دانشگاه صنعتی شریف

دانشکده‌ی مهندسی کامپیوتر

پایان‌نامه به عنوان تحقق بخشی از شرایط دریافت درجه‌ی کارشناسی ارشد
گرایش مهندسی نرم‌افزار

مدل پیش بینی خطا مبتنی بر معیارهای جهش

نگارش

علی محبی

استاد راهنما

دکتر سید حسن میریان حسین آبادی

شهریور ۱۳۹۷

بسم الله الرّحمن الرّحيم

تصویب‌نامه

به نام خدا
دانشگاه صنعتی شریف
دانشکده‌ی مهندسی کامپیوتر

پایان‌نامه‌ی کارشناسی ارشد

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

کمیته‌ی ممتحنین:

استاد راهنما: دکتر سید حسن میریان حسین آبادی **امضاء**.....
استاد مدعو: دکتر <نام استاد مدعو ۱> **امضاء**.....
استاد مدعو: دکتر <نام استاد مدعو ۲> **امضاء**.....
تاریخ: .....

اظهارنامه

(اصالت متن و محتوای رساله‌ی دکتری)

عنوان رساله: _____

_____

نام استاد راهنما: _____ نام استاد راهنمای هم‌کار: _____ نام استاد مشاور: _____

این‌جانب _____ اظهار می‌دارم:

۱. متن و نتایج علمی ارایه‌شده در این رساله اصیل بوده و منحصراً توسط این‌جانب 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 این‌جانب 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: _____

تاریخ:

امضاء:

نام دانشجو: _____

تاریخ:

امضاء:

تقدیم به پدر و مادر مهربانم که همواره پشتیان و مایه‌ی دلگرمی در تمام مراحل زندگی بوده‌اند.

قدردانی

از زحمات استاد فرهیخته دکتر سیدحسن میریان حسین‌آبادی که راهنما و راهگشای اینجانب در انجام این پایان‌نامه بوده‌اند، بدین وسیله تقدیر و تشکر می‌نمایم. همچنین از زحمات دوست گرانقدرم مهران ریواده که با راهنمایی خویش مرا یاری نمودند، تشکر می‌کنم. لازم است در اینجا از زحمات دوست عزیزم خشایار اعتمادی که در همفکری و کمک به من نقش مهمی داشته‌اند، قدردانی به عمل آورم.

مدل پیش بینی خطا مبتنی بر معیارهای جهش

چکیده

توسعه‌دهندگان نرم‌افزار از طریق گزارش خطا در سیستم‌های ردگیری خطا و یا شکست در آزمون نرم‌افزار متوجه حضور خطا می‌شوند و پس از آن به جستجوی محل خطا و درک مشکل نرم‌افزار می‌پردازند. کشف زود هنگام خطاها موجب صرفه‌جویی در زمان و هزینه می‌شود و فرآیند اشکال زدایی را تسهیل می‌بخشد. ابزارهای آماری نوین امکان ساخت و بهره‌برداری از مدل‌های پیش‌بینی را فراهم می‌سازند. اصلی‌ترین جزء مدل‌های پیش‌بینی، معیارهای نرم‌افزار می‌باشد و با به کارگیری معیارهای نوین و موثر می‌توان به مدل‌های کاراتر دست پیدا کرد. در این پژوهش از معیارهای فرآیند و معیارهایی که بر اساس تحلیل جهش ساخته شده‌اند استفاده شده عملکرد مدل‌های حاصل ارزیابی شده‌اند. علاوه بر بکارگیری معیارهای جهش در کنار معیارهای فرآیند دو دسته معیار جدید به نام‌های معیارهای *فرآیند مبتنی بر جهش* و معیارهای *ترکیبی جهش- فرآیند* نیز جهت به کارگیری در ساخت مدل‌های پیش‌بینی معرفی شده‌اند. نتایج ارزیابی نشان می‌دهد معیارهای جهش می‌تواند به قدرت پیش‌بینی معیارهای فرآیند بیافزاید. معیارهای فرآیند مبتنی بر جهش علارغم داشتن قدرت پیش‌بینی بهتر از معیارهای جهش عمل نمی‌کنند. همچنین معیارهای ترکیبی جهش- فرآیند بهبود قابل توجهی را در عملکرد مدل‌های پیش بینی ایجاد می‌کنند.

کلیدواژه‌ها: پیش‌بینی خطا، آزمون نرم‌افزار، معیارهای جهش، معیارهای فرآیند .

سرخط‌ها

[1] Dynamic

فهرست جدول‌ها

فهرست شکل‌ها

فصل ۱
سرآغاز

سامانه‌های نرم‌افزاری بسیار فراگیر شده‌اند و زندگی امروزی را ارتقا داده‌اند. در نتیجه کاربران کیفیت نرم‌افزار بالایی را تقاضا می‌کنند. کشف و برطرف کردن خطاها پرهزینه است و مدل‌های پیش‌بینی خطا از طریق اولویت‌دهی به فعالیت‌های تضمین کیفیت موجب افزایش بازدهی می‌گردند. پیش‌بینی خطا از سال ۱۹۹۲ تا کنون یک زمینه‌ی فعال تحقیقاتی بوده است. محققان همواره به دنبال روش‌هایی بوده‌اند که پیش‌بینی خطا را با کیفیت بهتری انجام دهند و یا دامنه‌ی کاربرد آن را گسترش بخشند.

به منظور افزایش کارایی پیش‌بینی‌خطا محققان معیارهای نوینی را ارائه داده‌اند[۱]، سعی داشته‌اند محدودیت‌های یادگیری ماشین را تقلیل بخشند[۲] و یا روش‌های بروزتری را به منظور *دسته‌بندی*[1] به کار گیرند[۳].

۱.۱ تعاریف مقدماتی

در این قسمت چند اصطلاح رایج در مبحث پیش‌بینی خطا و مورد استفاده در پایانامه نوشته شده است.

- *مورد آزمون*[2]:

یک مورد آزمون متشکل است از مقادیر ورودی‌های آزمون، نتایج مورد انتظار که با اجرای برنامه تحت آزمون یک یا چند عملکرد آنرا ارزیابی می‌کند.

- *سامانه‌ی کنترل نسخه*[3]:

این سامانه تغییرات اعمال شده بر روی یک یا چندین *پرونده*[4] را ذخیره می‌کند تا در آینده بتوان یک نسخه‌ی خاص را بازخوانی کرد.

[1] Classification
[2] Test Case
[3] Version Control System
[4] File

- *ثبت*[۵]:

ذخیره‌ی تغییرات ایجاد شده بر روی پرونده‌ها در سامانه‌ی کنترل نسخه را ثبت می‌نامند. یک ثبت را می‌تواند معادل یک نسخه از برنامه در نظر گرفت که البته این نسخه می‌تواند ناکامل باشد.

- *انتشار*[۶]:

انتشار به معنی توزیع نسخه‌ی نهایی یک نرم‌افزار است که قابل استفاده برای کاربر می‌باشد. یک انتشار ممکن است نسخه‌ای از یک برنامه‌ی جدید باشد و یا ارتقاء یافته‌ی نرم‌افزار موجود باشد. قبل از یک انتشار معمولا به ترتیب نسخه‌های *آلفا* [۷] و *بتا*[۸] توزیع می‌شود.

- *ماتریس درهم‌ریختگی*[۹]:

در زمینه‌ی یادگیری ماشین، به خصوص مسئله‌ی دسته‌بندی،یک ماتریس درهم‌ریختگی یک جدول است که اجازه می‌دهد عملکرد یک الگوریتم تصویرسازی گردد. هر سطر از ماتریس نشان دهنده‌ی نمونه‌هایی است که پیش‌بینی شده‌اند در حالی که هر ستون نمونه‌ها در کلاسهای واقعی را نشان می‌دهند(یا بالعکس). این ماتریس با توجه به این واقعیت نامگذاری شده است که اجازه می‌دهد به سادگی مشخص شود که آیا یک سیستم دو کلاس را با هم اشتباه گرفته است یا خیر. ماتریس درهم ریختگی برای دسته‌بندی دو کلاس فرضی (آ) و (ب) در جدول ۱.۱ آمده است.

در این جدول نمونه‌هایی که در دسته‌ی آ قرار می‌گیرند مثبت در نظر گرفته شده‌اند. این ماتریس از چهار عنصر اصلی تشکیل شده است که در زیر شرح داده شده اند.

- *مثبت واقعی*[۱۰]: تعداد نمونه‌هایی را نشان می‌دهد که به درستی در دسته‌ی آ پیش‌بینی شده‌اند.
- *مثبت اشتباه*[۱۱]: تعداد نمونه‌هایی را نشان می‌دهد که در دسته‌ی آ پیش‌بینی شده‌اند اما در واقع در دسته‌ی ب قرار دارند.
- *منفی اشتباه*[۱۲]: تعداد نمونه‌هایی را نشان می‌دهد که در دسته‌ی ب پیش‌بینی شده‌اند اما در واقع در درسته‌ی آ قرار دارند.

[5] Commit
[6] Release
[7] Alpha
[8] Beta
[9] Confusion Matrix
[10] True Positive (TP)
[11] False Positive (FP)
[12] False Negetive (FN)

– *منفی واقعی*[۱۳]: تعداد نمونه‌هایی را نشان می‌دهد که به درستی در دسته‌ی ب پیش‌بینی شده‌اند.

جدول ۱.۱: ماتریس درهم‌ریختگی

		دسته‌ی واقعی	
		دسته‌ی آ	دسته‌ی ب
دسته‌ی پیش‌بینی شده	دسته‌ی آ	مثبت واقعی	مثبت اشتباه
	دسته‌ی ب	منفی اشتباه	منفی واقعی

۲.۱ بیان مسئله

آزمون نرم‌افزار اصلی‌ترین فعالیت تیم تضمین کیفیت می‌باشد. آزمون نرم‌افزار می‌تواند تا ۵۰ درصد هزینه‌ی تولید نرم‌افزار را به خود اختصاص دهد. هدف از پیش‌بینی خطا افزایش بازدهی این فرآیند می‌باشد. حال با بهبود پیش‌بینی خطا می‌توان به دستیابی به این هدف کمک نمود. به منظور پیش‌بینی خطا معیارهایی در سطح مورد نظر استخراج می‌گردد. منظور از سطح مورد نظر سطوح مختلف برنامه مانند زیرسیستم، *بسته*[۱۴]، پرونده و تابع می‌باشد. سپس با استفاده از دسته‌بندی، خطادار بودن یا نبودن قطعه‌ی مورد بررسی پیش‌بینی می‌شود. یک دسته از معیارهای مورد استفاده در این زمینه معیارهای فرآیند است و معیارهای جهش نیز به تازگی در این راستا استفاده شده‌اند. این پایانامه قصد دارد تا بررسی کند که معیارهای جهش در کنار فرآیند چه میزان در پیش‌بینی خطا تاثیر گذار است و همچنین بر اساس مفاهیم تحلیل جهش معیارهای جدیدی ارائه دهد تا پیش‌بینی خطا بهبود یابد.

با توجه به اینکه معیارهای جهش به تازگی در پیش‌بینی خطا مورد استفاده قرار گرفته‌اند لازم است تا تحقیقات بیشتری در مورد آنها صورت گیرد و عملکرد آنها از ابعاد مختلف مورد بررسی قرار گیرد. همچنین با بررسی مطالعات پیشین نقاط ضعف و قوت معیارهایی که تا کنون ارائه شده‌اند مورد بررسی قرار می‌گیرد. در این پایانامه عملکرد معیارهای فرآیند و جهش مورد بررسی بیشتری قرار می‌گیرند و با توجه به نقاط ضعف و قدرت معیارهای قبلی، معیارهایی ارائه شده تا بخشی از نقاط ضعف را پوشش دهند و به پیش‌بینی بهتری بیانجامند.

[13] True Negetive (TN)
[14] Package

۳.۱ ساختار پایانامه

این پایانامه در ۶ فصل تهیه گردیده است. در فصل ۲ به مرور مطالعات پیشین پرداخته می‌شود که در قسمت ۱.۲ مباحث مربوط به پیش‌بینی خطا از جمله فرآیند پیش‌بینی، معیارهای ارزیابی، معیارهای پیش‌بینی و مدل‌های پیش‌بینی بررسی می‌شوند. در قسمت ۲.۲ مباحث مربوط به آزمون جهش بررسی شده‌اند و در قسمت ۳.۲ مطالعات مروری جمع‌بندی شده‌اند. در فصل ۳ معیارهای مورد استفاده و ارائه شده در این پایانامه معرفی می‌شوند. در فصل ۴ پنج پروژه‌ی صنعتی مورد مطالعه قرار گرفته‌اند و در فصل ۵ معیارها مورد ارزیابی قرار گرفته‌اند. در فصل ۶ مباحث مطرح شده در این پایانامه جمع‌بندی شده و کارهای آتی شرح داده شده است.

فصل ۲

مرور پژوهش‌های حوزه‌ی پیش‌بینی خطا و آزمون جهش

۱.۲ پیش بینی خطا

در این قسمت ابتدا نحوه‌ی پیش‌بینی خطا به طور کلی شرح داده می‌شود. سپس معیارهای متداول جهت ارزیابی مدل‌های پیش‌بینی بررسی می‌شوند. همانطور که اشاره شد جهت پیش‌بینی لازم است که معیارهای از کد استخراج شود این معیارها به دو دسته‌ی کلی معیارهای کد و معیارهای فرآیند تقسیم می‌گردند. معیارهای مختلف معرفی شده در پژوهش‌های محتلف معرفی می‌شوند. در انتها از مدل‌های که جهت پیش‌بینی استفاده می‌گردد بازبینی می‌شوند.

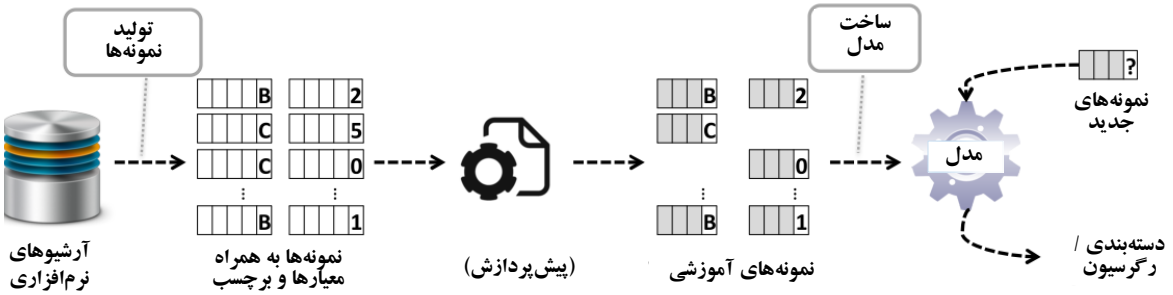
۱.۱.۲ فرآیند پیش‌بینی خطا

اکثریت پژوهش‌های پیش‌بینی خطا از روش‌های یادگیری ماشین استفاده کرده‌اند. اولین گام در ساخت مدل پیش‌بینی تولید داده‌هایی با استفاده از آرشیوهای نرم‌افزاری همانند سامانه‌های کنترل نسخه مانند گیت[۱]، سیستم‌های ردگیری مشکلات مانند جیرا[۲] و آرشیو ایمیل‌ها است. هر یک از این داده‌ها بر اساس درشت دانگی پیش‌بینی می‌توانند نمایانگر یک سیستم، یک *قطعه‌ی*[۳] نرم‌افزاری، بسته، 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 *خطادار* 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

[1] Git
[2] Jira
[3] Component

یک داده‌ی جدید حاوی خطا است یا خیر. تشخیص *خطاخیز*[۴] بودن داده معادل *دسته‌بندی دوتایی*[۵]. مقصود از دسته‌بندی دوتایی، دسته‌بندی عناصر مجموعه‌ی داده شده به دو گروه مجزا می‌باشد. همچنین پیش‌بینی تعداد خطاها معادل *رگرسیون*[۶] می‌باشد. منظور از رگرسیون فرآیند آماری است که در آن با استفاده از متغیرهای مستقل سعی می‌شود متغیر وابسته تخمین زده شود که در اینجا متغیرهای مستقل معیارهای پیش‌بینی خطا و معیار وابسته تعداد خطاها می‌باشد.

در شکل ۱.۲ فرآیند پیش‌بینی خطا نشان داده شده است. داده‌ها نمونه‌هایی هستند که می‌توانند خطادار 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.



شکل ۱.۲: فرآیند پیش‌بینی خطا [۴]

۲.۱.۲ اندازه‌های ارزیابی

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

- مثبت واقعی: تعداد داده‌های حاوی خطا که به درستی تشخیص داده شدند
- مثبت اشتباه: تعداد داده‌های سالم که به عنوان خطادار پیش‌بینی شدند
- منفی اشتباه: تعداد داده‌های سالم که به درستی تشخیص داده شدند
- منفی واقعی: تعداد داده‌های حاوی خطا که به عنوان داده‌ی سالم پیش‌بینی شدند

[4] Bug-proneness
[5] Binary Classification
[6] Regression

جدول ۱.۲: فرمول‌های محاسبه‌ی معیارهای ارزیابی

نام معیار	نام لاتین	نحوه‌ی محاسبه	توضیح
نرخ مثبت اشتباه	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	$\frac{۲ \times Precision \times Recall}{Precision+Recall}$	از آنجا که در بین معیارهای دقت و بازخوانی مصالحه وجود دارد معیار اف ترکیبی از آن دو را در نظر می‌گیرد

در ادامه به بررسی و تحلیل هر یک از این معیارها پرداخته می‌شود.

- نرخ مثبت اشتباه: نام دیگر این معیار *احتمال اخطار اشتباه*[۷] می‌باشد. هر چقد که یک مدل پیش‌بینی به اشتباه داده‌ها را خطادار پیش‌بینی کند مقدار این معیار بیشتر می‌شود تا جایی که اگر مدل پیش‌بینی هیچ داده‌ای را بدون خطا پیش‌بینی نکند مقدار آن یک می‌شود و اگر داده‌ای را به اشتباه حاوی خطا معرفی نکند مقدار معیار صفر می‌شود.

- صحت: این معیار نسبت تعداد پیش‌بینی‌های مثبت واقعی و منفی واقعی را به تعداد کل پیش‌بینی‌ها می‌سنجد. با این حال صحت نمی‌تواند در مواردی که مجموعه داده‌های نامتوازن وجود دارد معیار مناسبی داشته باشد. به عنوان مثال اگر در یک مجموعه‌داده ۱۰ درصد از داده‌ها حاوی خا باشد آنگاه مدلی که همواره داده‌ها را بدون خطا پیش‌بینی می‌کند این معیار مقدار ۹۰ درصد می‌گیرد در صورتی که این مدل مناسب نیست.

- دقت: نام دیگر این معیار *ارزش پیش‌بینی مثبت*[۸] می‌باشد. این معیار نشان دهنده‌ی آن است که به چه

[7] Probability of False Alarm (PF)
[8] Positive Predictive Value

میزان داده‌های پیش‌بینی شده به عنوان خطادار درست پیش‌بینی شده است. در صورتی که همه‌ی داده‌هایی که خطادار معرفی شده‌اند در واقعیت نیز حاوی خطا باشد این معیار مقدار یک می‌یابد.

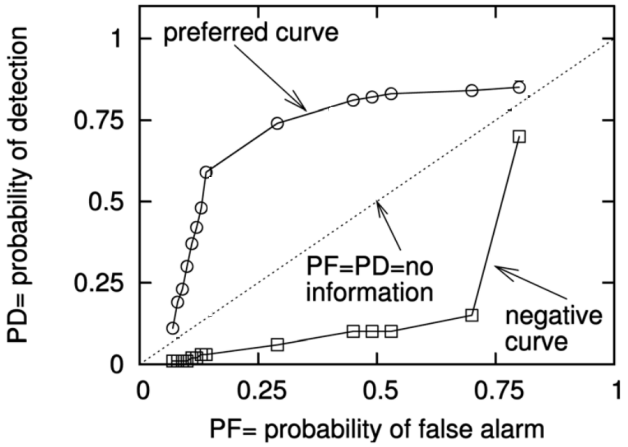
- بازخوانی: این معیار مشخص می‌کند که چه مقدار از داده‌هایی که باید به عنوان خطادار معرفی می‌شدند در واقع توسط مدل خطادار پیش‌بینی شده‌اند. زمانی که این معیار برابر یک می‌باشد بدان معنی است که تمام داده‌های حاوی خطا شناسایی شده‌اند. البته ممکن است برخی داده‌های بدون خطا نیز خطا دار پیش‌بینی شوند و همچنان معیار بازخوانی مقدار یک را داشته باشد. همانطور که در جدول ۱.۲ مشخص شده است بین دقت و بازخوانی یک *مصالحه*[9] وجود دارد. این بدان معنی است که اغلب می‌توان یکی را به هزینه‌ی کاهش دیگری افزایش داد.

- معیار اف: از آنجا که در محاسبه‌ی این معیار از ترکیب دقت و بازخوانی استفاده می‌شود از معایب بررسی جداگانه این دو معیار کاسته می‌شود. در برخی موارد اهمیت دقت و بازخوانی یکسان نیست که باید از نوع دیگری از معیار اف استفاده که دارای وزن‌دهی می‌باشد.

دو اندازه دیگر نیز که در پژوهش‌ها کاربرد دارند عبارتند از *مساحت زیر منحنی*[10] و *مساحت زیر منحنی هزینه_اثربخشی*[11]. در محاسبه‌ی مساحت زیر منحنی از نمودار *مشخصه‌ی عملکرد دریافت‌کننده*[12] استفاده می‌شود . در این نمودار محورهای عمودی و افقی را به ترتیب بازخوانی و نرخ مثبت کاذب تشکیل می‌دهد. با تغییر *آستانه‌ی تصمیم*[13] برای یک مدل می‌توان میزان بازخوانی و نرخ مثبت کاذب را تغییر داده و بدین ترتیب منحنی را رسم نمود. منظور از آستانه‌ی تصمیم مرزی است که یک مدل یک داده را حاوی خطا پیش‌بینی می‌کند یا سالم. به عنوان مثال زمانی که آستانه برابر ۳۰ درصد است در صورتی که یک داده به احتمال ۳۱ درصد حاوی خطا باشد آن داده به عنوان خطادار پیش‌بینی می‌شود.

یک مدل بی‌نقص دارای مساحت زیر نمودار ۱ است. مدل بی‌نقص مدلی است که تمام پیش‌بینی‌ها را به درستی انجام می‌دهد. این مدل در برخورد با داده‌ی حاوی خطا ۱۰۰ درصد احتمال می‌دهد که حاوی خطا است و برای داده‌ی سالم صفر درصد احتمال می‌دهد حاوی خطا است. اگر بخواهیم منحنی را برای مدل بی‌نقص رسم

[9] Trade-off
[10] Area under curve (AUC)
[11] Area under cost-effectiveness curve(AUCEC)
[12] Receiver Operating Characteristic
[13] Decition Threshold



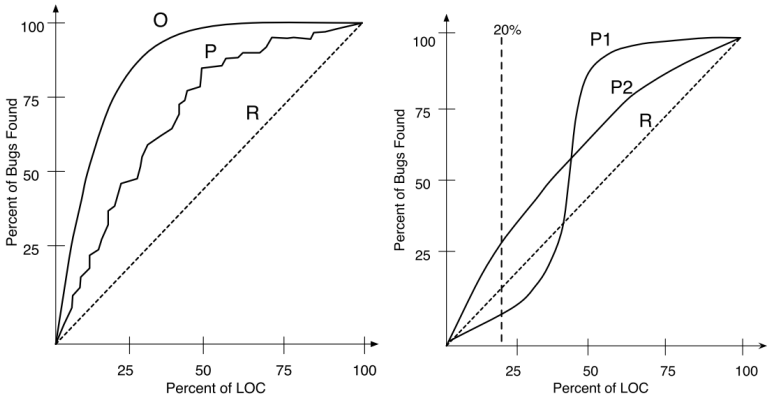
شکل ۲.۲: نمونه‌ای از نمودار ROC [۵]

کنیم در ابتدا آستانه را برابر یک در نظر گرفته می‌شود. در نتیجه همه‌ی داده‌ها بدون خطا دسته‌بندی می‌شوند. در این حالت نرخ مثبت اشتباه برابر صفر است زیرا هیچ داده‌ای به اشتباه خطادار معرفی نشده. بازخوانی نیز صفر است چون هیچ داده‌ای به درستی خطادار پیش‌بینی نشده. پس منحنی از نقطه‌ی صفر و صفر آغاز می‌شود. زمانی که آستانه اندکی از یک کمتر شود مدل همه‌ی پیش‌بینی‌ها را به درستی انجام می‌دهد و نرخ مثبت اشتباه برابر صفر و بازخوانی برابر یک خواهد بود. در نتیجه نقطه‌ی دیگر بر روی منحنی در بالا سمت چپ منحنی است. با کمتر کردن آستانه تغییری در کحل نقطه ایجاد نمی‌شود تا زمانی که آستانه به صفر برسد. در این حالت همه‌ی داده‌ها خطادار پیش‌بینی می‌شوند. نرخ مثبت اشتباه برابر یک خواهد شد چون هیچ داده‌ای سالم پیش‌بینی نشده است و بازخوانی برابر یک خواهد بود چون همه‌ی داده‌هایی که باید خطادار پیش‌بینی می‌شدند خطادار پیش‌بینی شده‌اند. در نتیجه نقطه‌ی دیگر در بالا راست نمودار خواهد بود و مساحت زیر نمودار برابر یک خواهد بود.

برای یک مدل تصادفی منحنی از مبدا به نقطه‌ی (۱,۱) رسم خواهد شد. یک نمونه از منحنی مشخصه‌ی عملکرد دریافت‌کننده در شکل ۲.۲ آمده است.

مساحت زیر منحنی هزینه‌ـاثربخشی معیاری است که تعداد خطوطی از برنامه که توسط تیم تضمین کیفیت و یا توسعه دهندگان نیاز است بررسی و آزمون شود را در نظر می‌گیرد. منظور از بررسی بازبینی کد جهت یافتن خطا بدون استفاده از روش‌های مرسوم آزمون نرم‌افزار می‌باشد. ایده‌ی موثر بودن از نظر هزینه برای مدل‌های

پیش‌بینی خطا برای اولین بار توسط آریشلم و همکاران [۶] ارائه گردید. موثر بودن از نظر هزینه به این معنا است که چه تعداد خطا با بررسی و یا آزمون n درصد اول خطوط می‌توان یافت. به عبارت دیگر اگر یک مدل پیش‌بینی خطا بتواند تعداد 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 درصد اول محور افقی در نظر گرفته می‌شود مدل P_2 کارایی بهتری دارد. نمودار سمت چپ مدل‌های تصادفی، عملی[۱۴] و بهینه را نشان می‌دهد.



R = random P = practical O = optimal

شکل ۳.۲: نمودار موثر بودن از نظر هزینه [۷]

معیارهایی که برای ارزیابی نتایج حاصل از روش رگرسیون به کار گرفته می‌شوند بر اساس همبستگی[۱۵] میان تعداد خطاهای پیش‌بینی شده و خطاهای واقعی محاسبه می‌شوند. نماینده‌ی این معیارها را می‌توان همبستگی اسپیرمن، پیرسون و R^2 دانست [۴].

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 جمع آوری 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 موثر تر از سایر معیارها هستند[۸]. در برخی از

[14] Practical
[15] Correlation

مقالات نیز معیارهای پیش‌بینی خطا به سه دسته‌ی: معیارهای کد منبع سنتی، معیارهای شیءگرایی و معیارهای فرآیند تقسیم شده‌اند[۹].

معیارهای کد

معیارهای کد تحت عنوان *معیارهای محصول*[۱۶] نیز شناخته می‌شوند و میزان پیچیدگی کد را می‌سنجند. *فرض زمینه‌ای*[۱۷] 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.

این معیارها با استفاده از اندازه‌های مطرح شده در جدول ۲.۲ استخراج شده‌اند.

[16] Product Metrics
[17] Ground Assumption

جدول ۲.۲: اندازه‌های به کارگرفته شده در معیارهای کد

نام	نام لاتین	علامت اختصاری	توضیح
تعداد خطوط کد	Line of Code	LOC	این اندازه را می‌توان به اندازه‌های جزیی‌تر مانند تعداد خطوط توضیح، قابل اجرا، خالی از نوشته تقسیم کرد
تعداد عملگرها	Number of Operators	N_1	تعداد عملگرهای موجود مانند + ، ـ ، &
تعداد عملوندها	Number of Operands	N_2	تعداد عملوندهای استفاده شده در کنار عملگرها
تعداد عملگرهای متمایز	Number of Unique Operators	η_1	—
تعداد عملوندهای متمایز	Number of Unique Operands	η_2	—
تعداد یال‌ها	Number of Edges	E	تعداد یال‌های گراف جریان کنترلی
تعداد گره‌ها	Number of Nodes	N	تعداد گره‌ها در گراف جریان کنترلی
تعداد قطعات متصل	Number of Connected Component	P	تعداد قطعات متصل به هم در گراف جریان کنترلی

- **معیار بزرگی:** معیارهای *بزرگی*[18] اندازه‌ی کلی و حجم کد را می‌سنجند. یکی از اندازه‌های برجسته که در محاسبه‌ی این معیارها و گاها خود به تنهایی به کار می‌رود "تعداد خطوط" می‌باشد. اولین بار *آکیاما*[19] [۱۰] رابطه‌ی میان خطا و تعداد خطوط را مطرح کرد. *هالستد*[20] [۱۱] چندین معیار بزرگی بر اساس تعداد عملگرها و عملوندها ارائه داده است و در مقاله‌ی [۱۲] مورد بازنگری قرار گرفته است. معیارهایی که توسط هالستد مطرح شده‌اند در زیر آمده آمده‌اند که با استفاده از اندازه‌های جدول ۲.۲ محاسبه می‌شوند.

Lenght: $N = N_1 + N_2$

[18] Size
[19] Akiyama
[20] Halstead

Volume: $V = N \times log_2(\eta_1 + \eta_2)$

Difficulty: $D = \eta_1/2 \times N_2/\eta_2$

Effort: $E = D \times V$

Program Time: $T = E/18$

- **معیار پیچیدگی حلقوی:** مک‌کیب[۲۱] معیارهای *پیچیدگی حلقوی*[۲۲] را پیشنهاد داد که این معیار با استفاده از تعداد گره‌ها، یالها و قطعات متصل در گراف *جریان کنترلی*[۲۳] کد منبع محاسبه می‌گردد[۱۳]. این معیارها نشان می‌دهند که راه‌های کنترلی به چه میزان پیچیده هستند. باوجود اینکه جز اولین معیارها بوده است همچنان در پیش‌بینی خطا کاربرد دارد [۱۴]. این معیار با استفاده با استفاده از فرمول زیر محاسبه می‌شود.

$V(G) = E - N + 2P$

- **معیار مربوط به شئ‌گرایی:** با ظهور زبان‌های شئ‌گرایی و محبوبیت آنها معیارهای کد برای این زبان‌ها ارائه شد تا فرآیند توسعه بهبود یابد. نماینده‌ی این معیارها CK می‌باشد که توسط *چدامبر و کمرر*[۲۴] ارائه شده است[۱۵]. این معیارها که در جدول ۳.۲ لیست آنها قرار داده شده، با توجه به خصیصه‌های زبان‌های شئ‌گرا مانند وراثت، *زوجیت*[۲۵]، همبستگی[۲۶] طراحی شده‌اند. بجز معیارهای CK، معیارهای شئ‌گرایی دیگری نیز بر اساس حجم و کمیت کد منبع پیشنهاد داده شده‌اند. مشابه معیارهای *اندازه*، معیارهای شئ‌گرایی تعداد نمونه‌های یک کلاس، توابع را می‌شمارند.

[21] McCabe
[22] Cyclomatic Complexity
[23] Control Flow
[24] Chidamber and Kemerer (CK)
[25] Coupling
[26] Cohesion

جدول ۳.۲: معیارهای CK

نام	توضیح	نحوه‌ی محاسبه
WMC	تعداد توابع وزن‌دهی شده	وزن دهی بر اساس پیچیدگی هر تابع انجام می‌شود
DIT	عمق درخت وراثت	حداکثر طول مسیر در در از نوادگان یک کلاس تا خود کلاس
NOC	تعداد فرزندان	تعداد نوادگان مستقیم کلاس
CBO	زوجیت میان اشیاء کلاس‌ها	تعداد کلاس‌هایی که کلاس مورد نظر با آن زوج شده‌است. دو کلاس با هم زوجیت دارند اگر یکی از توابع و یا متغرهای دیگری استفاده کرده باشد.
RFC	پاسخ برای یک کلاس	تعداد توابعی که با فراخوانی یک تابع از کلاس احتمال فراخوانی دارند. برابر است با تعداد کل توابع کلاس و توابعی از سایر کلاس‌ها که در آنها فراخوانی می‌شوند.
LCOM	کمبود همبستگی میان توابع	تعداد جفت توابعی که متغیر مشترک ندارند منهای جفت توابعی که متغیر مشترک دارند.

معیارهای فرآیند

در ادامه تعدادی از معیارهای فرآیند بررسی می‌شوند که در این دسته شاخص محسوب می‌شوند. در جدول ۴.۲ اندازه‌هایی که در محاسبه‌ی معیارهای فرآیند به کار گرفته شده‌اند آمده است.

جدول ۴.۲: اندازه‌های به کارگرفته شده در معیارهای فرآیند

نام	نام لاتین	علامت اختصاری	توضیح
تعداد خطوط تبدیلی	Churned LOC	—	تعداد خطوط اضافه شده به علاوه‌ی خطوط تغییر داده شده در دو نسخه‌ی متفاوت از برنامه
تعداد فایل‌های تبدیلی	Files Churned	—	تعداد فایل‌های تغییر یافته در یک قطعه
تعداد فایل‌ها	Files Count	—	تعداد فایل‌های موجود در یک قطعه
تجدیدنظرها	Revisions	—	تعداد تجدید نظرهایی (اصلاح‌ها) که در فایل انجام شده است
بازآرایی	Refactoring	—	تعداد دفعاتی که یک فایل بازآرایی شده است. در واقع تعداد ثبت‌هایی شمرده می‌شود که در توضیح آنها کلمه‌ی refactor وجود داشته باشد.
تعداد ایمیل‌ها	Number of Mails	POP_NOM	تعداد ایمیل‌هایی که در آنها نام کلاس مورد نظر آورده شده است.
تعداد نخ‌ها	Number of Threads	POP_NOT	تعداد نخ‌هایی که درباره‌ی یک کلاس صحبت می‌کنند
تعداد نویسندگان	Number of Authors	POP_NOA	تعداد نویسندگانی که درباره‌ی کلاس مورد نظر صحبت می‌کنند.

- **تغییر تبدیلی نسبی کد:** *ناگاپان و بال*[۲۷] هشت معیار تغییر *تبدیلی*[۲۸] نسبی کد را ارائه داده‌اند[۱۶]. دو مثال از این معیارها در زیر آمده است. در معیار M_1 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.

$$M_1 = ChurnedLOC/TotalLOC$$

$$M_2 = FilesChurned/FilesCount$$

- **معیارهای تغییر:** این معیارها گستره‌ی تغییرات در تاریخچه‌ی ذخیره شده در سامانه‌ی کنترل نسخه را اندازه می‌گیرند. به عنوان مثال تعداد رفع خطاها، تعداد *بازآرایی کد*[۲۹] و یا تعداد نویسندگان یک فایل

[27] Nagappan and Ball
[28] Churn
[29] Refactoring

را می‌شمارند. *موزر*[۳۰] و همکاران ۱۸ معیار تغییر را از مخازن *اکلیپس*[۳۱] استخراج کردند و یک تحلیل مقایسه‌ای میان معیارهای کد و معیارهای تغییر انجام دادند. آنها به این نتیجه رسیدند که معیارهای تغییر پیش‌بینی کننده‌ی بهتری از معیارهای کد هستند. به عنوان نمونه دو مورد از ۱۸ معیار مطرح شده برابر اندازه‌های *تجدیدنظرها* و *بازآرایی* است.

- **معیارهای شهرت:** بکچلی[۳۲] و همکاران معیارهای *شهرت*[۳۳] 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[۱]. بکچلی پنج معیار شهرت معرفی کرده است. به عنوان نمونه سه مورد از آنها برابر است با اندازه‌های *تعداد ایمیل‌ها*، *تعداد نخ‌ها* و *تعداد نویسندگان*.

راجنویک[۳۴] و همکاران در پژوهش خود به *بررسی قاعده‌مند*[۳۵] معیارهای پیش‌بینی خطا در مطالعات پیشین پرداخته‌اند. طبق این پژوهش در ۴۹% مطالعات از معیارهای شیءگرایی، 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 شیءگرایی 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 شیءگرایی 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 شیءگرایی به منظور بررسی یک نسخه از برنامه مفید هستند اما با هر *تکرار*[۳۶] در فرآیند توسعه‌ی 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[۹].

[30] Moser
[31] Eclipse
[32] Bacchelli
[33] Popularity
[34] Radjenovic
[35] Systematic Review
[36] Iteration

آسترند[۳۷] 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[۱۷] مشخص شده بود که تعداد کلی افراد توسعه‌دهنده در یک فایل می‌تواند در پیش‌بینی خطا تاثیر متوسطی داشته باشد. در مقاله‌ی [۱۸] 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 توسعه دهندگان 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[۱۸].

رحمان و دوانبو[۳۸] از جنبه‌های مختلف معیارهای فرآیند را با سایر معیارها مقایسه کرده‌اند[۱۹]. نتایج نشان می‌دهد زمانی که مدل پیش‌بینی بر روی یک نسخه آموزش می‌بیند و در نسخه‌ی بعدی آزمون می‌شود معیارهای کد، مساحت زیر منحنی قابل قبولی دارند اما مساحت آنها کمتر از معیارهای فرآیند است و از نظر معیار مساحت زیر نمودار هزینه‌ـ‌اثربخشی ۲۰ درصد بهتر از یک مدل تصادفی عمل نمی‌کنند و به آن معنی است که این معیارها از نظر هزینه چندان موثر نیستند. همچنین معیارهای کد ایستاتر 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 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*[۳۹] 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.

معیارهای استفاده شده در این مقاله در جدول ۴.۲ آورده شده‌اند. در ادامه هر یک از معیارها به طور مشروح توضیح داده می‌شوند.

[37] Ostrand
[38] Rahman and Devanbu
[39] Bias

جدول ۵.۲: معیارهای فرآیند [۱۹]

	نام معیار	توضیح
۱	COMM	تعداد ثبت در سیستم کنترل نسخه
۲	ADEV	تعداد توسعه‌دهندگان فعال
۳	DDEV	تعداد توسعه‌دهندگان متمایز
۴	ADD	مقدار نرمال‌سازی شده‌ی تعداد خطوط اضافه شده
۵	DEL	مقدار نرمال‌سازی شده‌ی تعداد خطوط حذف شده
۶	OWN	درصد خطوطی که مالک پرونده مشارکت کرده
۷	MINOR	تعداد مشارکت‌کنندگان جزئی
۸	NCOMM	تعداد ثبت‌های همسایگان
۹	NADEV	تعداد توسعه‌دهندگان فعال همسایگان
۱۰	NDDEV	تعداد توسعه‌دهندگان متمایز همسایگان
۱۱	OEXP	تجربه‌ی مالک پرونده
۱۲	AEXP	تجربه‌ی تمام مشارکت‌کنندگان

۱. **تعداد ثبت در سیستم کنترل نسخه:** تعداد ثبت‌هایی که در آن پرونده‌ی مورد نظر در طول انتشار قبلی تاکنون تغییر کرده است. برای محاسبه ی آن لازم است که تمام ثبت‌های پروژه بین ثبت کنونی و انتشار قبلی بررسی شود و ثبت‌هایی که در آن این پرونده تغییر کرده‌اند شمرده شوند.

۲. **تعداد توسعه‌دهندگان فعال:** تعداد توسعه‌دهندگانی که در طول انتشار قبلی تا کنون (زمان ثبت) پرونده را تغییر داده‌اند. لازم است ثبت‌های موجود در بازه‌ی زمانی خواسته شده بررسی شود و آنها که پرونده 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 پرونده 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 پرونده 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 پرونده مشارکت کرده:** درصد خطوطی از پرونده، در ثبت مورد نظر که به مالک پرونده تعلق دارد. مالک پرونده کسی است که در آن لحظه از زمان بیشترین تعداد خطوط موجود در پرونده 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 پرونده بر تعداد خطوط پرونده تقسیم می‌گردد.

۷. **تعداد مشارکت‌کنندگان جزئی:** مشارکت‌کننده‌ی جزئی کسی است که کمتر از ۵٪ خطوط موجود در پرونده به او تعلق داشته باشد. بدین منظور نویسنده‌ی هر خط مشخص می‌شود. تعداد خطوط هر نویسنده شمرده می‌شود و بر تعداد خطوط پرونده تقسیم می‌شود. سپس تعداد نویسندگانی که کمتر از ۵٪ مشارکت داشته‌اند شمرده می‌شود.

۸. **تعداد ثبت‌های همسایگان:** میانگین وزن دهی شده تعداد ثبت‌های همسایگان پرونده از انتشار قبلی تا کنون را اندازه‌گیری می‌کند. همسایگان یک پرونده در یک ثبت، پرونده‌هایی هستند که در آن نسخه از برنامه تغییر کرده‌اند. درواقع در هر ثبت از برنامه تعدادی پرونده نسبت به ثبت قبلی تغییر کرده‌اند که این پرونده‌ها همسایه‌ی یکدیگر محسوب می شوند. نحوه‌ی وزن دهی نیز به این صورت است که هرچقدر یک پرونده تعداد دفعات بیشتری را در طول انتشار با پرونده مورد نظر همسایه شده باشد وزن بیشتری می‌یابد. برای محاسبه ابتدا همسایگان پرونده در ثبت و تعداد دفعاتی که در طول انتشار همسایه شده‌اند مشخص می‌شوند. سپس برای هر پرونده‌ی 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 پرونده:** ابتدا لازم است که نحوه ی محاسبه تجربه را تعریف کنیم. هر چقدر 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 پرونده مشخص می شود. سپس تعداد ثبت‌هایی که مالک پرونده از ابتدای پروژه تا زمان مورد نظر انجام داده، شمرده می شود.

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

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*[۴۰] برای ساخت مدل‌های پیش‌بینی خطای کاراتر به کار گرفته شده است[۲۰]. علاوه بر مدل‌های یادگیری ماشین، مدل‌های غیر آماری مانند *باگ‌کَش*[۴۱] پیشنهاد داده شده است [۲۱]. در میان روش‌های دسته‌بندی، *رگرسیون منطقی*[۴۲] ، *بیز ساده*[۴۳] و *درخت تصمیم*[۴۴] بیش از سایرین در پژوهش‌ها مورد استفاده قرار گرفته‌اند. همچنین در میان روش‌های رگرسیون، *رگرسیون خطی*[۴۵] و *رگرسیون دوبخشی منفی*[۴۶] به طور گسترده به کار گرفته شده‌اند [۴].

کیم[۴۷] و همکاران *باگ‌کَش* را ارائه داده‌اند که اولویت موجودیت‌های خطاخیز در *حافظه‌ی موقت*[۴۸] 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 روش نسبت به دیگری متفاوت باشد، با این حال پژوهشی که توسط آریشلم و همکاران [۸] انجام شده است

[40] Semi-Supervised
[41] BugCache
[42] Logistic Regression
[43] Naive Bayes
[44] Decision Tree
[45] Linear Regression
[46] Negetive Binomial Regression
[47] Kim
[48] Cache

نشان می‌دهد که تاثیر تکنیک یادگیری در حد متوسطی است و کمتر از انتخاب معیار بر روی کارایی تاثیر گذار است.

مالهوترا[۴۹] با بکارگیری معیارهای سنتی کد، عملکرد تکنیک‌های یادگیری ماشین و رگرسیون را مقایسه کرده است[۱۴]. وی به منظور پیش پردازش نیز از *آماره‌های توصیفی*[۵۰] 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 ماژول‌ها را به خوبی متمایز نمی‌کنند و بعید است که مفید باشند و می‌توانند حذف شوند. در این مقاله 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. Logestic 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 بهتر از سایرین عمل کرده است.

۵.۱.۲ 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*[۵۱] 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 [۲۲]. کیم و همکاران نیز مدل جدیدی ارائه داده‌اند که دسته‌بندی تغییر 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[۲۳].

[49] Malhotra
[50] Descriptive Statistics
[51] Hata

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 و پژوهشگران به دنبال مقایسه‌ی روش‌های مختلف آزمون و *اشکال زدایی*[۵۲] 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*[۵۳] استفاده شود. یکی از اندازه‌گیری‌های شناخته شده *امتیاز جهش*[۵۴] 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*[۵۵] 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*[۵۶] و یا حذف یک عبارت است[۲۴]. تحلیل آزمون در موارد زیر کاربرد دارد:

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

[52] Debugging
[53] Proxy Measurement
[54] Mutation Score
[55] Mutant
[56] Branch 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 [۲۵]

جاست[۵۷] و همکاران در پژوهش خود به بررسی این موضوع پرداخته‌اند که آیا جهش‌یافته‌ها می‌توانند جایگزین مناسبی برای خطاهای واقعی باشند یا خیر[۲۴]. 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 متداول تولید شده‌اند وابستگی دارند. در این پژوهش خطاهایی که با جهش‌یافته‌ها وابستگی ندارند در سه دسته قرار می‌گیرند : دسته اول نیازمند عملگرهای قوی‌تری هستند، دسته‌ی دوم نیازمند عملگرهای جدیدی هستند و دسته‌ی سوم با جهش‌یافته‌ها وابستگی ندارند.

[57] Just

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 کد منبع استفاده می‌شود تا احتمال وجود خطا محاسبه شود. دو تفاوت عمده‌ی این دو حوزه این است که اولا در مکان‌یابی خطا از روش‌های یادگیری ماشین استفاده‌ی چندانی نمی‌شود، ثانیا 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[۵۸] 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 خطادار نسبت به جهش یک عبارت درست می‌تواند موارد آزمون بیشتری را با موفقیت بگذراند. 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*[۵۹] 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 [۲۵].

پاپادا‌کیس و تراوون[۶۰] در مقاله‌ی خود به این نکته اشاره کرده‌اند که استفاده از تحلیل جهش در گذشته به دلیل پر هزینه بودن چندان مورد توجه قرار نمی‌گرفته است اما امروزه با وجود ابزارهای مقیاس پذیر، نمونه گیری و انتخاب جهش می‌توان به خوبی از تحلیل جهش در انجام پژوهش‌های مختلف استفاده کرد[۲۶]. 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

[58] Moon
[59] Suspiciousness
[60] Papadakis and Traon

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.

۲.۲.۲ مدل‌های یادگیری و جهش‌یافته‌ها

هااو[۶۱] 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[۶۲] شدن یک مورد آزمون[۲۷]. هفت معیار ارائه شده در این پژوهش مرتبط با گراف فراخوانی، تغییر در فایل‌ها و تعداد شکست در آزمون‌ها بوده است. هااو 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 دسته‌ای*[۶۳] به آموزش و آزمایش مدل ساخته شده پرداخته می‌شود. نتایج پژوهش نشان می‌دهد که روش پیشنهادی زمانی که بر روی یک نسخه‌یا 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.

بوئز[۶۴] و همکاران معیارهایی را مبتنی بر جهش معرفی کردند و از ترکیب آنها با معیارهای سنتی و شئ‌گرایی، یک مدل پیش‌بینی ساخته شده است[۲۸]. هشت 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

[61] Hao
[62] Obsolete
[63] Cross-validation
[64] Bowes

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.

معیارهای به کار گرفته شده در این مقاله در جدول ۶.۲ آمده است. این معیارها توسط ابزارهای آزمون جهش و بدون نیاز به پردازش بیشتر، قابل استخراج هستند.

جدول ۶.۲: معیارهای جهش [۲۸]

	نام معیار	توضیح
۱	MuNOM	تعداد جهش‌یافته‌های تولید شده
۲	MuNOC	تعداد جهش‌یافته‌های پوشش‌داده شده توسط آزمون‌ها
۳	MuNMS	امتیاز جهش‌یافته‌های تولید شده
۴	MuNMSC	امتیاز جهش‌یافته‌های پوشش‌داده شده توسط آزمون‌ها

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 جمع آوری می‌شود. با توجه به معیارهای مختلف داده‌هایی استخراج می‌شود که هر داده دارای برچسب "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 مسلما 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 مساحت زیر منحنی 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 مساحت زیر منحنی هزینه_اثربخشی می‌توان موثر بودن مدل از نظر هزینه را سنجید. معمولا 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 شئ 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 شئ گرایی همبستگی فراوانی با معیارهای اندازه دارند. معیارهای شئ گرایی دارای وابستگی فراوانی با معیارهای اندازه هستند. با این حال معیارهای شئ 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 [۸] 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 [۲۴] نشان داده شده است که جهش‌یافته‌هایی که با عملگرهای جهش ساده تولید شده‌اند می‌توانند تا ۷۳ % خطاهای واقعی را شبیه سازی کنند و ازین جهت جایگزین مناسبی برای خطاهای واقعی باشند.

جدول ۷.۲: جدول مشخصات پژوهش‌های مرور شده در حوزه‌ی پیش بینی خطا

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

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 کاراتر در پیش‌بینی خطا پرداخت مشخص شد. مقاله‌ی [۲۸] 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 [۹] 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 [۱۹].

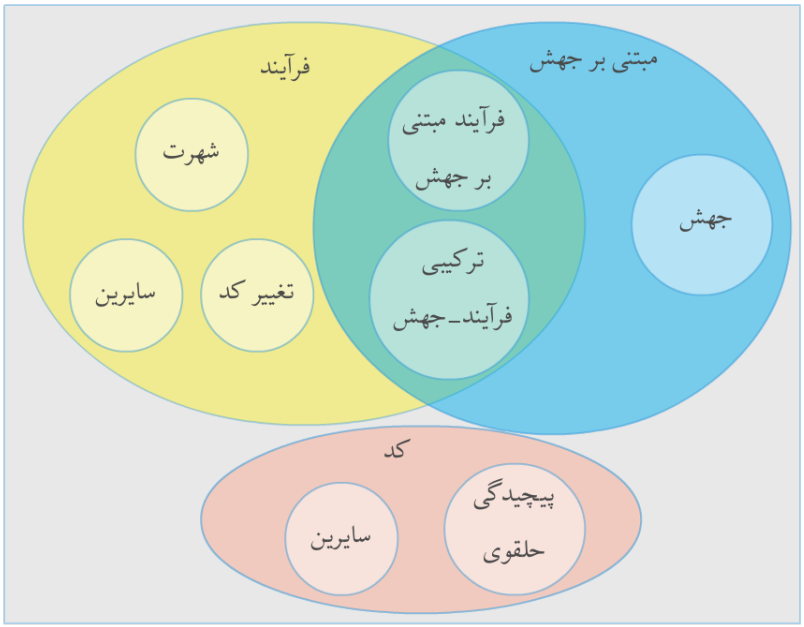
این پایانامه قصد دارد سه رویکرد پیشنهادی را به منظور بهبود پیش‌بینی خطا بررسی کند. این رویکردها عبارتند از:

۱. معیارهای جهش و معیارهای فرآیند در کنار یکدیگر استفاده می‌شوند و به وسیله‌ی آنها پیش‌بینی انجام می گیرد. 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.

همچنین جهت استخراج معیارها و انجام پیش‌بینی خطا در این پایانامه ابزاری به نام JPredict طراحی و ساخته می‌گردد. جهت مشخص‌تر شدن نحوه‌ی قرارگیری معیارهای مطح شده نمودار ون معیارهای پیش‌بینی خطا در شکل ۱.۳ نمایش داده شده است.



شکل ۱.۳: نمودار ون معیارهای پیش‌بینی خطا

۱.۳ معیارهای جهش و فرآیند

این رویکرد با توجه به مقاله‌ی [۲۸] 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 [۱۹] 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 چراکه 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 انجام این رویکرد مجموعه معیارهای جهش از پژوهش [۲۸] و معیارهای فرآیند از پژوهش [۱۹] انتخاب می‌شوند. در جداول ۵.۲ و ۶.۲ معیارهای مورد نظر آورده شده است و به ترتیب در قسمت‌های ۳.۱.۲ و ۲.۲ معرفی شده‌اند.

از آنجا که در این پایانامه پیش‌بینی‌ها در سطح پرونده انجام می‌شود، معیارها برای هر پرونده 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 و نیازمند توضیح بیشتر نیستند.

۲.۳ معیارهای جهش مبتنی بر فرآیند

در رویکرد دوم، چهار معیار جدید در این پایانامه 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*[1] نامیده شده‌اند.

جدول ۱.۳: نمادهای استفاده شده در تعاریف معیارها

نماد	توضیح
$Mutants(c, f)$	مجموعه‌ی جهش‌یافته‌های پرونده‌ی f در ثبت یا انتشار c
C_i	ثبت شماره‌ی i
R_i	انتشار شماره‌ی i
$C(R)$	ثبت متعلق به انتشار R
$Seq(C)$	شماره‌ی ثبت C
$LR(i)$	آخرین انتشار ماقبل ثبت شماره‌ی i
$MuScore(c, f)$	امتیاز جهش پرونده‌ی f در ثبت یا انتشار c
$\delta^+(x)$	$\begin{cases} x & x > 0 \text{ اگر} \\ 0 & x <= 0 \text{ اگر} \end{cases}$
$\delta^-(x)$	$\begin{cases} 0 & x > 0 \text{ اگر} \\ x & x <= 0 \text{ اگر} \end{cases}$

[1] Process Based Mutation Metrics (PBMM)

جدول ۲.۳: عملگرهای استفاده شده در مثالها

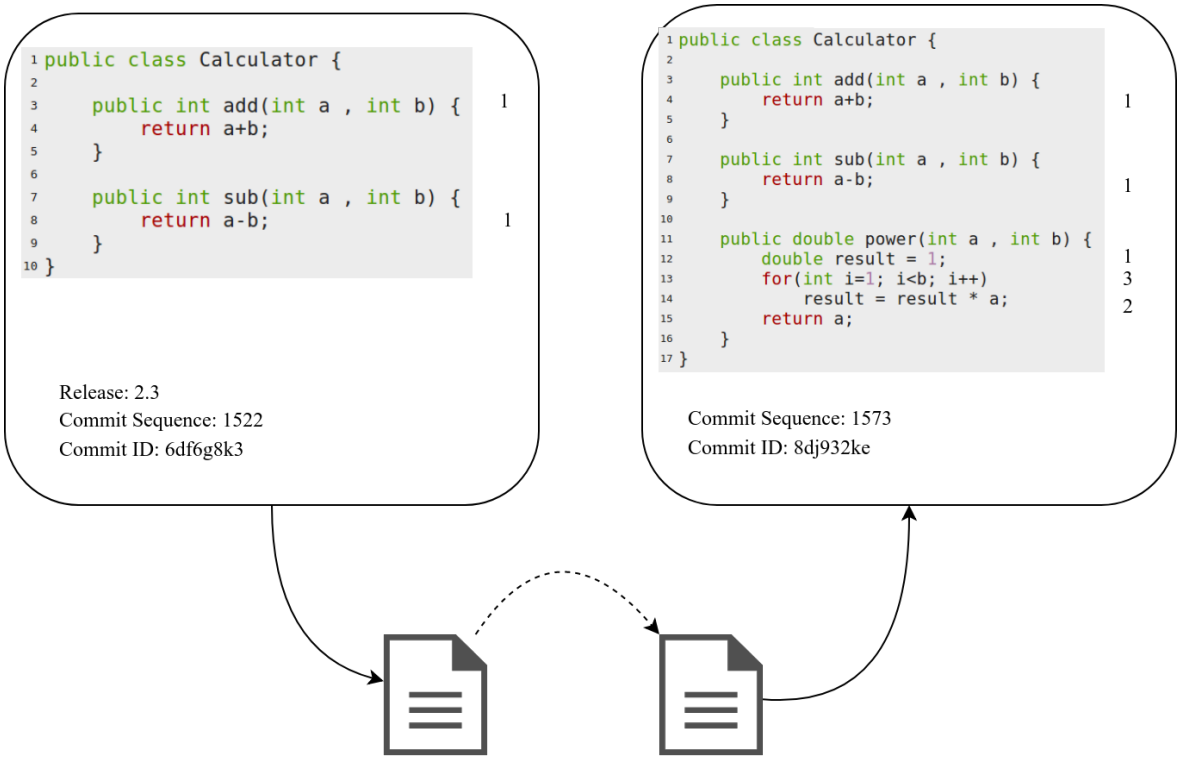
عملگر	نسخه‌ی اصلی	نسخه‌ی جهش‌یافته
Arithmetic Operator Replacement	b + a	b ـ a
Arithmetic Operator Replacement	b ـ a	b + a
Arithmetic Operator Replacement	b * a	b / a
Arithmetic Operator Replacement	b % a	b / a
Expression Value Replacement	int a = x	int a = 0
Relational Operator Replacemen	b > a	b < a
Relational Operator Replacemen	b > a	b => a

۱. **تعداد جهش‌یافته‌های تولید شده‌ی جدید نسبت به انتشار قبلی برنامه:** همانطور که در مقاله‌ی [۲۴] مطرح شده جهش‌یافته‌ها جایگزین خوبی برای خطاهای واقعی می‌باشند. زمانی که تعداد جهش‌یافته‌های جدید زیاد باشد یعنی تغییراتی که خطاخیزتر هستند بیشتر است. به منظور محاسبه‌ی این معیار لازم است خطوط اضافه شده به پرونده‌ی مورد نظر در ثبت کنونی، نسبت به انتشار قبلی مشخص شود و سپس تعداد جهش یافته‌هایی که این خطوط تولید می‌کنند شمرده شوند. این معیار در فرمول زیر خلاصه می‌شود.

$$NewMutants(C_i, f) = ||Mutants(C_i, f) - Mutants(LR(i), f)||$$

مثال اول : در شکل ۲.۳ مثالی از روند توسعه پرونده‌ی در طول یک انتشار آورده شده است. قسمت چپ این شکل نشان می‌دهد که پرونده‌ی Calculator در انتشار ۳.۲ دارای دو تابع جمع و تفریق بوده است. ثبت مربوط به انتشار ۳.۲، شماره‌ی دنباله‌ی ۱۵۲۲ را دارد یعنی از ابتدای پروژه تا این ثبت ۱۵۲۲ ثبت دیگر انجام گرفته است. با استفاده از CommitID نیز می‌توان این ثبت را از سامانه‌ی کنترل نسخه فراخوانی کرد. معیار توضیح داده شده در بالا را می‌خواهیم برای پرونده‌ی Calculator در ثبت نشان داده شد در قسمت راست شکل محاسبه کنیم. این شکل نشان می‌دهد که پروژه از انتشار قبلی تا ثبت مورد نظر ۵۱ ثبت دیگر داشته و در میان این آنها در دو ثبت پرونده‌ی Calculator تغییر کرده است. همانور که در شکل مشخص است برای محاسبه‌ی معیار تنها لازم است که ثبت مورد نظر و ثبت مربوط به آخرین انتشار در نظر گرفته شود. تعداد جهش‌یافته‌هایی که از هر خط تولید می‌شود در کنار آنها نوشته شده

است. عملگرهای استفاده شده جهت تولید جهش یافته در جدول ۲.۳ آمده است. در این مثال خطوط ۱۱ تا ۱۷ به پرونده اضافه شده است. از این خطوط می‌توان ۶ جهش یافته تولید کرد و در نهایت تعداد جهش‌یافته‌های جدید نسبت به انتشار قبلی برابر ۶ می‌شود.

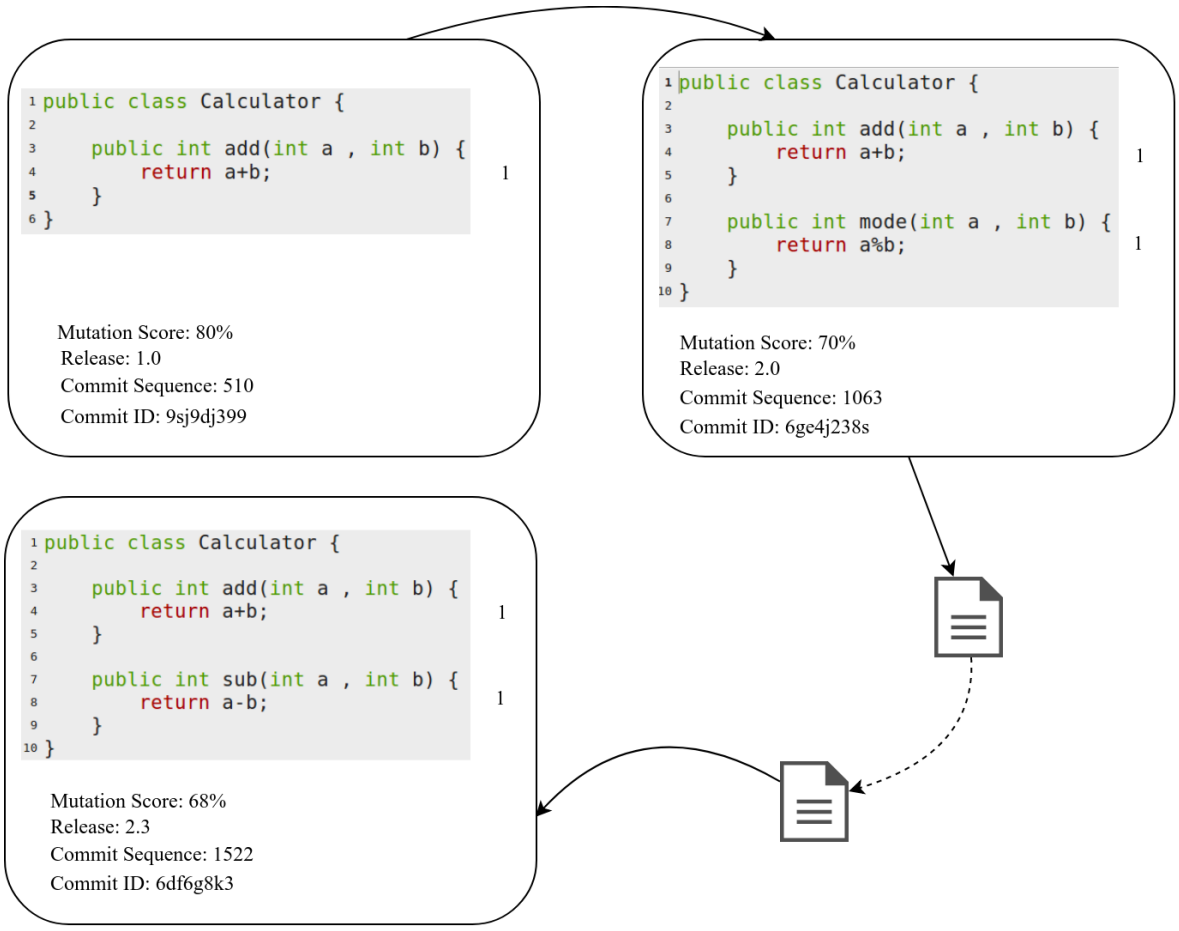


شکل ۲.۳: تاریخچه‌ی پرونده‌ی Calculator در مثال اول

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 چندانی با معیار قبلی نداشته 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 پرونده در طول توسعه‌ی نرم‌افزار اطلاعات اولیه مفید نخواهد بود. تعداد انتشارهای در نظر گرفته شده در این پایانامه 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. این معیار در فرمول زیر خلاصه می‌شود.

$$DistinctMutant(C_i, f) = \sum_{j=LR(c_i)-3}^{LR(c_i)} ||Mutants(R_{j+1}, f) - Mutants(R_j, f)||$$

مثال دوم: در شکل ۳.۳ روند توسعه‌ی پرونده‌ی Calculator از انتشار ۱ تا ۳.۲ نشان داده شده است. مشابه مثال قبل تعداد جهش‌یافته‌هایی که از هر خط توسط عملگرهای جدول ۲.۳ تولید می‌شود در کنار آن نوشته شده است. در این مثال قصد داریم معیار مطرح شده را برای پرونده‌ی Calculator در ثبت با شماره‌ی ۱۵۷۳ استخراج کنیم. انتشارهای ما قبل از این ثبت در شکل ۳.۳ نشان داده شده است. در انتشار دوم یک تابع اضافه شده که یک جهش یافته می‌توان در آن ایجاد کرد. در انتشار سوم تابع mode حذف شده تابع sub جایگزین آن شده است. در تابع جایگزین هم یک جهش یافته می‌توان استخراج کرد. در نهایت تعداد جهش یافته‌های متمایز در انتشارهای ما قبل برابر ۳ خواهد بود.



شکل ۳.۳: تاریخچه‌ی پرونده‌ی Calculator در مثال دوم و سوم

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 مثبت افزوده می‌شود. این معیار در فرمول زیر خلاصه شده است.

$$PositiveChanges(C_i, f) = \sum_{j=LR(c_i)-3}^{LR(c_i)} \delta^{+}\Big(MuScore(R_{j+1}, f) - MuScore(R_j, f)\Big)$$

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. این معیار در فرمول زیر خلاصه شده است.

$$NegetiveChanges(C_i, f) = \sum_{j=LR(c_i)-3}^{LR(c_i)} \delta^{-}\Big(MuScore(R_{j+1}, f) - MuScore(R_j, f)\Big)$$

مثال سوم: در این مثال تغییرات مثبت و منفی امتیاز جهش در انتشارهای قبل محاسبه می‌شود. برای این منظور روند توسعه‌ی پرونده‌ی Calculator در شکل ۳.۳ در نظر گرفته می‌شود. امتیاز جهش از انتشار اول به دوم ۱۰ درصد کاهش یافته و از انتشار دوم به سوم نیز دو درصد کاهش داشته است. در نتیجه مجموعه تغییرات مثبت صفر و مجموعه تغییرات منفی برابر ۱۲ و تغییرات مثبت صفر است.

۳.۳ معیارهای ترکیبی جهش_فرآیند

رویکرد سوم با توجه به مطالب گفته شده در مقاله‌ی [۱۹] 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 پرونده به یک اندازه بر پیچیدگی پرونده نمی‌افزایند و به عبارت دیگر موجب بروز خطا نمی‌شوند. به عنوان مثال در یک پرونده 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 و

[2] Comment
[3] Javadoc

میران پیچیدگی ندارند با این حال در محاسبه‌ی معیارهای پیش‌بینی خطا در نظر گرفته می‌شوند. هدف از ارائه‌ی معیارهای *ترکیبی جهش‌ـ فرآیند*[۴] بهبود کاستی‌های معیارهای فرآیند در چنین شرایطی است. در اینجا دو معیار *مقدار نرمال شده‌ی خطوط اضافه شده* و یا *کم شده* جهت اصلاح انتخاب شده‌اند. این دو معیار جز شاخص‌ترین معیارهای فرآیند هستند.

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

$$M_1 = number\ of\ lines\ added\ \times\ number\ of\ muatants\ derived$$

$$M_2 = number\ of\ lines\ deleted\ \times\ number\ of\ mutants\ derived$$

با وجود مناسب بودن ایده ی اولیه با بررسی‌های بیشتر دو مشکل در معیارهای فوق مشخص می‌شود.

مشکل اول : هدف از ارئه ی 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 پرونده اضافه شده است. معیار *مقدار نرمال شده‌ي 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 ارایه ی معیار 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 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 [۲۴] اشاره شده که جهش یافته ها جایگزین خوبی برای خطاهای واقعی هستند. این نگاشت برابر است با تعداد جهش یافته های تولید شده در آن خط.

[4]Process-Mutation Hybrid

مشکل دوم: این معیار برای عملکرد هرچه بهتر مشابه معیار *مقدار نرمال شده‌ی خطوط اضافه شده* نیاز به نرمال‌سازی دارد. به جهت نرمال‌سازی نمی‌توان از همان روش استفاده کنیم چراکه در آن وزن‌دهی به خطوط 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 مقاله‌ی [۳۱] اشاره شده که تعداد ثبت‌ها می‌تواند نشانگر میزان تغییرات باشد. بنابرین از تعداد ثبت‌های کل پروژه در طول یک انتشار به منظور نرمال‌سازی استفاده خواهد شد.

$$NormalWeightedAddedLines(c_i, f) = \frac{\sum_{j=Seq(C(LR(c_i)))}^{i} ||Mutants(C_j, f) - Mutants(C_{j-1,f})||}{i - Seq(C(LR(c_i)))}$$

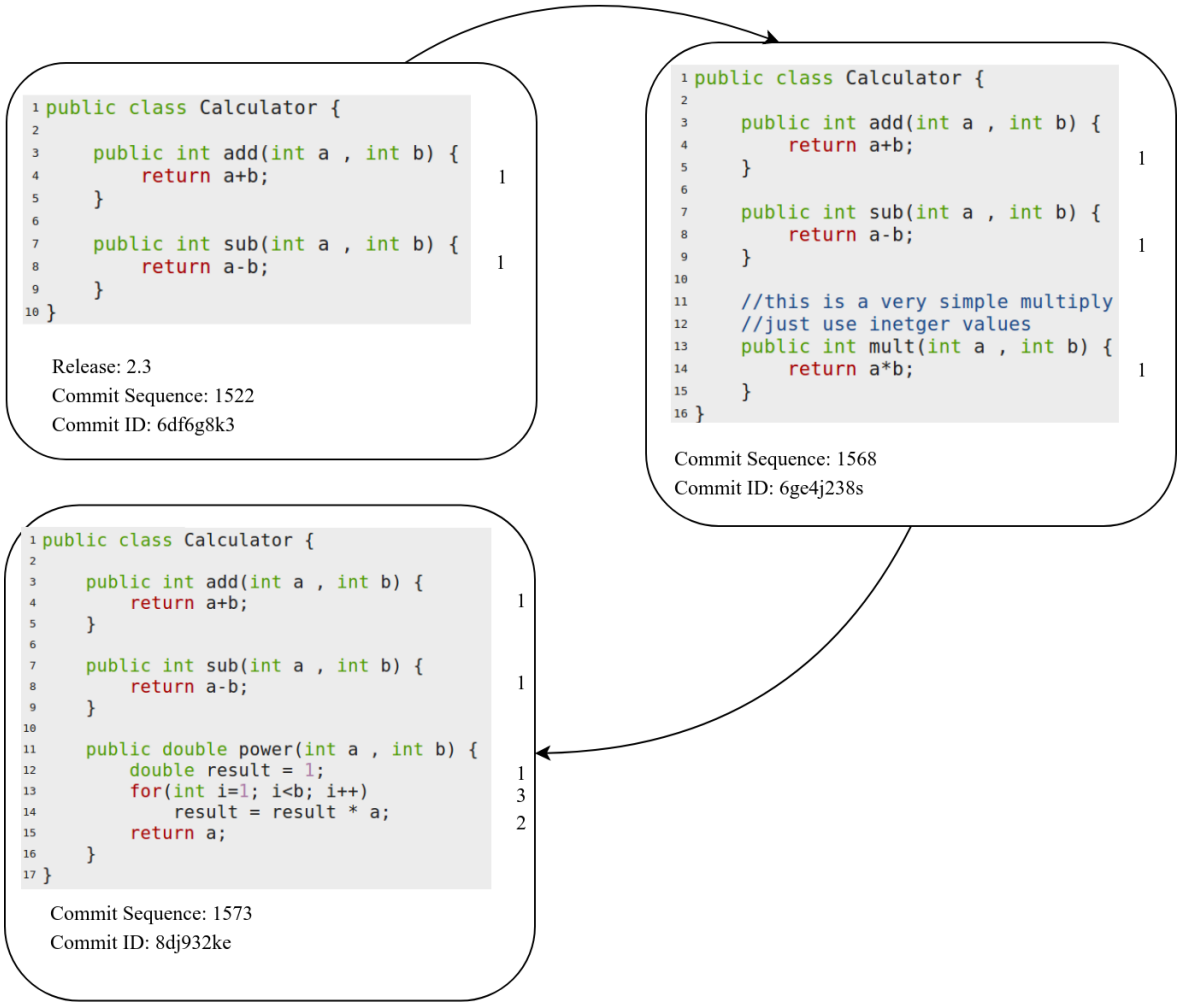
$$NormalWeightedDeletedLines(c_i, f) = \frac{\sum_{j=Seq(C(LR(c_i)))}^{i} ||Mutants(C_{j-i}, f) - Mutants(C_j, f)||}{i - Seq(C(LR(c_i)))}$$

مثال چهارم: در این مثال معیارهای مقدار نرمال شده‌ی خطوط اضافه‌ی وزن دهی شده و خطوط حذف شده برای شکل ۴.۳ محاسبه می‌شود. زمانی که قرار است برای پرونده‌ی Calculator در ثبت ۱۵۷۳ معیارها محاسبه شود لازم است آخرین انتشار و ثبت‌های ماقبل بین انتشار و ثبت ۱۵۷۳ بررسی شوند. آخرین انتشار برای این ثبت انتشار ۳.۲ می‌باشد و در این بین این پرونده تنها در ثبت شماره‌ی ۱۳۳۹ تغییر کرده است.

در ثبت ۱۵۶۸ خطوط ۱۱ تا ۱۶ نسبت به ثبت قبلی (انتشار ۳.۲) تغییر کرده است. از این خطوط تنها یک جهش‌یافته را می‌توان تولید کرد. در ثبت ۱۵۷۳ خطوط ۱۱ تا ۱۷ جایگزین شده‌اند که این خطوط می‌توانند ۶ جهش یافته را تولید کنند. تعداد کل ثبت‌های پروژه در این بازه برابر نیز برابر ۵۱ (۱۵۲۲_۱۵۷۳) است.

$$NormalWeightedAddedLine = \frac{1+6}{51} = 0.137$$

$$NormalWeightedDeletedLine = \frac{1}{51} = 0.019$$



شکل ۴.۳: تاریخچه‌ی پرونده‌یCalculator در مثال چهارم

JPredict ۴.۳

جهت آگاهی از عملکرد معیارهای مطرح شده لازم است این معیارها استخراج شوند با استفاده از آنها مدل پیش‌بینی ساخته شود و با یکدیگر مقایسه شوند. جهت انجام این وظایف ابزار در این پایانامه ابزار JPredict ارائه گردیده است که می‌تواند این وظایف را به صورت خودکار انجام دهد. همچنین قابلیت گسترش جهت کار با انواع مجموعه‌داده‌ها و استخراج معیارهای تعریف شده‌ی جدید توسط کاربر را دارد.

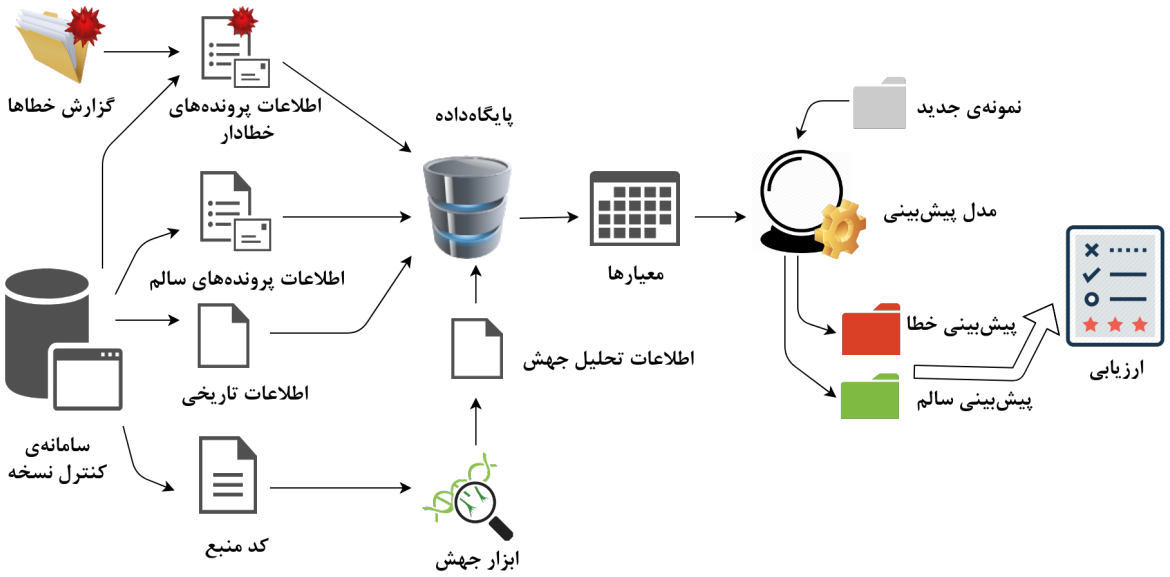
نمای کلی از فرآیندهایی که در ابزار Jpredict انجام می‌گیرد در شکل ۵.۳ آمده است. این شکل نشان

می‌دهد که در ابتدا انواع مختلفی از اطلاعات لازم است که از منابع متفاوت بدست آید. ابتدا اطلاعات پرونده‌های حاوی خطا از گزارش‌های خطا بیرون بدست می‌آید. این گزارش‌ها می‌توانند داده‌های موجود در سیستم ردگیری خطا و یا مجموعه‌داده‌ی مرتبط با خطاهای پروژه‌های نرم‌افزاری باشد. همچنین تعدادی از پرونده‌های بدون نیز انتخاب می‌شوند. این انتخاب به صورت تصادفی انجام می‌شود و بسته به خواست کاربر محدودیت‌هایی در انتخاب در نظر گرفته می‌شود. اطلاعات این دو نوع پرونده با استفاده از سامانه‌ی کنترل نسخه تکمیل می‌گردد و در پایگاه داده ذخیره می‌شود. نوع دیگری از اطلاعات که از سامانه‌ی کنترل نسخه استخراج می‌شود اطلاعات تاریخی مربوط به توسعه‌ی پروژه‌ی نرم‌افزاری است. این اطلاعات بسته به معیارهایی که قرار است از آنها استخراج شود می‌تواند برای تمامی پرونده‌های موجود در سامانه‌ی کنترل نسخه استخراج شود و یا تنها برای پرونده‌های حاوی خطا و سالم انتخاب شده. معمولا در صورتی که استخراج اطلاعات پرهزینه باشد تنها برای پرونده‌های انتخابی اطلاعات تاریخی استخراج می‌شود.

از آنجا که معیارهای جهش نیز باید محاسبه شوند لازم است تا برای پرونده‌های انتخابی کد منبع پروژه‌ی مرتبط از سامانه‌ی کنترل نسخه بازیابی شود سپس با استفاده از یکی از ابزارهای جهش بر روی پرونده تحلیل جهش انجام می‌گیرد. این دسته از اطلاعات نیز در پایگاه‌داده ذخیره می‌شود.

معیارهای مورد نظر با استفاده *پرسمان*[5] مناسب از اطلاعات موجود در پایگاه داده استخراج می‌شوند و از آنها در ساخت مدل پیش‌بینی استفاده می‌گردد. مدل پیش‌بینی نمونه‌ی جدید را دریافت می‌کند و سالم یا خطادار بودن آنرا پیش‌بینی خواهد کرد. در واقع این نمونه یک بردار ویژگی از معیارهای استفاده شده در ساخت مدل است. در نهایت با توجه به نتایج مدل ساخته شده ارزیابی می‌گردد.

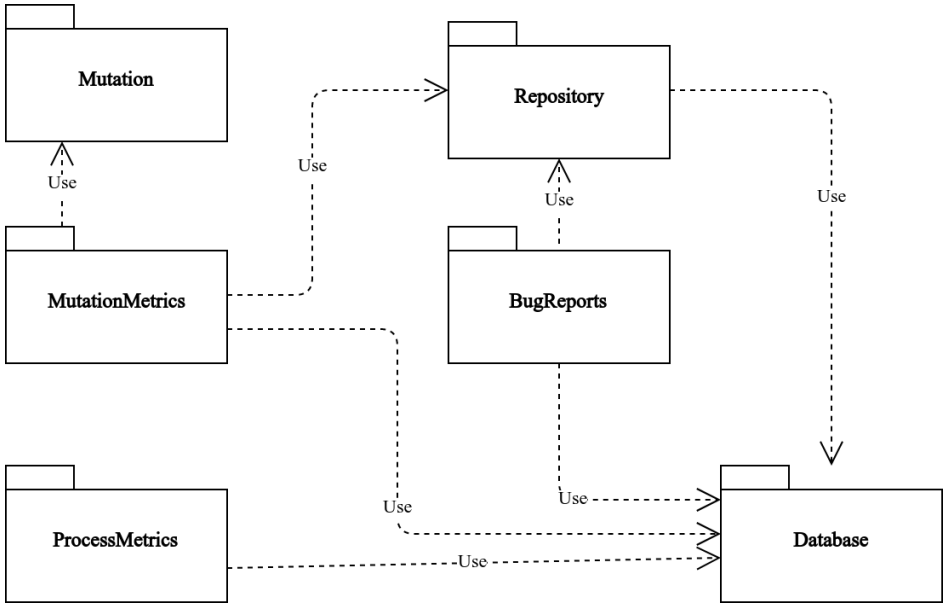
[5] Query



شکل ۳.۵: نمایی کلی از فرآیندهای موجود در JPredict

۱.۴.۳ ساختار Jpredict

واحدهای اصلی تشکیل دهنده‌ی Jpredict در شکل ۶.۳ نشان داده شده است. واحد Mutation وظیفه‌ی تولید جهش و انجام تحلیل را بر عهده دارد و در واقع رابط میان Jpredict و ابزار خارجی آزمون جهش می‌باشد. واحد Repository ارتباط با سامانه‌ی کنترل نسخه‌ی پروژه‌های مورد آزمایش را بر عهده دارند. اطلاعات لازم را بازیابی می‌کند و کد منبع ثبت‌های مختلف را در مسیری قرار می‌دهد تا ابزار جهش با آن کار کند. واحد BugReport اطلاعات پرونده‌های حاوی خطا را از منبع بیرونی دریافت می‌کند و با استفاده از واحد Repository آنها را تکمیل می‌کند. انتخاب پرونده‌های سالم نیز بر عهده‌ی این واحد می‌باشد. واحد MutationMetric دستورات مرتب با جهش به واحد Mutation می‌دهد و نتایج اولیه را در پایگاه داده ذخیره می‌کند و با اجرای پرسمان مناسب معیارها را محاسبه می‌نماید. واحد ProcessMetric نیز از داده‌هایی که واحد Reposiotry بر روی پایگاه داده ذخیره کرده است استفاده می‌کند و معیارهای فرآیند را محاسبه می‌کند. واحد DataBase وظیفه‌ی ارتباط با پایگاه داده را دارد و همچنین حاوی پرسمان‌های مختلف است. در ادامه به بررسی جزیی‌تر هریک از واحدها پرداخته می‌شود.



شکل ۶.۳: نمایی از واحدهای تشکیل دهنده‌ی JPredict

۲.۴.۳ واحد Repository

همانطور که اشاره شد این واحد دو وظیفه‌ی اصلی دارد: استخراج اطلاعات از سامانه‌ی کنترل نسخه و دریافت کد منبع برای یک پروژه‌ی خاص.

با توجه به مرور معیارهای استفاده شده در پژوهش‌های پیشین که در قسمت ۳.۱.۲معرفی شدند و معیارهای انتخاب شده در قسمت ۱.۳ لازم است اطلاعات زیر استخراج گردد.

- اطلاعات ثبت‌های مختلف در پروژه شامل شماره‌ی ثبت در سامانه‌ی کنترل نسخه، نام ثبت‌کننده، پرونده‌های تغییر یافته در ثبت، تعداد خطوط حذف و اضافه شده
- انتشار قبلی ثبت‌ها
- مشارکت‌کنندگان در یک پرونده در زمان ثبت و میزان مشارکت آنها

پس از استخراج هر یک از این اطلاعات لازم است آنها به نحو مناسبی در پایگاه داده قرار گیرند. این واحد چهار جدول در پایگاه داده می‌سازد که در زیر مشخص شده‌اند:

- CommitInfo : حاوی اطلاعات ثبت‌ها
- CommitChangedFiles : حاوی اطلاعات پرونده‌های تغییر یافته در هر ثبت نسبت به ثبت قبلی

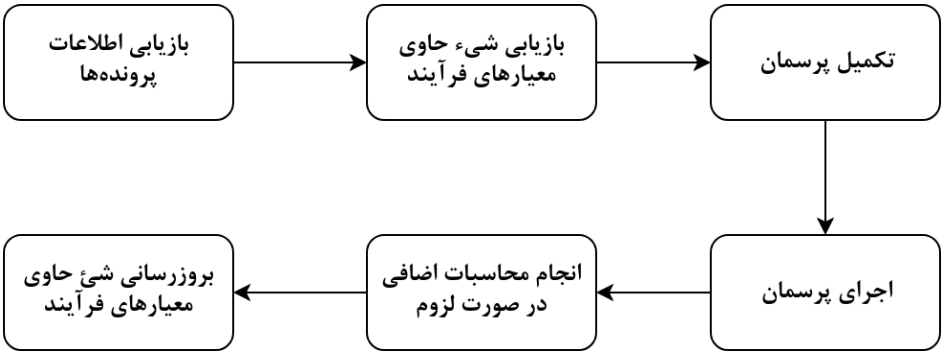
- ProjectRelease : انتشارهای موجود در پروژه‌ها
- Participation : مشارکت کنندگان در پرونده و میزان مشارکت آنها به تفکیک ثبت

وظیفه‌ی دیگر این واحد بازیابی کد منبع پروژه در یک ثبت خاص می‌باشد. جهت انجام این امر، برای هر سامانه‌ی کنترل نسخه لازم است از کتابخانه‌ی مناسب با آن کمک گرفته شود. یکی دیگر از وظایف این سامانه مشخص کردن تفاوت میان دو ثبت از پروژه است. از این قابلیت هم در مشخص کردن تعداد خطوط کم و اضافه شده در جدول CommitChangedFiles استفاده می‌شود و هم در محاسبه‌ی معیارهای ارائه شده‌ی جدید.

۳.۴.۳ واحد ProcessMetrics

معیارهای فرآیند در جدول متناظری ذخیره می‌شوند که این جدول قبل از محاسبه‌ی معیارها مقداردهی اولیه می‌شود. هر سطر از این جدول به یکی از پرونده‌هایی که لازم است معیارها برای آن محاسبه شود اختصاص می‌یابد. این امر سبب می‌شود محاسبه‌ی هر معیار مستقل از دیگر انجام گیرد و امکان بروزرسانی داشته باشد. نحوه‌ی محاسبه‌ی یک معیار فرآیند در واحد ProcessMetrics در شکل ۷.۳ آمده است. در این فرآیند ابتدا اطلاعات پرونده‌ها از پایگاه داده بازیابی می‌گردد. سپس برای هر پرونده شئ مربوط به آن که حاوی معیارهای فرآیند است بازیابی می‌گردد. این شئ معادل یک سطر از جدول حاوی معیارهای فرآیند در پایگاه داده است. سپس لازم است که یک یا چند پرسمان تکمیل شود. این پرسمان‌ها از قبل در واحد Database قرار دارند که نیازمند تکمیل هستند. پس از تکمیل آنها به واحد Database داده می‌شوند تا اجرا شوند. در صورت نیاز محاسبات بیشتری بر روی نتایج پرسمان انجام می‌گیرد. در نهایت شئ حاوی معیارهای فرآیند بروزرسانی می‌شود.

به منظور محاسبه‌ی معیار فرآیند یک کلاس انتزاعی در نظر گرفته شده است که شامل مراحل نشان داده شده در شکل ۷.۳ است. هر معیار به طور جداگانه گسترشی از این کلاس خواهد بود که قسمت‌های انتزاعی را پیاده‌سازی می‌کند. بدین ترتیب امکان افزودن معیار جدید فراهم می‌گردد.



شکل ۷.۳: فرآیند محاسبه‌ی یک معیار فرآیند در Jpredict

۴.۴.۳ واحد BugReports

این واحد با منبع خارجی ارتباط برقرار می‌کند و اطلاعات پرونده‌های حاوی خطا را دریافت می‌کند. همچنین بر طبق محدودیت‌های از پیش تعیین شده تعدادی فایل سالم را از پروژه در یک ثبت خاص انتخاب کند. این انتخاب می‌تواند به صورت تصادفی انجام پذیرد و فایل‌های سالم نیز توسط منبع خارجی مشخص شود. سپس به کمک واحد Repository اطلاعات پرونده تکمیل می‌گردد و در پایگاه داده ذخیره می‌شود. اطلاعات مربوط به پرونده‌های حاوی خطا در جدول BugInfo و پرونده‌های سالم در جدول CleanInfo قرار می‌گیرد. اطلاعاتی که در این دو جدول وجود دارد عبارتند از :

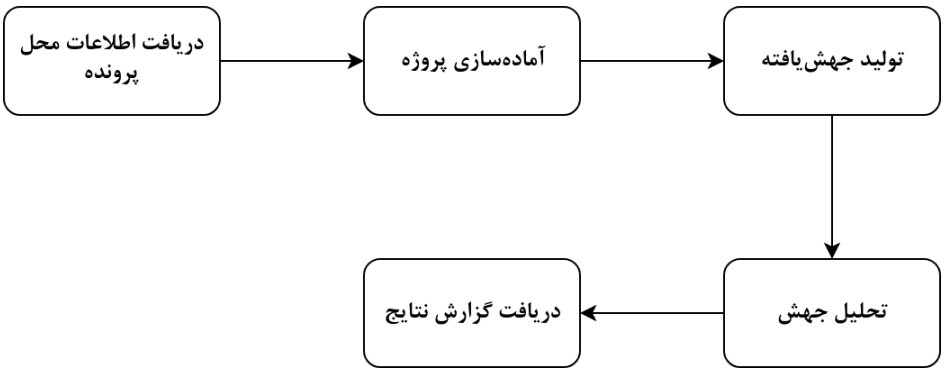
- نام پرونده‌ی حاوی خطا
- شماره‌ی خطا (در صورتی که یک خطا بیش از چند پرونده را شامل می‌شود)
- شماره‌ی ثبتی که در آن در پرونده‌ی مورد نظر خطا رخ داده
- نام انتشار قبلی
- شماره‌ی ثبت انتشار قبلی
- نام پروژه (در حالتی که اطلاعات خطای چندید پروژه‌ی مختلف وجود دارد)

۵.۴.۳ واحد Mutation

این واحد ارتباط میان Jpredict و ابزار جهش را برقرار می‌سازد. فرآیندهایی که در این واحد انجام می‌شوند در شکل ۸.۳ آمده است. محل پرونده‌ی مورد نظر توسط واحد MutationMetric به این واحد داده می‌شود.

این محل شامل سایر پرونده‌های پروژه نیز می‌شود. سپس پروژه آماده می‌گردد. این آماده‌سازی به این جهت است که ابزارهای جهش اغلب نیازمند آن هستند که پروژه کامپایل شود و برای کامپایل صحیح آنها لازم است که پیکربندی‌هایی انجام شود و یا وابستگی‌های پروژه اضافه شود. پس از آماده سازی جهش‌یافته‌ها برای پرونده‌ی مورد نظر ساخته می‌شوند. سپس در صورت لزوم تحلیل جهش انجام می‌گردد. ممکن است به تحلیل جهش نیازی نباشد چراکه برخی معیارها تنها با تولید جهش یافته محاسبه می‌گردد. در پایان باید گزارش از ابزار جهش دریافت شود و به شکل مناسب جهت استفاده در واحد MutationMetric در بیاید.

به منظور انجام این فرآیندها یک کلاس انتزاعی در نظر گرفته شده است که شامل مراحل نشان داده شده در شکل ۸.۳ است. برای انجام عملیات جهش بر روی هر پروژه می‌توان این کلاس را گسترش داد تا آماده‌سازی‌ها متناسب با آن پروژه پیاده‌سازی شود. بدین ترتیب امکان انجام جهش بر روی پروژه‌های جدید فراهم می‌گردد.



شکل ۸.۳: فرآیندهای واحد Mutation

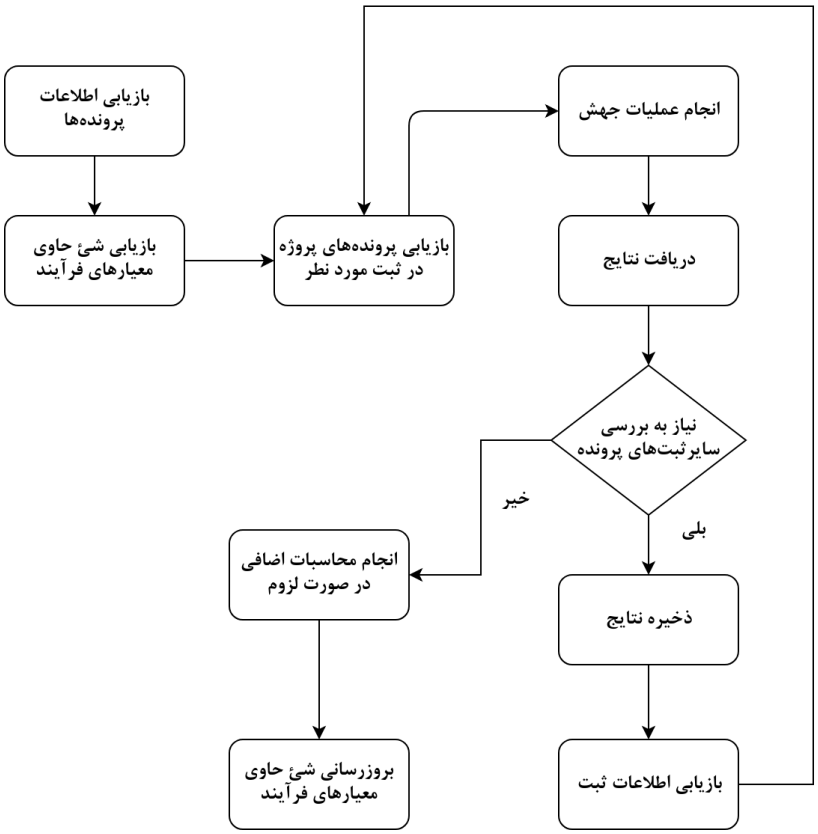
۶.۴.۳ واحد MutationMetrics

معیارهایی که به وسیله‌ی جهش به دست می‌آیند مشابه معیارهای فرآیند در جدول متناظری ذخیره می‌شوند که این جدول قبل از محاسبه‌ی معیارها مقداردهی اولیه می‌شود. هر سطر از این جدول به یکی از پرونده‌هایی که لازم است معیارها برای آن محاسبه شود اختصاص می‌یابد. نحوه‌ی محاسبه‌ی یک معیار بر اساس جهش در شکل ۹.۳ آمده است. در این فرآیند ابتدا اطلاعات پرونده‌ها از پایگاه داده بازیابی می‌گردد. سپس برای هر پرونده شئ مربوط به آن که حاوی معیارهای مبتنی بر جهش است بازیابی می‌گردد. با استفاده از واحد Repository پرونده‌های ثبت مورد نظر بازیابی می‌شود. سپس با استفاده از واحد Mutation عملیات جهش انجام می‌گیرد. در محاسبه‌ی معیارهای جهش نیاز به سایر ثبت‌های پرونده نیست. در دو دسته‌ی دیگر لازم است که سایر ثبت‌ها نیز بازیابی شود و عملیات جهش بر روی آنها نیز انجام شود در این صورت نتایج میانی در پایگاه داده ذخیره

می‌شود. در حالتی که نتایج میانی وجود دارد، محاسبات اضافی انجام خواهد گرفت. در نهایت معیار محاسبه شده در شئ حاوی معیارها قرار می‌گیرد.

داده‌هایی که برای محاسبه‌ی معیارهای جدید ارائه شده لازم است محاسبه شود در زیر آمده است. همچنین جدولی که داده‌ها در آن قرار می‌گیرد مشخص شده است.

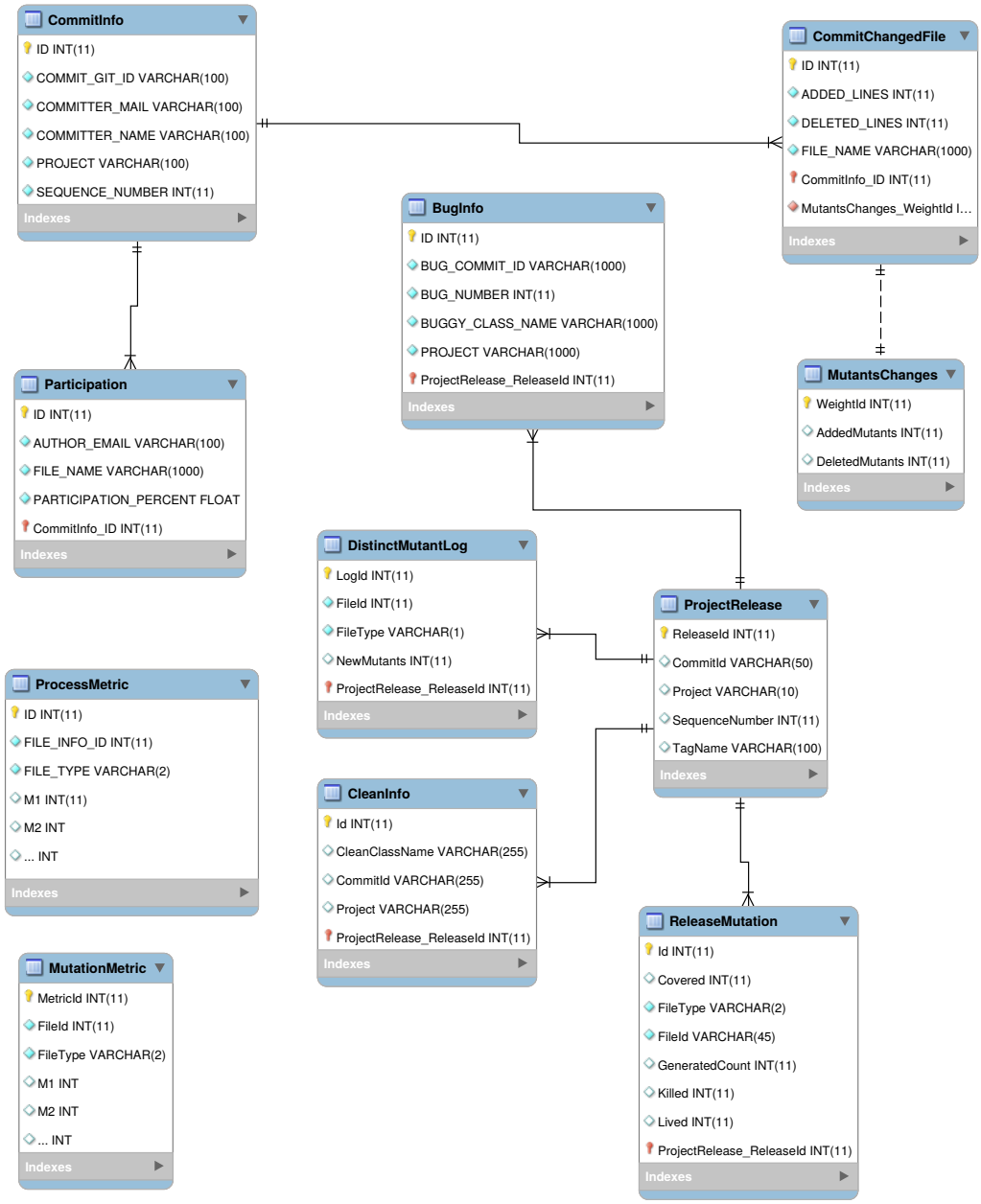
- تعداد جهش‌یافته‌های جدید برای یک پرونده در ثبت مورد نظر نسبت به آخرین انتشار قبلی: این معیار نتایج میانی ندارد و پس از محاسبات مستقیما در کنار سایر معیارهای مبتنی بر جهش قرار می‌گیرد.
- تعداد جهش‌یافته‌های جدید برای یک پرونده در یک انتشار نسبت به انتشار قبلی: این داده‌ها در جدولی به نام DistinctMutantLog قرار می‌گیرد.
- تعداد جهش‌یافته‌های اضافه و کم شده برای یک پرونده در یک ثبت نسبت به ثبت قبلی: داده‌ها در جدول MutantsChange قرار می‌گیرد.
- تغییرات امتیاز جهش برای یک پرونده در یک انتشار نسبت به انتشار قبلی: داده‌ها در جدول Release-Mutation قرار می‌گیرد.



شکل ۹.۳: فرآیندهای واحد MutationMetrics

۷.۴.۳ واحد Database

همانطور که بیان شد این واحد ارتباط میان پایگاه‌داده و JPredict را برقرار می‌کند. در این واحد یک کلاس انتزاعی به منظور دسترسی به جداول قرار داده شده است. برای هر جدول یک گسترش از کلاس انتزاعی ساخته می‌شود به نمونه‌های این کلاس‌ها، شئ دسترسی به داده گفته می‌شود. پرسمان‌های مربوط به هر جدول نیز از طریق کلاس متناظر اجرا می‌گردد. نمودار EER جداول ساخته شده در شکل ۱۰.۳ آمده است.



شکل ۳.۱۰: نمودار EER جداول ساخته شده

۸.۴.۳ محاسبه‌ی معیارها در روش پیشنهادی

محاسبه‌ی معیارهای مطرح شده با استفاده از روش پیشنهادی به طور کلی به این صورت است که پرسمان‌هایی طراحی می‌شود تا به وسیله‌ی آنها معیارها محاسبه شود. در قسمت زیر پرسمان‌های لازم برای هر معیار آورده شده است. این پرسمان‌ها را می‌توان متناسب با هر نوع پایگاه داده پیاده‌سازی نمود. لازم به ذکر است که در صورتی

که از جدوال برای نگهداری اطلاعات چندین پروژه‌ی مختلف استفاده شود لازم است شرط تعلق به پروژه‌ی مورد نظر به پرسمان‌های زیر افزوده شود. در این پرسمان‌ها به منظور تمایز و شناسایی افراد از ایمیل آنها می‌توان استفاده کرد. امکان استفاده از هر شناساگر دیگر که بتوان به وسیله‌ی آن افراد را شناسایی و متمایز کرد وجود دارد. البته این شناساگر باید در تمام جداول یکسان باشد.

- تعداد ثبت: تعداد سطرهای جدول CommitChangedFile که نام آنها برابر پرونده‌ی مورد نظر است و ثبت متناظر با آن سطر در بازه‌ی انتشار قبلی تا ثبت مورد نظر پرونده قرار دارد.

- تعداد توسعه‌دهندگان فعال: تعداد متمایز ثبت‌کننده‌گان از جدول CommitInfo که ثبت متناظر با آن در بازه‌ی آخرین انتشار و ثبت مورد نظر است. همچنین شناسه‌ی ثبت در بین شناسه‌هایی از جدول CommitChangedFile باشد که برای آن شناسه‌ی ثبت فایل مورد نظر تغییر کرده است.

- تعداد توسعه‌دهندگان متمایز: تعداد متمایز ثبت‌کننده‌گان از جدول CommitInfo که ثبت متناظر با آن در بازه‌ی اولین ثبت از پروژه تا ثبت مورد نظر است. همچنین شناسه‌ی ثبت در بین شناسه‌هایی از جدول CommitChangedFile باشد که برای آن شناسه‌ی ثبت فایل مورد نظر تغییر کرده است.

- مقدار نرمال‌سازی شده‌ی تعداد خطوط اضافه شده: برای این معیار لازم است از دو پرسمان استفاده شود. پرسمان اول مجموع خطوط اضافه شده در پرونده را می‌شمارد و پرسمان دوم مجموع خطوط اضافه شده در پروژه را می‌شمارد.

  پرسمان اول: مجموع خطوط اضافه شده در سطرهایی از جدول CommitChangedInfo که ثبت متناظر با آنها در بازه‌ی آخرین انتشار و ثبت مورد نظر است و همچنین آن سطر متعلق به پرونده‌ی مورد نظر می‌باشد. پرسمان دوم: مجموع خطوط اضافه شده در سطرهایی از جدول CommitChangedInfo که ثبت متناظر با آنها در بازه‌ی آخرین انتشار و ثبت مورد نظر است.

  به منظور محاسبه معیار پرسمان اول بر دوم تقسیم می‌گردد.

- مقدار نرمال‌سازی شده‌ی تعداد خطوط اضافه شده: مشابه معیار قبل با این تفاوت که مجموع خطوط حذف شده در نظر گرفته می‌شود.

- درصد خطوطی که مالک پرونده مشارکت کرده: مقدار حداکثر از میان سطرهای جدول Participation که متعلق به پرونده‌ی مورد نظر در ثبت مورد نظر است.

- تعداد مشارکت‌کنندگان جزیی: تعداد ایمیل‌های نویسندگان در جدول Participation در سطرهایی که متعلق به پرونده‌ی مورد نظر در ثبت مورد نظر است و میزان مشارکت آنها کمتر از ۵ درصد است.

- تعداد ثبت‌های همسایگان: برای این معیار دو پرسمان لازم است. پرسمان اول همسایگان و تعداد همسایگی پیدا می‌کند. پرسمان دوم تعداد ثبت‌ها را می‌شمارد که درباره‌ی آن توضیح داده شد.
پرسمان اول: سطرهایی از جدول CommitChangedFile که ثبت متناظر در میان ثبت‌هایی است که در آنها فایل مورد نظر تغییر کرده و آن ثبت در بازه‌ی انتشار قبلی تا ثبت مورد نظر انجام شده. همچنین آن سطر نباید متعلق به پرونده‌ی مورد نظر باشد. سطرهای انتخاب شده باید بر اساس نام پرونده‌ی آنها گروهبندی شود و در این گروه‌بندی تعداد سطرها در هر گروه مشخص شود.

- تعداد توسعه‌دهندگان فعال همسایگان: مشابه معیار قبل با این تفاوت که در پرسمان دوم معیار توسعه‌دهنگان فعال محاسبه می‌شود.

- تعداد توسعه‌دهندگان متمایز همسایگان: مشابه معیار قبل با این تفاوت که در پرسمان دوم معیار توسعه‌دهنگان متمایز محاسبه می‌شود.

- تجربه‌ی مالک پرونده: این معیار به دو پرسمان نیاز دارد. یک پرسمان جهت یافتن مالک پرونده و پرسمان دیگر تعداد ثبت‌های فرد یافت شده در پروژه را می‌شمارد.
پرسمان اول: فردی که در جدول Participation میزان مشارکت وی در پرونده در ثبت مورد نظر برابر بیشترین میزان مشارکت در بین سطرهای متعلق به آن پرونده در آن سطر است.
پرسمان دوم: تعداد ثبت‌هایی که توسط فرد مشخص شده در پرسمان اول در بازه‌ی آخرین انتشار و ثبت مورد نظر انجام شده است.

- تجربه‌ی تمام توسعه‌دهندگان: این معیار دو پرسمان احتیاج دارد. پرسمان اول لیست تمام توسعه‌دهندگان را پیدا می‌کند و پرسمان دوم برای هر فرد تجربه‌ی وی را استخراج می‌کند که برابر پرسمان دوم معیار قبلی است.

-

اکثر معیارهای مبتنی بر جهش به پرسمان خاصی جهت محاسبه نیاز ندارند و بیشتر پرسمان‌ها جهت بازیابی اطلاعات پرونده‌ها و نتایج میانی است. دو پرسمان پرکاربرد در زیر آمده‌اند. علت ایجاد جداول اضافی در

محاسبه‌ی این معیارها پایداری در انجام محاسبات است. به علت زمان‌بر بودن عملیات جهش در صورت توقف محاسبات امکان از سرگیری محاسبات از محل توقف وجود دارد و همچنین با نگهداری به عنوان یک مجموعه داده می‌تواند در پژوهش‌های دیگر به کار گرفته شود. همچنین مزایای عمومی استفاده از پایگاه داده در محاسبه‌ی معیارها در مورد این دسته نیز صادق است.

- چهار انتشار اخیر یک ثبت: از جدول ProjectRelease انتشارهایی از پروژه انتخاب می‌شوند و شماره‌ی دنباله‌ی آنها از شماره دنباله‌ی انتشار قبلی ثبت مورد نظر کمتر است. این سطرها بر اساس شماره‌ی دنباله نزولی مرتب می‌شوند و چهار انتشار برتر انتخاب می‌شوند.

- ثبت‌هایی که در آنها پرونده‌ی مورد نظر تغییر کرده: ثبت‌هایی از جدول CommitInfo انتخاب می‌شوند که در بازه‌ی انتشار قبلی تا ثبت کنونی انجام شده‌اند و در جدول CommitChangedFile سطری برای آن ثبت وجود داشته باشد که نشان دهد در آن پرونده‌ی مورد نظر تغییر کرده است. همچنین ثبت‌های انتخاب شده باید بر اساس شماره‌ی دنباله مرتب شوند.

فصل ۴

مورد مطالعاتی

در این فصل مطالعه‌ی موردی بر روی مجموعه‌داده‌ی defects4j [۳۲] 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 مجموعه‌داده‌ی فراهم شده معیارهای بیان شده در فصل ۳ استخراج شوند. مجموعه‌داده‌ی 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*[1] 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.

[1] Configuration

۱.۲.۴ مجموعه داده defect4j

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

- اطلاعات خطاهای پروژه وجود داشته باشد و این اطلاعات نشان دهد که خطا متعلق به کدام پرونده در کدام ثبت است.
- پروژه‌ها متن‌ـباز باشد تا بتوان با استفاده از *کد منبع*[۲] 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 *متن‌ـباز*[۳] 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*[۴] 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

[2] Source Code
[3] Open-source
[4] Framework

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.

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*[5] 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 توزیع‌کننده‌ی جاوا است. جاوا دو توزیع کننده ی عمده دارد. یکی OpenJDK و دیگری Oracle می‌باشد. با 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*[6] لازم با شکست مواجه می شوند.

راه ارتباط با این ابزار *خط دستور*[7] می‌باشد و یک نمونه از دستورات قابل استفاده در این ابزار در شکل ۱.۴ است که این دستور اطلاعات مربوط به پروژه‌ی Lang و خطای شماره‌ی یک را خواهد داد.

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

[5] Github
[6] Dependency
[7] Command 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 جهش‌یافته و تحلیل جهش استفاده می‌شود. یک ابزار دیگر در این حوزه پیت[8] 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*[9] 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 جهش‌یافته استفاده می‌کند که گسترش یافته‌ی یک کامپایلر جاوا است. استفاده از این ابزار را می‌توان در سه مرحله خلاصه کرد :

[8] PIT - http://pitest.org/
[9] Gradle

۱. **پیکربندی تولید جهش‌یافته به وسیله‌ی 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 و سپس با MMLC کامپایل می‌شود و نتیجه به عنوان یکی از پارامترها در هنگام فراخوانی به ابزار Major ارسال می شود. نمونه‌ای از این کد در شکل ۲.۴ آمده است.

```
targetOp{
    // Define the replacements for ROR
    BIN(>)->{>=,!=,FALSE};
    BIN(<)->{<=,!=,FALSE};
    BIN(>=)->{>,==,TRUE};
    BIN(<=)->{<,==,TRUE};
    BIN(==)->{<=,>=,FALSE,LHS,RHS};
    BIN(!=)->{<,>,TRUE,LHS,RHS};
    // Define the replacements for COR
    BIN(&&)->{==,LHS,RHS,FALSE};
    BIN(||)->{!=,LHS,RHS,TRUE};
    // Define the type of statement that STD should delete
    DEL(RETURN);

    // Enable the STD, COR, and ROR mutation operators
    STD;
    COR;
    ROR;
}
// Call the defined operator group for the target method
targetOp<"triangle.Triangle::classify(int,int,int)">;
```

شکل ۲.۴: نمونه کد MML در Major

۲. **تولید جهش‌یافته‌ها:** همانطور که اشاره شد ابزار Major جهت تولید نسخ جهش‌یافته نیاز به کامپایل 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 کامپال پروژه
- معرفی وابستگی‌ها و کامپایل موارد آزمون
- اجرای موارد آزمون و ارائه‌ی 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 کافیست کامپایلر مورد استفاده در قسمت کامپایل پروژه را کامپایلر 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*[۱۰] در ابزار Maven این پرونده تبدیل به یک پرونده 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 کافیست کامپایلر مورد استفاده در قسمت کامپایل پروژه را کامپایلر 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

[10] Plugin

```
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 آزمون بر روی جهش‌یافته‌هایی که تاکنون کشته نشده‌اند اجرا می‌شود. در پایان نتایج در خروجی چاپ می‌شوند. همچنین نتایج در پرونده‌های با پسوند csv قرار می‌گیرد. نمونه‌ای از اجرای تحلیل جهش در شکل ۵.۴ و پرونده‌ی نتایج خروجی در شکل ۶.۴ نشان داده شده.

```
compile.tests:
    [javac] Compiling 3 source files to /home/ali/project/defects4j/major/example/a
nt/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 ۶.۴: نتایج خروجی تحلیل جهش

۳.۲.۴ کتابخانه‌ی Jgit

این کتابخانه جهت کار با مخازن نرم‌افزاری که 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 بر طرف 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 ابزار ها *ORM*[11] می‌گویند. در ابتدا تصمیم بر این بود که داده‌های بدست آمده در پرونده متنی ذخیره شوند و در هنگام نیاز آن‌ها خوانده شوند یا همه ی 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 پرونده 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 پرونده 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

[11] Object Relational Mapping

ندارد بجز پیکربندی اولیه مانند آدرس پایگاه داده. همچنین در تمام مراحل سعی شده است که اصول لازم 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*[۱۲] : جهت پاسخ به این نیازمندی از پایگاه داده استفاده شده است.

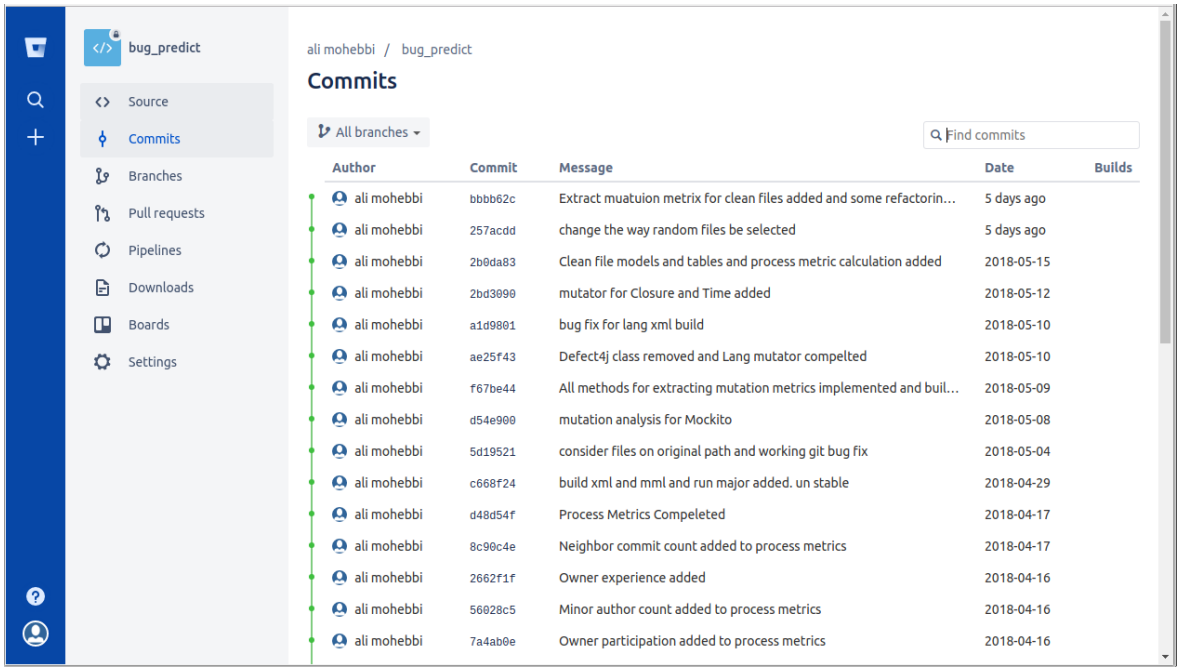
۲. قابلیت نگهداری: این قابلیت از سایرین بیشتر حائز اهمیت است. زیرا پروژه‌های پروژهشی معمولا به صورت مستقیم کاربران عمومی ندارند و از این جهت نیازمند کارایی بالا یا رابط گرافیکی کاربر پسند نیستند. استفاده آن‌ها معمولاً در گسترش آن‌ها توسط سایر محققین است که راه را ادامه خواهند داد.

- برای پاسخ به این نیازمندی اصول مربوط به کدنویسی در فصل سوم و چهارم کتاب [۳۳] به کار گرفته شده است.
- از الگوهای نرم افزاری پرکاربرد مانند *اداپتور*[۱۳]، *فکتوری*[۱۴] و *سینگلتون*[۱۵] استفاده شده است.
- به منظور جلوگیری از قطعه کد تکراری از وراثت و توابع *عمومی*[۱۶] 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 استفاده‌ی عموم برسد و کاربران عمل متخاصمانه‌ای انجام نخواهند داد به نوع خاصی از امنیت نسبت به انواع متداول نیاز 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*[۱۷] قرار دارد. مزیت این سایت نسبت به گیت‌هاب این است مخازن خصوصی را به صورت رایگان ارائه می دهد. 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

[12] Performance
[13] Adaptor
[14] Factory
[15] Singelton
[16] Generic
[17] Bitbucket - https://bitbucket.org/alimohebbi/bug_predict

است.



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 لازم است اطلاعات مربوط به ثبت‌های حاوی خطا از ابزار defects4j بازیابی شود و سپس این اطلاعات با استفاده از مخرن نرم‌افزاری تکمیل شود. در مراحل بعد 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 پرونده‌ی 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 defect4j قابل استخراج است بجز دو مورد آخر. همچنین شماره‌ی ثبت تعمیر مورد استفاده قرار نگرفت ولی نگهداری شد چراکه ممکن است در پژوهش‌های دیگر لازم شود. برای بدست آوردن اطلاعات مربوط به هر انتشار لازم است که مخزن نرم‌افزاری هر پروژه مورد بررسی قرار گیرد. در مخازن پروژه‌های نرم‌افزاری از نوع گیت برای مشخص کردن یک رویداد مهم از تگ[۱۸] 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. بنابرین با استفاده از تگ می‌تواند انتشار را پیدا کرد.

تگ‌های مخازن گیت دو نوع *سبک‌وزن*[۱۹] و *حاشیه‌نویسی شده*[۲۰] دارد 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 جزییات صرف نظر می‌شود.
ابتدا همه‌ی تگ‌های موجود در مخازن نرم‌افزاری استخراج می‌شود و در پایگاه داده قرار می‌گیرد. از میان تگ‌های استخراج شده تگ‌های نامرتبط با انتشار از پایگاه داده حذف می‌شود. تگ‌های نامرتبط با توجه به نام آنها مشخص می‌شود به عنوان مثال تگ‌هایی که حاوی لغات Beta یا Dev هستند نامرتبط 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.

[18] Tag
[19] Lightweight
[20] Annotated

#	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 ماقبل هر ثبت حاوی خطا کدام است. برای این منظور لیست 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.

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

#	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	4ddbd99c5805781bd3c2...	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....	cff0f1ae37bb2b7ab2dcdb1...	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 انتخاب پرونده‌های سالم

همانطور که مطرح شد تعداد پرونده‌های حاوی خطا برابر ۴۰۵ عدد است که از تعداد کل پرونده‌ها کمتر است. بنابرین جهت ساخت مدل‌های بدون جهت‌گیری به همین تعداد پرونده‌های بدون خطا به طور تصادفی انتخاب می‌شود. بدین ترتیب یک مجموعه داده‌ی *متعادل*[۲۱] حاوی ۸۱۰ پرونده ساخته شده است. این روش طبق مقاله‌ی

[21] Balanced

[۳۴] به کار گرفته شده است. در این انتخاب به تعداد پرونده‌های حاوی خطا، پرونده‌های بدون خطا انتخاب می‌شود.

به ازای هر پرونده‌ی دارای خطا در همان ثبت از پروژه‌ی مربوط، یک پرونده‌ی بدون خطا به صورت تصادفی انتخاب می‌شود. برای این کار لیست تمام پرونده‌های داخل پروژه در ثبت پرونده حاوی خطا در نظر گرفته می‌شود و یک پرونده به صورت تصادفی انتخاب می‌شود. این پرونده نباید جز پرونده‌های حاوی خطا در آن ثبت از پروژه باشد. همانطور که گفته شد یک ثبت ممکن است بیش از یک پرونده حاوی خطا داشته باشد. همچنین ممکن است این پرونده در ثبت‌های بعدی یا قبلی خطا داشته باشد و از این نظر محدودیتی ندارد. سپس مشخصات این پرونده در جدول CleanInfo قرار می گیرد. نمایی از این جدول در تصویر ۱۰.۴ آورده شده است.

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

#	Id	CleanClassName	CommitId	PriorTagName	Project	RelatedBugI	TagCommit
1	1	org.apache.commons.la...	2c454a4ce3fe7710987468...	LANG_3_1	Lang	1	b1340f422f68be...
2	2	org.apache.commons.la...	aefc12c38171e1a84a90dc...	LANG_3_1	Lang	2	b1340f422f68be...
3	3	org.apache.commons.la...	1f001d06a2bde5ee4e3204...	LANG_3_1	Lang	3	b1340f422f68be...
4	4	org.apache.commons.la...	4ddbd99c5805781bd3c22...	LANG_3_1	Lang	4	b1340f422f68be...

شکل ۱۰.۴: نمایی از جدول محتوای اطلاعات پرونده‌های سالم

۳.۴.۴ 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 استفاده ی مستقیم از Jgit برای این کار است. به این صورت که تعداد ثبت‌های بین ثبت کنونی و انتشار قبلی را بررسی کرده و تعداد ثبت‌هایی که در آن‌ها پرونده حاوی خطا تغییر کرده است شمرده شوند. مشکل این راه این است که بسیار پر هزینه خواهد بود زیرا مرتبا باید عملیات *ورودی/خروجی*[۲۲] 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.

[22] Input/Output (IO)

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 بنابرین استفاده از تاریخ غیر ممکن می‌شود. علت داشتن تاریخ یکسان احتمالاً مهاجرت از یک نوع مخزن نرم‌افزاری به نوع گیت بوده است.

بنابرین 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 ۱۱.۴ و جدول CommitChangeFile در شکل ۱۲.۴ زیر آمده است:

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: | Fetch rows:

#	ID	COMMIT_GIT_ID	COMMITTER_MAIL	COMMITTER_NAME	PROJECT	SEQUENCE_NUMBER
35	9506	1e11bf4fcfb934f6bd3788a2d47089...	szczepiq@gmail.c...	Szczepan Faber	Mockito	34
36	9507	f28e6c5ebecee8cb75f6ab79b7ad3...	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 ۱۱.۴: نمایی از جدول اطلاعات ثبت‌ها

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: | Fetch rows:

#	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 ۱۲.۴: 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 ۲.۴ استفاده می‌شود و معیار محاسبه می‌گردد.

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

قطعه کد ۱.۴: بازیابی اطلاعات ثبت

```
SELECT  count(*) from CommitChangedFile CCzzz where CC.COMMIT_INFO_ID IN
  (SELECT CI.ID from CommitInfo CI WHERE CI.SEQUENCE_NUMBER BETWEEN
    :startSeq AND :endSeq AND CI.PROJECT = :project)
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 پرونده‌ی مورد نظر در آن ثبت‌ها تغییر کرده است. به عبارت دیگر ثبت‌هایی که نام پرونده در جدول CommitChangeFile برای آن‌ها وجود دارد.

```
SELECT  count(DISTINCT CI.COMMITTER_MAIL) from CommitInfo CI WHERE
CI.SEQUENCE_NUMBER BETWEEN :startSeq AND :endSeq AND CI.PROJECT =
project AND CI.ID IN
  (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 تعداد 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.

```
SELECT sum(CC.ADDED_LINES) from CommitChangedFile CC where
CC.COMMIT_INFO_ID IN
  (SELECT CI.ID from CommitInfo CI WHERE CI.SEQUENCE_NUMBER BETWEEN
  :startSeq AND :endSeq AND CI.PROJECT = :project)
AND CC.FILE_NAME = :fileName

```

قطعه کد ۴.۴: محاسبه‌ی تعداد خطوط اضافه شده به پرونده

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

قطعه کد ۵.۴: محاسبه‌ی تعداد خطوط اضافه شده به پروژه

مقدار نرمال‌سازی شده‌ی تعداد خطوط حذف شده: به طور مشابه معیار قبلی محاسبه می‌گردد.

درصد خطوطی که مالک پرونده مشارکت کرده: دستور Blame در Jgit نشان می‌دهد که هر خط از پرونده در یک ثبت در کدام یک از ثبت‌های گذشته اضافه شده است. با یافتن ثبت مسئول اضافه کردن آن خط نویسنده‌ی آن خط مشخص می‌شود که همان ثبت‌کننده است. با کمک این دستور به دلایل مشابه ساخت جداول مربوط به ثبت‌ها، جدولی با عنوان Participation ساخته شده که در آن هر سطر نشان می‌دهد که یک نویسنده در یک نسخه از برنامه چند درصد از خطوط به وی اختصاص دارد. در شکل ۱۳.۴ نمایی از این جدول آورده شده است. از این جدول علاوه بر محاسبه‌ی 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 ساده‌ی ۶.۴ محاسبه خواهد شد.

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: | Fetch rows:

#	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	stevencaswell@apa...	2c454a4ce3fe771098...	org.apache.commons.lang3.time.FastDateFormat	src/main/java/org/ap...	0.00508475

شکل ۱۳.۴: نمایی از جدول مشارکت‌کنندگان در ویرایش پرونده‌ها

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

قطعه کد ۶.۴: محاسبه‌ی درصد خطوط مالک پرونده

تعداد مشارکت‌کنندگان جزئی: با استفاده از جدول Participation و پرسمان ۷.۴ معیار محاسبه می‌شود. مقدار minorThereshold برابر ۵ درصد قرار می‌گیرد.

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

قطعه کد ۷.۴: محاسبه‌ی تعداد مشارکت‌کنندگان جزئی

تعداد ثبت‌های همسایگان: ابتدا لازم است که همسایگان پرونده در یک ثبت و نیز تعداد دفعات همسایگی در طول انتشار مشخص شود. این عمل به وسیله‌ی پرسمان ۸.۴ انجام می‌شود. سپس معیار تعداد ثبت‌ها در سیستم کنترل نسخه مشابه قبل با استفاده از کد ۲.۴ محاسبه می‌گردد و از آنها میانگین وزن‌دهی شده گرفته می‌شود.

```
SELECT FILE_NAME as 'name', count(ID) as 'frequency' FROM
CommitChangedFile WHERE COMMIT_INFO_ID IN
  (SELECT COMMIT_INFO_ID FROM CommitChangedFile WHERE FILE_NAME = :
    fileName)
AND COMMIT_INFO_ID IN
  (SELECT CI.ID from CommitInfo CI WHERE CI.SEQUENCE_NUMBER BETWEEN :
    startSeq AND :endSeq AND PROJECT = :project)
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 پرونده: 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.

```
SELECT AUTHOR_EMAIL FROM Participation P WHERE COMMIT_ID = :commitId
AND FILE_NAME = :fileName AND PARTICIPATION_PERCENT =
  (SELECT max(PARTICIPATION_PERCENT) FROM Participation P2
   WHERE P2.COMMIT_ID = :commitId AND P2.FILE_NAME = :fileName)
```

قطعه کد ۹.۴: یافتن مالک پرونده

```
SELECT  count(*) from CommitInfo CI where CI.SEQUENCE_NUMBER BETWEEN
```

```
:startSeq AND :endSeq AND CI.PROJECT = :project AND CI.COMMITTER_MAIL =
: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 می شود.

```
SELECT AUTHOR_EMAIL FROM Participation P WHERE COMMIT_ID = :commitId
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 ۱۴.۴ آورده شده است.

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

#	ID	ACTIVE_D	COMM	DEV_COUNT	FILE_INFO_ID	NORMAL_ADD	NORMAL_DE	FILE_1	MINOR	OWNER_PARTI	ALL_AUTH	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 ۱۴.۴: نمایی از جدول معیارهای فرآیند

۴.۴.۴ 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 یک ثبت خاص از پروژه از مخزن نرم‌افزاری فراخوانی می‌شود و در آن قرار می گیرد. سپس به پرونده 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*[۲۳] 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 یک رایانه به صورت اختصاصی برای انجام آن در *آزمایشگاه*

[23] Perl

کیفیت نرم‌افزار[۲۴] واقع در دانشگاه صنعتی شریف در نظر گرفته شد. این رایانه به یک سرور *لینوکس*[۲۵] 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 ب 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*[۲۶]:** این فضا یک هیپ[۲۷] مخصوص است که از فضای هیپ اصلی جاوا مجزا است و در آن *ماشین مجازی جاوا*[۲۸] *فراداده‌های*[۲۹] 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*[۳۰] :** 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 بوئز و همکاران[۲۸] استفاده شده که مطابق عملگرهای پیش‌فرض در ابزار PIT می‌باشد.

[24] Software Quality Research Lab - http://sqrlab.ce.sharif.edu/
[25] Linux
[26] Premanent Generation
[27] Heap
[28] Java Virtual Machine
[29] Metadata
[30] Timeout

پرونده‌ی MML ساخته شده در شکل ۱۵.۴ آمده است.

```
targetOp{

BIN(+)->{-};
BIN(-)->{+};
BIN(*)->{/};
BIN(/)->{*};
BIN(%)->{*};

BIN(>>)->{<<};
BIN(<<)->{>>};
BIN(>>>)->{<<};

BIN(&)->{|};
BIN(|)->{&};
BIN(^)->{&};

UNR(+)->{-};
UNR(-)->{+};

// Use sufficient replacements for ROR
BIN(>)->{>=,<=};
BIN(<)->{<=,>=};
BIN(>=)->{>,<};
BIN(<=)->{<,>};
BIN(==)->{!=};
BIN(!=)->{==};

// Delete all types of supported statements
DEL(CALL);

// Enable all operators
AOR;
EVR;
LOR;
SOR;
ROR;
ORU;
STD;
}
```

شکل ۱۵.۴: 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 ۱۶.۴ آمده است.

Result Grid | Filter Rows: | Edit: | Export/Import:

#	MetricId	Covered	FileInfoId	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 ۱۶.۴: 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.

- **تعداد جهش‌یافته‌های متمایز در چند انتشار اخیر:**

به منظور افزایش کارایی ابتدا بررسی می‌شود که فایل مورد نظر در آن انتشار وجود دارد یا خیر در صورت عدم وجود محاسبات برای آن انتشار انجام نمی‌گیرد. برای محاسبه ابتدا چهار انتشار قبلی با استفاده از پرسمان ۱۲.۴ از جدول ProjectRelease بازیابی می‌شود. سپس مشابه معیار قبلی جهش‌یافته‌های جدید نسبت به انتشار قبلی برای هر انتشار محاسبه می‌شود و با هم جمع زده می‌شود. یک جدول برای نتایج تولید جهش‌یافته‌ها به نام DistinctMutantLog در نظر گرفته شده که تعداد جهش‌یافته‌های

جدید برای هر انتشار نسبت به انتشار قبلی در آن ذخیره می‌گردد. از مزایای ایجاد این جدول پایداری در انجام محاسبات است به عنوان مثال در صورت توقف محاسبات امکان از سرگیری محاسبات از محل توقف وجود دارد و همچنین با نگهداری به عنوان یک مجموعه داده می‌تواند در پژوهش‌های دیگر به کار گرفته شود. نمایی از جدول در شکل زیر آمده است. به طور مثال سطر اول جدول بیان می‌کند که در انتشاری از برنامه با شماره ثبت 21a.. پرونده‌ی شماره یک شماره یک از فایلهای حاوی خطا ۴۳۰ جهش‌یافته‌ی جدید نسبت به انتشار قبلی داشته است.

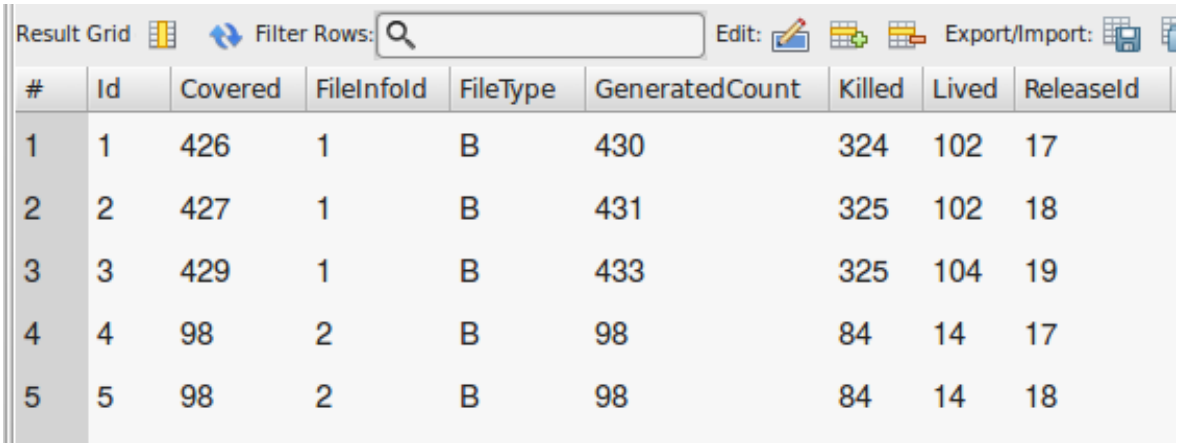
```
SELECT * FROM ProjectRelease WHERE Project = :project AND
 SequenceNumber <
  (SELECT SequenceNumber FROM ProjectRelease WHERE
    Project = :project AND CommitId = :releaseCommit)
 ORDER BY SequenceNumber DESC LIMIT 4
```

قطعه کد ۱۲.۴: بازیابی چهار نسخه‌ی اخیر یک ثبت

#	LogId	CommitId	FileId	FileType	NewMutants
1	1	aad55e0d568d152e7290a18136d247b1abbaa21a	1	B	430
2	2	9ee116a6a54763f0e86567df2a290cf81d8a3437	1	B	44
3	3	b1340f422f68be7c237bbc9127d1f12a92be16a2	1	B	4
4	4	aad55e0d568d152e7290a18136d247b1abbaa21a	2	B	98
5	5	9ee116a6a54763f0e86567df2a290cf81d8a3437	2	B	0
6	6	b1340f422f68be7c237bbc9127d1f12a92be16a2	2	B	0

شکل ۱۷.۴: نمایی از جدول تعداد جهش‌یافته‌های متمایز در انتشارها

- **میزان تغییرات مثبت امتیاز جهش در چند انتشار اخیر:** ابتدا انتشارها مشابه معیار قبلی بازیابی می‌شوند و سپس برای هر یک تحلیل جهش انجام می‌گردد. نتایج جهش در جدولی به نام ReleaseMutation قرار می‌گیرد. نمایی از این جدول در شکل ۱۸.۴ آمده است. سپس هر انتشار با انتشار قبلی مقایسه می‌شود و در صورتی که تغییر امتیاز جهش مثبت باشد با مجموعه تغییرات مثبت جمع می‌گردد.



Result Grid | Filter Rows: | Edit: | Export/Import:

#	Id	Covered	FileInfoId	FileType	GeneratedCount	Killed	Lived	ReleaseId
1	1	426	1	B	430	324	102	17
2	2	427	1	B	431	325	102	18
3	3	429	1	B	433	325	104	19
4	4	98	2	B	98	84	14	17
5	5	98	2	B	98	84	14	18

شکل ۱۸.۴: نمایی از جدول نتایج تحلیل جهش در انتشارها

- **میزان تغییرات منفی امتیاز جهش در چند انتشار اخیر:** به طور مشابه با معیار قبلی عمل می‌گردد با این تفاوت که تغییرات منفی در نظر گرفته می‌شود.

۶.۴ رویکرد سوم: معیارهای ترکیبی جهش‌ـفرآیند

نحوه‌ی محاسبه به این صورت خواهید بود که ابتدا ثبت‌هایی از برنامه در طول آخرین انتشار که در آن فایل مورد نظر تغییر کرده است توسط پرسمان ۱۳.۴ بازیابی می شود. سپس برای هر ثبت تعداد جهش یافته‌های جدید نسبت به ثبت قبلی محاسبه می‌شود و برای محاسبه‌ی جهش‌یافته‌های حذف شده تعداد جهش یافته‌ها در ثبت قبلی را یافته و آن‌ها که جز خطوط حذف شده در ثبت بعدی است شمرده می شود. تعداد جهش‌یافته‌های اضافه و حذف شده در ثبت‌ها جمع شده و بر تعداد ثبت‌های کل پروژه در طول انتشار تقسیم می گردد.

```
1 SELECT CC.* from CommitChangedFile CC, CommitInfo CI where CC.
    COMMIT_INFO_ID = CI.ID
2 AND CI.SEQUENCE_NUMBER BETWEEN :startSeq AND :endSeq
3 AND CI.PROJECT = :project
4 AND CC.FILE_NAME = :fileName ORDER BY CI.SEQUENCE_NUMBER asc
```

قطعه کد ۱۳.۴: بازیابی اطلاعات ثبت‌هایی که یک فایل در بازه‌ی مشخص در آنها تغییر کرده است

فصل ۵

ارزیابی

در این بخش به تشریح نحوه‌ی ساخت مدلهای پیش‌بینی و ارزیابی معیارهای شرح داده شده در فصل ۳ پرداخته می‌شود. با استفاده از معیارهای استخراج شده در فصل ۴ مدل‌های مورد نظر شاخته می‌شوند. ساخت مدل‌ها در زبان R و به وسیله‌ی بسته‌ی کرت [1] [۳۵] انجام می‌شود.

در ابتدا ۱۰ درصد از داده‌ها به عنوان داده‌ی آزمون جدا می‌شود. با استفاده از ۹۰ درصد باقی‌مانده به ساخت مدل پرداخته می‌شود. در ساخت و ارزیابی مدل‌ها از روش ارزیابی میان دسته‌ای استفاده می‌شود که تعداد دسته‌ها ۱۰ و تعداد تکرار نیز ۱۰ مورد می‌باشد. لازم به ذکر است که دسته‌بندی‌ها به طور تصادفی انجام می‌شود. همچنین در بسته‌ی کرت در هر روش دسته‌بندی پارامترهای مختلفی به طور پیش فرض به کار گرفته می‌شود تا بهترین مدل ممکن ساخته شود. با استفاده از ارزیابی میان‌دسته‌ای و تنظیم خودکار پارامترهای مختلف مدل نهایی ساخته شده و از این مدل برای پیش‌بینی داده‌های آزمون مورد استفاده قرار گرفته است.

در این ارزیابی از روش‌های دسته‌بندی Decition Tree ، SVM ، Logestic Regression و Neural Network استفاده شده است. این روش‌های دسته‌بندی بیش از سایرین در مقالات مورد استفاده قرار گرفته‌اند.

در ادامه هر یک از رویکردها به طور جداگانه ارزیابی شده و نتایج در زیر آمده است.

۱.۵ ارزیابی معیارهای فرآیند و جهش

همانطور که اشاره شد هدف از این آزمایش این است که مشخص شود قرارگیری معیارهای جهش در کنار معیارهای فرآیند باعث بهبود پیش‌بینی خطا می‌گردد یا خیر و این تاثیر تا چه میزان است. به همین منظور با استفاده از ۱۲ معیار فرآیند یک مدل پیش‌بینی ساخته شده و مدل دیگری با استفاده از ۱۲ معیار فرآیند و ۴ معیار جهش ساخته شده است. در نهایت این دو مدل با استفاده از معیارهای ارزیابی مختلف با هم مقایسه شده‌اند.

[1] Caret

بدیهی است که دو مدلی که با هم مقایسه می‌شوند بجز در معیارهای استفاده شده (بردار ویژگی) به منظور ساخت مدل از هیچ منظری تفاوت ندارند و داده‌های پرونده‌های یکسانی در ساخت و ارزیابی آنها استفاده شده.

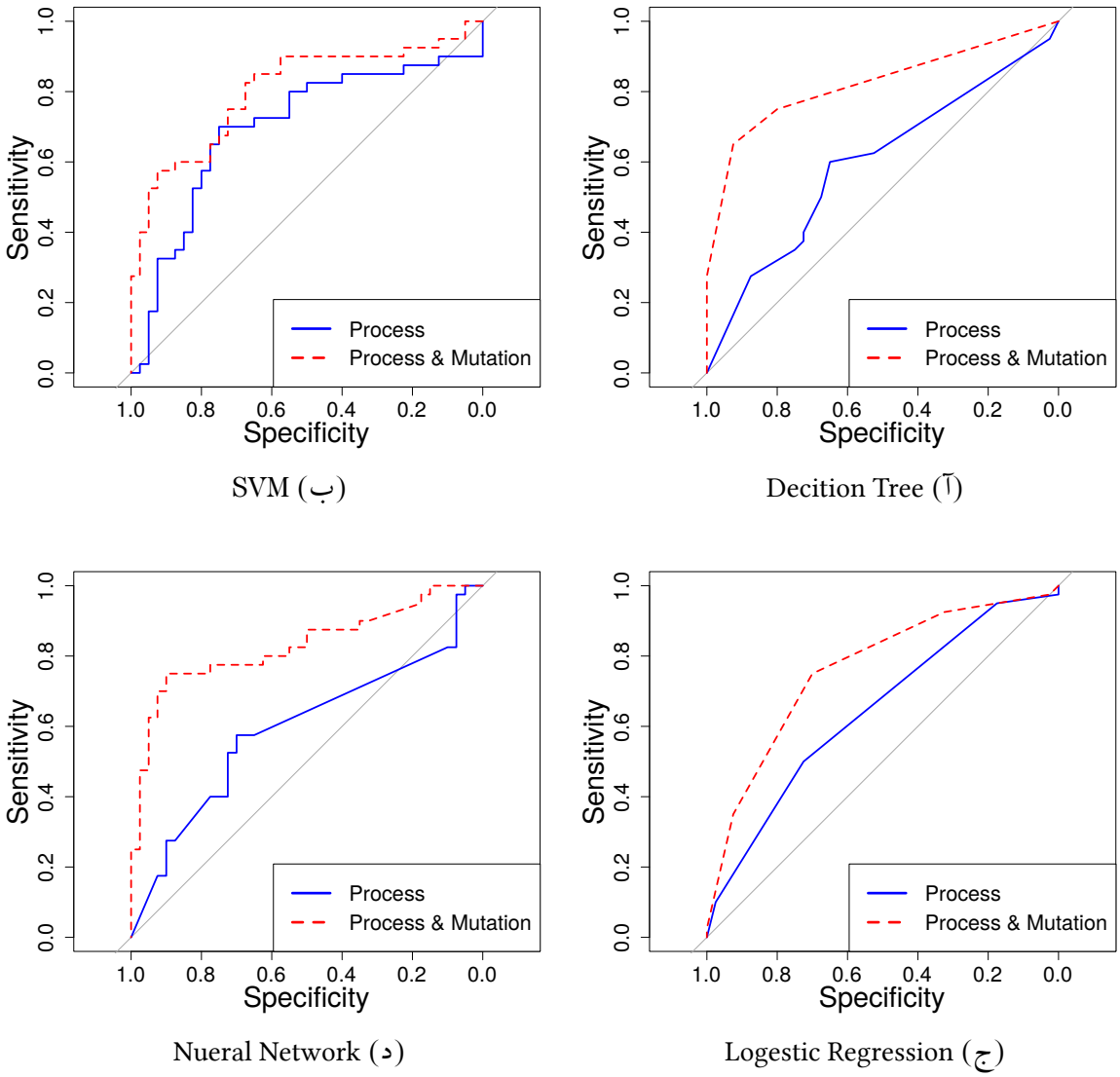
در جدول ۱.۵ بخشی از نتایج آمده است. این نتایج نشان می‌دهد که قرار گیری معیارهای جهش در کنار معیارهای فرآیند موجب بهبود پیش‌بینی خطا به مقدار قابل ملاحظه‌ای می‌شود و در تمام روشهای یادگیری موجب بهبود پیش‌بینی می‌گردد. از میان روشهای دسته‌بندی بهترین عملکرد پس از افرودن معیارهای جهش از نظر صحت و دقت را روش Neural Network داشته است. روش Decition Tree نیز بهترین عملکرد از نظر معیار بازخوانی را داشته است. همچنین بیشترین تغییر مثبت در صحت پیش‌بینی پس از افزودن معیارهای جهش را روش Neural Network و Decition Tree با مقدار ۲۰ درصد داشته است. کمترین تاثیر با مقدار ۷/۵ درصد در روش SVM بوده است. بیشترین افزایش دقت در روش Decition Tree بوده است که مقدار آن ۱۵/۱ درصد می‌باشد. از نظر معیار بازخوانی بیشترین تغییر مثبت را درخت تصمیم دارد که رشد ۲۵ درصدی داشته و روش Logestic Regression کاهش ۲/۵ درصدی داشته است. به طور کلی می‌توان این نتیجه را برداشت کرد که بیشترین بهبود در روش Decision Tree و کمترین در SVM روی داده است.

جدول ۱.۵: مقایسه‌ی معیارهای فرآیند به تنهایی و به همراه جهش

معیار	نام روش	صحت	دقت	بازخوانی
فرآیند	Decition Tree	۰/۵۸۷	۰/۵۷۴	۰/۶۷۵
فرآیند و جهش	Decition Tree	۰/۷۸۷	۰/۷۲۵	۰/۹۲۵
فرآیند	SVM	۰/۶۶۲	۰/۶۸۵	۰/۶۰۰
فرآیند و جهش	SVM	۰/۷۳۷	۰/۸۰۶	۰/۶۲۵
فرآیند	Logestic Regression	۰/۶۱۲	۰/۵۹۱	۰/۷۲۵
فرآیند و جهش	Logestic Regression	۰/۷۲۵	۰/۷۳۶	۰/۷۰۰
فرآیند	Nueral Network	۰/۶۱۲	۰/۷۲۵	۰/۵۹۱
فرآیند و جهش	Nueral Network	۰/۸۱۲	۰/۷۷۷	۰/۸۷۵

در شکل ۱.۵ نمودارهای ROC به تفکیک روش دسته‌بندی آمده است. در هر یک از زیر شکل‌ها منحنی ROC مربوط به دو مدل با هم مقایسه شده است. درمدل اول که در ساخت آن از معیارهای فرآیند استفاده شده با خط ممتد نمایش داده شده است و مدل دوم از معیارهای فرآیند به همراه معیارهای جهش ساخته شده است و با خط‌چین نمایش داده شده. همانطور که قابل مشاهده است در تمامی روش‌ها دسته‌بندی مدل حاوی معیار جهش

مساحت زیر نمودار بیشتری نسبت به مدل دیگر دارند و نشان از عملکرد بهتر این مدل‌ها می‌باشد.



(ب) SVM

(آ) Decition Tree

(د) Nueral Network

(ج) Logestic Regression

شکل ۱.۵: نمودارهای ROC معیارهای فرآیند و به همراه جهش

در جدول ۲.۵ مساحت زیر نمودار ROC در هر یک از روش‌های دسته‌بندی آورده شده است.در میان روش‌های یادگیری به کار گرفته شده بیشترین افزایش مساحت زیر نمودار را Neural Network به مقدار ۰٫۲۲۶ واحد داشته است و کمترین تغییر را نیز SVM با مقدار ۰٫۱۰۵ واحد داشته است. به طور متوسط ۰٫۱۵۱ واحد در مدلها بهبود مشاهده می‌شود. این موضوع نشان از تاثیر قابل توجه معیارهای جهش می‌باشد.

جدول ۲.۵: مقادیر زیر نمودار ROC معیارهای فرآیند و به همراه جهش

معیار	Decition Tree	SVM	Logestic Regression	Neural Network
فرآیند	٪۵۹۶	٪۶۹۷	٪۶۴۳	٪۵۹۳
فرآیند و جهش	٪۸۲۲	٪۸۰۲	٪۷۶۱	٪۸۲۹

۲.۵ ارزیابی معیارهای فرآیند مبتنی بر جهش

ارزیابی این معیارها در دو مرحله انجام می‌شود. در مرحله‌ی اول سه مدل ساخته می‌شود. این مدل‌ها به ترتیب با استفاده از معیارهای فرآیند، فرآیند و جهش و مدل آخر با استفاده از معیارهای فرآیند و فرآیند مبتنی بر جهش ساخته می‌شود. در مرحله‌ی دوم دو مدل ساخته می‌شود. در مدل اول معیارهای فرآیند و جهش مدل پیش‌بینی را خواهد ساخت و در مدل دوم معیارهای فرآیند مبتنی بر جهش نیز به مجموعه‌ی معیارها افزوده می‌شود.

۱.۲.۵ مرحله‌ی اول

مقایسه‌ی این مدل‌ها امکان را فراهم می‌کند مشخص شود آیا معیارهای فرآیند مبتنی بر جهش دارای قابلیت پیش‌بینی هستند یا خیر. همچنین در صورت داشتن این قابلیت مشخص شود که این قابلیت از معیارهای جهش کمتر است یا بیشتر.

مقایسه‌ی نتایج بدست آمده در جدول ۳.۵ با جدول ۱.۵ نشان می‌دهد که در تمامی روشهای دسته‌بندی بجز SVM معیار صحت در مدل سوم از مدل اول مقدار بیشتری دارد. در مدل ساخته شده توسط SVM نیز اختلاف معیار صحت کم می‌باشد(۳ درصد). این مدل در مقایسه با مدل دوم عملکرد بهتری از نظر معیار صحت و بازخوانی در هیچکدام از روشهای دسته‌بندی نداشته است. از نظر معیار دقت در تمامی روشها مدل سوم از مدل اول عملکرد بهتری داشته و حتی در روش Decition Tree مدل سوم از مدل دوم نیز بهتر عملکرده است. از نظر معیار بازخوانی مدل سوم نسبت به مدل اول تنها در روش Neural Network عملکرد بهتری داشته، در Decision Tree بدون تغییر مانده و در دو روش دیگر کاهش یافته است.

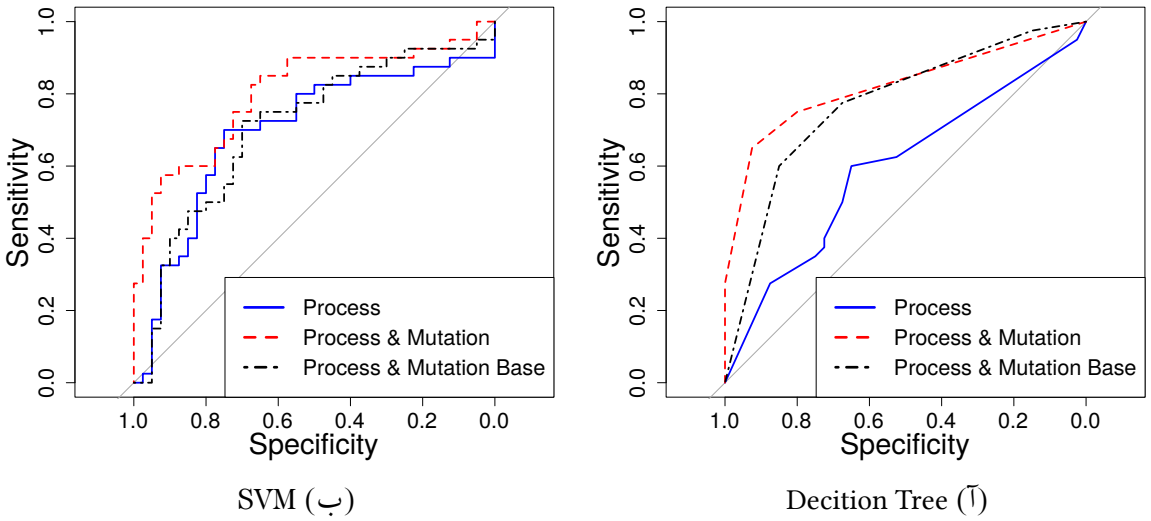
می‌توان این نتیجه را برداشت کرد که معیارهای ارائه شده دارای توانایی پیش‌بینی بیشتری نسبت به معیارهای فرآیند به تنهایی هستند.

جدول ۳.۵: نتایج پیش‌بینی‌خطای معیارهای فرآیند مبتنی بر جهش ـ مرحله‌ی اول

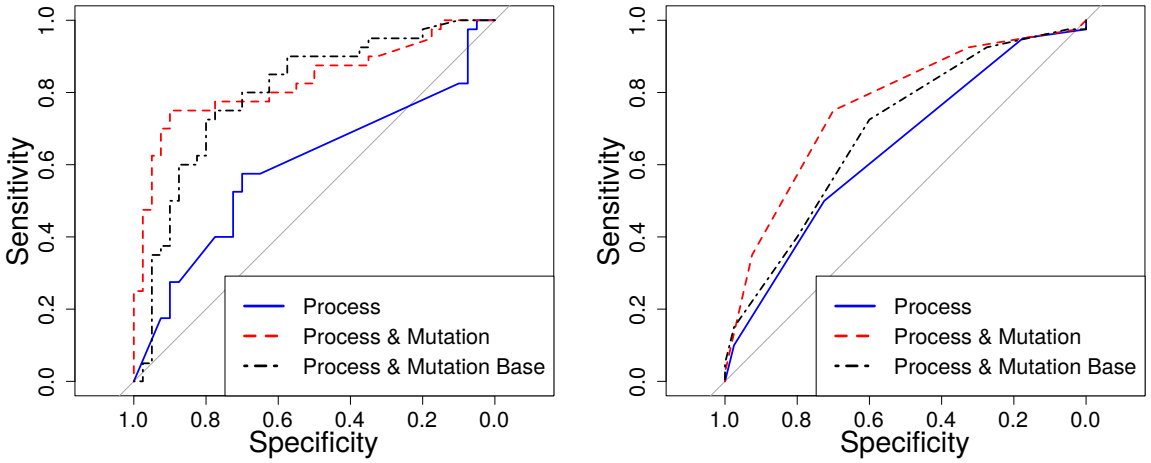
نام روش	صحت	دقت	بازخوانی
Decition Tree	۰/۷۲۵	۰/۷۵۰	۰/۶۷۵
SVM	۰/۶۳۷	۰/۶۸۹	۰/۵۰۰
Logestic Regression	۰/۶۶۲	۰/۶۸۵	۰/۶۰۰
Neural Network	۰/۷۶۲	۰/۷۵۶	۰/۷۷۵

در شکل ۲.۵ نمودارهای ROC سه مدل ساخته شده نشان داده شده است. در زیرشکل‌های (آ)(ج)(د) به وضوح عملکرد بهتر مدل سوم از مدل اول قابل مشاهده است. در زیرشکل (ب)نیز که متعلق به SVM است با رجوع به جدول ۴.۵ مشخص می‌شود که در این شکل نیز مساحت زیر نمودار ROC در مدل سوم بیشتر از اول است. همچنین مساحت زیر نمودار در مدل سوم در زیرشکل (ج) به مقدار ۰/۰۱۵ واحد از مدل دوم نیز بیشتر است.

این نتایج در راستای نتایج بدست آمده از جدول ۳.۵ می‌باشد. در نهایت می‌توان این نتیجه را گرفت که معیارهای فرآیند مبتنی بر جهش معرفی شده پیش‌بینی خطا را بهبود می‌بخشند اما عملکرد بهتری نسبت به معیارهای جهش ندارند. همچنین از آنجا که هزینه‌ی محاسباتی بیشتری نسبت به معیارهای جهش دارند جایگزینی آنها به جای یکدیگر مزیتی ندارد.



(آ) Decition Tree (ب) SVM



(ج) Logestic Regression (د) Nueral Network

شکل ۲.۵: نمودارهای ROC معیارهای فرآیند ، فرآیند و جهش ، فرآیند مبتنی بر جهش

جدول ۴.۵: مقادیر زیر نمودار ROC معیارهای فرآیند مبتنی جهش

Neural Network	Logestic Regression	SVM	Decition Tree
٪۷۹۸	٪۶۹۳	٪۷۰۷	٪۷۷۲

۲.۲.۵ مرحله‌ی دوم

همانطور که اشاره شد دو مدل ساخته می‌شود که مدل اول از معیارهای فرآیند و جهش استفاده می‌کند و در مدل دوم از همگی معیارها (با افزودن معیارهای فرآیند مبتنی بر جهش) استفاده می‌شود. هدف از این آزمایش این

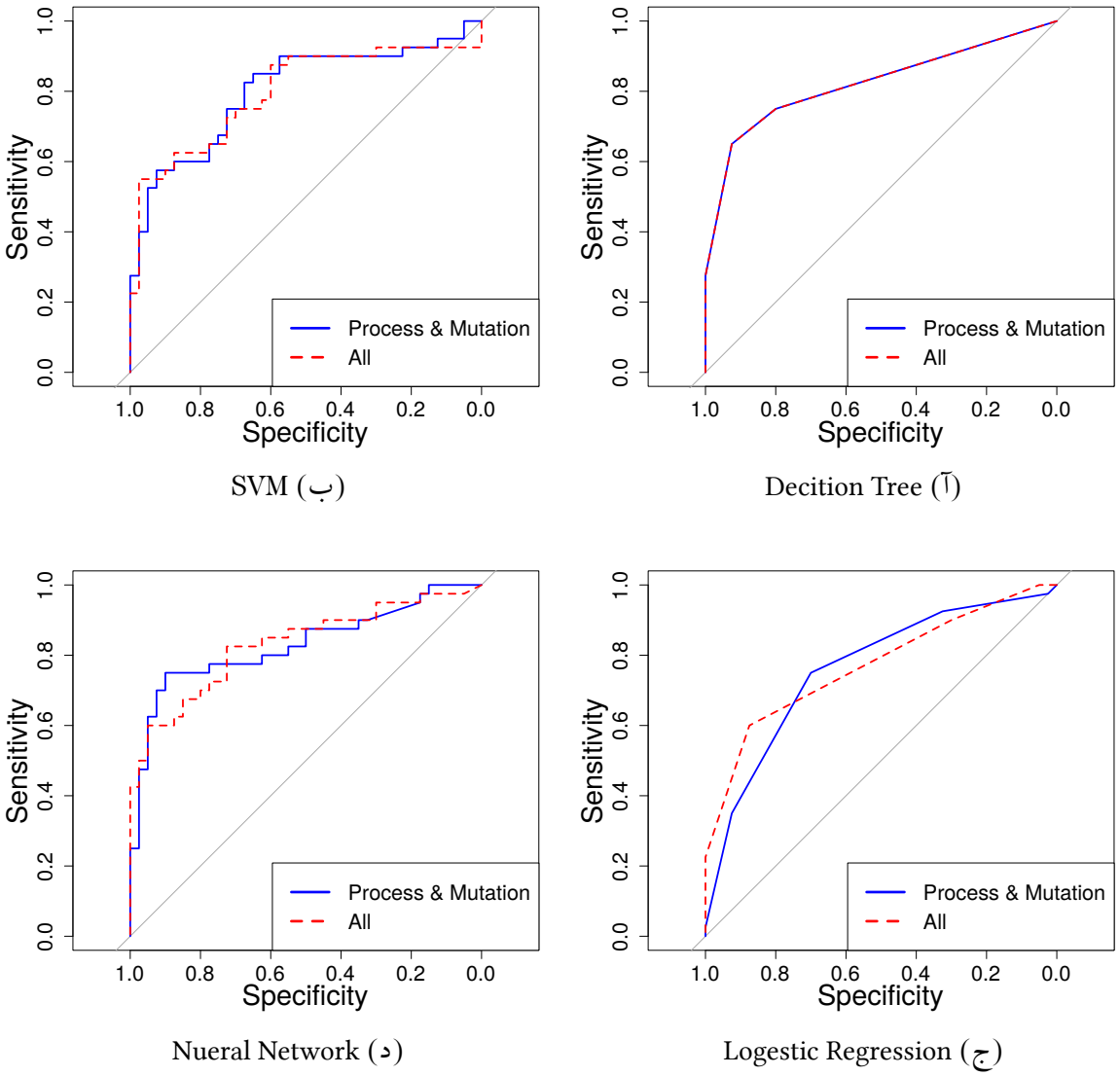
است که مشخص شود در صورتی که معیارهای ارائه شده‌ی جدید در کنار معیارهای قبلی قرار گیرد، در پیش‌بینی بهبودی حاصل می‌گردد یا خیر.

نتایج بدست آمد در جدول ۵.۵ نشان می‌دهد که مدل دوم در هیچ یک از روش‌ها بجز Logestic Regression از نظر معیارهای صحت، دقت و بازخوانی نسبت به مدل اول بهبودی پیدا نکرده است. همچنین در روش Decition Tree نتایج دو مدل یکسان است. در روش Logestic Regression مدل دوم در معیار صحت ۱٫۲ درصد افزایش، در معیار دقت ۵ درصد کاهش و ۱۷٫۵ درصد در بازخوانی افزایش داشته است.

جدول ۵.۵: نتایج پیش‌بینی خطای مدل حاصل از بکارگیری تمامی معیارها

نام روش	صحت	دقت	بازخوانی
Decition Tree	۰٫۷۸۷	۰٫۷۲۵	۰٫۹۲۵
SVM	۰٫۷۱۲	۰٫۷۷۴	۰٫۶۰۰
Logestic Regression	۰٫۷۳۷	۰٫۶۸۶	۰٫۸۷۵
Nueral Network	۰٫۷۵۰	۰٫۷۱۷	۰٫۸۲۵

نمودارهای ROC هر یک از این دو مدل در روش‌های دسته‌بندی مختلف در شکل ۳.۵ آمده است. در روش‌های مختلف مدل اول با دوم تفاوت چندانی ندارند و طبق جدول ۶.۵ تنها در مدل‌های حاصل از روش Logestic Regression به مقدار ۰٫۰۰۹ واحد مساحت زیر نمودار افزایش پیدا کرده است. بنابرین افزودن معیارهای فرآیند مبتنی بر جهش به سایر معیارهای مورد بررسی نمی‌تواند به بهبود پیش‌بینی بیانجامد.



(آ) Decition Tree

(ب) SVM

(ج) Logestic Regression

(د) Nueral Network

شکل ۳.۵: نمودارهای ROC معیارهای جهش و فرآیند و تمامی معیارها

جدول ۶.۵: مقادیر زیر نمودار ROC تمامی معیارها

Neural Network	Logestic Regression	SVM	Decition Tree
٪۸۳۰	٪۷۷۰	٪۷۸۶	٪۸۲۲

۳.۵ ارزیابی معیارهای ترکیبی فرآیند-جهش

در این قسمت به ارزیابی دو معیار مطرح شده پرداخته می‌شود. به منظور ارزیابی آنها دو مدل با استفاده از هر یک از روشهای دسته‌بندی ساخته می‌شود. در مدل اول معیارهای فرآیند استفاده می‌شود و در مدل دوم معیار *مقدار*

نرمال شده‌ی خطوط اضافه شده با معیار *تعداد خطوط اضافی وزن‌دهی شده* جایگزین می‌شود و معیار *مقدار نرمال شده‌ی خطوط حذف شده* نیز به طور مشابه جایگزین می‌شود. سایر معیارهای مدل دوم با مدل اول یکسان خواهد بود.

نتایج به دست آمده در جدول ۷.۵ نشان می‌دهد که معیارهای صحت، دقت و بازخوانی برای تمامی مدل‌ها بجز مدل ساخته شده توسط روش SVM افزایش قابل ملاحظه‌ای داشته است. بیشترین افزایش صحت در روش Neural Network به میزان ۱۳٫۸ درصد روی داده است. از نظر افزایش دقت بیشترین تغییر مثبت در روش Decition Tree بوده است که ۱۳٫۱ درصد رشد داشته است. معیار بازخوانی در دو روش Logestic Regression و Neural Network به ترتیب ۸٫۴ و ۲٫۵ رشد داشته و در دو روش دیگر کاهش داشته است.

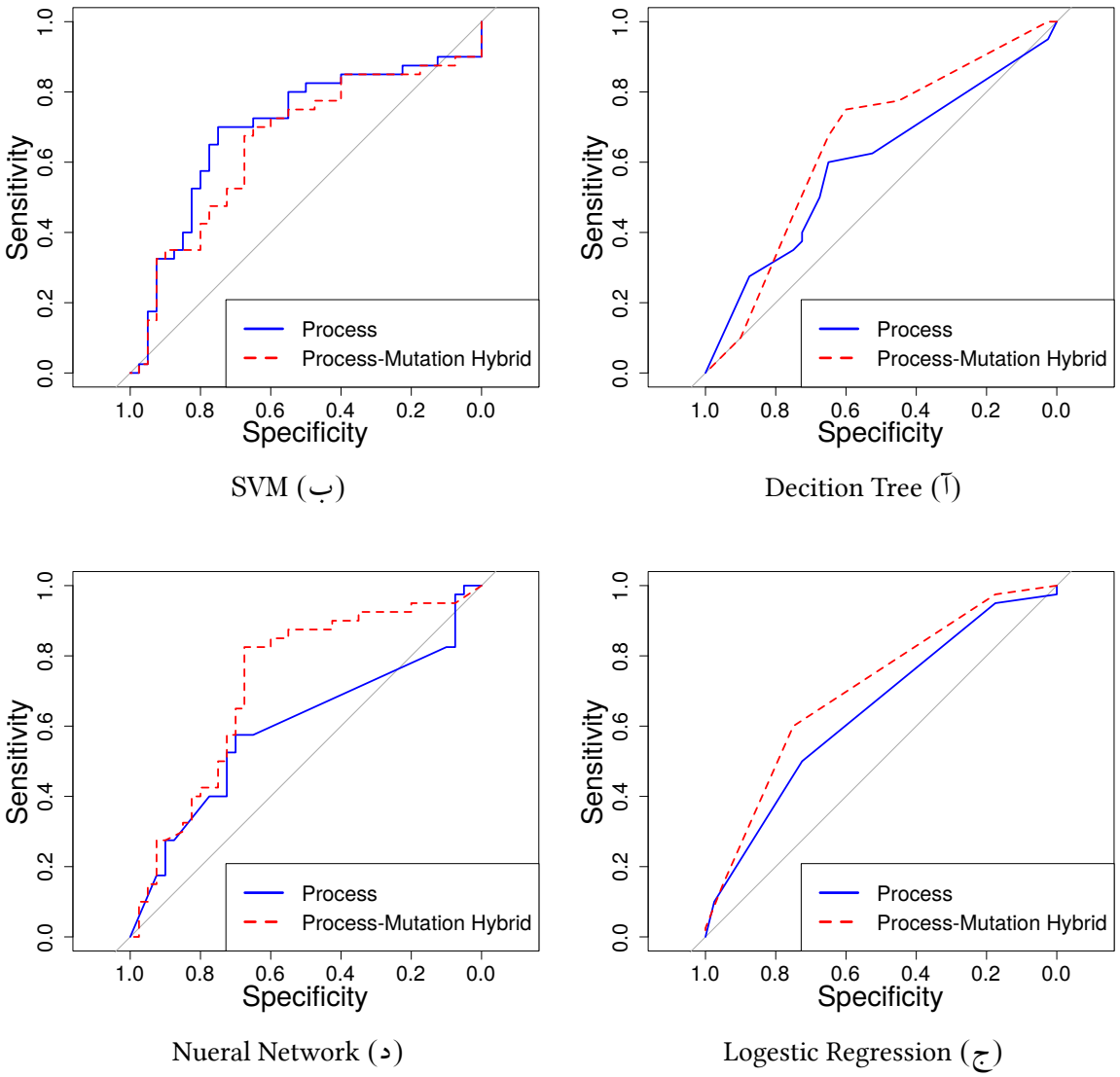
به طور میانگین معیار صحت ۶٫۶ درصد افزایش، معیار دقت ۶ درصد افزایش و معیار بازخوانی ۰٫۴ درصد کاهش داشته است. در نهایت می‌توان این نتیجه را گرفت که معیارهای ترکیبی جهش_فرآیند موجب بهبود در صحت و دقت پیش‌بینی می‌شوند و تاثیر چندانی در بازخوانی ندارند. لازم به ذکر است که تنها دو معیار از ۱۲ معیار مورد استفاده در دو مدل ساخته شده با هم متفاوت هستند که این دو معیار توانسته‌اند حدود ۶ درصد صحت و دقت را بهبود بخشند. این امر نشان از تاثیر قابل ملاحظه‌ی این معیارها می‌باشد.

جدول ۷.۵: مقایسه‌ی معیارهای فرآیند و معیارهای ترکیبی جهش_فرآیند

معیار	نام روش	صحت	دقت	بازخوانی
فرآیند	Decition Tree	۰٫۵۸۷	۰٫۵۷۴	۰٫۶۷۵
ترکیبی جهش_فرآیند	Decition Tree	۰٫۶۷۵	۰٫۷۰۵	۰٫۶۰۰
فرآیند	SVM	۰٫۶۶۲	۰٫۶۸۵	۰٫۶۰۰
ترکیبی جهش_فرآیند	SVM	۰٫۶۳۷	۰٫۶۶۶	۰٫۵۵۰
فرآیند	Logestic Regression	۰٫۶۱۲	۰٫۵۹۱	۰٫۷۲۵
ترکیبی جهش_فرآیند	Logestic Regression	۰٫۶۷۵	۰٫۶۵۲	۰٫۷۵۰
فرآیند	Nueral Network	۰٫۶۱۲	۰٫۷۲۵	۰٫۵۹۱
ترکیبی جهش_فرآیند	Nueral Network	۰٫۷۵۰	۰٫۷۹۴	۰٫۶۷۵

در شکل ۴.۵ نمودارهای ROC به تفکیک روش دسته‌بندی آمده است. در هر یک از زیر شکل‌ها منحنی ROC مربوط به دو مدل با هم مقایسه شده است. درمدل اول که در ساخت آن از معیارهای فرآیند استفاده شده با خط ممتد نمایش داده شده است و مدل دوم از جایگزینی دو معیار فرآیند با معیارهای ترکیبی جهش_فرآیند ساخته

شده و با خط چین نمایش داده شده. همانطور که قابل مشاهده است در تمامی روش‌ها بجز SVM مدل دوم مساحت زیر نمودار بیشتری نسبت به مدل اول داشته است.



(آ) Decition Tree (ب) SVM

(ج) Logestic Regression (د) Nueral Network

شکل ۴.۵: نمودارهای ROC معیارهای فرآیند و به همراه جهش

در جدول ۸.۵ مساحت زیر نمودار ROC دو مدل به تفکیک روش دسته‌بندی آورده شده است. در میان روش‌های یادگیری به کار گرفته شده بیشترین افزایش مساحت زیر نمودار را روش Neural Network به مقدار ۰٫۱۲۸ واحد داشته است. به طور متوسط ۰٫۰۵۱ واحد در مدل‌ها بهبود مشاهده می‌شود. این موضوع نشان می‌دهد که معیارهای ترکیبی جهش‌ـفرآیند از نظر مساحت زیر نمودار ROC نیز موجب تغییر مثبت ایجاد می‌شوند.

با توجه به اینکه تنها روش SVM نتایج ضعیفی نسبت به سایرین داشته است این موضوع را می‌توان با توجه نحوه‌ی عملکرد این روش توجیه کرد. به طور خلاصه این روش سعی می‌کند که *فضای ویژگی*[۲] را با ایجاد یک *ابرصفحه*[۳] به دسته‌های مختلف تقسیم کند اما توزیع نقاط داده در فضای ویژگی به نحوی نیست که این روش بتواند به خوبی عمل کند.

جدول ۸.۵: مقادیر زیر نمودار ROC معیارهای فرآیند و معیارهای ترکیبی جهش_

معیار	Decition Tree	SVM	Logestic Regression	Neural Network
فرآیند	٪۵۹۶	٪۶۹۷	٪۶۴۳	٪۵۹۳
جهش_فرآیند	٪۶۵۴	٪۶۵۶	٪۷۰۵	٪۷۲۱

[2]Feature Space
[3]Hyperplane

فصل ۶
نتیجه‌گیری و کارهای آتی

در این پایانامه سعی شد که تاثیر معیارهای جهش بر پیش‌بینی خطا در هنگام قرار گیری در کنار معیارهای فرآیند ارزیابی شود و معیارهای جدیدی با استفاده از مفاهیم تحلیل جهش و تاریخچه‌ی توسعه‌ی نرم‌افزار ارائه گردد. در فصل ۱ به بیان مسئله و مفاهیم مقدماتی پرداخته شد. در فصل ۲ پژوهش‌های پیشین در حوزه‌ی پیش‌بینی خطا مورد بررسی قرار گرفت. پژوهشگران به طرق مختلف سعی در دستیابی به نتایج بهتری در پیش‌بینی خطا هستند. در این بررسی مشخص شد که در پژوهش‌های پیشین دو دسته‌ی کلی از معیارها مورد استفاده قرار گرفته است. این دسته‌ها عبارتند از معیارهای کد و معیارهای فرآیند. معیارهای فرآیند دارای مزیت‌ها بیشتری نسبت به معیارهای کد هستند و پژوهش‌های کمتری نیز به بررسی آنها پرداخته است. در یکی از پژوهش‌های اخیر از معیارهای جهش در کنار معیارهای کد به منظور پیش‌بینی‌خطا استفاده گردیده و موجب بهبود پیش‌بینی شده است.

پس از مشخص شدن بخش‌هایی از این حوزه که نیازمند تحقیق بیشتر هستند و شناسایی پتانسیل‌های موجود در معیارهای فرآیند و جهش در فصل ۳ راهکارهایی ارائه شدند تا با استفاده از معیارهای فرآیند و مفاهیم تحلیل جهش پیش‌بینی خطا بهبود یابد. در رویکرد اول معیارهای فرآیند در کنار معیارهای جهش قرار می‌گیرند و پیش‌بینی خطا با استفاده از آنها انجام می‌پذیرد. در رویکرد دوم، چهار معیار فرآیند مبتنی بر مفاهیم تحلیل جهش ارائه شده‌اند و در رویکرد سوم دو معیار فرآیند با استفاده از مفاهیم جهش اصلاح شدند و معیارهای ترکیبی جهش_فرآیند به وجود آمدند.

در فصل ۴ نحوه‌ی پیاده‌سازی هر یک از سه رویکرد ارائه شده و ابزارهای مورد استفاده شرح داده شد. به منظور انجام مطالعه‌ی موردی، پنج پروژه‌ی صنعتی جاوا مورد استفاده قرار گرفتند و معیارهای مورد بررسی در آنها استخراج شد. این معیارها برای دو گروه از پرونده‌ها که یکی حاوی خطا و دیگری سالم هستند محاسبه شده

است. در این دو گروه تعداد یکسانی پرونده وجود دارد. پرونده‌های حاوی خطا در مجموعه‌داده‌ی defects4j مشخص شده‌اند و پرونده‌های سالم به طور تصادفی انتخاب شدند.

معیارهای استخراج شده در فصل ۵ ارزیابی شدند. مدل‌های پیش‌بینی با استفاده از چهار روش دسته‌بندی ساخته شدند و عملکرد مدل‌ها با یکدیگر مقایسه گردید. نتایج ارزیابی نشان داد که معیارهای جهش زمانی که در کنار معیارهای فرآیند قرار گیرند می‌توانند تاثیر قابل توجهی در بهبود پیش‌بینی داشته باشند.
معیارهایی که تحت عنوان فرآیند مبتنی بر جهش ارائه شدند، زمانی که در کنار معیارهای فرآیند قرار می‌گیرند موجب بهبود پیش‌بینی خطا می‌شوند اما توانایی آنها بیشتر از معیارهای جهش نیست. از آنجا که این دسته از معیارها هزینه‌ی محاسباتی بیشتری دارند جایگزینی آنها با معیارهای جهش نمی‌تواند مزیتی داشته باشد. همچنین قرارگیر همه‌ی این معیارها در کنار هم نیز تاثیر مثبت چندانی نخواهد داشت.
معیارهای ترکیبی جهش‌ـفرآیند به طور میانگین ۶ درصد در صحت، ۶/۶ درصد در دقت و ۵/۱ در مساحت زیر نمودار ROC تغییر مثبت ایجاد کرده است و از نظر معیار بازخوانی تغییر قابل توجهی ایجاد نمی‌کند. این تغییرات نشان می‌دهد که اصلاح معیارهای فرآیند موفق آمیز بوده است و عرصه‌ی جدیدی را می‌توان به منظور ساخت معیارهای پیش‌بینی در نظر گرفت و این عرصه ارائه‌ی معیارهای ترکیبی است. همچنین با توجه به این نکته که تولید جهش‌یافته نیازمند وجود موارد آزمون نیست می‌توان برای این معیارها دامنه‌ی کاربرد وسیع‌تری در نظر گرفت.
در ادامه به گام‌هایی اشاره می‌شود که می‌توانند به نتایج این پایانامه جامعیت بخشند شود و ابعاد دیگری از بکارگیری این معیارها مورد بررسی قرار گیرد.

- **بررسی تاثیر استفاده از عملگرهای متفاوت:**

  در این پایانامه مجموعه‌ی محدودی از عملگرها جهت ساخت جهش استفاده شده است. در پژوهش‌های آتی می‌توان به این موضوع پرداخت که افزایش و یا کاهش مجموعه‌ی عملگرهای جهش‌یافته چه تاثیری بر پیش‌بینی خطا داشته باشد. همچنین اینکه کدام نوع از عملگرهای مورد استفاده در استخراج معیارهای ارائه شده تاثیر بیشتری بر پیش‌بینی خطا دارد.

- **ارزیابی معیارهای کد در کنار معیارهای ارائه شده:**

  همانطور که بیان شد معیارهای جهش می‌توانند به معیارهای فرآیند کمک کنند تا پیش‌بینی دقیقتری

انجام شود. از طرف دیگر استفاده از معیارهای کد نیز می‌تواند به معیارهای جهش کمک کند و این معیارها هزینه‌ی محاسباتی کمتری دارند. با توجه به پر هزینه بودن معیارهای جهش لازم است میزان بهبود پیش‌بینی خطا توسط آنها با معیارهای کد مقایسه شود و مشخص شود در هنگام قرار گیری در کنار معیارهای فرآیند مزیتی در مقابل معیارهای کد دارند یا خیر.

- **ساخت چهارچوب پیش‌بینی خطا با استفاده از پژوهش موجود:**

استخراج معیارها و ساخت مدل‌های پیش‌بینی در این پایانامه به صورت خودکار انجام می‌گیرد. با ایجاد تغییرات لازم می‌توان چهارچوبی ارائه داده که برای سایر پروژه‌های نرم‌افزاری نیز این معیارها را استخراج کند. با ایجاد یک چهارچوب هم انجام پژوهش‌های آتی توسط سایرین سهولت می‌یابد و هم زمینه‌ی به کارگیری پیش‌بینی خطا در صنعت توسعه می‌یابد.

پیوست آ

ساخت مدل‌های پیش‌بینی و ارزیابی

در این قسمت قطعه کدهای ساخت مدل‌های پیش‌بینی و ارزیابی آنها آورده شده است. قطعه کد آ.۱ مجموعه‌داده‌ها را آماده می کند و تنظیمات مربوط به آموزش مدل‌ها را انجام می‌دهد.

```
1  library(RMySQL);
2  library(caret);
3  library(pROC);
4  library(e1071)
5
6  mydb = dbConnect(MySQL(), user='root', password='1', dbname='
       bug_predict', host='127.0.0.1');
7
8  rs_mutation_metric = dbSendQuery(mydb, "select * from
       MutationMetric");
9  mutation_metircs = fetch(rs_mutation_metric, n=-1);
10 rs_process_metric = dbSendQuery(mydb, "select * from
       ProcessMetric ");
11 process_metircs = fetch(rs_process_metric, n=-1);
12
13 ##### clean up data #####
14 source("/home/ali/project/R-scripts/kill-live-to-score.R");
15
16 merged_metrics <- merge(x=clean_mutation_metircs,y=process_
       metircs, by.x="MetricId", by.y="ID")
17 lables<- as.factor(merged_metrics[,names(merged_metrics) %in%
       c("FILE_TYPE")]);
18
19
20 ############test and train#############
21 b_number<-nrow(merged_metrics[merged_metrics$FILE_TYPE == "B"
       ,])
22 c_number<- nrow(merged_metrics[merged_metrics$FILE_TYPE == "C
       ",])
23 smp_size_b <- floor(0.9 * b_number);
24 smp_size_c <- floor(0.9 * c_number);
25
```

```
## set the seed to make your partition reproducible
set.seed(1423)
train_ind_b <- sample(seq_len(b_number), size = smp_size_b)
train_ind_c <- sample(seq_len(c_number), size = smp_size_c)
train_ind_c <- train_ind_c + b_number
train_ind <- c(train_ind_b, train_ind_c)

#########train control##########
MyFolds <- createMultiFolds(merged_metrics[train_ind,4], k =
    10, times=10)
train_control <- trainControl(method = "cv", index = MyFolds,
savePredictions = TRUE,
classProbs = TRUE
#     ,summaryFunction = twoClassSummary
)


```

قطعه کد آ.۱: آماده‌سازی مجموعه داده

در قطعه کد آ.۲ پاک‌سازی داده‌ها و تبدیل داده‌های جهش به امتیاز جهش انجام می‌شود.

```
  clean_mutation_metircs<- mutation_metircs[!is.na(mutation_
   metircs$Covered),];

  for(i in 1:dim(clean_mutation_metircs)[1])
  {
  if(clean_mutation_metircs[i,'Lived']==-1)
  {
  clean_mutation_metircs[i,'Lived']<- 0;
  clean_mutation_metircs[i,'Killed']<-clean_mutation_metircs[
   i,'Killed']-1;
  }
  }

  temp<-clean_mutation_metircs;
  temp[,6]<-clean_mutation_metircs[,6]/clean_mutation_metircs
   [,5]
  temp[,7]<-clean_mutation_metircs[,6]/clean_mutation_metircs
   [,2]
  clean_mutation_metircs <- temp

  for(i in 1:dim(clean_mutation_metircs)[1])
  {
  if(is.nan(clean_mutation_metircs[i,'Lived']))
  clean_mutation_metircs[i,'Lived']<- 0;
  if(is.nan(clean_mutation_metircs[i,'Killed']))
```

```
    clean_mutation_metircs[i,'Killed']<- 0;
    }

```

قطعه کد آ.۲: تبدیل داده‌های جهش به امتیاز جهش

در قطعه کد آ.۳ مدل‌های پیش‌بینی ساخته می‌شوند و با استفاده از داده‌های آزمون پیش‌بینی انجام می‌گیرد. تمامی معیارها در متغیر merged_metrics وجود دارند و با انتخاب ستون‌های مورد نظر زیر مجموعه‌ی مناسب انتخاب می‌شود. همچنین در تابع train روش دسته‌بندی انتخاب می‌گردد.

```

############Process Metrics#############
p_features<-merged_metrics[,-c(seq(1,13),17,20)];
model1 <- train(p_features[train_ind,], lables[train_ind],
    trControl = train_control, method="nnet");
predict1_raw<-predict.train(model1, p_features[-train_ind,],
    type="raw")
predict1_prob<-predict.train(model1, p_features[-train_ind,],
     type="prob")


############Process Metrics with mutation#############

m_features1<-merged_metrics[,!names(merged_metrics) %in% c("
    FILE_TYPE","MetricId","FileType","FileInfoId","FILE_INFO_
    ID")];
m_features1<-m_features1[,-c(seq(5,10))];
model2 <- train(m_features1[train_ind,], lables[train_ind],
    trControl=train_control, method="nnet");
predict1_raw<-predict.train(model2, m_features1[-train_ind,],
     type="raw")
predict1_prob<-predict.train(model2, m_features1[-train_ind
    ,], type="prob")


m_features2<-merged_metrics[,!names(merged_metrics) %in% c("
    FILE_TYPE","MetricId","FileType","FileInfoId","FILE_INFO_
    ID")];
m_features2<-m_features2[,-c(7,8)];
model3 <- train(m_features2[train_ind,], lables[train_ind],
    trControl=train_control, method="nnet");
predict2_raw<-predict.train(model3, m_features2[-train_ind,],
     type="raw")
predict2_prob<-predict.train(model3, m_features2[-train_ind
    ,], type="prob")

```

قطعه کد آ.۳: ساخت مدل‌های پیش‌بینی

در قطعه کد آ.۴ پیش‌بینی‌های انجام شده ارزیابی می‌شوند.

```
auc(lables[-train_ind], predict1_prob$B)
auc(lables[-train_ind], predict2_prob$B)
auc(lables[-train_ind], predict3_prob$B)

plot(roc(lables[-train_ind], predict1_prob$B), col="blue",
    cex.lab=2, cex.axis=1.5);
plot(roc(lables[-train_ind], predict2_prob$B), add=TRUE, col=
    "red", lty=2);
plot(roc(lables[-train_ind], predict3_prob$B), add=TRUE, col=
    "black", lty=4);
 legend(x="bottomright",col=c("blue", "red", "black"),lwd=3,
    legend=c("Process","Process & Mutation", "Process &
    Mutation Base"),bty="6", lty=c(1,2,4), cex = 1.5)


confusionMatrix(predict1_raw, lables[-train_ind])
confusionMatrix(predict2_raw, lables[-train_ind])
confusionMatrix(predict3_raw, lables[-train_ind])
```

قطعه کد آ.۴: ارزیابی مدل‌های پیش‌بینی

پیوست ب

آماده‌سازی رایانه به عنوان سرور

انجام تحلیل جهش امری زمان‌بر است. به همین علت لازم است که رایانه‌ای به این فرآیند اختصاص یابد تا این کار بدون وقفه انجام شود و رفع خطا در زمان توسعه‌ی کد در هر مکان و زمانی امکان پذیر باشد. در ادامه گام‌های لازم برای تبدیل رایانه به سرور آمده است.

ب.۱ تنظیمات پایگاه داده

پایگاه‌داده‌ی مورد استفاده در این پژوهش MySQL 5.7.22 می‌باشد. در ابتدا لازم است امکان برقراری ارتباط از راه دور توسط *آی‌پی*[1] های خارج از رایانه فرآهم شود. در فایل mysqld.cnf پیکربندهای پایگاه داده وجود دارد و این فایل در آدرس /etc/mysql/mysql.conf.d قرار دارد. در این فایل لازم است که پارامتر bind-address با استفاده از # به *کامنت*[2] تبدیل شود.

سپس لازم است که یک کاربر مشخص شود که با هر آی‌پی بتواند به پایگاه‌داده وارد شود. این عمل می‌تواند با استفاده از نرم‌افزار Workbench به سادگی انجام شود. این نرم‌افزار ابزار طراحی، توسعه و مدیریت پایگاه‌داده است. در قسمت server و سپس users and previlages می‌توان به سادگی کاربر مورد نیاز را تعریف کرد.

پس از انجام این تنظیمات لازم است که پایگاه داده راه‌اندازی مجدد شود.

ب.۲ ارتباط با اینترنت

پیشنیاز اولیه هر سرور ارتباط با اینترنت می‌باشد. در برخی از شبکه‌ها برای برقراری این ارتباط لازم است از VPN مخصوص به آن شبکه استفاده شود. مشکلی که اغلب یک vpn دارد قطع شدن ارتباط آن است و لازم است این ارتباط پس از قطع دوباره ایجاد شود. قطعه کد ب.۱ هر ۳۰ ثانیه ارتباط را چک می‌کند و در صورت قطع vpn را مجددا راه اندازی می‌کند.

```
1 #!/bin/bash +x
```

[1] IP
[2] Comment

```
while [ "true" ]
  do
  CON="Sharif-ID2"
  STATUS=`nmcli con show --active | grep $CON | cut -f1 -d "
    "`
  if [ -z "$STATUS" ]; then
    echo "Disconnected, trying to reconnect..."
    (sleep 1s && nmcli con up $CON)
  else
    echo "Already connected !"
  fi
  sleep 30
done
```

قطعه کد ب.۱: راه اندازی مجدد vpn

ب.۳ رفع مشکل آی‌پی *پویا*[۳]

برای ارتباط با هر رایانه از راه دور لازم است که آدرس آن رایانه را داشته باشیم که این آدرس همان آی‌پی می‌باشد. در بسیاری از شبکه‌ها این آدرس به دلایل مختلف ثابت نیست. به منظور حل این مشکل می‌توان از سرویس‌هایی استفاده کرد که امکان *انقیاد*[۴] آی‌پی به آدرس URL را فراهم می‌کنند و همواره با تغییر آی‌پی، آدرس URL را به آی‌پی جدید متصل می‌کنند. سرویسی که در این پژوهش استفاده شد متعلق به سایت *noip* [۵] بود. این سایت یک برنامه هم فراهم می‌کند که این برنامه بر روی رایانه‌ی مورد نظر نصب می‌شود و در بازه‌های زمانی مشخص آدرس آی‌پی را برای سرویس ارسال می‌کند و انقیاد آدرس انجام می‌شود.

ب.۴ ارتباط با ترمینال

ترمینال این امکان را فراهم می‌سازد تا تمام عملیات‌های ممکن در یک سیستم عامل از طریق آن انجام شود. یکی از راههای متداول و مطمئن ارتباط از راه دور استفاده از پروتکل ssh می‌باشد. برای استفاده از این پروتکل لازم است یک سرویس SSH بر روی سرور راه اندازی شود یکی از نرم‌افزارهایی که این کار را انجام می‌دهد OpenSSH می‌باشد. نسخه‌ی Client از این ابزار نیز بر روی رایانه‌ی مشتری نصب می‌گردد. کاربران مجاز می‌توانند از طریق گذرواژه مشخص شده با سرویس SSH بر روی سرور ارتباط برقرار کنند. اما راه ایمن‌تر شناسایی از طریق کلید است. کاربر یک کلید عمومی و خصوصی تولید می‌کند. کلید عمومی

[3] Dynamic
[4] Bind
[5] www.noip.com

در نزد سرور نگهداری می‌شود و کلید خصوصی در نزد کاربر و برای برقراری ارتباط از این کلیدها استفاده می‌شود.

در ارتباط SSH یک ترمینال برای کاربر ایجاد می‌شود که برای اجرای پروژه در پس‌زمینه کافی نیست. زیرا با قطع ارتباط اجرا نیز متوقف می‌شود. برای دسترسی به چند ترمینال از طریق یک ترمینال می‌توان از ابزار Screen استفاده نمود. پس از برقراری ارتباط SSH در ترمینال باز شده ترمینال‌های مختلفی از طریق ابزار Screen می‌توان ایجاد و مدیریت کرد. هر یک از این ترمینال‌ها در پس زمینه می‌توانند کار خود را بدون توجه به وجود یا عدم وجود ارتباط SSH ادامه دهند.

ب.۵ ساخت و اجرای پروژه‌ی جاوا

برای *ساخت*[۶] و اجرای یک پروژه‌ی جاوا به صورت خودکار ابزارهای مختلفی وجود دارد. یکی از ابزارهای مناسب که به آن اشاره شد Maven است. لازم است که پیکربندی‌های مناسب جهت استفاده از وابستگی‌ها، کامپایل و ساخت فایل اجرایی jar در فایل pom.xml انجام شود. این تنظیمات در قطعه کد ب.۲ آمده است. سپس با استفاده از قطعه کد ب.۳ پروژه ساخته و اجرا می‌شود. از آنجا که پروژه به منظور کامپایل به نسخه‌ی ۸ جاوا کامپایلر نیاز دارد و در هنگام اجرا به نسخه‌ی ۷، این تنظمات به صورت خودکار انجام می‌شود. همچنین لازم نیست که با تغییر کد بر روی رایانه‌ی *مشتری*[۷] کدها مستقیما بر به سرور منتقل شوند. کافیست از سیستم کنترل نسخه استفاده شود. کدها با استفاده از ابزار گیت در سیستم کنترل نسخه‌ی بر روی *ابر*[۸] آپلود شود و سپس توسط همین ابزار در سرور دانلود شود. اصطلاحا به این عمل Pull و Push می‌گویند.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <build>
3  <plugins>
4  <plugin>
5  <groupId>org.apache.maven.plugins</groupId>
6  <artifactId>maven-compiler-plugin</artifactId>
7  <configuration>
8  <source>1.8</source>
9  <target>1.8</target>
10 </configuration>
11 </plugin>
12 <plugin>
13 <!-- Build an executable JAR -->
14 <groupId>org.apache.maven.plugins</groupId>
15 <artifactId>maven-jar-plugin</artifactId>
16 <version>3.1.0</version>
17 <configuration>
18 <archive>
19 <manifest>
```

[6] Build
[7] Client
[8] Cloud

```
<addClasspath>true</addClasspath>
<classpathPrefix>lib/</classpathPrefix>
<mainClass>Main</mainClass>
</manifest>
</archive>
</configuration>
</plugin>
<plugin>
<artifactId>maven-assembly-plugin</artifactId>
<configuration>
<archive>
<manifest>
<mainClass>Main</mainClass>
</manifest>
</archive>
<descriptorRefs>
<descriptorRef>jar-with-dependencies</descriptorRef>
</descriptorRefs>
</configuration>
</plugin>
</plugins>
</build>
```

قطعه کد ب.۲: پیکربندی pom.xml

```
#!/bin/bash +x
cd ~/IdeaProjects/bug_predict
echo 2 | sudo update-alternatives --config javac
mvn clean compile assembly:single
echo 1 | sudo update-alternatives --config javac
/usr/lib/jvm/java-8-openjdk-amd64/bin/java -jar target/com.bug.predict-1.0-
    SNAPSHOT-jar-with-dependencies.jar
```

قطعه کد ب.۳: ساخت و اجرای پروژه

کتاب‌نامه

[1] 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.

[2] N. Limsettho, K. E. Bennin, J. W. Keung, H. Hata, and K. Matsumoto, "Cross project defect prediction using class distribution estimation and oversampling," ***Information and Software Technology***, vol. 100, pp. 87–102, 2018.

[3] L. Chen, B. Fang, and Z. Shang, "Software fault prediction based on one-class svm," in ***Machine Learning and Cybernetics (ICMLC), 2016 International Conference on***, IEEE, vol. 2, 2016, pp. 1003–1008.

[4] J. Nam, "Survey on software defect prediction," ***Department of Compter Science and Engineerning, The Hong Kong University of Science and Technology, Tech. Rep***, 2014.

[5] 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.

[6] E. Arisholm, L. C. Briand, and M. Fuglerud, "Data mining techniques for building fault-proneness models in telecom java software," in ***Software Reliability, 2007. ISSRE'07. The 18th IEEE International Symposium on***, IEEE, 2007, pp. 215–224.

[7] 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.

[8] 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.

[9] D. Radjenović, M. Heričko, R. Torkar, and A. Živkovič, "Software fault prediction metrics: A systematic literature review," ***Information and Software Technology***, vol. 55, no. 8, pp. 1397–1418, 2013.

[10] F. Akiyama, "An example of software system debugging.," in ***IFIP Congress (1)***, vol. 71, 1971, pp. 353–359.

[11] M. H. Halstead, ***Elements of software science***. Elsevier New York, 1977, vol. 7.

[12] 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.

[13] T. J. McCabe, "A complexity measure," ***IEEE Transactions on software Engineering***, no. 4, pp. 308–320, 1976.

[14] R. Malhotra, "Comparative analysis of statistical and machine learning methods for predicting faulty modules," ***Applied Soft Computing***, vol. 21, pp. 286–297, 2014.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Don't touch my code!: Examining the effects of ownership on software quality," in ***Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering***, ACM, 2011, pp. 4–14.

[32] 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. [Online]. Available: http://doi.acm.org/10.1145/2610384.2628055.

[33] R. C. Martin, ***Clean code: a handbook of agile software craftsmanship***. Pearson Education, 2009.

[34] E. B. Johannessen, "Data mining techniques, candidate measures and evaluation methods for building practically useful fault-proneness prediction models," Master's thesis, University of Oslo, 2008.

[35] M. Kuhn ***et al.***, "Caret package," ***Journal of statistical software***, vol. 28, no. 5, pp. 1–26, 2008.

The Bug Prediction Model Based On Mutation Metrics

Abstract

Software developers notice existance of faults by report of a fault issue tracking systems or failure in software test. Then they try to locate bug and understanding the problem. Early detection of dault results in saving time and money and faciliates debugginh process. Prediction models can be built and used easly by modern statestical tools. Software metrics are the most important part of prediction models. Therfore higher perfomance in model can be achived using new and effective metrics. In this study, process metrics and metrics that built base on mutation analysis used and resulting models evaluted. In addition to using process metrics with mutation metrics, two group of metrics named ***mutation base process*** metrics and ***mutation-process hybrid*** introduced for building prediction models. Results showed that can mutation metrics can improve prediction prefromance of process metrics. Although mutation base process metrics have a predective value, the can not perform better than mutation metrics. Also mutation-process hybrid metrics can improve performance in prediction models significantly.

Keywords: Bug Prediction, Software Testing, Mutation Metrics, Process Metrics.

Sharif University of Technology
Computer Engineering Department

A thesis submitted in partial fulfillment of the requirements for the M.Sc. degree
Software Engineering

The Bug Prediction Model Based On Mutation Metrics

By:

Ali Mohebbi

Supervisor:

Dr. Hassan Mirian

August 2018