



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: .....		



## اظهارنامه (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
۳۳	۳.۲.۴	کتابخانه‌ی Jgit

۳۴	..... Hibernate چهارچوب	۴.۲.۴
۳۵	ارزیابی	۵
۳۷	نتیجه‌گیری و کارهای آتی	۶
۳۹	کتاب‌نامه	
۴۲	واژه‌نامه انگلیسی به فارسی	
۴۵	واژه‌نامه 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	نمونه کد MML در Major	۳.۴
۳۲	نمونه کد MML در Major	۴.۴
۳۳	نمونه کد MML در Major	۵.۴
۳۳	نمونه کد MML در Major	۶.۴

[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<sup>۱</sup> مانند گیت<sup>۲</sup>، سیستم‌های ردگیری مشکلات مانند جیرا و آرشیو ایمیل‌ها است. هر یک از این داده‌ها بر اساس درشت‌دانی پیش‌بینی می‌توانند نمایانگر یک سیستم، یک قطعه‌ی<sup>۳</sup> نرم‌افزاری، بسته<sup>۴</sup>، 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/دار 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<sup>۵</sup> 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.

---

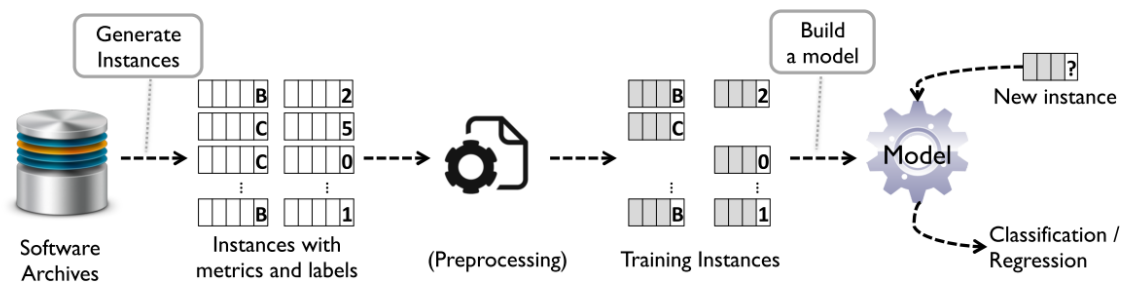
<sup>1</sup>Version Control System

<sup>2</sup>Git

<sup>3</sup>Component

<sup>4</sup>Package

<sup>5</sup>Bug-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<sup>۶</sup> 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

<sup>۶</sup>Confusion Matrix

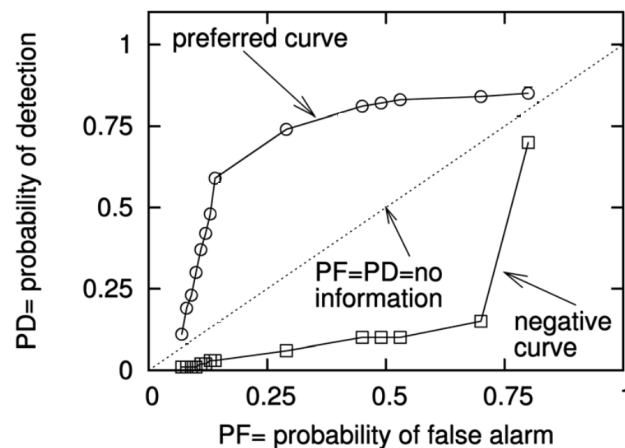
جدول ۱.۲: فرمول‌های محاسبه‌ی معیارهای ارزیابی

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  $AUC$ <sup>۷</sup> و  $AUCEC$ <sup>۸</sup> که هر دو به مساحت زیر یک منحنی اشاره می‌کنند.  $AUC$  مساحت زیر نمودار  $ROC$ <sup>۹</sup> 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.

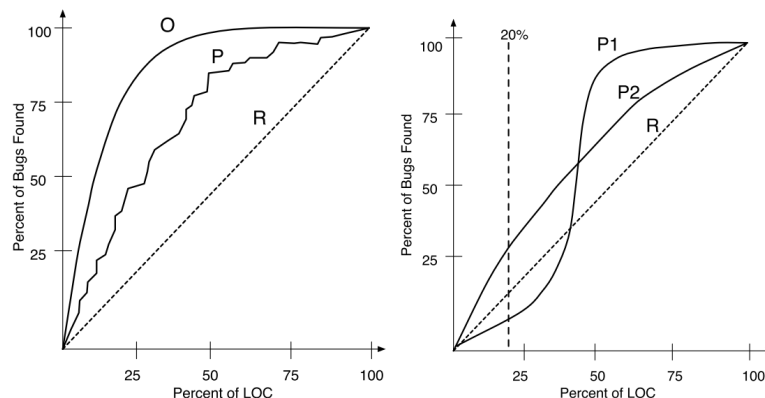
معیار  $AUCEC$  معیاری است که تعداد خطوطی از برنامه که توسط تیم تضمین کیفیت و یا توسعه دهندگان نیاز است بررسی و آزموده شود را در نظر می‌گیرد. ایده‌ی موثر بودن از نظر هزینه<sup>۱۰</sup> برای مدل‌های خطا برای اولین بار توسط آریشلم و همکاران [۳] ارائه گردید. موثر بودن از نظر هزینه به این معنا است که چه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

<sup>۷</sup>Area under curve<sup>۸</sup>Area under cost-effectiveness curve<sup>۹</sup>Receiver operating characteristic<sup>۱۰</sup>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<sup>۱۱</sup> 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<sup>۱۲</sup> میان تعداد خطاهای پیش‌بینی شده و خطاهای واقعی محاسبه می‌شوند. نماینده‌ی این معیارها را می‌توان همبستگی اسپیرمن، پیرسون و  $R^2$  دانست [۱].

<sup>۱۱</sup>Practical

<sup>۱۲</sup>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*<sup>۱۳</sup> نیز شناخته می‌شوند و میزان پیچیدگی کد را می‌سنجند. فرض زمینه‌ی<sup>۱۴</sup> 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*<sup>۱۵</sup> [۷] ارائه شد. *هالستد*<sup>۱۶</sup> [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<sup>۱۷</sup> معیارهای پیچیدگی حلقوی<sup>۱۸</sup> را پیشنهاد داد که این معیار با استفاده از تعداد گره‌ها، یالها و قطعات متصل در گراف جریان کنترلی<sup>۱۹</sup> 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

<sup>13</sup>Product Metrics

<sup>14</sup>Ground Assumption

<sup>15</sup>Akiyama

<sup>16</sup>Halstead

<sup>17</sup>McCabe

<sup>18</sup>Cyclomatic Complexity

<sup>19</sup>Control Flow

تا فرآیند توسعه بهبود یابد. نماینده‌ی معیارهای شیء‌گرایی چد/مبر و کمرر<sup>۲۰</sup> می‌باشند [۱۲]. این معیارها با توجه به خصیصه‌های زبان‌های شیء‌گرا مانند وراثت، زوجیت<sup>۲۱</sup>، همبستگی<sup>۲۲</sup> 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<sup>۲۳</sup> 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<sup>۲۴</sup> و یا تعداد نویسندگان یک فایل را می‌شمارند. موزر<sup>۲۵</sup> 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<sup>۲۶</sup> می‌شوند در نظر گرفته می‌شود [۱۴].
- **معیار شهرت:** بکچلی<sup>۲۷</sup> و همکاران معیارهای شهرت<sup>۲۸</sup> 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.

<sup>20</sup>Chidamber and Kemerer (CK)

<sup>21</sup>Coupling

<sup>22</sup>Cohesion

<sup>23</sup>Nagappan and Ball

<sup>24</sup>Refactoring

<sup>25</sup>Moser

<sup>26</sup>Commit

<sup>27</sup>Bacchelli

<sup>28</sup>Popularity

راجنویک<sup>۲۹</sup> و همکاران در پژوهش خود به بررسی قاعده‌مند<sup>۳۰</sup> 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<sup>۳۱</sup> 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<sup>۳۲</sup> 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<sup>۳۳</sup> 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

<sup>29</sup>Radjenovic<sup>30</sup>Systematic Review<sup>31</sup>Iteration<sup>32</sup>Ostrand<sup>33</sup>Rahman 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<sup>۳۴</sup> 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<sup>۳۵</sup> 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<sup>۳۶</sup> برای ساخت مدل‌های پیش‌بینی خطای کاراتر به کار گرفته شده است [۱۹]. علاوه بر مدل‌های یادگیری ماشین، مدل‌های غیر آماری مانند باگ‌کش<sup>۳۷</sup> 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<sup>۳۸</sup> 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

<sup>34</sup>Release

<sup>35</sup>Bias

<sup>36</sup>Semi-Supervised

<sup>37</sup>BugCache

<sup>38</sup>Kim

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<sup>۳۹</sup> 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<sup>۴۰</sup> 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<sup>۴۱</sup> بهتر از سایرین عمل کرده است.

## ۱.۴.۱.۲ 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<sup>۴۲</sup> 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<sup>۴۳</sup> 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

<sup>39</sup> Malhotra

<sup>40</sup> Descriptive Statistics

<sup>41</sup> Decision Tree

<sup>42</sup> Hata

<sup>43</sup> Change Classification

و پژوهشگران به دنبال مقایسه‌ی روش‌های مختلف آزمون و *اشکال زدایی*<sup>۴۴</sup> 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*<sup>۴۵</sup> استفاده شود. یکی از اندازه‌گیری‌های شناخته شده *امتیاز جهش*<sup>۴۶</sup> 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*<sup>۴۷</sup> 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<sup>۴۸</sup> 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

---

<sup>44</sup>Debugging

<sup>45</sup>Proxy Measurement

<sup>46</sup>Mutation Score

<sup>47</sup>Mutant

<sup>48</sup>Branch Condition

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

شکل ۴.۲: نمونه‌ای از جهش‌یافته‌های یک برنامه [۲۴]

جاست<sup>۴۹</sup> 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

<sup>49</sup>Just

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<sup>۵۰</sup> 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<sup>۵۱</sup> 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<sup>۵۲</sup> 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.

<sup>50</sup>Moon<sup>51</sup>Suspiciousness<sup>52</sup>Papadakis and Traon



## ۲.۲.۲ مدل‌های یادگیری و جهش‌یافته‌ها

ها/و<sup>۵۳</sup> 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<sup>۵۴</sup> 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<sup>۵۵</sup> 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<sup>۵۶</sup> 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.

<sup>53</sup>Hao

<sup>54</sup>Obsolete

<sup>55</sup>Cross-validation

<sup>56</sup>Bowes

## 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 متفاوت
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<sup>۱</sup> 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

<sup>۱</sup>Process 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<sup>۲</sup> و یا مستندجاوا<sup>۳</sup> وجود داشته باشد که بروزرسانی یا اضافه و کم شدن آنها تاثیری بر روند اجرای برنامه

<sup>۲</sup>Comment

<sup>۳</sup>Javadoc

و میران پیچیدگی ندارند با این حال در محاسبه‌ی معیارهای پیش‌بینی خطا در نظر گرفته می‌شوند. هدف از ارائه‌ی معیارهای ترکیبی جهش-فرآیند<sup>۴</sup> 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

<sup>۴</sup>Process-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<sup>۱</sup> 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:

---

<sup>1</sup>Configuration

• 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 مجموعه داده‌های موجود مجموعه داده‌ی defects4j تنها موردی است که تمام ویژگی‌ها را دارد.

این مجموعه شامل شش پروژه می‌باشد که پنج مورد از آن‌ها مربوط به شرکت آپاچی<sup>۲</sup> است و دیگری نرم‌افزار jfreechart است که به شرکت خاصی تعلق ندارد. این پروژه‌ها متن-باز<sup>۳</sup> هستند و با استفاده از نرم‌افزارهای کنترل نسخه‌ی گیت و svn می‌توان به کدهای آن‌ها در طول فرآیند توسعه‌ی آنها دسترسی پیدا کرد. مجموعه داده‌ی defect4j به صورت یک چهارچوب<sup>۴</sup> ارائه شده است که کارهایی بیش از نگهداری اطلاعات درباره‌ی پروژه‌ها انجام می‌دهد. مهم‌ترین کارهایی که می‌توان به وسیله‌ی این ابزار انجام داده عبارت است در جدول زیر آمده است.

جدول ۱۰۴: 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 کدهای 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 زیر اطلاعات مربوط به تعداد خطاهای هر پروژه آمده است.

<sup>۲</sup>Apache

<sup>۳</sup>Open-source

<sup>۴</sup>Framework

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

به منظور نصب و راه اندازی این ابزار ابتدا از صفحه ی github آن پروژه بر روی PC ، کلون می شود. سپس یک script باید اجرا کرد تا سایر تعلقات پروژه دانلود شود. این تعلقات شامل مخزن نرم افزاری مربوط به این شش پروژه است که کدهای پروژه ها در آن قرار دارد. نکته ی قابل توجه در این پروژه این است بجز دستور info سایر دستورات عملیاتهای مربوط را بر روی کامپیوتر کاربر انجام می دهند و خروجی را نمایش می دهند نه اینکه از یک پایگاه داده اطلاعات صرفاً بارخوانی شوند. در نیازمندی های این ابزار اشاره شده که باید از جاوا نسخه ی ۷ استفاده شود. اما مسأله ای که به آن اشاره نشده توزیع کننده ی جاوا است. جاوا دو توزیع کننده ی عمده دارد. یکی OpenJDK و دیگر Oracle. در ابتدا از نسخه ی openJDK استفاده شد زیرا نصب آن در linux ساده تر است و همچنین oracle به ایرانیان اجازه ی دانلود نمی دهد. اما با openJDK ابزار defects4j و ابزارهایی که به آن وابسته است به خوبی کار نمی کنند. به عنوان مثال برخی مجموعه تست ها که باید بدون خطا اجرا شوند به دلیل نبود dependency های لازم با شکست مواجه می شوند. این مسأله موجب سردرگمی و صرف مدت زمانی تا حل آن شد. راه ارتباط با این ابزار line command می باشد و نمونه ای از دستورات قابل استفاده در این ابزار در شکل ۱.۴ است که این دستور اطلاعات مربوط به پروژه ی Lang خطای شماره ی یک را خواهد داد.

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

```

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 می شود. در ابتدا تصمیم بر این بود از ابزار دیگری به نام PIT استفاده شود ولی ابزار defect4j از major استفاده می کند بنابراین به دلیل سازگاری با JD و نیز قابلیت های منحصر به فرد این ابزار تصمیم به استفاده از Major گرفته شد. چند مورد از ویژگی ها عبارتند از:

- راحتی استفاده: قابلیت تعامل از روشهای مختلف مانند command line، بکارگیری به وسیله ی ابزار ant build و دستورات کم نسبت به PIT

- مجموعه عملگرهای کاملتر

- راحتی در پیکربندی: امکان انجام تحلیل تنها برای یک کلاس یا تابع، تنظیمات راحت و کامل جهت مشخص کردن مجموعه عملگرها

لازم به ذکر است که این ابزار از compiler مخصوص به خود جهت کامپایل برنامه و ساخت جهش یافته استفاده می کند که گسترش یافته ی یک کامپایلر جاوا است. البته در انتها مشخص شد که این برنامه کاستی هایی هم دارد از جمله کامل نبود مستندات برای بکارگیری پیشرفته و خطاهایی که در قسمت مربوطه به آنها اشاره خواهد شد. استفاده از این ابزار را می توان در سه مرحله خلاصه کرد:



۱. پیکربندی تولید جهش یافته به وسیله script: MML این ابزار برای مشخص نمودن اینکه از چه عملگرهای 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 mmle 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: برای این منظور می‌توان از line command یا فایل build.xml مربوط به پروژه استفاده کرد که روش دوم مورد استفاده قرار گرفت. برای این منظور لازم است که یک قطعه کد به شکل زیر به فایل build.xml اضافه شود. و سپس برنامه کامپایل شود. حاصل به شکل زیر خواهد بود که نشان می‌دهد ۸۶ جهش یافته تولید شده است. همچنین ابزار یک فایل به نام mutation.log تولید می‌کند که نشان می‌دهد چه جهش یافته‌هایی در کجا تولید شده‌اند.

```

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: نمونه کد MML در Major

```

mutants.log (-/project/defects4j/major/example/ant)
File Edit View Search Tools Documents Help
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: نمونه کد MML در Major

۳. اجرای تحلیل جهش : در این قسمت نیز قطعه کدی به فایل build.xml اضافه می شود و سپس فایل اجرا می شود. در اجرا ابتدا فایل های تست کامپایل می شود و سپس هر مجموعه تست بر روی جهش یافته هایی که تا کنون کشته نشده اند اجرا می شود. در پایان نتایج را در خروجی چاپ می کند. همچنین نتایج را در فایل های با پسوند csv قرار می دهد. نمونه ای از اجرای تحلیل جهش و فایل های خروجی در زیر آمده است:

```

ali@ali-GL553VD ~/project/defects4j/major/example/ant
File Edit View Search Terminal Help

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
ali@ali-GL553VD ~/project/defects4j/major/example/ant $

```

شکل ۵.۴: نمونه کد MML در Major

The screenshot shows the Microsoft Excel application window. The top ribbon includes menus for File, Edit, View, Insert, Format, Sheet, Data, Tools, Window, and Help. Below the ribbon is a toolbar with icons for various functions like saving, printing, undo, redo, and formatting. The formula bar at the top displays "B6" as the active cell, followed by function symbols (fx, Σ, =) and a dropdown menu. The spreadsheet grid below has columns labeled A through F and rows numbered 1 through 4. Red dashed lines are drawn across the first row of data (row 1). The data in row 1 is as follows:

	A	B	C	D	E	F
1	MutantsGenerated	MutantsCovered	MutantsKilled	MutantsLive	RuntimePreprocSeconds	RuntimeAnalysisSeconds
2	86	86	76	10	0.06	0.
3						
4						

شکل ۶.۴: نمونه کد MML در Major

### ۳.۲.۴ کتابخانه‌ی Jgit

این کتابخانه جهت کار با مخازن نرم افزاری از نوع git به کار گرفته می شود و به زبان جاوا است. تمام عملیات های مهم و اساسی که در نرم افزار اصلی git وجود دارد در این کتابخانه نیز قابل انجام است. مشکلی که کار با

این کتابخانه دارد نبود منابع آموزشی به اندازه ی کافی است. چراکه کاربران زیادی ندارد و آموزش های ابتدایی معمولاً نیازهایشان را بر طرف می کند.

## ۴.۲.۴ چهارچوب Hibernate

به وسیله ی این چارچوب می توان اشیاء موجود در برنامه ی جاوا را به داده های موجود در پایگاه داده تبدیل کرد. اصطلاحاً به این ابزار ها (object ORM) relational (object mapping) می گویند. در ابتدا تصمیم بر این بود که داده های بدست آمده در فایل متنی ذخیره شوند و در هنگام نیاز آن ها خوانده شوند یا همه ی اشیاء با هر بار اجرا ساخته شوند نکات زیر سبب شد که هزینه ی 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، داده ها در قالب جداولی به صورت چشمی کنترل شوند ۳- فراخوانی و جستجو در پایگاه داده سریع است و کارایی بالا می رود ۴- نگهداری از برنامه در دراز مدت راحت تر 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

## 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>.

[این صفحه آگاهانه خالی گذاشته شده است.]

# واژه‌نامه انگلیسی به فارسی

**B**      آماره‌های توصیفی ..... Descriptive Statistics

**F**      جهت‌گیری ..... Bias  
شرط شاخه ..... Branch Condition  
چارچوب ..... Framework

**G**      خط‌اخیزی ..... Bug-proneness  
فرض زمینه‌ای ..... Ground Assumption

**I**      دسته‌بندی تغییر ..... Change Classification  
همبستگی ..... Cohesion  
تکرار ..... Iteration

**J**      توضیح ..... Comment  
ثبت ..... Commit  
مستند جاوا ..... Javadoc

**M**      قطعه ..... Component  
پیکربندی ..... Configuration  
ماتریس درهم‌ریختگی ..... Confusion Matrix  
جهش یافته ..... Mutant  
امتیاز جهش ..... Mutation Score  
جریان کنترلی ..... Control Flow

**O**      زوجیت ..... Coupling  
ارزیابی میان دسته‌ای ..... Cross-validation  
پیش‌چیدگی حلقوی ..... Cyclomatic Complexity  
منسوخ ..... Obsolete  
متن-باز ..... Open-source

**P**      اشکال زدایی ..... Debugging  
درخت تصمیم ..... Decision Tree  
بسته ..... Package  
شهرت ..... Popularity

Product Metrics ..... معیارهای محصول

Proxy Measurement ..... اندازه‌گیری وکالتی

## **R**

Refactoring ..... بازآرایی کد

Release ..... انتشار

## **S**

Semi-Supervised ..... نیمه-نظارتی

Suspiciousness ..... مشکوک بودن

Systematic Review ..... بررسی قاعده‌مند

## **V**

Version Control System .. سیستم کنترل نسخه

# واژه‌نامه فارسی به انگلیسی

## ت

۱

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

## 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

## خ

## ق

Component ..... قطعه      Bug-proneness ..... خطاخیزی

## د

## م

Confusion Matrix ..... ماتریس درهم‌ریختگی      Decision Tree ..... درخت تصمیم

Open-source ..... متن-باز      Change Classification ..... دسته‌بندی تغییر

Javadoc ..... مستندجاوا

Suspiciousness ..... مشکوک بودن

Product Metrics ..... معیارهای محصول

## ز

Obsolete ..... منسوخ

Coupling ..... زوجیت

## ن

## س

Semi-Supervised ..... نیمه-نظارتی

Version Control System .. سیستم کنترل نسخه

## ه

## ش

Cohesion ..... همبستگی

Branch Condition ..... شرط شاخه

Popularity ..... شهرت

## ف

Ground Assumption ..... فرض زمینه‌ای



# **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

