مدل‌های خطا برای اولین بار توسط آریشلم و همکاران [۳] ارائه گردید. موثر بودن از نظر هزینه به این معنا است که چه تعداد خطا با بررسی و یا تست $n\%$ اول خطوط می‌توان یافت. به عبارت دیگر اگر یک مدل پیش‌بینی خطا بتواند تعداد

^۷Area under curve^۸Area under cost-effectiveness curve^۹Receiver operating characteristic^{۱۰}Cost-effectiveness



شکل ۲.۲: نمونه‌ای از نمودار ROC [۲]

خطای بیشتری را با بررسی و تلاش در آزمون کمتر، نسبت به باقی مدل‌ها بیابد می‌توان گفت که تاثیر آن از نظر هزینه بیشتر است. دو منحنی در قسمت راست شکل ۳.۲ برای دو مدل پیش‌بینی مختلف آمده است. هر دو مدل دارای سطح زیر نمودار یکسانی هستند اما زمانی که ۲۰٪ اول محور افقی در نظر گرفته می‌شود مدل P_2 کارایی بهتری دارد. نمودار سمت چپ مدل‌های تصادفی، عملی^{۱۱} و بهینه را نشان می‌دهد.



R = random P = practical O = optimal

شکل ۳.۲: نمودار موثر بودن از نظر هزینه [۴]

معیارهایی که برای ارزیابی نتایج حاصل از روش رگرسیون به کار گرفته می‌شوند بر اساس همبستگی^{۱۲} میان تعداد خطاهای پیش‌بینی شده و خطاهای واقعی محاسبه می‌شوند. نماینده‌ی این معیارها را می‌توان همبستگی اسپیرمن، پیرسون و R^2 دانست [۱].

^{۱۱}Practical

^{۱۲}Correlation

۳.۱.۲ معیارهای پیش‌بینی خطا

معیارهای پیش‌بینی خطا نقش مهمی را در ساخت مدل پیش‌بینی ایفا می‌کنند. اکثریت معیارهای پیش‌بینی خطا را می‌توان به دو دسته‌ی کلی تقسیم کرد: معیارهای کد و معیارهای فرآیند. معیارهای کد می‌توانند به طور مستقیم از کدهای منبع موجود جمع‌آوری شوند در حالی که معیارهای فرآیند از اطلاعات تاریخی که در مخازن نرم‌افزاری مختلف آرشیو شده‌اند استخراج می‌گردند. نمونه‌ای از این مخازن نرم‌افزاری سیستم‌های کنترل نسخه و سیستم‌های ردگیری خطا است. معیارهای فرآیند از نظر هزینه موثرتر از سایر معیارها هستند [۵]. در برخی از مقالات نیز معیارهای پیش‌بینی خطا به سه دسته‌ی: معیارهای کد منبع سستی، معیارهای شیء‌گرایی و معیارهای فرآیند تقسیم شده‌اند [۶].

معیارهای کد

معیارهای کد تحت عنوان *معیارهای محصول*^{۱۳} نیز شناخته می‌شوند و میزان پیچیدگی کد را می‌سنجند. فرض زمینه‌ی^{۱۴} آنها این است که هرچه کد پیچیده‌تر باشد خطاخیزتر است. برای اندازه‌گیری پیچیدگی کد پژوهش‌گران معیارهای مختلفی را ارائه داده‌اند که در ادامه مهم‌ترین آنها معرفی خواهند شد.

- **معیار اندازه:** معیارهای "اندازه" اندازه‌ی کلی و حجم کد را می‌سنجند. نماینده‌ی این معیارها "تعداد خطوط" می‌باشد و اولین بار توسط *آکیاما*^{۱۵} [۷] ارائه شد. *هالستد*^{۱۶} [۸] چندین معیار اندازه بر اساس تعداد عملگرها و عملوندها ارائه داده است و در مقاله‌ی [۹] مورد بازنگری قرار گرفته است.

- **معیار پیچیدگی حلقوی:** مک‌کیب^{۱۷} معیارهای پیچیدگی حلقوی^{۱۸} را پیشنهاد داد که این معیار با استفاده از تعداد گره‌ها، یالها و قطعات متصل در گراف جریان کنترلی^{۱۹} کد منبع محاسبه می‌گردد [۱۰]. این معیارها نشان می‌دهند که راه‌های کنترلی به چه میزان پیچیده هستند. با وجود اینکه جز اولین معیارها بوده است همچنان در پیش‌بینی خطا کاربرد دارد [۱۱].

- **معیار شیء‌گرایی:** با ظهور زبان‌های شیء‌گرایی و محبوبیت آنها معیارهای کد برای این زبان‌ها ارائه شد

¹³Product Metrics

¹⁴Ground Assumption

¹⁵Akiyama

¹⁶Halstead

¹⁷McCabe

¹⁸Cyclomatic Complexity

¹⁹Control Flow

تا فرآیند توسعه بهبود یابد. نماینده‌ی معیارهای شیء‌گرایی چد/مبر و کمرر^{۲۰} می‌باشند [۱۲]. این معیارها با توجه به خصیصه‌های زبان‌های شیء‌گرا مانند وراثت، زوجیت^{۲۱}، همبستگی^{۲۲} طراحی شده‌اند. بجز معیارهای معیارهای شیء‌گرایی دیگری نیز بر اساس حجم و کمیت کد منبع پیشنهاد داده شده‌اند. مشابه معیارهای اند/زه، معیارهای شیء‌گرایی تعداد نمونه‌های یک کلاس، توابع را می‌شمارند.

معیارهای فرآیند

در ادامه تعدادی از معیارهای فرآیند بررسی می‌شوند که در این دسته شاخص محسوب می‌شوند.

- **تغییر نسبی کد:** ناگاپان و بال^{۲۳} هشت معیار تغییر نسبی کد را ارائه داده‌اند [۱۳]. به عنوان مثال یکی از معیارهای آنها تعداد تجمعی خطوط اضافه و حذف شده بین دو نسخه از برنامه را می‌شمارد و بر تعداد خطوط برنامه تقسیم می‌کند. معیار دیگر تعداد فایل‌های تغییر یافته از یک قطعه برنامه را بر تعداد کل فایل‌ها تقسیم می‌کند.
- **تغییر کد:** این معیارها به عنوان مثال تعداد رفع خطاها، تعداد بازرایی کد^{۲۴} و یا تعداد نویسندگان یک فایل را می‌شمارند. موزر^{۲۵} و همکاران معیارهایی را ارائه داده‌اند که تعداد خطوط اضافه و کم شده را بدون در نظر گرفتن تعداد کل خطوط می‌شمارد. در عوض سن فایل‌ها و تعداد فایل‌هایی که در سیستم کنترل نسخه ثبت^{۲۶} می‌شوند در نظر گرفته می‌شود [۱۴].
- **معیار شهرت:** بکچلی^{۲۷} و همکاران معیارهای شهرت^{۲۸} را بر اساس تحلیل ایمیل‌های آرشیو شده‌ی نویسندگان ارائه داده‌اند. ایده‌ی اصلی این معیارها این است که یک قطعه‌ی نرم‌افزاری که در ایمیل‌ها درباره‌ی آن بیشتر صحبت شده است خطاخیزتر می‌باشد [۱۵]. برد و همکاران چهار معیار مالکیت بر اساس نویسندگان یک قطعه ارائه داده‌اند. مالکیت یک قطعه بر اساس نسبت تعداد ثبت‌های افراد در سیستم کنترل نسخه برای یک قطعه (مشارکت آنها) تعریف می‌شود.

²⁰Chidamber and Kemerer (CK)

²¹Coupling

²²Cohesion

²³Nagappan and Ball

²⁴Refactoring

²⁵Moser

²⁶Commit

²⁷Bacchelli

²⁸Popularity

راجنویک^{۲۹} و همکاران در پژوهش خود به بررسی قاعده‌مند^{۳۰} معیارهای پیش‌بینی خطا در مطالعات پیشین پرداخته‌اند. طبق این پژوهش در ۴۹٪ مطالعات از معیارهای شی‌گرایی، در ۲۷٪ معیارهای سنتی کد و در ۲۶٪ از معیارهای فرآیند استفاده شده است. با توجه به مطالعات بررسی شده دقت پیش‌بینی خطا با انتخاب معیارهای مختلف، تفاوت قابل توجهی پیدا می‌کند. معیارهای شی‌گرایی و فرآیند موفق‌تر از معیارهای سنتی هستند. معیارهای سنتی پیچیدگی کد، قویا با معیارهای اندازه مانند تعداد خطوط کد همبستگی دارند و این دو توانایی پیش‌بینی خطا دارند اما جز بهترین معیارها نیستند. معیارهای شی‌گرایی بهتر از اندازه و پیچیدگی عمل می‌کنند و با این که با معیارهای اندازه همبستگی دارند اما ویژگی‌های بیشتری علاوه بر اندازه را دارند. معیارهای ایستای کد همانند اندازه، پیچیدگی و شی‌گرایی به منظور بررسی یک نسخه از برنامه مفید هستند اما با هر تکرار^{۳۱} در فرآیند توسعه نرم‌افزار دقت پیش‌بینی آنها کاسته می‌شوند و معیارهای فرآیند در چنین شرایطی بهتر عمل می‌کنند. با این وجود که معیارهای فرآیند دارای توانمندی بالقوه‌ای هستند، اما در تعداد کمتری از پژوهش‌ها مورد استفاده قرار گرفته‌اند[۶].

آسترند^{۳۲} و همکاران به بررسی این موضوع پرداخته‌اند که آیا اطلاعاتی درباره‌ی اینکه کدام توسعه‌دهنده یک فایل را اصلاح می‌کند قادر است که پیش‌بینی خطا را بهبود بخشد. در پژوهش قبلی آنها[۱۶] مشخص شده بود که تعداد کلی افراد توسعه‌دهنده در یک فایل می‌تواند در پیش‌بینی خطا تاثیر متوسطی داشته باشد. در مقاله‌ی [۱۷] تعدادی از متغیرهای کد منبع و فرآیند به همراه معیار مرتبط به توسعه‌دهنده در نظر گرفته شده است. در این پژوهش مشخص شد که تعداد خطاهایی که یک توسعه‌دهنده تولید می‌کند ثابت است و با سایر توسعه‌دهندگان فرق دارد. این تفاوت با حجم کدی که یک توسعه‌دهنده اصلاح می‌کند مرتبط است و در نتیجه در نظر گرفتن یک نویسنده خاص نمی‌تواند به بهبود پیش‌بینی خطا کمک کند[۱۷].

رحمان و دونبو^{۳۳} از جنبه‌های مختلف معیارهای فرآیند را با سایر معیارها مقایسه کرده‌اند[۱۸]. نتایج نشان می‌دهد زمانی که مدل پیش‌بینی بر روی یک نسخه آموزش می‌بیند و در نسخه‌ی بعدی آزموده می‌شود معیارهای کد، AUC قابل قبولی دارند اما AUC آنها کمتر از معیارهای فرآیند است و از نظر معیار ۲۰٪ AUCEC بهتر از

²⁹Radjenovic³⁰Systematic Review³¹Iteration³²Ostrand³³Rahman and Devanbu

یک مدل تصادفی عمل نمی‌کنند و به آن معنی است که این معیارها از نظر هزینه چندان موثر نیستند. همچنین معیارهای کد ایستار هستند، یعنی با تغییرات پروژه و تغییر در توزیع خطاها همچنان معیارها بدون تغییر باقی می‌مانند. معیار ایستا تمایل دارد یک فایل را در انتشارهای^{۳۴} متوالی همچنان حاوی خطا معرفی کند. معیارهای ایستا به مدل‌های راکد منجر می‌شوند که این مدل‌ها به سمت فایل‌های بزرگ با تراکم خطای کمتر جهت‌گیری^{۳۵} دارند. به عنوان مثال حالتی را در نظر بگیرید که در یک پروژه فایل‌های بزرگ و پیچیده‌ای وجود دارد که پس از چندین انتشار خطاهای آنها برطرف می‌شود اما مدل‌هایی که بر اساس معیارهای کد ساخته شده‌اند همچنان این فایل‌ها را به عنوان خطاخیز معرفی می‌کنند. از طرف دیگر حالتی را در نظر بگیرید که یک فایل با اندازه و پیچیدگی کم به تازگی به وجود آمده و یا تغییرات فراوان یافته است. مدل‌های مبتنی بر کد به این فایل‌ها توجه چندانی نخواهند کرد در حالیکه که این فایل‌ها مستعد وجود خطا هستند. بدین ترتیب معیارهای فرآیند بهتر از معیارهای کد عمل می‌کنند.

۴.۱.۲ مدل‌های پیش‌بینی خطا

اکثریت مدل‌های پیش‌بینی خطا بر اساس یادگیری ماشین می‌باشند. بر اساس اینکه چه چیزی پیش‌بینی شود (خطاخیز بودن یا تعداد خطا)، مدل‌ها به دو دسته‌ی کلی تقسیم می‌شوند، که عبارتند از دسته‌بندی و رگرسیون. با توسعه‌ی روش‌های جدیدتر یادگیری ماشین تکنیک‌های فعال و نیمه-نظارتی^{۳۶} برای ساخت مدل‌های پیش‌بینی خطای کاراتر به کار گرفته شده است [۱۹]. علاوه بر مدل‌های یادگیری ماشین، مدل‌های غیر آماری مانند باگ‌کش^{۳۷} پیشنهاد داده شده است [۲۰]. در میان روش‌های دسته‌بندی، Logistic Regression، Naive Bayes و Decision Tree بیش از سایرین در پژوهش‌ها مورد استفاده قرار گرفته‌اند. همچنین در میان روش‌های رگرسیون Linear Regression و Negative Binomial Regression به طور گسترده به کار گرفته شده‌اند [۱].

کیم^{۳۸} و همکاران باگ‌کش را ارائه داده‌اند که اولویت موجودیت‌های خطاخیز در کش را نگهداری می‌کند. این روش از اطلاعات محلی خطاها مانند اطلاعات زمانی و مکانی بهره می‌گیرد. به عنوان مثال اگر خطا در یک موجودیت به تازگی به وجود آمده یا همراه با سایر موجودیت‌ها تغییر کرده است، آن موجودیت با احتمال بیشتری حاوی خطا خواهد بود.

اگرچه مدل‌های یادگیری مختلف می‌تواند با توجه به داده‌های ورودی یکسان، متفاوت عمل کنند و کارایی یک

³⁴Release

³⁵Bias

³⁶Semi-Supervised

³⁷BugCache

³⁸Kim

روش نسبت به دیگری متفاوت باشد، با این حال پژوهشی که توسط آریشل و همکاران [۵] انجام شده است نشان می‌دهد که تاثیر تکنیک یادگیری در حد متوسطی است و کمتر از انتخاب معیار بر روی کارایی تاثیر گذار است.

مالهوتر^{۳۹} با بکارگیری معیارهای سنتی کد، عملکرد تکنیک‌های یادگیری ماشین و رگرسیون را مقایسه کرده است [۱۱]. وی به منظور پیش پردازش نیز از آماره‌های توصیفی^{۴۰} استفاده کرده است و داده‌های نامناسب را شناسایی نموده است. آماره‌های توصیفی می‌توانند شامل میانگین، کمینه، بیشینه و واریانس باشد. متغیرهای مستقلی که واریانس کمی دارند مازول‌ها را به خوبی متمایز نمی‌کنند و بعید است که مفید باشند و می‌توانند حذف شوند. یک روش رگرسیون و شش روش دسته‌بندی مورد آزمایش قرار گرفته‌اند که در میان آنها سه روش رایج و سه روش که کمتر مورد استفاده قرار می‌گیرند انتخاب شده‌اند. Logistic Regression به عنوان روش رگرسیون انتخاب شده و نتایج نشان می‌دهد که روش‌های دسته‌بندی بهتر از روش رگرسیون عمل می‌کند. در میان روش‌های دسته‌بندی درخت تصمیم^{۴۱} بهتر از سایرین عمل کرده است.

۱۰۴.۱.۲ درشت‌دانگی پیش‌بینی

در پژوهش‌های انجام شده مدل‌های پیش‌بینی در سطوح مختلفی از ریزدانگی ساخته شده‌اند از جمله: زیر سیستم، قطعه یا بسته، فایل یا کلاس، تابع و تغییر. هتا^{۴۲} و همکاران پیش‌بینی در سطح تابع را ارائه داده‌اند و به این نتیجه رسیده‌اند که پیش‌بینی خطا در سطح تابع نسبت به سطوح درشت‌دانه‌تر از نظر هزینه موثرتر است [۲۱]. کیم و همکاران نیز مدل جدیدی ارائه داده‌اند که دسته‌بندی تغییر^{۴۳} نام دارد. بر خلاف سایر مدل‌های پیش‌بینی، ”دسته‌بندی تغییر می‌تواند به طور مستقیم به توسعه دهنده کمک کند. این مدل می‌تواند زمانی که توسعه دهنده تغییری در کد منبع ایجاد می‌کند و آنرا در سیستم کنترل نسخه ثبت می‌کند، نتایج آنی را فراهم کند. از آنجا که این مدل بر اساس بیش از ده هزار ویژگی ساخته می‌شود، سنگین‌تر از آن است که در عمل مورد استفاده قرار گیرد [۲۲].

۲.۲ آزمون جهش و کاربردهای آن

توسعه‌دهندگان و پژوهش‌گران حوزه‌ی نرم‌افزار علاقه‌مند به اندازه‌گیری موثر بودن مجموعه‌های آزمون می‌باشند. توسعه‌دهندگان به دنبال آن هستند که بدانند مجموعه آزمون‌های آنها می‌تواند به خوبی خطاها را تشخیص دهد

³⁹ Malhotra

⁴⁰ Descriptive Statistics

⁴¹ Decision Tree

⁴² Hata

⁴³ Change Classification

و پژوهشگران به دنبال مقایسه‌ی روش‌های مختلف آزمون و *اشکال زدایی*^{۴۴} هستند. به طور ایده آل افراد تمایل دارند که بدانند تعداد خطاهایی که یک مجموعه آزمون می‌تواند شناسایی کند چه مقدار است اما از آنجا که خطاها ناشناخته هستند باید از *اندازه‌گیری وکالتی*^{۴۵} استفاده شود. یکی از اندازه‌گیری‌های شناخته شده *امتیاز جهش*^{۴۶} می‌باشد که توانایی مجموعه آزمون در تمیز دادن نسخه‌ی اصلی برنامه از تعداد زیادی نسخه‌های متفاوت را اندازه‌گیری می‌کند. این نسخه‌های متفاوت که تنها یک تفاوت کوچک نحوی نسبت به برنامه‌ی اصلی دارند *جهش‌یافته*^{۴۷} نامیده می‌شوند. امتیاز جهش درصد جهش‌یافته‌هایی است که توسط مجموعه آزمون از برنامه‌ی اصلی تمیز داده می‌شوند. به این صورت که این جهش‌یافته‌ها باعث شکست یک مورد آزمون می‌شوند در حالی که در نسخه‌ی اصلی مجموعه‌ی آزمون با موفقیت اجرا می‌گردد. جهش‌یافته‌ها با تزریق خطاهای ساختگی به برنامه‌ی تحت آزمون ساخته می‌شوند. نمونه‌ای از جهش‌یافته‌ها برای یک قطعه کد در شکل ۴.۲ آمده است. این خطاهای ساختگی با استفاده از عملگرهای جهش که از پیش تعریف شده‌اند ساخته می‌شود. نمونه‌ی این عملگرها جایگزینی عملگرهای ریاضی یا رابطه‌ای، تغییر شرط شاخه^{۴۸} و یا حذف یک عبارت است [۲۳].

تحلیل آزمون در موارد زیر کاربرد دارد:

- ارزیابی مجموعه آزمون
- انتخاب مجموعه آزمون
- کمینه سازی مجموعه آزمون
- تولید مجموعه آزمون
- مکان‌یابی خطا
- پیش‌بینی خطا

⁴⁴Debugging

⁴⁵Proxy Measurement

⁴⁶Mutation Score

⁴⁷Mutant

⁴⁸Branch Condition

Statements	Mutants
s ₁ : max = -x;	m1: max -= x-1; m2: max=x;
s ₂ : if(max < y) {	m3: if(!(max<y)) { m4: if(max==y) {
s ₃ : max = y;	m5: max = -y; m6: max = y+1;
s ₄ : if(x*y<0) {	m7: if(!(x*y<0)) m8: if(x/y<0)
s ₅ : print(`diff.sign`);}	m9: return; m10:;
s ₆ : print(max); }	m11: printf(0);} m12:;

شکل ۴.۲: نمونه‌ای از جهش‌یافته‌های یک برنامه [۲۴]

جاست^{۴۹} و همکاران در پژوهش خود به بررسی این موضوع پرداخته‌اند که آیا جهش‌یافته‌ها می‌توانند جایگزین مناسبی برای خطاهای واقعی باشند یا خیر [۲۳]. در پژوهش‌های گذشته بررسی شده بود که میان جهش‌یافته‌های ساده و پیچیده وابستگی وجود دارد ولی وابستگی میان جهش‌یافته‌های ساده و خطاهای واقعی مشخص نیست. جاست و همکاران دو مجموعه‌ی آزمون برای هر خطا در نظر گرفتند که مجموعه‌ی اول در نسخه‌ی حاوی خطا با موفقیت گذرانده می‌شود. مجموعه‌ی دوم در نسخه‌ی حاوی خطا شکست می‌خورد و در نسخه‌ی رفع خطا با موفقیت اجرا می‌شود. نتایج نشان می‌دهد که مجموعه‌ی آزمون دوم دارای امتیاز جهش بالاتری می‌باشد که نشان می‌دهد هر خطا به یک جهش‌یافته وابستگی دارد. لازم به ذکر است که سعی شده دو مجموعه‌ی آزمون دارای پوشش یکسانی باشند زیرا پوشش بیشتر می‌تواند امتیاز جهش بیشتر بیانجامد. همچنین مشخص شد که ۷۳٪ خطاهای واقعی با جهش‌یافته‌هایی که با عملگرهای متوال تولید شده‌اند وابستگی دارند. در این پژوهش خطاهایی که با جهش‌یافته‌ها وابستگی ندارند در سه دسته قرار می‌گیرند: دسته اول نیازمند عملگرهای قوی‌تری هستند، دسته دوم نیازمند عملگرهای جدیدی هستند و دسته سوم با جهش‌یافته‌ها وابستگی ندارند.

۱.۲.۲ مکان‌یابی خطا

روش‌هایی که از جهش‌یافته‌ها به منظور مکان‌یابی خطا استفاده می‌کنند دارای شباهت‌هایی با روش‌های پیش‌بینی خطا هستند. در هر دوی این روش‌ها از معیارهایی که منبع استفاده می‌شود تا احتمال وجود خطا

⁴⁹Just

محاسبه شود. دو تفاوت عمده‌ی این دو حوزه این است که اولاً در مکان‌یابی خطا از روش‌های یادگیری ماشین استفاده‌ی چندانی نمی‌شود، ثانیاً در مکان‌یابی خطا وجود خطا به وسیله شکست مورد آزمون یا گزارش خطا محرز شده است. با توجه به شباهت‌های موجود میان این دو حوزه در ادامه چند مقاله که با استفاده از آزمون جهش خطا را مکان‌یابی کرده‌اند، بررسی می‌کنیم.

موون^{۵۰} و همکاران در مقاله‌ی خود بر اساس دو فرض روشی به منظور مکان‌یابی خطا ارائه داده‌اند. فرض اول بیان می‌کند که در یک برنامه‌ی حاوی خطا جهش و یا اصلاح یک عبارت خطا دار نسبت به جهش یک عبارت درست می‌تواند موارد آزمون بیشتری را با موفقیت بگذرانند. فرض دوم بیان می‌کند که جهش عبارات صحیح نسبت به جهش یک عبارت غلط موجب می‌شود موارد آزمون بیشتری شکست بخورند. بر اساس این دو فرض معیاری به نام مشکوک بودن^{۵۱} ارائه گردیده است که دو فرض را فرموله می‌کند. این معیار بر اساس تعداد شکست و موفقیت موارد آزمون در نسخه‌ی اصلی و جهش‌یافته عمل می‌کند. سپس با رتبه‌بندی عبارات بر اساس این معیار عبارت حاوی خطا مشخص می‌گردد. در این پژوهش روش جدیدی نیز به منظور ارزیابی روش پیشنهادی ارائه شده است که برخی از مشکلات روش پیشین را بر طرف نموده است. در نهایت روش مکان‌یابی ارائه شده با دو روش ارزیابی شده و نتایج نشان می‌دهد فرضیات پژوهش درست بوده‌اند [۲۴].

پاپاداکیس و تراون^{۵۲} در مقاله‌ی خود به این نکته اشاره کرده‌اند که استفاده از تحلیل جهش در گذشته به دلیل پرهزینه بودن چندان مورد توجه قرار نمی‌گرفته است اما امروزه با وجود ابزارهای مقیاس پذیر، نمونه‌گیری و انتخاب جهش می‌توان به خوبی از تحلیل جهش در انجام پژوهش‌های مختلف استفاده کرد [۲۵]. آنها روشی را برای مکان‌یابی خطا بر اساس دو مشاهده ارائه کرده‌اند. در مشاهده‌ی اول دیده می‌شود که خطای موجود در یک عبارت رفتار مشابهی با جهش در همان عبارت نشان می‌دهد. در مشاهده‌ی دیگر دیده می‌شود که اگر خطا و جهش در دو عبارت متفاوت باشند رفتار متفاوتی خواهند داشت. منظور از رفتار مشابه موفقیت یا شکست در یک آزمون است. بر اساس این دو مشاهده معیاری برای مشکوک بودن عبارات تعیین می‌گردد. این پژوهش بیان می‌کند که مناسب بودن موارد آزمون تاثیر مستقیمی بر عملکرد روش مکان‌یابی خطا دارد. همچنین یک مجموعه‌ی کوچک از جهش‌یافته‌ها می‌تواند به اندازه‌ی مجموعه‌ای کامل تاثیر گذار باشد.

⁵⁰Moon⁵¹Suspiciousness⁵²Papadakis and Traon

۲.۲.۲ مدل‌های یادگیری و جهش‌یافته‌ها

ها/و^{۵۳} و همکاران با ارایه‌ی مجموعه‌ای از معیارها و استفاده از یادگیری ماشین مدلی را ارائه داده‌اند که به وسیله‌ی آن بتوان تشخیص داد علت شکست در آزمون رگرسیون وجود خطا است یا منسوخ^{۵۴} شدن یک مورد آزمون [۲۶]. هفت معیار ارائه شده در این پژوهش مرتبط با گراف فراخوانی، تغییر در فایل‌ها و تعداد شکست در آزمون‌ها بوده است. هاو و همکاران به منظور به دست آوردن مجموعه داده‌ی حاوی خطا، به صورت دستی بر اساس استانداردهایی از پیش تعریف شده خطاهایی را در کد قرار داده‌اند. بدین منظور عباراتی به صورت تصادفی که در سراسر کد محصول قرار دارند انتخاب شدند و به وسیله‌ی عملگرهای جهش خطاهایی تولید شده است. به منظور بدست آوردن آزمون‌های منسوخ شده، مجموعه آزمون‌هایی از نسخه‌ی قبلی برنامه بر روی کد نسخه‌ی بعدی به کار گرفته شده است. سپس با استفاده از روش ارزیابی میان دسته‌ی^{۵۵} به آموزش و آزمایش مدل ساخته شده پرداخته می‌شود. نتایج پژوهش نشان می‌دهد که روش پیشنهادی زمانی که بر روی یک نسخه یا نسخه‌های مختلف از یک برنامه اعمال شود نتایج خوبی دارد (۸۰٪ دقت) اما زمانی که بر روی برنامه‌های مختلف اعمال شود (مجموعه آموزش از یک برنامه و آزمون بر روی برنامه‌ای دیگر) موثر نیست. نتایج نشان می‌دهد تکنیک‌ها مکان‌یابی خطا نتیجه‌ی مثبتی بر تشخیص نوع خطا که مربوط به محصول است یا آزمون، ندارد.

بوئر^{۵۶} و همکاران معیارهایی را مبتنی بر جهش معرفی کردند و از ترکیب آنها با معیارهای سنتی و شی‌گرایی، یک مدل پیش‌بینی ساخته شده است [۲۷]. ۸ عملگر جهش در نظر گرفته شده و برای هر یک از آنها یک معیار ایستا (بدون اجرای کد) و چهار معیار پویا ساخته شده و در مجموع ۴۰ معیار جهش ارائه شده است. به این دلیل میان معیار ایستا و پویا تمایز قائل شده‌اند که اگر معیارهای ایستا به تنهایی پیش‌بینی را بهبود بخشند بدون نیاز به موارد آزمون می‌توان از آنها استفاده کرد، در واقع دامنه‌ی کاربرد روش گسترده‌تر می‌گردد. نتایج پژوهش نشان می‌دهد که استفاده از معیارهای جهش بهبود قابل توجهی را در پیش‌بینی خطا به وجود می‌آورد. همچنین معیارهای پویا و ایستا در کنار یکدیگر توانایی پیش‌بینی مناسبی دارند ولی استفاده‌ی جداگانه از آنها تاثیر چندان مثبتی نخواهد داشت. این پژوهش از دو جنبه حائز اهمیت می‌باشد. یکی اینکه اولین پژوهش در زمینه‌ی پیش‌بینی خطاست که از تحلیل جهش استفاده کرده است. دوم آنکه مشابه‌ترین پژوهش به پژوهش کنونی می‌باشد.

⁵³Hao

⁵⁴Obsolete

⁵⁵Cross-validation

⁵⁶Bowes

۳.۲ جمع بندی مطالعات پیشین

هدف از پیش‌بینی خطا کمک به توسعه‌دهندگان نرم‌افزار و کاهش هزینه‌های نرم‌افزاری می‌باشد. روند پیش‌بینی خطا به این صورت است که با استفاده از مخازن نرم‌افزاری همانند سیستم کنترل نسخه و سیستم ردگیری خطا، اطلاعات کد منبع، خطا و اطلاعات تاریخی پروژه جمع‌آوری می‌شود. با توجه به معیارهای مختلف داده‌هایی استخراج می‌شود که هر داده دارای برجسته‌ترین "سالم" یا "حاوی خطا" می‌باشد. قسمتی از این داده‌ها با استفاده از روش‌های یادگیری ماشین، مدل‌های پیش‌بینی خطا را تولید می‌کنند و قسمت دیگر جهت آزمایش مدل به کار گرفته می‌شود.

معیارهای متداول در ارزیابی پیش‌بینی دقت و فراخوانی می‌باشند. این معیارها دارای نواقصی هستند. به عنوان مثال مدلی که همه‌ی داده‌ها را خطا دار معرفی می‌کند دارای فراخوانی برابر یک است و مسلماً این مدل کارایی مناسبی ندارد. معیار اف میانگین هارمونیک دو معیار قبلی است و نواقص آنها را بر طرف می‌کند. یکی از معیارهای رایج برای مقایسه‌ی مدل‌های یادگیری ماشین AUC می‌باشد. هرچه این مساحت بیشتر باشد و منحنی مربوطه سریعتر در راستای محور عمودی به یک برسد مدل کارایی بهتری دارد. با استفاده از معیار AUCEC می‌توان موثر بودن مدل از نظر هزینه را سنجید. معمولاً چند درصد اول از منحنی مربوطه در نظر گرفته می‌شود و مساحت آن محاسبه می‌شود.

معیارهای مورد استفاده را می‌توان به سه دسته‌ی معیار سنتی کد، معیار شیء گرایی و معیار فرآیند تقسیم کرد. در برخی از منابع نیز به دو دسته‌ی کلی معیار کد و معیار فرآیند تقسیم شده‌اند. معیارهای اندازه جزء معیارهای ابتدایی و موثر هستند و معیارهای پیچیدگی و شیء گرایی همبستگی فراوانی با معیارهای اندازه دارند. معیارهای شیء گرایی دارای وابستگی فراوانی با معیارهای اندازه هستند. با این حال معیارهای شیء گرایی دارای توانایی بیشتری هستند. معیارهای فرآیند از جنبه‌های مختلفی مانند عدم رکود در تکرارهای چرخه‌ی تولید نرم‌افزار و موثر بودن از نظر هزینه از سایر معیارها برتری دارد. علی‌رغم توانمندی بالقوه‌ی معیارهای فرآیند در پیش‌بینی خطا، این معیارها در پژوهش‌های کمتری مورد تحقیق قرار گرفته‌اند.

در پژوهش‌های مختلف از روش‌های یادگیری ماشین متفاوتی استفاده شده است. در صورتی که هدف پیش‌بینی تعداد خطاها باشد از رگرسیون و در صورتی که هدف پیش‌بینی حاوی خطا بودن باشد از دسته‌بندی

استفاده می‌شود. پژوهش [۵] نشان داده است که روش دسته‌بندی تاثیر متوسطی بر کارایی پیش‌بینی خطا دارد و انتخاب معیار مهم‌تر است.

در ابتدا از امتیاز جهش برای میزان موثر بودن مجموعه آزمون استفاده می‌شد و سپس کاربردهای دیگری همچون انتخاب، رتبه‌بندی و کمینه کردن مجموعه آزمون پیدا کرده است. همچنین در پژوهش‌های اخیر جهت مکان‌یابی خطا و پیش‌بینی خطا مورد استفاده قرار گرفته است. در پژوهش [۲۳] نشان داده شده است که جهش‌یافته‌هایی که با عملگرهای جهش ساده تولید شده‌اند می‌توانند تا ۷۳٪ خطاهای واقعی را شبیه‌سازی کنند و ازین جهت جایگزین مناسبی برای خطاهای واقعی باشند.

جدول ۲.۲: جدول مشخصات پژوهش‌های مرور شده در حوزه‌ی پیش‌بینی خطا

مقاله	معیار	تکنیک یادگیری	ریزدانگی	روش ارزیابی	نوع پروژه‌ها	زبان پروژه‌ها
[۱۷]	فرآیند - سنتی	NBR	فایل	مشابه AUCEC	خصوصی	جاوا
[۱۸]	فرآیند - سنتی - شی‌گرایی	Naive Bayes - Logestic Regression - SMV - J48	فایل	AUC - AUCEC - F-Measure	متن باز	جاوا
[۲۷]	سنتی - شی‌گرایی	Naive Bayes - Logestic Regression - Random Forest - J48	کلاس	غیره	متن باز	جاوا
[۱۱]	سنتی	LR - ANN - DT - SVM - CCN - GMDH - GEP	NA	AUC - Precision	متن باز	سی
[۲۸]	سنتی - فرآیند	Naive Bayes - DT - kNN - RF	سیستم	AUC - Precision - Recall - F-Measure	متن باز	اندروید
[۲۹]	سنتی - شی‌گرایی	LR - ANN - RBFN	کلاس	Accuracy - F-Measure	متن باز	جاوا

[این صفحه آگاهانه خالی گذاشته شده است.]

فصل ۳

معیارهای جهش و فرآیند

با مطالعات مروری انجام شده نقاطی از این حوزه که نیازمند پژوهش بیشتر هستند تا بتوان به وسیله‌ی آن به ارائه‌ی روشی کارا تر در پیش‌بینی خطا پرداخت مشخص شد. مقاله‌ی [۲۷] اولین مقاله‌ای است که یک روش پیش‌بینی خطا با استفاده از تحلیل جهش ارائه نموده است و این موضوع نیازمند تحقیق بیشتر است. از طرف دیگر بر طبق مقاله‌ی [۶] استفاده از معیارهای فرآیند علی‌رغم توانایی بالقوه‌ای که در پیش‌بینی خطا دارند، در پژوهش‌های کمتری مورد بررسی قرار گرفته‌اند. یکی از دلایل آن می‌تواند نو ظهور بودن این معیارها نسبت به سایرین باشد. معیارهای فرآیند از جنبه‌های مختلف نیز از سایر معیارها برتری دارند [۱۸].

این پژوهش قصد دارد سه رویکرد پیشنهادی را به منظور بهبود پیش‌بینی خطا بررسی کند. این رویکردها عبارتند از:

۱. در این رویکرد معیارهای جهش و معیارهای فرآیند در کنار یکدیگر استفاده می‌شوند و به وسیله‌ی آنها پیش‌بینی انجام می‌گیرد. این دو دسته معیار در پژوهش‌های گذشته مطرح شده‌اند اما تاکنون در کنار یکدیگر قرار نگرفته‌اند.

۲. معیارهای جدیدی مطرح می‌شوند که مبتنی بر مفاهیم آزمون جهش و فرآیند توسعه‌ی نرم‌افزار است.

۳. معیارهای جدیدی مطرح می‌شوند که با کمک مفاهیم جهش سعی در بهبود معیارهای فرآیند دارند.

۱.۳ معیارهای جهش و فرآیند

این رویکرد با توجه به مقاله‌ی [۲۷] مطرح شده که در آن بررسی به کارگیری معیارهای جهش و فرآیند را در پژوهش‌های آتی توصیه می‌کند. همچنین معیار جهش یک معیار مرتبط با کد است. مقاله‌ی [۱۸] بیان می‌کند که معیارهای کد ایستا هستند و تمایل دارند که یک موجودیت را در انتشارهای متوالی حاوی خطا معرفی کنند. حال شرایطی را در نظر بگیرید که که امتیاز جهش در یک موجودیت کم باشد و دلیل آن کافی نبودن

مجموعه آزمون باشد چرا که توسعه‌دهندگان از درست بودن کد اطمینان دارند یا اینکه پس از انتشارهای متوالی خطاها بر طرف شده است. چنین موجودیتی حاوی خطا نیست اما با توجه به معیار جهش خطاخیز است. با در نظر گرفتن معیارهای فرآیند در مورد این موجودیت که نشان می‌دهند پایدار و بدون تغییر است از میزان خطاخیز بودن آن کاسته می‌شود و انتظار می‌رود کارایی مدل پیش‌بینی بهبود یابد. برای پاسخ به این پرسش مجموعه معیارهای جهش از پژوهش [۲۷] و معیارهای فرآیند از پژوهش [۱۸] انتخاب می‌شوند. در جداول ۱.۳ و ۲.۳ معیارهای مورد نظر آورده شده است و در ادامه معرفی شده و

جدول ۱.۳: معیارهای فرآیند [۱۸]

نام معیار	توضیح
۱	تعداد ثبت در سیستم کنترل نسخه
۲	تعداد توسعه‌دهندگان فعال
۳	تعداد توسعه‌دهندگان متمایز
۴	مقدار نرمال‌سازی شده‌ی تعداد خطوط اضافه شده
۵	مقدار نرمال‌سازی شده‌ی تعداد خطوط حذف شده
۶	درصد خطوطی که مالک فایل مشارکت کرده
۷	تعداد مشارکت‌کنندگان جزئی
۸	تعداد ثبت‌های همسایگان
۹	تعداد توسعه‌دهندگان فعال همسایگان
۱۰	تعداد توسعه‌دهندگان متمایز همسایگان
۱۱	تجربه‌ی مالک فایل
۱۲	تجربه‌ی تمام مشارکت‌کنندگان

جدول ۲.۳: معیارهای جهش [۲۷]

نام معیار	توضیح
۱	تعداد جهش‌یافته‌های تولید شده
۲	تعداد جهش‌یافته‌های پوشش داده شده توسط آزمون‌ها
۳	امتیاز جهش‌یافته‌های تولید شده
۴	امتیاز جهش‌یافته‌های پوشش داده شده توسط آزمون‌ها

از آنجا که در این پژوهش پیش‌بینی‌ها در سطح فایل انجام می‌شود، معیارها برای هر فایل جداگانه محاسبه می‌شوند. در ادامه هر یک از معیارهای فرآیند معرفی و نحوه‌ی محاسبه‌ی آن‌ها بیان می‌شود. معیارهای جهش به طور مستقیم توسط ابزارهای موجود محاسبه می‌گردد.

۱. **تعداد ثبت در سیستم کنترل نسخه:** تعداد ثبت‌هایی که در آن فایل مورد نظر در طول انتشار قبلی تاکنون تغییر کرده است. برای محاسبه‌ی آن لازم است که تمام ثبت‌های پروژه بین ثبت کنونی و انتشار قبلی بررسی شود و ثبت‌هایی که در آن این فایل تغییر کرده‌اند شمرده شوند.
۲. **تعداد توسعه‌دهندگان فعال:** تعداد توسعه‌دهندگانی که در طول انتشار قبلی تا کنون (زمان ثبت) فایل را تغییر داده‌اند. لازم است ثبت‌های موجود در بازه‌ی زمانی خواسته شده بررسی شود و آنها که فایل مورد نظر را تغییر داده‌اند انتخاب شوند. نام کسانی که ثبت را انجام داده‌اند بازیابی شود و تعداد نامهای متمایز شمرده شود.
۳. **تعداد توسعه‌دهندگان متمایز:** مشابه معیار قبلی با این تفاوت که در طول انتشار محاسبه نمی‌شود. بلکه از ابتدای پروژه تا زمان ثبت در نظر گرفته می‌شود.
۴. **مقدار نرمال‌سازی شده‌ی تعداد خطوط اضافه شده:** این معیار تعداد خطوط اضافه شده در یک فایل را در طول انتشار قبلی می‌شمارد. سپس جهت نرمال سازی آنرا بر تعداد کل خطوط اضافه شده در پروژه در طول انتشار قبلی تقسیم می‌کند. برای بدست آوردن تعداد خطوط اضافه شده در یک فایل هر ثبت نسبت به ثبت قبلی مقایسه می‌شود و تعداد خطوط اضافه شده جمع زده می‌شود.
۵. **مقدار نرمال‌سازی شده‌ی تعداد خطوط حذف شده:** مشابه معیار قبلی می‌باشد.
۶. **تعداد خطوطی که مالک فایل مشارکت کرده:** درصد خطوطی از فایل، در ثبت مورد نظر که به مالک فایل تعلق دارد. مالک فایل کسی است که در آن لحظه از زمان بیشترین تعداد خطوط موجود در فایل به او تعلق دارد. ابتدا نویسنده‌ی هر خط مشخص می‌شود سپس برای هر نویسنده تعداد خطوطی که به وی تعلق دارد شمرده می‌شود. تعداد خطوط مالک فایل بر تعداد خطوط فایل تقسیم می‌گردد.
۷. **تعداد مشارکت‌کنندگان جزئی:** توسعه‌دهنده‌ی جزئی کسی است که کمتر از ۵٪ خطوط موجود در فایل به او تعلق داشته باشد. بدین منظور نویسنده‌ی هر خط مشخص می‌شود. تعداد خطوط هر نویسنده شمرده می‌شود و بر تعداد خطوط فایل تقسیم می‌شود. سپس تعداد نویسندگانی که کمتر از ۵٪ مشارکت داشته‌اند شمرده می‌شود.
۸. **تعداد ثبت‌های همسایگان میانگین وزن دهی شده:** تعداد ثبت‌های همسایگان فایل از انتشار قبلی تا کنون را اندازه‌گیری می‌کند. همسایگان یک فایل در یک ثبت، فایل‌هایی هستند که در آن نسخه از

برنامه تغییر کرده‌اند. درواقع در هر ثبت از برنامه تعدادی فایل نسبت به ثبت قبلی تغییر کرده‌اند که این فایل‌ها همسایه‌ی یکدیگر محسوب می‌شوند. نحوه‌ی وزن دهی نیز به این صورت است که هرچقدر یک فایل تعداد دفعات بیشتری را در طول انتشار با فایل مورد نظر همسایه شده باشد وزن بیشتری می‌یابد. برای محاسبه ابتدا همسایگان فایل در ثبت و تعداد دفعاتی که در طول انتشار همسایه شده‌اند مشخص می‌شوند. سپس برای هر فایل همسایه، معیار تعداد ثبت در سیستم کنترل نسخه محاسبه می‌شود. هر معیار در تعداد دفعاتی همسایگی ضرب می‌شود و با هم جمع زده می‌شوند. در انتها بر تعداد کل دفعات همسایگی همسایگان تقسیم می‌شود.

۹. **تعداد توسعه‌دهندگان فعال همسایگان:** مشابه معیار قبلی عمل می‌شود با این تفاوت که معیار توسعه‌دهندگان فعال در نظر گرفته خواهد شد.

۱۰. **تعداد توسعه‌دهندگان متمایز همسایگان:** مشابه معیار قبلی عمل می‌شود با این تفاوت که معیار توسعه‌دهندگان متمایز در نظر گرفته خواهد شد.

۱۱. **تجربه‌ی مالک فایل:** ابتدا لازم است که نحوه‌ی محاسبه تجربه را تعریف کنیم. هرچقدر یک فرد تعداد تغییرات بیشتری را در یک پروژه انجام دهد تجربه بیشتری را در آن پروژه دارد و ثبت را می‌توان به ایجاد تغییر تعبیر کرد. برای محاسبه‌ی معیار ابتدا مالک فایل مشخص می‌شود. سپس تعداد ثبت‌هایی که مالک فایل از ابتدای پروژه تا زمان مورد نظر انجام داده، شمرده می‌شود.

۱۲. **تجربه‌ی تمام مشارکت‌کنندگان:** تمام مشارکت‌کنندگان در فایل تا زمان ثبت مورد نظر یافت می‌شوند. برای هر یک مشابه معیار قبلی تجربه، محاسبه می‌شود و از مقدار تجربه‌ها میانگین هندسی گرفته می‌شود.

۲.۳ معیارهای جهش مبتنی بر فرآیند

در رویکرد دوم، چهار معیار جدید در این پژوهش معرفی می‌شوند که با استفاده از مفاهیم آزمون جهش و تاریخچه‌ی توسعه‌ی نرم‌افزار ساخته می‌شوند. از این رو این معیارها معیارهای جهش مبتنی بر فرآیند^۱ نامیده شده‌اند.

۱. **تعداد جهش‌یافته‌های تولید شده‌ی جدید نسبت به انتشار قبلی برنامه:** همانطور که در مقاله‌ی

[۲۳] مطرح شده جهش‌یافته‌ها جایگزین خوبی برای خطاهای واقعی می‌باشند. زمانی که تعداد

^۱Process Based Mutation Metrics (PBMM)

جهش‌یافته‌های جدید زیاد باشد یعنی تغییراتی که خطاخیزتر هستند بیشتر است. به منظور محاسبه‌ی این معیار لازم است خطوط اضافه شده به فایل مورد نظر در ثبت کنونی، نسبت به انتشار قبلی مشخص شود و سپس تعداد جهش یافته‌هایی که این خطوط تولید می‌کنند شمرده شوند.

۲. **تعداد جهش‌یافته‌های متمایز در چند انتشار اخیر:** این معیار نشان می‌دهد موجودیت مورد بررسی به چه میزان سابقه‌ی تغییراتی را دارد که احتمال بروز خطا را افزایش می‌دهد. تعداد انتشارها باید به گونه‌ای باشد که کم یا زیاد نباشد. زیرا تعداد انتشارهای کم سبب می‌شود تفاوت جندانی با معیار قبلی نداشت باشد و سابقه‌ی تغییرات به اندازه‌ی کافی مد نظر قرار نگیرد. از طرف دیگر در نظر گرفتن تعداد زیادی انتشار هم هزینه‌بر است و هم به دلیل تغییرات زیاد فایل در طول توسعه‌ی نرم‌افزار اطلاعات اولیه مفید نخواهد بود. عدد در نظر گرفته شده برای تعداد انتشارها چهار می‌باشد. نحوه‌ی محاسبه به این شکل است که برای هر انتشار تعداد جهش‌یافته‌ها در انتشار جدید، نسبت به قبلی شمرده می‌شود و با یکدیگر جمع زده می‌شوند.

۳. **میزان تغییرات مثبت امتیاز جهش در چند انتشار اخیر:** تغییرات امتیاز جهش نشان از تغییرات در برنامه و آزمون‌های نرم‌افزار است. این معیار نشان می‌دهد این تغییرات به چه میزان در جهت بهبود کیفیت نرم‌افزار بوده. چراکه امتیاز بالاتر جهش نشان از کیفیت بهتر آزمون‌ها و در نتیجه نرم‌افزار است. به منظور محاسبه‌ی این معیار در هر انتشار امتیاز جهش محاسبه می‌شود و در صورتی که نسبت به انتشار قبلی تغییر مثبت بود به مجموع تغییرات افزوده می‌شود.

۴. **میزان تغییرات منفی امتیاز جهش در چند انتشار اخیر:** این معیار مشابه معیار سوم عمل می‌کند با این تفاوت که میزان تغییرات در خلاف جهت بهبود نرم‌افزار را می‌سنجد.

۳.۳ معیارهای ترکیبی جهش-فرآیند

رویکرد سوم با توجه به مطالب گفته شده در مقاله‌ی [۱۸] مطرح شده که بیان می‌کند معیارها هر چقدر هم که پویا باشند (دچار رکود نشوند، مانند معیارهای فرآیند) زمانی در پیش‌بینی خطا مفید هستند که همراه با ایجاد خطا باشند. نکته‌ی قابل توجه این است که همه‌ی تغییرات در یک فایل به یک اندازه موجب بر پیچیدگی فایل نمی‌افزایند و به عبارت دیگر موجب بروز خطا نمی‌شوند. به عنوان مثال در یک فایل به زبان جاوا ممکن است توضیح^۲ و یا مستندجاوا^۳ وجود داشته باشد که بروزرسانی یا اضافه و کم شدن آنها تاثیری بر روند اجرای برنامه

^۲Comment

^۳Javadoc

و میران پیچیدگی ندارند با این حال در محاسبه‌ی معیارهای پیش‌بینی خطا در نظر گرفته می‌شوند. هدف از ارائه‌ی معیارهای ترکیبی جهش-فرآیند^۴ بهبود کاستی‌های معیارهای فرآیند در چنین شرایطی است. در اینجا دو معیار مقدار نرمال شده‌ی خطوط اضافه شده و یا کم شده است. این دو معیار جز شاخص‌ترین معیارهای فرآیند هستند.

در نگاه اول این ایده به ذهن می‌رسد که با توجه به تعداد جهش‌یافته‌هایی که اضافه و یا حذف هر خط ایجاد می‌کند، اضافه یا کم شدن خطوط وزن دهی شود و به منظور اجرای آن از دو فرمول زیر بهره گرفت.

$$M_1 = \text{number of lines added} \times \text{number of mutants derived}$$

$$M_2 = \text{number of lines deleted} \times \text{number of mutants derived}$$

با وجود مناسب بودن ایده‌ی اولیه با بررسی‌های بیشتر دو مشکل در معیارهای فوق مشخص می‌شود. مشکل اول: هدف از ارائه‌ی این معیارها وزن دهی به خطوط اضافه و کم شده است. نکته قابل توجه این است که هر خط باید به صورت جداگانه وزن دهی شود و وزن یک خط بر وزن خط دیگر تأثیری نداشته باشد. مثال زیر را در نظر بگیرید.

```
//this method is important → 0 mutant
// this method get root of → 0 mutant
// sum of a plus b → 0 mutant
b = sqrt(a+b) → 2 mutant
```

فرض کنید ۴ خط بالا به یک فایل اضافه شده است. معیار مقدار نرمال شده‌ی خطوط اضافه شده قبل از نرمال سازی عدد چهار را نمایش می‌دهد در حالی که از این چهار خط ۳ خط توضیح است. حال معیار اولیه پیشنهادی برابر ۸ خواهد بود که بدیهی است، از هدف ارایه‌ی متریک فاصله گرفته است. حال اگر تنها جهش یافته‌های تولید شده در خطوط اضافه شده را در نظر بگیریم این مقدار می‌تواند جایگزین مناسبی باشد. در واقع نگاشتی را ارائه می‌شود که هر خط از برنامه را به یک عدد نگاشت می‌دهد. این عدد میزان پیچیدگی آن خط و یا احتمال بروز خطا را تعریف می‌کند. لازم به یادآوری است که در مقاله‌ی [۲۳] اشاره شده که جهش یافته‌ها جایگزین خوبی برای خطاهای واقعی هستند. این نگاشت برابر است با تعداد جهش یافته‌های تولید شده در آن خط.

مشکل دوم: این معیار برای عمل‌کرد هرچه بهتر مشابه معیار مقدار نرمال شده‌ی خطوط اضافه شده نیاز به نرمال سازی دارد. به جهت نرمال سازی نمی‌توان از همان روش استفاده کنیم چراکه در آن وزن دهی به خطوط

^۴Process-Mutation Hybrid

وجود ندارد و از آن مهم‌تر توضیحات را نیز در نظر می‌گیرد. از طرف دیگر این امکان وجود ندارد که برای تمام خطوط اضافه یا کم شده در کل پروژه در طول یک انتشار جهش یافته تولید شود (به دلیل زمانبر بودن و پیچیدگی‌های فراوان در پیاده‌سازی). در مقالات گذشته اشاره شده که تعداد ثبت‌ها می‌تواند نشانگر میزان تغییرات باشد. بنابراین از تعداد ثبت‌های کل پروژه در طول یک انتشار به منظور نرمال‌سازی استفاده خواهد شد. در نهایت نحوه‌ی محاسبه به این صورت خواهد بود که ابتدا ثبت‌هایی از برنامه در طول آخرین انتشار که در آن فایل مورد نظر تغییر کرده است بازیابی می‌شود. سپس برای هر ثبت تعداد جهش یافته‌های جدید نسبت به ثبت قبلی محاسبه می‌شود و برای محاسبه‌ی جهش یافته‌های حذف شده تعداد جهش یافته‌ها در ثبت قبلی را یافته و آن‌ها که جز خطوط حذف شده در ثبت بعدی است شمرده می‌شود. تعداد جهش یافته‌های اضافه و حذف شده در ثبت‌ها جمع شده و بر تعداد ثبت‌های کل پروژه در طول انتشار تقسیم می‌گردد.

[این صفحه آگاهانه خالی گذاشته شده است.]

فصل ۴

مورد مطالعاتی

در این فصل مطالعه‌ی موردی بر روی مجموعه داده‌ی defects4j [۳۰] انجام می‌گیرد. ابتدا نحوه‌ی کلی برپایی آزمایش شرح داده می‌شود و سپس چگونگی استخراج معیارها و پیاده‌سازی آزمایش توضیح داده خواهد شد.

۱.۴ طراحی آزمایش

به منظور ارزیابی رویکردهای گفته شده لازم است که برای مجموعه معیارهای هر رویکرد مدل‌های پیش‌بینی ساخته شود و هر عملکرد هر مدل نسبت به پژوهش‌های گذشته مقایسه شود. به این ترتیب ابتدا لازم است از مجموعه داده‌ی فراهم شده معیارهای بیان شده در فصل ۳ استخراج شوند. مجموعه داده‌ی defects4j که در قسمت‌های آتی معرفی می‌شود شامل اطلاعات خطا در چندین فایل است و به همین تعداد، فایل بدون خطا در ثبت و پروژه‌ی متناظر به طور تصادفی انتخاب می‌گردد. برای فایل‌های حاوی خطا و سالم، معیارها استخراج می‌شود. معیارهای استخراج شده برای هر فایل به عنوان بردار ویژگی در مدل‌های دسته‌بندی عمل می‌کند. مدل‌های دسته‌بندی به منظور پیش‌بینی حاوی خطا بودن ساخته می‌شود و عملکرد آنها مقایسه می‌گردد. مدل‌هایی که با هم مقایسه می‌شود در الگوریتم و پیکربندی^۱ یکسان هستند و تنها تفاوت آنها در معیارهای استفاده شده به منظور یادگیری است. بدین ترتیب تاثیر معیارها بر پیش‌بینی خطا سنجیده می‌شود.

۲.۴ آشنایی با ابزارها و مجموعه داده

این قسمت به معرفی ابزارهای استفاده شده در این پژوهش می‌پردازد. آشنایی با این ابزارها به درک هرچه بهتر نحوه‌ی استخراج معیارها و روند آزمایش کمک می‌کند.

۱.۲.۴ مجموعه داده defect4j

مجموعه داده‌ی انتخابی به منظور انجام مورد مطالعاتی لازم است که دارای ویژگی‌های زیر باشد:

¹Configuration

• اطلاعات خطاهای پروژه وجود داشته باشد و این اطلاعات نشان دهد که خطا متعلق به کدام فایل در کدام ثبت است.

• پروژه‌ها متن-باز باشد تا بتوان با استفاده از کد منبع آنها معیارها را استخراج نمود.

• برای پروژه‌ها موارد آزمون مناسب وجود داشته باشد تا بتوان معیارهای جهش را استخراج کرد.

در میان مجموعه‌داده‌های موجود مجموعه‌داده‌ی defects4j تنها موردی است که تمام ویژگی‌ها را دارد.

این مجموعه شامل شش پروژه می‌باشد که این پروژه‌ها متن-باز^۲ هستند و با استفاده از نرم‌افزارهای کنترل نسخه‌ی گیت و svn می‌توان به کدهای آن‌ها در طول فرآیند توسعه‌ی آنها دسترسی پیدا کرد. بجز پروژه‌ی Chart سایرین از سیستم گیت استفاده می‌کنند. همچنین این پروژه به دلیل نداشتن ساختار مناسب کنار گذاشته شد و از پرونده‌های حاوی خطای موجود در آن استفاده نشد. مجموعه‌داده‌ی defects4j به صورت یک چهارچوب^۳ ارائه شده است که کارهایی بیش از نگهداری اطلاعات درباره‌ی پروژه‌ها انجام می‌دهد. مهم‌ترین کارهایی که می‌توان به وسیله‌ی این ابزار انجام داد در جدول ۱.۴ آمده است.

جدول ۱.۴: عملیات‌های موجود در defects4j

نام عملیات	توضیح
info	نمایش پیکربندی یک پروژه‌ی خاص یا خلاصه‌ی یک خطای خاص
checkout	وارسی یک نسخه‌ی حاوی خطا یا تعمیر شده از پروژه
compile	کامپایل کدها و آزمون‌های نوشته شده توسط توسعه‌دهندگان
test	اجرای یک آزمون یا مجموعه‌ی آزمون در یک نسخه‌ی حاوی خطا یا تعمیر شده از پروژه
mutation	اجرای تحلیل جهش در یک نسخه‌ی حاوی خطا یا تعمیر شده از پروژه

این ابزار در اجرای عملیات‌های بالا دارای محدودیت است و تنها آن‌ها را بر روی ثبت‌های از پیش تعیین شده انجام می‌دهد. ثبت‌های از پیش تعیین شده شامل ثبت‌های حاوی خطا و تعمیر آن خطا می‌باشد. در جدول ۲.۴ اطلاعات مربوط به تعداد خطاهای هر پروژه آمده است.

^۲Open-source

^۳Framework

جدول ۲.۴: پروژه‌های موجود در defects4j

نام مختصر	نام کامل	تعداد خطا
Chart	JFreeChart	۲۶
Closure	Closure compiler	۱۳۳
Lang	Apache commons-lang	۶۵
Math	Apache commons-math	۱۰۶
Mockito	Mockito	۳۸
Time	Joda - Time	۲۷
-	کل پروژه‌ها	۳۹۵

به منظور نصب و راه اندازی ابزار defects4j ابتدا از صفحه‌ی گیت‌هاب^۴ آن کدهای مربوطه دریافت می‌شود. سپس باید دستوراتی را اجرا کرد تا سایر متعلقات دریافت شود. این تعلقات شامل مخزن نرم‌افزاری مربوط به شش پروژه‌ی یاد شده است که کدهای پروژه‌ها در آن قرار دارد. نکته‌ی قابل توجه در این ابزار این است که بجز دستور info سایر دستورات عملیات‌های مربوط را بر روی کامپیوتر کاربر انجام می‌دهد و خروجی را نمایش داده می‌شود، نه اینکه از یک پایگاه داده اطلاعات را صرفاً بارخوانی کند.

در نیازمندی‌های این ابزار اشاره شده که باید از جاوا نسخه‌ی ۷ استفاده شود. اما مسأله‌ای که به آن اشاره نشده توزیع کننده‌ی جاوا است. جاوا دو توزیع کننده‌ی عمده دارد. یکی OpenJDK و دیگر با Oracle استفاده از OpenJDK ابزار defects4j و ابزارهایی که به آن وابسته است به خوبی کار نمی‌کنند. به عنوان مثال برخی مجموعه آزمون‌ها که باید بدون خطا اجرا شوند به دلیل نبود وابستگی‌های^۵ لازم با شکست مواجه می‌شوند.

راه ارتباط با این ابزار خط دستور^۶ می‌باشد و یک نمونه از دستورات قابل استفاده در این ابزار در شکل ۱.۴ است که این دستور اطلاعات مربوط به پروژه‌ی Lang خطای شماره‌ی یک را خواهد داد.

```
defects4j info -p Lang -b 1
```

^۴Github

^۵Dependency

^۶Command line

```

ali@ali-GL553VD ~ $ defects4j info -p Lang -b 1
Determine revision date..... OK
Summary of configuration for Project: Lang
-----
Script dir: /home/ali/project/defects4j/framework
Base dir: /home/ali/project/defects4j
Major root: /home/ali/project/defects4j/major
Repo dir: /home/ali/project/defects4j/project_repos
-----
Project ID: Lang
Program: commons-lang
Build file: /home/ali/project/defects4j/framework/projects/Lang/Lang.build.xml
-----
Vcs: Vcs::Git
Repository: /home/ali/project/defects4j/project_repos/commons-lang.git
Commit db: /home/ali/project/defects4j/framework/projects/Lang/commit-db
Number of bugs: 65
-----
Summary for Bug: Lang-1
-----
Revision ID (fixed version):
687b2e62b7c6e81cd9d5c872b7fa9cc8fd3f1509
-----
Revision date (fixed version):
2013-07-26 01:03:52 +0000
-----
Root cause in triggering tests:
- org.apache.commons.lang3.math.NumberUtilsTest::TestLang747
--> java.lang.NumberFormatException: For input string: "80000000"
-----
List of modified sources:
- org.apache.commons.lang3.math.NumberUtils
-----

```

شکل ۱.۴: اجرای دستور info در defects4j

۲.۲.۴ ابزار Major

این ابزار جهت تولید جهش یافته و تحلیل جهش استفاده می‌شود. یک ابزار دیگر در این حوزه پیت^۷ می‌باشد اما به دلیل سازگاری ابزار defects4j با Major و نیز قابلیت‌های ویژه‌ی آن از این ابزار استفاده شد. چند مورد از ویژگی‌های مهم ابزار Major عبارتند از:

- راحتی استفاده به دلیل نیاز به دستورات کمتر نسبت به پیت
- امکان اجرای تحلیل جهش در پروژه‌هایی که از گریدل^۸ استفاده می‌کنند
- مجموعه عملگرهای کاملتر
- انعطاف در پیکربندی: امکان انجام تحلیل تنها برای یک کلاس یا تابع، تنظیمات ساده و کامل جهت مشخص کردن مجموعه عملگرها

لازم به ذکر است که این ابزار از کامپایلر مخصوص به خود جهت کامپایل برنامه و ساخت جهش یافته استفاده می‌کند که گسترش یافته‌ی یک کامپایلر جاوا است. استفاده از این ابزار را می‌توان در سه مرحله خلاصه کرد:

^۷PIT - <http://pitest.org/>

^۸Gradle

۱. **پیکربندی تولید جهش یافته به وسیله دستورات MML:** این ابزار برای مشخص نمودن اینکه از چه عملگرهایی استفاده شود و آن‌ها در چه محل‌هایی از برنامه به کار گرفته شوند یک زبان ساده ابداع کرده است به نام MML که یک کامپایلر نیز دارد. ابتدا کد MML نوشته می‌شود سپس با MMLC کامپایل می‌شود و نتیجه به عنوان یکی از پارامترها به در هنگام فراخوانی ابزار ارسال می‌شود. نمونه‌ای از این کد در شکل ۲.۴ آمده است.

```
1 targetOp{
2     // Define the replacements for ROR
3     BIN(>)->{>=,!=,FALSE};
4     BIN(<)->{<=,!=,FALSE};
5     BIN(>=)->{>==,TRUE};
6     BIN(<=)->{<==,TRUE};
7     BIN(==)->{<=,>=,FALSE,LHS,RHS};
8     BIN(!=)->{<,>,TRUE,LHS,RHS};
9     // Define the replacements for COR
10    BIN(&&)->{==,LHS,RHS,FALSE};
11    BIN(||)->{!=,LHS,RHS,TRUE};
12    // Define the type of statement that STD should delete
13    DEL(RETURN);
14
15    // Enable the STD, COR, and ROR mutation operators
16    STD;
17    COR;
18    ROR;
19 }
20 // Call the defined operator group for the target method
21 targetOp<"triangle.Triangle::classify(int,int,int)">;
```

شکل ۲.۴: نمونه کد MML در Major

۲. **تولید جهش یافته‌ها:** همانطور که اشاره شد ابزار Major جهت تولید نسخ جهش یافته نیار به کامپایلر پروژه دارد. امروزه پروژه‌های نرم‌افزاری از جمله پروژه‌های موجود در defects4j از ابزارهایی استفاده می‌کنند که فرآیند ساخت را خودکارسازی می‌کنند. فرآیند ساخت به طور کلی شامل مراحل زیر است:

- پاک سازی پوشه‌های کاری از پرونده‌های ساخت‌های قبلی
- معرفی وابستگی‌ها و کامپایل پروژه
- معرفی وابستگی‌ها و کامپایل موارد آزمون
- اجرای موارد آزمون و ارائه گزارش

سه نوع از مهم ترین ابزارهای خودکارسازی مورد استفاده در صنعت عبارتند از Ant ، Maven و Gradle. در پروژه‌های مورد مطالعه نیز این سه نوع به کار گرفته شده است. هر یک از روشهای

خودکارسازی دارای دستورات مربوط به خود می‌باشد و برای تولید جهش‌یافته باید متناسب با آنها عمل نمود که در زیر خلاصه شده است.

- Ant : این دسته از پروژه‌ها دارای یک پرونده به نام build.xml است که دستورات لازم جهت پیکربندی و انجام عملیات ساخت در آن قرار دارد. به منظور تولید جهش‌یافته کاپایلر مورد استفاده در قسمت کاپایلر پروژه را کاپایلر توسعه‌یافته‌ی Major قرار داد و پارامترهای لازم به آن ارسال شود.

- Maven : در این دسته از پروژه‌ها دستورات لازم در پرونده‌ی pom.xml قرار دارد. به وسیله‌ی یک افزونه^۹ در ابزار Maven این فایل تبدیل به یک فایل build.xml می‌شود که قابل استفاده توسط ابزار Ant است. پس از این تبدیل مشابه حالت قبل عمل می‌شود.

- Gragle : در این دسته از پروژه‌ها دستورات لازم در پرونده‌ی build.gradle قرار دارد. به منظور تولید جهش‌یافته کاپایلر مورد استفاده در قسمت کاپایلر پروژه را کاپایلر توسعه‌یافته‌ی Major قرار داد و پارامترهای لازم به آن ارسال شود.

نمونه‌ای از حاصل اجرای عملیات جهش برای یک پرونده در شکل ۳.۴ آمده است که نشان می‌دهد ۸۶ جهش یافته تولید شده است. همچنین ابزار یک پرونده به نام mutants.log تولید می‌کند که نشان می‌دهد چه جهش‌یافته‌هایی در کجا تولید شده‌اند. نمونه‌ای از محتویات این پرونده در شکل ۴.۴ آمده است.

```
Compiling and mutating project
(ant -DmutOp="$MAJOR_HOME/mml/tutorial.mml.bin" clean compile)
Buildfile: /home/ali/project/defects4j/major/example/ant/build.xml
clean:
  [delete] Deleting directory /home/ali/project/defects4j/major/example/ant/bin
init:
  [mkdir] Created dir: /home/ali/project/defects4j/major/example/ant/bin
compile:
  [javac] Compiling 1 source file to /home/ali/project/defects4j/major/example/ant/bin
  [javac] #Generated Mutants: 86 (66 ms)

BUILD SUCCESSFUL
Total time: 1 second
```

شکل ۳.۴: اجرای عملیات جهش برای یک پرونده

^۹Plugin

```
1 1:ROR:<=(int,int):<(int,int):triangle.Triangle@classify(int,int,int):11:a <= 0 |==> a < 0
2 2:ROR:<=(int,int):==(int,int):triangle.Triangle@classify(int,int,int):11:a <= 0 |==> a == 0
3 3:ROR:<=(int,int):TRUE(int,int):triangle.Triangle@classify(int,int,int):11:a <= 0 |==> true
4 4:ROR:<=(int,int):<(int,int):triangle.Triangle@classify(int,int,int):11:b <= 0 |==> b < 0
5 5:ROR:<=(int,int):==(int,int):triangle.Triangle@classify(int,int,int):11:b <= 0 |==> b == 0
```

شکل ۴.۴: نمونه‌ای از پرونده‌ی mutants.log

۳. **اجرای تحلیل جهش:** ابتدا پرونده‌های آزمون کامپایل می‌شود و سپس هر مجموعه تست بر روی جهش‌یافته‌هایی که تا کنون کشته نشده‌اند اجرا می‌شود. در پایان نتایج را در خروجی چاپ می‌کند. همچنین نتایج را در فایل‌های با پسوند CSV قرار می‌دهد. نمونه‌ای از اجرای تحلیل جهش در شکل ۵.۴ و پرونده‌ی نتایج خروجی در شکل ۶.۴ آمده است.

```
compile.tests:
[javac] Compiling 3 source files to /home/ali/project/defects4j/major/example/ant/bin

mutation.test:
[echo] Running mutation analysis ...
[junit] MAJOR: Mutation analysis enabled
[junit] MAJOR: -----
[junit] MAJOR: Run 3 ordered tests to verify independence
[junit] MAJOR: -----
[junit] MAJOR: Preprocessing time: 0.06 seconds
[junit] MAJOR: -----
[junit] MAJOR: Mutants generated: 86
[junit] MAJOR: Mutants covered: 86 (100.00%)
[junit] MAJOR: -----
[junit] MAJOR: Export test map to (testMap.csv)
[junit] MAJOR: -----
[junit] MAJOR: Run mutation analysis with 3 individual tests
[junit] MAJOR: -----
[junit] MAJOR: 1/3 - triangle.test.TestSuite (3ms / 86):
[junit] MAJOR: 312 (76 / 86 / 86) -> AVG-RTPM: 3ms
[junit] MAJOR: Mutants killed / live: 76 (76-0-0) / 10
[junit] MAJOR: -----
[junit] MAJOR: 2/3 - triangle.test.TestSuite2 (1ms / 86):
[junit] MAJOR: 545 (0 / 86 / 86) -> AVG-RTPM: 2ms
[junit] MAJOR: Mutants killed / live: 76 (152-0-0) / 10
[junit] MAJOR: -----
[junit] MAJOR: 3/3 - triangle.test.TestSuite3 (1ms / 86):
[junit] MAJOR: 737 (0 / 86 / 86) -> AVG-RTPM: 2ms
[junit] MAJOR: Mutants killed / live: 76 (228-0-0) / 10
[junit] MAJOR: -----
[junit] MAJOR: Summary:
[junit] MAJOR: -----
[junit] MAJOR: Analysis time: 0.7 seconds
[junit] MAJOR: Mutation score: 88.37% (88.37%)
[junit] MAJOR: Mutants killed / live: 76 (228-0-0) / 10
[junit] MAJOR: Mutant executions: 258
[junit] MAJOR: -----
[junit] MAJOR: Export summary of results (to summary.csv)
[junit] MAJOR: Export run-time results (to results.csv)
[junit] MAJOR: Export mutant kill details (to killed.csv)
[junit] MAJOR: Export kill map (to km.csv)!

BUILD SUCCESSFUL
Total time: 1 second
```

شکل ۵.۴: اجرای تحلیل جهش

	A	B	C	D	E	F
1	MutantsGenerated	MutantsCovered	MutantsKilled	MutantsLive	RuntimePreprocSeconds	RuntimeAnalysisSeconds
2	86	86	76	10	0.06	0.7
3						
4						

شکل ۴.۶: نتایج خروجی تحلیل جهش

۳.۲.۴ کتابخانه‌ی Jgit

این کتابخانه جهت کار با مخازن نرم افزاری از نوع گیت هستند به کار گرفته می‌شود و به زبان جاوا است. تمام عملیات‌های مهم و اساسی که در نرم‌افزار اصلی گیت وجود دارد در این کتابخانه نیز قابل انجام است. مشکلی که کار با این کتابخانه دارد نبود منابع آموزشی به اندازه‌ی کافی است. چراکه کاربران زیادی ندارد و آموزش‌های ابتدایی معمولاً نیازهای عموم کاربران را برطرف می‌کند.

۴.۲.۴ چهارچوب Hibernate

به وسیله‌ی این چارچوب می‌توان اشیاء موجود در برنامه‌ی جاوا را به داده‌های موجود در پایگاه داده تبدیل کرد. اصطلاحاً به این ابزارها *ORM*^{۱۰} می‌گویند. در ابتدا تصمیم بر این بود که داده‌های بدست آمده در فایل متنی ذخیره شوند و در هنگام نیاز آن‌ها خوانده شوند یا همه‌ی اشیاء با هر بار اجرا ساخته شوند نکات زیر سبب شد که هزینه‌ی اول کار با پایگاه داده و مزایای بلند مدت آن به سادگی استفاده از فایل متنی ترجیح داده شود.

۱. هر بار ساخت اشیاء با اجرای برنامه بسیار زمانبر است و اتلاف وقت زیادی دارد.
۲. لازم است برای اطمینان از درستی برنامه، داده‌ها در قالب جداولی به صورت چشمی کنترل شوند.
۳. فراخوانی و جستجو در پایگاه داده سریع است و کارایی بالا می‌رود.
۴. نگهداری از برنامه در دراز مدت راحت‌تر خواهد بود و خوانایی کدها بیشتر خواهد بود چرا که کار با پایگاه داده دارای اصول مشخصی است و سایرین از آن اطلاع دارند اما فایل متنی اینگونه نیست.

۳.۴ نکات پیاده‌سازی پروژه

پیاده‌سازی پروژه در زبان جاوا انجام گرفت. یکی از نکات مهم و قابل توجه در پیاده‌سازی این پروژه این است که تمام مراحل انجام کار به طور کاملاً خودکار انجام شود و در هیچ مرحله‌ای نیاز به دخالت عامل خارجی ندارد بجز پیکربندی اولیه مانند آدرس پایگاه داده. همچنین در تمام مراحل سعی شده است که تمام اصول لازم

¹⁰Object Relational Mapping

در طراحی معماری نرم‌افزار به کار گرفته شود و نیازمندی‌های کیفی پروژه نیز مد نظر قرار گیرد. این نیازمندی‌ها شامل موارد زیر است:

۱. کارایی^{۱۱}: جهت پاسخ به این نیازمندی از پایگاه داده استفاده شده است.
۲. قابلیت نگهداری: این قابلیت از سایرین بیشتر حائز اهمیت است. زیرا پروژه‌های پژوهشی خود به صورت مستقیم کاربران عمومی ندارند و از این جهت نیازمند کارایی بالا یا رابط گرافیکی کاربر پسند نیستند. استفاده آن‌ها معمولاً در گسترش آن‌ها توسط سایر محققین است که راه را ادامه خواهند داد.
 - برای پاسخ به این نیازمندی اصول مربوط به کدنویسی در فصل سوم و چهارم کتاب [۳۱] به کار گرفته شده است.
 - از الگوهای نرم‌افزاری پر کاربرد مانند /دپتور^{۱۲}، فکتوری^{۱۳} و سینگلتون^{۱۴} استفاده شده است.
 - به منظور جلوگیری از قطعه کد تکراری از وراثت و توابع عمومی^{۱۵} استفاده شده است. همین‌طور عمق وراثت از عدد ۳ بیشتر نشده است زیرا وراثت عمیق از خوانایی کد می‌کاهد و محل اشتباه خواهد بود.
۳. امنیت: از آنجا که پروژه قرار نیست به استفاده ی عموم برسد و کاربران عمل متخاصمانه‌ای انجام نخواهند داد به نوع خاصی از امنیت نسبت به انواع متداول دارد. باید روند توسعه‌ی پروژه دارای امنیت باشد. از این نظر که کدها مفقود نشوند یا در صورت اشتباه در توسعه بتوان پروژه را به حالت قبل بازگرداند. در این راستا کدهای پروژه در مخزن نرم‌افزاری از نوع گیت نگهداری شده که یک مخزن در کامپیوتر شخصی و دیگری در سایت بیت‌باکت^{۱۶} قرار دارد. مزیت این سایت نسبت به گیت‌هاب این است مخازن خصوصی را به صورت رایگان ارائه می‌دهد. در مخازن خصوصی اجازه‌ی دسترسی تنها به افراد تعیین شده از طرف مالک داده می‌شود و عموم کاربران به آن دسترسی ندارند. از ابتدای شروع پیاده‌سازی کدها در مخازن بروزرسانی شده است. نمایی از ثبت‌های مختلف پروژه در مخزن در شکل ۷.۴ آورده شده است.

¹¹Performance

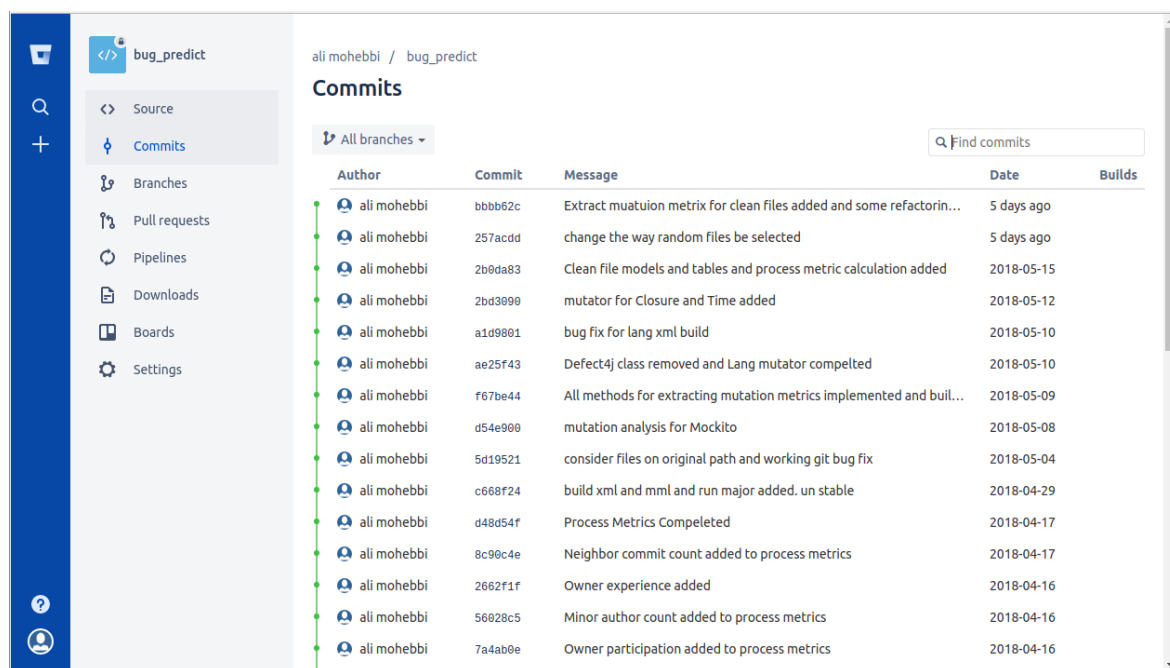
¹²Adaptor

¹³Factory

¹⁴Singelton

¹⁵Generic

¹⁶Bitbucket - https://bitbucket.org/alimohebbi/bug_predict



Author	Commit	Message	Date	Builds
ali mohebbi	bbbb62c	Extract mutauion matrix for clean files added and some refactorin...	5 days ago	
ali mohebbi	257acdd	change the way random files be selected	5 days ago	
ali mohebbi	2b0da83	Clean file models and tables and process metric calculation added	2018-05-15	
ali mohebbi	2bd3090	mutator for Closure and Time added	2018-05-12	
ali mohebbi	a1d9801	bug fix for lang xml build	2018-05-10	
ali mohebbi	ae25f43	Defect4j class removed and Lang mutator compelted	2018-05-10	
ali mohebbi	f67be44	All methods for extracting mutation metrics implemented and buil...	2018-05-09	
ali mohebbi	d54e900	mutation analysis for Mockito	2018-05-08	
ali mohebbi	5d19521	consider files on original path and working git bug fix	2018-05-04	
ali mohebbi	c668f24	build xml and mml and run major added. un stable	2018-04-29	
ali mohebbi	d48d54f	Process Metrics Completed	2018-04-17	
ali mohebbi	8c90c4e	Neighbor commit count added to process metrics	2018-04-17	
ali mohebbi	2662f1f	Owner experience added	2018-04-16	
ali mohebbi	56028c5	Minor author count added to process metrics	2018-04-16	
ali mohebbi	7a4ab0e	Owner participation added to process metrics	2018-04-16	

شکل ۷.۴: نمایی از مخزن نرم‌افزاری

۴.۴ رویکرد اول: معیارهای فرآیند در کنار جهش

در این قسمت چگونگی استخراج معیارهای رویکرد اول شرح داده می‌شود. ابتدا بازم است اطلاعات مربوط به ثبت‌های حاوی خطا از ابزار defects4j بازیابی شود و سپس این اطلاعات با استفاده از مخزن نرم‌افزاری تکمیل شود. در مراحل بعد ابتدا معیارهای فرآیند و سپس معیارهای جهش استخراج خواهند شد.

۱.۴.۴ استخراج اطلاعات مربوط به ثبت‌های حاوی خطا

اطلاعاتی که درباره‌ی ثبت‌های حاوی خطا قابل بازیابی است در زیر آمده است:

۱. شناسه‌ی ثبت در مخزن

۲. نام فایل حاوی خطا

۳. شماره‌ی خطا در ابزار defects4j

۴. شماره‌ی ثبت تعمیر خطا

۵. نام پروژه

۶. نام انتشار قبلی پروژه

۷. شماره‌ی ثبت انتشار قبلی پروژه

از میان اطلاعات بالا همگی به سادگی با استفاده از ابزار j4defect قابل استخراج است بجز دو مورد آخر. همچنین شماره‌ی ثبت تعمیر مورد استفاده قرار نگیرد ولی نگهداری شد چراکه ممکن بود لازم شود. برای بدست آوردن اطلاعات مربوط به هر انتشار لازم است که مخزن نرم افزاری هر پروژه مورد بررسی قرار گیرد. در مخازن پروژه‌های نرم‌افزاری از نوع گیت برای مشخص کردن یک رویداد مهم از تگ^{۱۷} استفاده می‌شود. هر تگ می‌تواند به یک ثبت از برنامه اشاره کند. تگ می‌تواند نمایانگر رویدادهایی چون انتشار برنامه، انتشار بتا، و یا کاندید انتشار باشد. بنابراین با استفاده از تگ می‌تواند انتشار را پیدا کرد.

تگ‌های مخازن گیت دو نوع سبک‌وزن^{۱۸} و حاشیه‌نویسی شده^{۱۹} که در میان پروژه‌های مورد مطالعه از هر دو نوع جهت مشخص کردن انتشار استفاده شده است. کار کردن با این دو نوع تگ دارای تفاوت‌هایی در پیاده‌سازی است که در اینجا از پرداختن به جزئیات صرف نظر می‌شود. ابتدا همه‌ی تگ‌های موجود در مخازن نرم‌افزاری استخراج می‌شود و در پایگاه داده قرار می‌گیرد. از میان تگ‌های استخراج شده تگ‌های نامرتب با انتشار از پایگاه داده حذف می‌شود. تگ‌های نامرتب با توجه به نام آنها مشخص می‌شود به عنوان مثال تگ‌هایی که حاوی لغات Beta یا Dev هستند نامرتب محسوب می‌شوند. در نهایت جدولی به نام ReleaseProject ساخته می‌شود که در آن اطلاعات انتشارهای مختلف وجود دارد. نمایی از این جدول در شکل ۸.۴ آمده است.

#	ReleaseId	CommitId	Project	SequenceNumbe	TagName
1	1	bd267505764488494ff13ba76ce53...	Lang	1	LANG_1_0
2	2	57be549cd8ffed876aafe0982f039d...	Lang	2	LANG_1_0_1
3	4	2caf1dd699d55338dae167333f676...	Lang	4	LANG_2_0
4	5	0aa8426b3f16d4fd0e6903d269669...	Lang	5	LANG_2_1
5	6	9eb821253181a7e075d7a3ed317f...	Lang	6	LANG_2_2

شکل ۸.۴: نمایی از جدول محتوای انتشارها

در قدم بعدی باید مشخص شود اولین انتشار ما قبل هر ثبت حاوی خطا کدام است. برای این منظور لیست

¹⁷Tag¹⁸Lightweight¹⁹Annotated

ثبت‌ها در یک پروژه به ترتیب زمانی بررسی می‌شود. اولین ثبت ماقبل ثبت مورد نظر که مربوط به یک انتشار است یافت می‌شود و به عنوان انتشار ماقبل آن ثبت در نظر گرفته می‌شود.

در نهایت جدولی به نام BugInfo تولید شده که نمایی از آن در شکل ۹.۴ آمده است. این جدول ۴۰۵ سطر دارد که بیشتر از تعداد کل خطاهای ذکر شده در مجموعه داده‌ی defects4j است. علت این است که یک خطا می‌تواند خطا در چندین پرونده به طور همزمان باشد و از آنجا که پیش‌بینی در سطح پرونده انجام می‌شود لازم است اطلاعات برای پرونده‌ها ذخیره شود.

#	ID	BUG_COMMIT_ID	BUG_NUMBER	BUGGY_CLASS_NAME	FIX_COMMIT_ID	TAG_ID	TAG_NAME	PROJECT	TAG_COMMIT
1	1	2c454a4ce3fe771098746...	1	org.apache.commons.lang3....	687b2e62b7c6e81cd9d5c8...	e1fb41239459e1f82...	LANG_3_1	Lang	b1340f422f68be7c237...
2	2	aefc12c38171e1a84a90d...	2	org.apache.commons.lang3....	09d39029b16dee61022dc...	e1fb41239459e1f82...	LANG_3_1	Lang	b1340f422f68be7c237...
3	3	1f001d06a2bde5ee4e32...	3	org.apache.commons.lang3....	2c9c8753165dc7ce5dd1d5...	e1fb41239459e1f82...	LANG_3_1	Lang	b1340f422f68be7c237...
4	4	4ddb99c5805781bd3c2...	4	org.apache.commons.lang3....	fb47b96ab635d7cc6e9edef...	e1fb41239459e1f82...	LANG_3_1	Lang	b1340f422f68be7c237...
5	5	379151bad9c5402c335d...	5	org.apache.commons.lang3....	75944e541d358d5b06ebb...	e1fb41239459e1f82...	LANG_3_1	Lang	b1340f422f68be7c237...
6	6	6823c3742ee16f5b28e5...	6	org.apache.commons.lang3....	cff0f1ae37bb2b7ab2dcd1...	e1fb41239459e1f82...	LANG_3_1	Lang	b1340f422f68be7c237...
7	7	f0c7e60bbaf975b64ab5b...	7	org.apache.commons.lang3....	e71f6dd3f2f70c640ae73d2...	e1fb41239459e1f82...	LANG_3_1	Lang	b1340f422f68be7c237...

شکل ۹.۴: نمایی از جدول محتوای اطلاعات پرونده‌های حاوی خطا

۲.۴.۴ استخراج معیارهای فرآیند

در این قسمت نحوه‌ی استخراج هر یک از معیارهای ذکر شده در قسمت ۱.۳ بیان می‌شود. **تعداد ثبت در سیستم کنترل نسخه:** اولین راه حلی که به ذهن می‌رسد استفاده‌ی مستقیم از Jgit برای این کار است. به این صورت که تعداد ثبت‌های بین ثبت کنونی و انتشار قبلی بررسی کرده و تعداد ثبت‌هایی که در آن‌ها فایل حاوی خطا تغییر کرده است شمرده شوند. مشکل این راه این است که بسیار پرهزینه خواهد بود زیرا مرتباً باید عملیات ورودی/خروجی^{۲۰} بر روی دیسک انجام پذیرد و همچنین بررسی‌های تکراری بسیاری انجام می‌گیرد. به عنوان مثال دو ثبت حاوی خطا را در نظر بگیرید که دارای انتشار ما قبل یکسانی هستند. تعدادی از بررسی‌های ثبت‌های ما بین آن‌ها تا ثبت مربوط به انتشار دارای همپوشانی خواهد بود. از طرف دیگر می‌توان اطلاعاتی که در بررسی ثبت‌ها بدست می‌آید در محاسبه‌ی معیارهای دیگر نیز مورد استفاده قرار گیرد.

همچنین برای یافتن ثبت‌های بین انتشار و ثبت مورد نظر نمی‌توان از تاریخ ثبت آنها استفاده کرد. زیرا تعداد زیادی از ثبت‌های ابتدای برخی پروژه‌های مورد مطالعه دارای تاریخ یکسانی هستند استفاده از تاریخ غیر ممکن می‌شود. علت داشتن تاریخ یکسان احتمالاً مهاجرت از یک نوع مخزن نرم‌افزاری به نوع گیت بوده است.

²⁰Input/Output (IO)

بنابراین کل ثبت‌های پروژه‌ها مورد بررسی قرار گرفت و دو جدول تولید شد. جدول اول به نام CommitInfo که اطلاعات کلی ثبت‌ها را در بر می‌گیرد و جدول دوم CommitChangedFile که اطلاعات مربوط به پرونده‌هایی که در یک ثبت از برنامه نسبت به ثبت قبلی تغییر کرده است نگهداری می‌شود. در این جدول برای هر پرونده تعداد خطوط اضافه و کم شده نسبت به ثبت قبلی ذخیره شده است. در جدول اول Sequence_Number نشان می‌دهد که چندمین نسخه از ابتدای پروژه می‌باشد و این عدد در هنگام بررسی‌ها به آن ثبت داده شده زیرا برای یافتن ثبت‌های بین ثبت کنونی و ثبت مربوط به انتشار قبلی لازم است از آنها استفاده شود. هر سطر از جدول دوم یک کلید خارجی دارد به سطری از جدول اول. قسمتی از جدول CommitInfo در شکل ۱۰.۴ و جدول CommitChangeFile در شکل ۱۱.۴ زیر آمده است:

#	ID	COMMIT_GIT_ID	COMMITTER_MAIL	COMMITTER_NAME	PROJECT	SEQUENCE_NUMBER
35	9506	1e11bf4cfb934f6bd3788a2d47089...	szczepiq@gmail.c...	Szczepan Faber	Mockito	34
36	9507	f28e6c5ebee8cb75f6ab79b7ad3...	szczepiq@gmail.c...	Szczepan Faber	Mockito	35
37	9508	01b9ebd8ab76460d8b2b59ec581a...	szczepiq@gmail.c...	Szczepan Faber	Mockito	36
38	9509	99e14ba7fd4ce6a101485d65a4949...	szczepiq@gmail.c...	Szczepan Faber	Mockito	37
39	9510	4d48778d08c11825c9c4f089c1730...	iczechowski@gm...	Igor Czechowski	Mockito	38
40	9511	8af6740c7ec9d65a2a7f68c7ca8ea...	iczechowski@gm...	Igor Czechowski	Mockito	39
41	9512	8e871ae69e946c89bccd8ee8f93fc...	szczepiq@gmail.c...	Szczepan Faber	Mockito	40

شکل ۱۰.۴: نمایی از جدول اطلاعات ثبت‌ها

#	ID	ADDED_LINES	DELETE	FILE_NAME	PATH	COMMIT_INFO_ID
433	433	2	3	org.apache.commons.lang.builder.T...	src/java/org/apache/com...	371
434	434	19	4	org.apache.commons.lang.builder.T...	src/java/org/apache/com...	372
435	435	10	5	org.apache.commons.lang.builder....	src/java/org/apache/com...	373
436	436	10	5	org.apache.commons.lang.builder....	src/java/org/apache/com...	373
437	437	14	11	org.apache.commons.lang.builder....	src/java/org/apache/com...	373
438	438	12	1	org.apache.commons.lang.WordWr...	src/java/org/apache/com...	376

شکل ۱۱.۴: نمایی از جدول تغییرات پرونده‌ها در ثبت‌ها

در نهایت با استفاده از قطعه کد ۱.۴ اطلاعات مربوط به ثبت مورد نظر و ثبت انتشار بازیابی می‌شوند و سپس از شماره‌ی دنباله‌ی آنها در پرسمان موجود در قطعه کد ؟؟ استفاده می‌شود و معیار محاسبه می‌گردد.

```
1 SELECT * from CommitInfo CI where CI.COMMIT_GIT_ID = : gitId AND CI.
PROJECT = : project
```

قطعه کد ۱.۴: بازیابی اطلاعات ثبت

```

1 SELECT count(*) from CommitChangedFile CC where CC.COMMIT_INFO_ID IN
2 (SELECT CI.ID from CommitInfo CI WHERE CI.SEQUENCE_NUMBER BETWEEN
3 :startSeq AND :endSeq AND CI.PROJECT = :project)
4 AND CC.FILE_NAME = :fileName

```

قطعه کد ۲.۴: محاسبه‌ی معیار تعداد ثبت در سیستم کنترل نسخه

تعداد توسعه‌دهندگان فعال: به منظور محاسبه‌ی این معیار تعداد آدرس ایمیل‌های ثبت‌کننده‌های ثبت‌هایی شمرده می‌شود که آن ثبت‌ها شماره‌ی دنباله‌ی آنها بین شماره‌ی دنباله‌ی ثبت پرونده‌ی مورد نظر و ثبت انتشار قبلی است و همچنین در آن ثبت پرونده‌ی مورد نظر در آن ثبت‌ها تغییر کرده است. به عبارت دیگر ثبت‌هایی که نام پرونده در جدول CommitChangeFile برای آن‌ها وجود دارد.

```

1 SELECT count(DISTINCT CI.COMMITTER_MAIL) from CommitInfo CI WHERE
2 CI.SEQUENCE_NUMBER BETWEEN :startSeq AND :endSeq AND CI.PROJECT =
3 project AND CI.ID IN
4 (SELECT CC.COMMIT_INFO_ID from CommitChangedFile CC where CC.FILE_NAME
   = :fileName)

```

قطعه کد ۳.۴: محاسبه‌ی تعداد توسعه‌دهندگان فعال

تعداد توسعه‌دهندگان متمایز: برای محاسبه‌ی معیار از پرسمان قبلی استفاده می‌شود اما اینبار به جای استفاده Sequence_Number انتشار قبلی، عدد یک قرار داده می‌شود که از ابتدای پروژه توسعه‌دهندگان شمرده شوند.

مقدار نرمال‌سازی شده‌ی تعداد خطوط اضافه شده: از پرسمان ۴.۴ جهت محاسبه‌ی مجموع تعداد خطوط اضافه شده به پرونده در طول انتشار استفاده می‌شود و از پرسمان ۵.۴ جهت محاسبه‌ی مجموع خطوط اضافه شده به پروژه استفاده می‌شود. سرانجام حاصل پرسمان اول بر دوم تقسیم می‌شود.

```

1 SELECT sum(CC.ADDED_LINES) from CommitChangedFile CC where
2 CC.COMMIT_INFO_ID IN
3 (SELECT CI.ID from CommitInfo CI WHERE CI.SEQUENCE_NUMBER BETWEEN
4 :startSeq AND :endSeq AND CI.PROJECT = :project)
5 AND CC.FILE_NAME = :fileName
6

```

قطعه کد ۴.۴: محاسبه‌ی تعداد خطوط اضافه شده به پرونده

```

1 SELECT sum(CC.ADDED_LINES) from CommitChangedFile CC where
2 CC.COMMIT_INFO_ID IN
3 (SELECT CI.ID from CommitInfo CI WHERE CI.SEQUENCE_NUMBER
4 BETWEEN :startSeq AND :endSeq AND CI.PROJECT = :project)

```

قطعه کد ۵.۴: محاسبه‌ی تعداد خطوط اضافه شده به پروژه

مقدار نرمال سازی شدهی تعداد خطوط حذف شده: به طور مشابه معیار قبلی محاسبه می گردد.

درصد خطوطی که مالک فایل مشارکت کرده: دستور Blame در Jgit نشان می دهد که هر خط از پرونده در یک ثبت در کدام یک از ثبت های گذشته اضافه شده است. با یافتن ثبت مسئول اضافه کردن آن خط نویسندهی آن خط مشخص می شود که همان ثبت کننده است. با کمک این دستور به دلایل مشابه ساخت جداول مربوط به ثبت ها، جدولی با عنوان Participation ساخته شده که در آن هر سطر نشان می دهد که یک نویسنده در یک نسخه از برنامه چند درصد از خطوط به وی اختصاص دارد. در شکل ۱۲.۴ نمایی از این جدول آورده شده است. از این جدول علاوه بر محاسبه ی این معیار برای یافت سایر معیارها نیز استفاده خواهد شد. در نهایت معیاری که در ابتدا بسیار پیچیده به نظر می رسید به کمک پرسمان ساده ی ۶.۴ محاسبه خواهد شد.

#	ID	AUTHOR_EMAIL	COMMIT_ID	FILE_NAME	FILE_PATH	PARTICIPATION_PERCENT
1	1	dirkv@apache.org	2c454a4ce3fe771098...	org.apache.commons.lang3.time.FastDateFormat	src/main/java/org/ap...	0.00169492
2	2	oheger@apache.org	2c454a4ce3fe771098...	org.apache.commons.lang3.time.FastDateFormat	src/main/java/org/ap...	0.0745763
3	3	joehni@apache.org	2c454a4ce3fe771098...	org.apache.commons.lang3.time.FastDateFormat	src/main/java/org/ap...	0.0118644
4	4	stevenecaswell@apa...	2c454a4ce3fe771098...	org.apache.commons.lang3.time.FastDateFormat	src/main/java/org/ap...	0.00508475

شکل ۱۲.۴: نمایی از مخزن نرم افزاری

```
1 SELECT max(PARTICIPATION_PERCENT) from Participation P
2 where COMMIT_ID = :commitId AND FILE_NAME = :fileName")
```

قطعه کد ۶.۴: محاسبه ی درصد خطوط مالک پرونده

تعداد مشارکت کنندگان جزئی: با استفاده از جدول Participation و پرسمان ۷.۴ معیار محاسبه می شود.

مقدار minorThreshold برابر ۵ درصد قرار می گیرد.

```
1 SELECT count(AUTHOR_EMAIL) from Participation P
2 where COMMIT_ID = :commitId AND FILE_NAME = :fileName
3 and PARTICIPATION_PERCENT < :minorThreshold
```

قطعه کد ۷.۴: محاسبه ی تعداد مشارکت کنندگان جزئی

تعداد ثبت های همسایگان: ابتدا لازم است که همسایگان پرونده در یک ثبت و نیز تعداد دفعات همسایگی در طول انتشار مشخص شود. این عمل به وسیله ی پرسمان ۸.۴ انجام می شود. سپس معیار تعداد ثبت ها در سیستم کنترل نسخه مشابه قبل با استفاده از کد ۲.۴ محاسبه می گردد و از آنها میانگین وزن دهی شده گرفته می شود.

```
1 SELECT FILE_NAME as 'name', count(ID) as 'frequency' FROM
2 CommitChangedFile WHERE COMMIT_INFO_ID IN
3 (SELECT COMMIT_INFO_ID FROM CommitChangedFile WHERE FILE_NAME = :
4   fileName)
5 AND COMMIT_INFO_ID IN
```

```

5 (SELECT CI.ID from CommitInfo CI WHERE CI.SEQUENCE_NUMBER BETWEEN :
   startSeq AND :endSeq AND PROJECT = :project)
6 AND FILE_NAME != :fileName GROUP BY FILE_NAME

```

قطعه کد ۸.۴: یافتن همسایگان و تعدد همسایگی

تعداد توسعه‌دهندگان فعال همسایگان: به طور مشابه با معیار قبلی محاسبه می‌شود.

تعداد توسعه‌دهندگان متمایز همسایگان: به طور مشابه با معیار قبلی محاسبه می‌شود.

تجربه‌ی مالک فایل: برای محاسبه‌ی معیار ابتدا با استفاده از پرسمان ۹.۴ مالک پرونده مشخص می‌شود.

سپس تعداد ثبت‌هایی که مالک پرونده از ابتدای پروژه تا آن زمان ثبت کرده است با استفاده از پرسمان ۱۰.۴

شمرده می‌شود. به ترتیب از دو جدول Participation و CommitInfo استفاده می‌شود.

```

1 SELECT AUTHOR_EMAIL FROM Participation P WHERE COMMIT_ID = :commitId
2 AND FILE_NAME = :fileName AND PARTICIPATION_PERCENT =
3 (SELECT max(PARTICIPATION_PERCENT) FROM Participation P2
4 WHERE P2.COMMIT_ID = :commitId AND P2.FILE_NAME = :fileName)

```

قطعه کد ۹.۴: یافتن مالک پرونده

```

1 SELECT count(*) from CommitInfo CI where CI.SEQUENCE_NUMBER BETWEEN
2 :startSeq AND :endSeq AND CI.PROJECT = :project AND CI.COMMITTER_MAIL =
3 :authorEmail

```

قطعه کد ۱۰.۴: شمارش تعداد ثبت‌های یک ثبت کننده در بازه‌ی زمانی داده شده

تجربه‌ی تمام مشارکت‌کنندگان: ابتدا همه‌ی توسعه‌دهندگان پرونده با استفاده از پرسمان ۱۱.۴ مشخص

می‌شوند. سپس میزان تجربه‌ی هر یک با استفاده از پرسمان ۱۰.۴ جداگانه محاسبه می‌شود و از آن‌ها میانگین

هندسی گرفته می‌شود.

```

1 SELECT AUTHOR_EMAIL FROM Participation P WHERE COMMIT_ID = :commitId
2 AND FILE_NAME = :fileName

```

قطعه کد ۱۱.۴: یافتن مشارکت‌کنندگان در پرونده

در نهایت جدولی برای معیارهای فرآیند تولید می‌شود که نمایی از آن در شکل ۱۳.۴ آورده شده است.

#	ID	ACTIVE_D	COMM	DEV_COUNT	FILE_INFO_ID	NORMAL_ADE	NORMAL_DE	FILE_1	MINOR	OWNER_PARTI	ALL_AUTHI	OWNER_EXPERIEN	NEIGHBORS_COM	NEIGHBORS_A	NEIGHBORS_TOTAL_DEV
98	98	0	0	1	98	0	0	B	0	1	536	536	0	0	0
99	99	1	3	2	99	0.0716029	0.060241	B	0	0.595238	158	525	4	1	2
100	100	1	1	2	100	0.000898...	0.0019084	B	1	0.990909	148	524	1.5	1.5	2
101	101	3	8	4	101	0.000521...	0.001285...	B	2	0.892635	717	1324	4.39672	1.93115	3.94754
102	102	6	22	11	102	0.00794188	0.0134352	B	6	0.497545	213	450	4.08507	1.85764	4.02778
103	103	4	32	4	103	0.0671348	0.0951634	B	2	0.996774	570	441	5.73	2.25	3.16

شکل ۱۳.۴: نمایی از جدول معیارهای فرآیند

۳.۴.۴ استخراج معیارهای جهش

روند کلی به این صورت است که برای هر سطر از جدول BugInfo یا CleanInfo که معادل یک پرونده در یک نسخه است ابتدا آن نسخه از برنامه در پوشه‌ی کاری قرار می‌گیرد. منظور از پوشه‌ی کاری محلی است که پرونده‌های پروژه از مخزن نرم‌افزاری فراخوانی می‌شود و در آن قرار می‌گیرد. سپس به فایل build.xml و یا build.gradle قطعه کدهایی به منظور اجرای صحیح فرآیند ساخت اضافه می‌شود.

همچنین جهت تولید جهش‌یافته و تحلیل جهش لازم است برای هر پروژه پیکربندی‌هایی انجام شود که این پیکربندی‌ها با اجرای عملیات مهندسی معکوس در ابزار Defects4j به دست آمد. به منظور انجام مهندسی معکوس کدهای ابزار که به زبان پرل^{۲۱} نوشته شده‌اند مورد بررسی قرار گرفتند و نحوه‌ی عملکرد ابزار با پروژه‌های مختلف و پیکربندی‌ها مشخص شد.

از آنجا که اجرای تحلیل جهش زمان زیادی می‌گیرد برای انجام آن یک رایانه به صورت اختصاصی برای انجام آن در آزمایشگاه کیفیت نرم‌افزار^{۲۲} واقع در دانشگاه صنعتی شریف در نظر گرفته شد. این رایانه به یک سرور لینوکس^{۲۳} تبدیل شد تا امکان نظارت و رفع خطا در استخراج معیارهای جهش همواره امکان پذیر باشد و استخراج معیارها و توسعه‌ی سایر قسمت‌های این پژوهش به صورت موازی انجام گیرد. جزییات تبدیل رایانه به سرور لینوکس در پیوست آمده است.

از آنجا که انجام تحلیل جهش بر روی موارد مطالعاتی صنعتی انجام گرفته است و پروژه‌های انتخاب شده حجم زیادی دارند لازم است تا پیکربندی‌هایی در نظر گرفته شود تا از بروز خطا و توقف محاسبات جلوگیری شود. این پیکربندی‌ها در زیر آمده است.

- افزایش فضای **PermGen**^{۲۴}: این فضا یک هیپ^{۲۵} مخصوص است که از فضای هیپ اصلی جاوا

²¹Perl

²²Software Quality Research Lab - <http://sqrllab.ce.sharif.edu/>

²³Linux

²⁴Permanant Generation

²⁵Heap

مجزا است و در آن ماشین مجازی جاوا^{۲۶} فراداده‌های^{۲۷} کلاس‌های بارگذاری شده را ردگیری می‌کند. به دلیل حجم زیاد پروژه‌های مورد مطالعه لازم است که این فضا بیشتر از حالت پیش‌فرض قرار داده شود. برای انجام این پژوهش فضای ۲ گیگابایت در نظر گرفته شده است.

- **افزایش فضای Codecache :** کدهای ترجمه شده به زبان ماشین در این فضا قرار می‌گیرد که به دلیل مشابه پیکربندی قبلی لازم است این فضا از حالت پیش‌فرض بیشتر باشد. فضای در نظر گرفته شده ۵۱۲ مگابایت می‌باشد.

- **قرار دادن زمان خروج^{۲۸} :** زمانی که یک جهش یافته از کد اصلی ساخته می‌شود ممکن است که جریان کنترلی به نحوی تغییر کند که برنامه در حلقه‌ی بی‌نهایت یا بن‌بست قرار گیرد. برای جلوگیری از چنین حالتی لازم است تا در تنظیمات ابزار JUnit مهلت زمانی در نظر گرفته شود تا در صورت قرارگیری در چنین شرایطی پس از مدت زمان معین اجرای مورد آزمون متوقف شود و مورد آزمون شکست خورده تلقی شود. مدت زمان تعیین شده جهت خروج ۱۳ ثانیه می‌باشد. **عملگرهای جهش انتخابی** با توجه به هزینه‌ی زمانی تحلیل جهش به کارگیری تمامی عملگرهای موجود در ابزار Major به صرفه نمی‌باشد. برای تولید جهش‌یافته‌ها از مجموعه عملگرهای استفاده شده در مقاله‌ی بوئر و همکاران^[۲۷] استفاده شده که مطابق عملگرهای پیش‌فرض در ابزار PIT می‌باشد. پرونده‌ی MML ساخته شده در شکل ۱۴.۴

²⁶Java Virtual Machine

²⁷Metadata

²⁸Timeout

```
1 targetOp{
2
3  BIN(+) ->{-};
4  BIN(-) ->{+};
5  BIN(*) ->{/};
6  BIN(/) ->{*};
7  BIN(%) ->{*};
8
9  BIN(>>) ->{<<};
10 BIN(<<) ->{>>};
11 BIN(>>>) ->{<<<};
12
13 BIN(&) ->{|};
14 BIN(|) ->{&};
15 BIN(^) ->{&};
16
17 UNR(+) ->{-};
18 UNR(-) ->{+};
19
20 // Use sufficient replacements for ROR
21 BIN(>) ->{>=,<=};
22 BIN(<) ->{<=,>=};
23 BIN(>=) ->{>,<};
24 BIN(<=) ->{<,>};
25 BIN(==) ->{!=};
26 BIN(!=) ->{==};
27
28 // Delete all types of supported statements
29 DEL(CALL);
30
31 // Enable all operators
32 AOR;
33 EVR;
34 LOR;
35 SOR;
36 ROR;
37 ORU;
38 STD;
39 }
```

شکل ۱۴.۴: پرونده‌ی mml ساخته شده جهت تولید جهش‌یافته‌ها

پس از انجام تحلیل جهش برای پرونده‌های حاوی خطا و سالم نتایج در جدول MutationMetrics قرار داده شد که نمایی از این جدول در شکل ۱۵.۴ آمده است.

Result Grid							
Filter Rows: <input type="text"/>							
#	MetricId	Covered	FileInfold	FileType	GeneratedCount	Killed	Lived
1	1	369	1	B	373	310	59
2	2	82	2	B	82	80	2
3	3	364	3	B	368	304	60
4	4	26	4	B	26	20	6

شکل ۱۵.۴: نمایی از جدول نتایج تحلیل جهش

۵.۴ رویکرد دوم: معیارهای فرآیند مبتنی بر جهش

همانطور که در قسمت ۲.۳ اشاره شده چهار معیار معرفی شدند و مبتنی بر جهش نامیده شدند. این قسمت به نحوه‌ی پیاده‌سازی دسته‌ی دوم از معیارها را شرح خواهد داد.

- **تعداد جهش‌یافته‌های تولید شده‌ی جدید نسبت به انتشار قبلی برنامه:** به منظور محاسبه‌ی این معیار ابتدا لازم است که مشخص شود که پرونده‌ی مورد نظر نسبت به انتشار قبلی چه تغییراتی داشته است. این کار با استفاده از ابزار JGit انجام می‌شود. JGit این امکان را فراهم می‌کند که دو پرونده در دو ثبت متفاوت مقایسه شوند و مشخص می‌کند که کدام خطوط حذف شده‌اند و کدام خطوط اضافه شده‌اند. در اینجا لازم است خطوط اضافه شده مشخص شود. سپس با استفاده از ابزار Major جهش‌یافته‌ها تولید می‌شود. در قسمت ۲.۲.۴ توضیح داده شد که پس تولید جهش‌یافته‌ها یک فایل خروجی نیز به نام mutant.log تولید می‌شود که در آن مشخص شده در هر خط از برنامه چه جهش‌یافته‌هایی تولید شده است. حال کافیست تعداد جهش‌یافته‌های تولید شده در خطوطی شمرده شوند که ابزار JGit آن‌ها را به عنوان خطوط جدید نسبت به انتشار قبلی معرفی کرده است. بدین ترتیب این معیار محاسبه خواهد شد. لازم به ذکر است روش یاد شده پایه‌ی محاسبه‌ی معیار بعدی و معیارهای رویکرد سوم است.

فصل ۵

ارزیابی

[این صفحه آگاهانه خالی گذاشته شده است.]

فصل ۶

نتیجه‌گیری و کارهای آتی

این فصل به جمع‌بندی کارهای انجام شده در پایان‌نامه و بیان نقاط قوت و کاستی‌ها به طور خلاصه اختصاص می‌یابد. در این فصل هم می‌توان از بخش‌های مختلف برای سازمان‌دهی متن بهره برد. ولی نگارش همه‌ی این فصل بدون هیچ بخشی نیز متداول است.

[این صفحه آگاهانه خالی گذاشته شده است.]

کتاب نامه

- [1] J. Nam, "Survey on software defect prediction," *Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Tech. Rep*, 2014.
- [2] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE transactions on software engineering*, vol. 33, no. 1, pp. 2–13, 2007.
- [3] E. Arisholm, L. C. Briand, and M. Fuglerud, "Data mining techniques for building fault-proneness models in telecom java software," in *Software Reliability, 2007. IS-SRE'07. The 18th IEEE International Symposium on*, IEEE, 2007, pp. 215–224.
- [4] F. Rahman, D. Posnett, A. Hindle, E. Barr, and P. Devanbu, "Bugcache for inspections: Hit or miss?" In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, ACM, 2011, pp. 322–331.
- [5] E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *Journal of Systems and Software*, vol. 83, no. 1, pp. 2–17, 2010.
- [6] D. Radjenović, M. Heričko, R. Torkar, and A. Živković, "Software fault prediction metrics: A systematic literature review," *Information and Software Technology*, vol. 55, no. 8, pp. 1397–1418, 2013.
- [7] F. Akiyama, "An example of software system debugging,," in *IFIP Congress (1)*, vol. 71, 1971, pp. 353–359.
- [8] M. H. Halstead, *Elements of software science*. Elsevier New York, 1977, vol. 7.
- [9] D. Pawade, D. J. Dave, and A. Kamath, "Exploring software complexity metric from procedure oriented to object oriented," in *Cloud System and Big Data Engineering (Confluence), 2016 6th International Conference*, IEEE, 2016, pp. 630–634.
- [10] T. J. McCabe, "A complexity measure," *IEEE Transactions on software Engineering*, no. 4, pp. 308–320, 1976.
- [11] R. Malhotra, "Comparative analysis of statistical and machine learning methods for predicting faulty modules," *Applied Soft Computing*, vol. 21, pp. 286–297, 2014.
- [12] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on software engineering*, vol. 20, no. 6, pp. 476–493, 1994.
- [13] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*, IEEE, 2005, pp. 284–292.

- [14] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *Proceedings of the 30th international conference on Software engineering*, ACM, 2008, pp. 181–190.
- [15] A. Bacchelli, M. D'Ambros, and M. Lanza, "Are popular classes more defect prone?" In *International Conference on Fundamental Approaches to Software Engineering*, Springer, 2010, pp. 59–73.
- [16] E. J. Weyuker, T. J. Ostrand, and R. M. Bell, "Do too many cooks spoil the broth? using the number of developers to enhance defect prediction models," *Empirical Software Engineering*, vol. 13, no. 5, pp. 539–559, 2008.
- [17] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Programmer-based fault prediction," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, ACM, 2010, p. 19.
- [18] F. Rahman and P. Devanbu, "How, and why, process metrics are better," in *Proceedings of the 2013 International Conference on Software Engineering*, IEEE Press, 2013, pp. 432–441.
- [19] M. Li, H. Zhang, R. Wu, and Z.-H. Zhou, "Sample-based software defect prediction with active and semi-supervised learning," *Automated Software Engineering*, vol. 19, no. 2, pp. 201–230, 2012.
- [20] S. Kim, T. Zimmermann, E. J. Whitehead Jr, and A. Zeller, "Predicting faults from cached history," in *Proceedings of the 29th international conference on Software Engineering*, IEEE Computer Society, 2007, pp. 489–498.
- [21] H. Hata, O. Mizuno, and T. Kikuno, "Bug prediction based on fine-grained module histories," in *Proceedings of the 34th International Conference on Software Engineering*, IEEE Press, 2012, pp. 200–210.
- [22] S. Kim, E. J. Whitehead Jr, and Y. Zhang, "Classifying software changes: Clean or buggy?" *IEEE Transactions on Software Engineering*, vol. 34, no. 2, pp. 181–196, 2008.
- [23] R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes, and G. Fraser, "Are mutants a valid substitute for real faults in software testing?" In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ACM, 2014, pp. 654–665.
- [24] S. Moon, Y. Kim, M. Kim, and S. Yoo, "Ask the mutants: Mutating faulty programs for fault localization," in *Software Testing, Verification and Validation (ICST), 2014 IEEE Seventh International Conference on*, IEEE, 2014, pp. 153–162.
- [25] M. Papadakis and Y. Le Traon, "Metallaxis-fl: Mutation-based fault localization," *Software Testing, Verification and Reliability*, vol. 25, no. 5-7, pp. 605–628, 2015.
- [26] D. Hao, T. Lan, H. Zhang, C. Guo, and L. Zhang, "Is this a bug or an obsolete test?" In *European Conference on Object-Oriented Programming*, Springer, 2013, pp. 602–628.
- [27] D. Bowes, T. Hall, M. Harman, Y. Jia, F. Sarro, and F. Wu, "Mutation-aware fault prediction," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, ACM, 2016, pp. 330–341.

- [28] X. Xia, E. Shihab, Y. Kamei, D. Lo, and X. Wang, “Predicting crashing releases of mobile applications,” in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ACM, 2016, p. 29.
- [29] L. Kumar, S. Rath, and A. Sureka, “An empirical analysis on effective fault prediction model developed using ensemble methods,” in *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1, Jul. 2017, pp. 244–249. DOI: [10.1109/COMPSAC.2017.53](https://doi.org/10.1109/COMPSAC.2017.53).
- [30] R. Just, D. Jalali, and M. D. Ernst, “Defects4j: A database of existing faults to enable controlled testing studies for java programs,” in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ser. ISSTA 2014, San Jose, CA, USA: ACM, 2014, pp. 437–440, ISBN: 978-1-4503-2645-2. DOI: [10.1145/2610384.2628055](https://doi.org/10.1145/2610384.2628055). [Online]. Available: <http://doi.acm.org/10.1145/2610384.2628055>.
- [31] R. C. Martin, *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2009.

[این صفحه آگاهانه خالی گذاشته شده است.]

واژه‌نامه انگلیسی به فارسی

A پیچیدگی حلقوی Cyclomatic Complexity

D

حاشیه‌نویسی شده Annotated

اشکال زدایی Debugging

درخت تصمیم Decision Tree

وابستگی Dependency

آماره‌های توصیفی Descriptive Statistics

F

جهت‌گیری Bias

شرط شاخه Branch Condition

خط‌اخیزی Bug-proneness

چارچوب Framework

G

عمومی Generic

فرض زمینه‌ای Ground Assumption

I

ورودی/خروجی Input/Output (IO)

تکرار Iteration

J

ماشین مجازی جاوا Java Virtual Machine

C

دسته‌بندی تغییر Change Classification

همبستگی Cohesion

خط دستور Command line

توضیح Comment

ثبت Commit

قطعه Component

پیکربندی Configuration

ماتریس درهم‌ریختگی Confusion Matrix

جریان کنترلی Control Flow

زوجیت Coupling

ارزیابی میان دسته‌ای Cross-validation

Systematic Review بررسی قاعده‌مند Javadoc مستند جاوا

T

Timeout زمان خروج

V

Version Control System .. سیستم کنترل نسخه

L

Lightweight سبک‌وزن

M

Metadata فراداده

Mutant جهش‌یافته

Mutation Score امتیاز جهش

O

Obsolete منسوخ

Open-source متن-باز

P

Package بسته

Performance کارایی

Plugin افزونه

Popularity شهرت

Product Metrics معیارهای محصول

Proxy Measurement اندازه‌گیری وکالتی

R

Refactoring بازآرایی کد

Release انتشار

S

Semi-Supervised نیمه-نظارتی

Suspiciousness مشکوک بودن

واژه‌نامه فارسی به انگلیسی

ت

۱

Iteration	تکرار	Descriptive Statistics	آماره‌های توصیفی
Comment	توضیح	Cross-validation	ارزیابی میان دسته‌ای
		Debugging	اشکال زدایی
		Plugin	افزونه
		Mutation Score	امتیاز جهش
		Release	انتشار
Commit	ثبت	Proxy Measurement	اندازه‌گیری وکالتی

ب

ج

		Refactoring	بازآرایی کد
Control Flow	جریان کنترلی	Systematic Review	بررسی قاعده‌مند
Bias	جهت‌گیری	Package	بسته
Mutant	جهش یافته		

پ

چ

		Cyclomatic Complexity	پیچیدگی حلقوی
Framework	چارچوب	Configuration	پیکربندی

ح

ش

Branch Condition شرط شاخه Annotated حاشیه‌نویسی شده
Popularity شهرت

خ

ع

Generic عمومی Command line خط دستور
Bug-proneness خط‌اخیزی

ف

د

Metadata فراداده
Ground Assumption فرض زمینه‌ای
Decision Tree درخت تصمیم
Change Classification دسته‌بندی تغییر

ق

Component قطعه

ز

Timeout زمان خروج
Coupling زوجیت

ی

Performance کارایی

س

م

Lightweight سبک‌وزن
Confusion Matrix ماتریس درهم‌ریختگی
Version Control System سیستم کنترل نسخه
Java Virtual Machine ماشین مجازی جاوا

متن-باز Open-source
مستندجاوا Javadoc
مشکوک بودن Suspiciousness
معیارهای محصول Product Metrics
منسوخ Obsolete

ن

نیمه-نظارتی Semi-Supervised

و

وابستگی Dependency
ورودی/خروجی Input/Output (IO)

ه

همبستگی Cohesion

[این صفحه آگاهانه خالی گذاشته شده است.]

Title of thesis

Abstract

The abstract of thesis in English language should be written after completing this document. The abstract is consisted of 300 words (or less) and is followed with 4 to 7 keywords. The keywords are written (in both Persian and English) within the main file and the abstract itself, based on its language, is written in two distinct files within the general folder.

Keywords: First Key Word, Second Key Word, Final Key Word.

[این صفحه آگاهانه خالی گذاشته شده است.]



Sharif University of Technology
Computer Engineering Department

A thesis submitted in partial fulfillment of the requirements
for the M.Sc. degree
Your Major in English Language

Title of thesis

By:

Behnam Momeni

Supervisor:

Dr. <name of your supervisor prof.>

August 2017

