Scientific Journal of Electronic & Cyber Defense
Vol. 11, No. 4, Winter 2024, Serial No. 44
ISSN: 2322-4347 , E-ISSN: 2980-8979

Fault Proness Estimation of Software Modules Using Machine learning

reza torkashvan | Saeed Parsa * | babak vaziri

* Associate Professor, Faculty of Computer Engineering, Iran University of Science and Technology, Tehran, Iran.

(Received: 2023/09/07, Revised: 2023/12/06, Accepted: 2023/12/22, Published: 2024/01/18)
DOR: https://dorl.net/dor/20.1001.1.23224347.1402.11.4.4.1

ABSTRACT

To evaluate the software quality it is required to measure factors affecting the qualityof the software . Reliability and number of faults are examples of quality factors. If these factors are measured during the software development life cycle, more efficient and optimal activities can be performed to improve the software quality. The difficulty is that these factors could be measured only at the ending steps of the life cycle. To resolve the difficulty, these factors are indirectly measured through some software metrics which are available at the early stages of life cycle. These metrics are used as the input to fault prediction models and software components which may be faulty are the output of these models. Prediction of fault prone modules is a well known approach in software testing phase. When a module is predicted to be faulty, apparently more efforts have to be paid for correcting it. In addition to the module, all its dependent modules require specific consideration. When modifying a module all its dependent modules may be affected. The difficulty is that current known metrics for fault prediction do not reflect this situation. To resolve the difficulty, in this thesis, new metrics are introduced. Our experimental results show that the more the dependees of a module are changed, the more fault prone the module will be .

Keywords: Software quality, Fault prediction, Measurement, Software metrics, Fault prone modules, Change metrics, Dependency

* Corresponding Author Email: parsa@iust.ac.ir

نشریه علمی "پدافند الکترونیکی و سایبری"
سال یازدهم، شماره ۴، زمستان ۱۴۰۲، ص۴۵-۵۹
شاپا الکترونیکی: ۸۹۷۹-۲۹۸۰ شاپا چاپی: ۴۳۴۷-۲۳۲۲

علمی – پژوهشی

تخمین میزان خطا خیزی ماژول‌ها با استفاده از یادگیری ماشین

رضا ترکاشون[1*]، سعید پارسا[2]، بابک وزیری[3]
۱. دانشجوی دکتری ۲. دانشیار،دانشگاه علم و صنعت ایران، تهران، ایران.
۳. استادیار، گروه نرم افزار، واحد تهران مرکز، دانشگاه ازاد اسلامی، تهران، ایران.

(دریافت: ۱۴۰۲/۰۶/۱۶، بازنگری: ۱۴۰۲/۰۹/۱۵، پذیرش: ۱۴۰۲/۱۰/۰۱، انتشار: ۱۴۰۲/۱۰/۲۸)
DOR: https://dorl.net/dor/20.1001.1.23224347.1402.11.4.4.1

ناشر: دانشگاه جامع امام حسین (ع) نویسندگان

چکیده

برای آگاهی از میزان کیفیت نرم‌افزار لازم است عامل‌های مؤثر در کیفیت را اندازه بگیریم. میزان قابلیت اطمینان و تعداد خطا در نرم‌افزار از مهم‌ترین عامل‌های کمی هستند که کیفیت آنها را می‌سنجند. اگر بتوان این عامل‌ها را در حین چرخهٔ توسعهٔ نرم‌افزار اندازه‌گیری کرد، می‌توان فعالیت‌های مؤثر و بهتری را در راستای بهبود کیفیت نرم‌افزار انجام داد. مشکل اینجا است که این دسته از عامل‌ها در مراحل آخر توسعهٔ نرم‌افزار در دسترس خواهند بود. برای حل این مشکل، این عامل‌ها توسط معیارهایی اندازه‌گیری می‌شوند که در چرخهٔ توسعهٔ نرم‌افزار به‌صورت زودهنگام در دسترس هستند. معیارهای اندازه‌گیری شده ورودی‌های مدل پیش‌بینی خطا هستند و خروجی این مدل، ماژول‌هایی از نرم‌افزار هستند که احتمال بروز خطا در آنها وجود دارد. پیش‌بینی ماژول‌ها مستعد خطا، رویکردی اثبات شده در جهت تخصیص به‌موقع منابع پروژه، در مرحلهٔ آزمون نرم‌افزار است. هنگامی که یک ماژول به‌عنوان یک ماژول مستعد خطا پیش‌بینی می‌شود، تلاش بیشتری برای آزمون و تصحیح آن باید صرف شود. علاوه‌برآن ماژول، تمامی ماژول‌ها وابسته به آن نیز نیاز به رسیدگی ویژه‌ای دارند. زمانی که یک ماژول تغییر می‌کند تمامی ماژول‌ها وابسته به آن نیز ممکن است تحت‌تأثیر قرار بگیرند. مشکل در این است که معیارهای شناخته شده‌ای که در حوزه‌ی پیش‌بینی خطا مورداستفاده قرار می‌گیرند، این وضعیت را در نظر نمی‌گیرند. برای حل این مشکل، در این مقاله معیارهای جدیدی بر اساس تغییرات در موارد وابستگی ارائه شده است. نتایج تجربی به‌دست‌آمده نشان داد هرچه میزان تغییرات در ماژول‌ها مورد وابستگی بیشتر باشد، احتمال خطا در ماژول وابسته بیشتر می‌شود. با ارزیابی‌های صورت‌گرفته مشاهده شد معیار پیشنهادی قدرت پیش‌بینی نسبتاً بالایی دارد و به‌کاربردن آن برای ساخت مدل‌های پیش‌بینی خطا نیز افزایش قدرت پیش‌بینی را برای آنها در پی داشت.

کلید واژه‌ها: کیفیت نرم‌افزار، پیش‌بینی خطا، اندازه‌گیری، معیارهای نرم‌افزاری، ماژول‌ها مستعد خطا، تغییر، وابستگی.

۱. مقدمه[1]

در فرآیند توسعهٔ یک نرم‌افزار، مرحلهٔ آزمون از اهمیت بسیاری برخوردار است. هرچه این مرحله بادقت بیشتری انجام شود، کیفیت نرم‌افزار تولید شده بیشتر و نرم‌افزار تولید شده مطمئن‌تر خواهد بود. درعین‌حال، این فرآیند بسیار وقت‌گیر است و خصوصاً در مواردی که اندازهٔ نرم‌افزار بزرگ باشد، امری طاقت‌فرسا است. در این‌گونه موارد تخصیص منابع پروژه در مرحلهٔ آزمون بر روی مؤلفه‌های ضروری‌تر امری بسیار مهم است.

در مرحلهٔ آزمون، هدف افزایش کیفیت نرم‌افزار است. البته معمولاً به تمام جنبه‌های کیفی نمی‌توان پرداخت، بلکه برخی عامل‌های کیفی که برای ما حیاتی هستند را انتخاب و سعی می‌کنیم بر روی آنها تمرکز کنیم. این که کیفیت نرم‌افزار را بهبود داده‌ایم یا نه با اندازه‌گیری این عامل‌ها مشخص می‌شود. اما مشکل در این است که اندازه‌گیری آنها در مرحلهٔ توسعهٔ نرم‌افزار امکان‌پذیر نیست. به طور مثال اگر عامل کیفی را تعداد خطا در نظر بگیریم، در آغاز مرحلهٔ آزمون نمی‌توان آن را تعیین کرد تا سعی شود آن را بهبود داد. برای حل این مشکل، عامل کیفی موردنظر را از روی معیارهای نرم‌افزاری برای مؤلفه‌ها پیش‌بینی می‌کنند. با پیش‌بینی این عامل کیفی، می‌توان مؤلفه‌هایی که احتمال مطلوب نبودن و یا به عبارتی کم‌کیفیت بودن برای آنها وجود دارد را شناسایی و تمرکز خود را در مرحلهٔ آزمون برای بهبود آنها و در نتیجه بالاتر بردن کیفیت نرم‌افزار تولیدی، قرارداد [۲][۳][۴].

*Corresponding Author E-mail: Parsa@iust.ac.ir

پیش‌بینی خطا یک فعالیت مهم در تمرین مهندسی نرم‌افزار است. هدفی که این فعالیت دنبال می‌کند، ساخت مدل‌های پیش‌بینی کیفیت نرم‌افزار است که کیفیت ماژول‌های برنامه را تخمین می‌زنند، بدین صورت که تعیین می‌کنند آیا یک ماژول دارای خطا و یا عاری از خطا می‌باشد. این رویکرد امکان استفادهٔ مقرون‌به‌صرفه از منابع پروژه را فراهم می‌آورد. این‌گونه مدل‌ها می‌توانند خطادار و یا عاری از خطا بودن را برای مؤلفه‌های نرم‌افزاری پیش‌بینی کنند. این نوع از مدل‌ها، به‌عنوان مدل‌های رده‌بندی شناخته می‌شوند. اصلی‌ترین هدف در بحث پیش‌بینی خطای نرم‌افزار، افزایش قابلیت اطمینان و به‌طورکلی کیفیت نرم‌افزار از طریق تشخیص ماژول‌ها مستعد خطا در نرم‌افزار می‌باشد. پس از این که توسط مدل کیفی ایجاد شده برای نرم‌افزار، ماژول‌ها مستعد خطا شناسایی شدند، منابع مربوط به افزایش قابلیت اطمینان پروژه بین این ماژول‌ها تقسیم می‌شوند [۱] [۲].

در ادامه،در بخش دوم مقاله، مروری بر معیارهای مرسوم پیش‌بینی خطا آرایه شده است. در بخش سوم، نحوه افزودن معیار جدید به مدل‌های پیش‌بینی خطا تشریح شده است.. بخش چهارم مدل پیشنهادی معرفی شده، و در نهایت، بخش پنجم نتیجه‌گیری و کارهای آینده را بیان می‌کند.

۲- مروری بر معیارهای مرسوم پیش‌بینی خطا

بخشی از مقاله، روش‌های معیارهای مرسوم بر روش اندازه-گیری آنها ارائه می‌شود.

۲-۱- غفلت از اندازه‌گیری در مهندسی نرم‌افزار

قوانین مهندسی نرم‌افزار از روش‌هایی که مبتنی بر مدل‌ها و تئوری‌ها هستند استفاده می‌کنند. به طور مثال در طراحی مدارهای الکتریکی ما از تئوری‌هایی همچون قانون اهم استفاده می‌کنیم که ارتباط بین مقاومت، جریان و ولتاژ را در مدار بیان می‌کند.

زیربنای پردازش‌های علمی اندازه‌گیری است: اندازه‌گیری متغیرها برای متمایز ساختن حالت‌ها، اندازه‌گیری تغییرات در رفتار و اندازه‌گیری علل و تأثیرات. هنگامی که یک روش علمی اعتبار یک مدل و یا صحت یک تئوری را مشخص می‌کند، ما برای اعمال تئوری در عمل از اندازه‌گیری استفاده می‌کنیم و ارزیابی منابع حاصل از به‌کارگیری آن به‌صورت جدی در اغلب پروژه‌های نرم‌افزاری پیگیری نمی‌شود.

۱) در تعیین اهداف قابل اندازه‌گیری برای محصولات نرم‌افزاری خود شکست می‌خوریم. به طور مثال ما وعده می‌دهیم که محصول کاربرپسند، قابل اطمینان و قابل نگهداری خواهد بود بدون اینکه به طور روشن و هدفمندی مشخص کنیم که این واژگان چه معنایی دارند. گیلب می‌گوید که پروژه‌های بدون اهداف روشن، به اهداف خود نخواهند رسید[۷].

۲) درتعیین و فهم هزینه‌های مربوط به مؤلفه‌ها در پروژه‌های نرم‌افزاری شکست می‌خوریم. به طور مثال، اغلب پروژه‌ها نمی‌توانند بین هزینه‌ی طراحی و هزینه‌ی پیاده سازی و آزمون تمایز قائل شوند. در نتیجه چون هزینه‌ی اضافی یک امر پذیرفته شد توسط مشتری‌ها است، اگر هزینه مربوط به مؤلفه‌ها را نتوانیم اندازه‌گیری کنیم، نمی‌توانیم امید کنترل هزینه‌ها را داشته باشیم.

۳) نمی‌توانیم کیفیت محصولات را اندازه‌گیری و یا پیش‌بینی کنیم. بنابراین نمی توان به کاربرانی که قصد استفاده از سیستم نرم‌افزاری را دارند، گفت که یک محصول چقدر قابل اطمینان خواهد بود، نظر احتمال شکست در یک بازه‌ی زمانیِ استفاده از آن چه خواهد بود.

۲-۲- اهداف اندازه‌گیری در نرم‌افزار

اندازه‌گیری برای تعیین وضعیت پروژه‌ها، محصولات، فرآیندها و میزان منابع موردنیاز است. از آن جا که ما همیشه نمی‌دانیم چگونه محصول نرم‌افزار شکست می‌خورد، لذا لازم است تا ویژگی‌های نرم‌افزار خوب همانند نرم‌افزار بد اندازه‌گیری و ذخیره شوند. تام دی مارکو که یک پشتیبان قوی دررابطه‌با به اندازه‌گیری معیارهای کیفت مدل کیفی که نرم‌افزار برای اندازه-گیری کیفیت نرم‌افزار در فرایند تولید باید استفاده شوند که معیارهای سنجش کیفیت نرم‌افزار را اندازه‌گیری نمایند در فرآیند توسعه نرم‌افزار است، می‌گوید: چیزی را که نمی‌توانی اندازه‌گیری کنید، نمی‌توانی کنترل کنید [۳] [۶].

اندازه‌گیری، یک فعالیت مهندسی است. اندازه‌گیری باید برای هدفی خاص یا نیازی که به‌روشنی تعریف شده و قابل‌فهم است، انجام گردد.

معیارهایی استفاده می‌شود که به ما کمک می‌کند که رویدادهای در حین توسعه بی‌پرده است سپس مرزهایی را معین می‌کنیم که بهتری تولید نماییم. در این جا، اندازه‌گیری‌ها درستی فرآیند و محصول‌هایی را برای ما بیشتر قابل‌رؤیت می‌کنند و فهم بهتری از روابط بین فعالیت‌ها و موجودیت‌هایی که بر روی آنها اثر می‌گذارند می‌دهد [۴] [۷]. در دومین گام، اندازه‌گیری به ما کمک می‌کند تا آن چه در پروژه در حال رخدادن است را کنترل کنیم. با استفاده از حدّ و مرزها، اهداف و فهم روابط ما می‌توانیم پیش‌بینی کنیم که محتمل است چه چیزی اتفاق بیفتد و تغییراتی را در محصولات و فرآیندها انجام دهیم که به ما کمک کند به اهداف خود برسیم. در سومین گام، اندازه‌گیری ما را تشویق می‌کند تا فرآیندها و محصولات بهبود و یا پیش‌بینی کنید.

۲-۳- معیارهای نرم‌افزاری

معیارهای نرم‌افزاری واژه‌ای است که فعالیت‌های زیادی را در بر می‌گیرد. ویژگی‌های مختلف نرم‌افزار را اندازه‌گیری می‌کنند.

برخی از این معیارها و فعالیت‌ها عبارت‌اند از:

- ❖ تخمین تلاش و هزینه
- ❖ مدل‌ها و اندازه‌گیری‌های بهره‌وری
- ❖ جمع‌آوری داده
- ❖ اندازه‌گیری‌ها و مدل‌های کیفی
- ❖ مدل‌های قابلیت اطمینان
- ❖ مدل‌ها و ارزیابی کارایی
- ❖ معیارهای پیچیدگی و ساختاری
- ❖ ارزیابی قابلیت بلوغ
- ❖ مدیریت توسط معیارها
- ❖ ارزیابی روش‌ها و ابزار

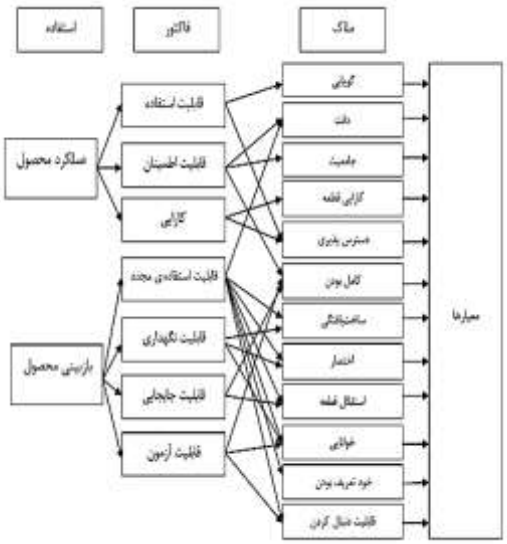
باتوجه‌به موضوع بحث این مقاله چهار فعالیت جمع‌آوری داده، اندازه‌گیری و طراحی مدل‌های کیفی و طراحی مدل‌های قابلیت اطمینان را اندکی شرح خواهیم داد.

۲-۴- جمع‌آوری داده‌ها

کیفیت هر عملیات اندازه‌گیری به طور دقیقی وابسته به جمع‌آوری داده است. اما جمع‌آوری داده در گفتار آسان‌تر از عمل است، خصوصاً وقتی که داده می‌بایست از بین مجموعهٔ مختلفی از پروژه‌ها جمع‌آوری شود؛ بنابراین جمع‌آوری داده خود به یک حوزهٔ پژوهشی تبدیل شده و متخصصین تلاش می‌کنند تا اطمینان یابند داده‌های جمع‌آوری‌شده بر اساس معیارهای هدف ارزیابی و مورد تأیید می باشد [۷].

۲-۵- اندازه‌گیری و مدل‌های کیفی

بدون درنظرگرفتن برآورد کیفیت محصول، سرعت تولید معنا ندارد. این دیدگاه باعث شده است تا مهندسین نرم‌افزار مدل‌های کیفی را توسعه دهند که اندازه‌گیری‌های مربوط به این مدل‌ها می‌تواند با اندازه‌های مربوط به مدل‌های بهره‌وری ترکیب شود. برای مثال مدل پیشرفتهٔ تخمین هزینهٔ بوهم (کوکومو) یک مدل کیفی است. مدل کیفی اِم‌سی‌کال مربوط به بهره‌وری است. این مدل‌ها معمولاً در یک ساختار درختی ساخته می‌شوند، در شکل (۱) شاخه‌های بالایی شامل عامل‌های کیفی سطح بالا برای محصولات نرم‌افزاری هستند، همچون قابلیت اطمینان و قابلیت استفاده که ما می‌خواهیم این عامل‌ها را اندازه‌گیری کنیم. هر عامل کیفی شامل معیارهای سطح پایین‌تری است، همچون ساخت‌یافتگی یا قابلیت ردیابی. ملاک‌ها راحت‌تر از پارامترهای سنجه فهمیده و اندازه‌گیری می‌شوند؛ بنابراین ما می‌توانیم پارامترهای سنجه را از طریق اندازه‌های مربوط به ملاک‌های وابسته به آنها اندازه‌گیری کنیم. این رویکرد تقسیم و حل به‌عنوان یک رویکرد استاندارد برای اندازه‌گیری کیفیت نرم‌افزار پیاده‌سازی شده است [۲] [۷].



شکل (۱): مدل کیفی نرم‌افزار دی‌مارکو[۷]

۲-۶-پیش‌بینی خطای نرم‌افزار

باتوجه‌به گسترش روزافزون استفاده از نرم‌افزارها در سیستم های حساس به‌سلامت، توجه به فرایندهای سنجش معیارهای مختلف ارزیابی کیفیت نرم‌افزار شده است عموماً این معیارها مربوط به سنجش کیفیت، پس از تولید به‌کارگیری نرم‌افزار قابل‌سنجش هستند. امروزه توجه زیادی به توسعهٔ مدل‌ها می‌شود که اندازه‌های مربوط به کیفیت نرم‌افزار را که دیرهنگام در فرآیند توسعه به دست می‌آیند، از روی اندازه‌هایی همچون پیچیدگی که زودهنگام به دست می‌آیند، پیش‌بینی کنند. به طور مثال اگر ما بخواهیم از معیارهای نرم‌افزاری برای پیش‌بینی تعداد کل تغییرات برنامه در ماژول‌ها یک سیستم نرم‌افزاری استفاده کنیم، یک مدل پیش‌بینی خوب امکان شناسایی زودهنگام ماژول‌هایی که به‌احتمال بیشتری نیاز به اصلاح دارند را به ما خواهد داد. این مدل‌ها به‌عنوان‌مثال: معیارهای پیچیدگی همچون اندازه‌های عددی مربوط به تفاوت‌های ذاتی بین ماژول‌ها نرم‌افزاری اندازه-گیری می‌کنند. اطلاعاتی که ازاین‌گونه طریق به دست می‌آید ممکن است در توسعه و مدیریت سیستم‌های پیچیدهٔ نرم‌افزاری به‌خصوص در شرایطی که معیارهای پیچیدگی بسیار مرتبط با تغییرات برنامه هستند، به کار گرفته شوند. در این صورت، منابع پروژه ممکن است به طور مؤثری به ماژول‌های برنامه تخصیص داده شوند و بر روی ماژول‌هایی متمرکز شود که به‌احتمال بیشتر نیاز به اصلاح دارند.

۳- افزودن معیار جدید به مدل‌های پیش‌بینی خطا

در برخی از مطالعات، بنا به کاستی‌های دیده شده در مجموعه معیارهای نرم‌افزاری موجود که در ساخت مدل‌های پیش‌بینی خطا استفاده می‌شوند، به تعریف یک یا چند معیار برای

سنجش کیفیت نرم افزار پرداخته شده است. از جمله می‌توان به [۳][۶] اشاره کرد که معیار برش‌بندی را تعریف کرده‌اند.

۳-۱- معیارهای مبتنی بر توالی تغییرات

ناگاپان و همکاران در [۱۸] برای اولین بار مجموعه معیاری براساس توالی تغییرات ارائه کردند. به این صورت که می‌توان تاریخچه‌ی یک سیستم نرم‌افزاری را به دنباله‌ای از رویدادها تقسیم کرد و در آن نقاط تعیین کرد آیا تغییراتی صورت گرفته است یا نه. یک نمونه از این رویدادها می‌تواند براساس تاریخ روزها باشد و به این صورت می‌توان آخر هفته‌ها و روزهای تعطیل را به عنوان نقاطی برای شکستن سری‌های متوالی در نظر گرفت. به طور مثال در مورد سیستم ویندوز عنوان شده است که در هر روز کاری مجموعه‌ی تغییرات در سیستم ثبت می‌شوند و قابل استخراج از سیستم کنترل نسخه می‌باشند. بنابراین یک سیستم S به صورت دنباله‌ای از سازه[1]ها $S = <s_1, s_2, \dots>$ درنظر گرفته می‌شوند که هر سازه با دیگری تفاوت دارد. هم‌چنین هر سازه از چندین مؤلفه شکل می‌گیرد. بنابراین، هر مؤلفه در یک سیستم خود تاریخچه‌ای در طی سازه‌ها دارد؛ یعنی دارای یک توالی از نسخه‌ها می‌باشد: $C = <c_1, c_2, \dots, c_{|S|}>$. اگر رابطه‌ی $c_i \neq c_{i-1}$ برقرار باشد، بدین معنی است که مؤلفه‌ی C در سازه S_i تغییر کرده است.

حال برای هر مؤلفه، توالی تغییرات تعیین می‌گردد. این توالی تغییرات بر اساس دو پارامتر مشخص می‌شوند:

۱. اندازهٔ شکاف[2]. حداقل فاصله بین دو تغییر را مشخص می‌کند. اگر دو تغییر فاصله‌ای کم‌تر از اندازهٔ شکاف داشته باشد، آن دو تغییر جزء یک توالی خواهند بود.
۲. اندازهٔ توالی. این عامل حداقل تعداد تغییرات در یک توالی را مشخص می‌کند. اگر تعداد تغییرات در یک توالی b کم‌تر از اندازهٔ توالی باشد، آن تغییرات به‌عنوان یک توالی در نظر گرفته نمی‌شوند.

۳-۲- بررسی تأثیر وابستگی در پیش‌بینی خطا

تغییر کد در سیستم‌های نرم‌افزاری بزرگ کاری بس دشوار است و نیاز به فهم بالایی از وابستگی‌های بین مؤلفه‌های نرم‌افزاری دارد. تصحیح مؤلفه‌ها بدون توجه کافی به وابستگی‌ها ممکن است اثر سوئی بر روی کیفیت دیگر مؤلفه‌ها داشته باشد، بدین معنی که ریسک آنها برای شکست را افزایش می‌دهد. در برخی تحقیقات بر روی حوزه‌ی پیش‌بینی خطا به بررسی تأثیر وابستگی‌ها در ایجاد خطا پرداخته شده است. از آن جمله می‌توان به [۲۱] [۲۶] [۳۲] اشاره کرد. در ادامه بنا به جایگاه بحث، به دلیل مرتبط بودن [۱۸] به کار صورت گرفته در بستر این مقاله به توضیح آن خواهیم پرداخت.

۳-۳- وابستگی

وابستگی بین مؤلفه‌های نرم‌افزاری در سیستم‌های نرم‌افزاری بسیار گسترده هستند و برای کارکرد موفقیت‌آمیز آنها بسیار حیاتی هستند. این وابستگی‌ها گاهی اوقات به دلایل استراتژیکی همچون فهم کارایی‌های خاصی در سیستم و ایجاد امکان استفادهٔ مجدد از نرم‌افزار ایجاد می‌شوند و یا اثر جانبی ساختار سازمانی هستند.

وابستگی‌ها جریان اطلاعاتی درون سیستم نرم‌افزاری را نشان می‌دهند و بر روی موفقیت و کیفیت محصول اثرگذار هستند. در [۱۸]، تأثیر وابستگی‌ها بر روی کیفیت یک مؤلفه‌ی نرم‌افزاری (در واقع بر روی احتمال خطادار بودن آن) از طریق بررسی وضعیت همسایه‌های آن مؤلفه مورد مطالعه قرار گرفته است.

منظور از همسایه‌های یک مؤلفه، مؤلفه‌هایی هستند که مؤلفهٔ موردنظر به آنها وابسته است و یا آنها به مؤلفهٔ موردنظر وابسته هستند. به طور مثال، دو مؤلفهٔ A و B را در نظر بگیرید که A وابسته به B هست. اگر مؤلفهٔ B تحت مقداری درهم‌ریختگی در کد در بین نسخه‌های نرم‌افزار قرار بگیرد، معقول است که انتظار داشته باشیم مؤلفهٔ A نیز دستخوش تغییرات شود تا با B هماهنگ شود شکل(۲)؛ بنابراین اثر درهم‌ریختگی کد ممکن است در بین وابستگی‌ها انتشار یابد و در نتیجه کیفیت آنها را با معرفی خطاهای جدید در آنها تحت‌تأثیر منفی قرار دهد. فرضیه‌ای که در [۲۱] سعی در بررسی آن شده است، این است که کیفیت یک مؤلفه ممکن است توسط ویژگی‌های مؤلفه‌های وابسته به آن همچون: اندازه، درهم‌ریختگی کد، پیچیدگی، پوشش آزمون و ساختار سازمانی تحت تأثیر قرار بگیرد.

شکل (۲): مؤلفه‌ی A به مؤلفه‌ی B وابسته است

۳-۴-بررسی تأثیر تعداد معیارها بر روی مدل پیش‌بینی خطا

یک مدل پیش‌بینی خطا با استفاده از معیارهای نرم‌افزاری و داده‌های خطای جمع‌آوری شده از نسخه‌های قبلی نرم‌افزار و یا پروژه‌های مشابه آموزش داده می‌شود. پس از آموزش مدل می‌توان آن را بر روی ماژول‌های برنامه که داده‌های خطادار آنها مشخص نیست اعمال کرد. تعداد ویژگی‌ها و مشخصات معیارهای نرم‌افزاری، کارایی و میزان تأثیر مدل پیش‌بینی خطا را تحت تأثیر قرار می‌دهد. یکی از بحث‌هایی که مطرح می‌شود این است که چه تعداد معیار برای ساخت مدل مورد نیاز است. در [۲۳] پژوهشی بر روی تعداد معیارهای مورد استفاده صورت گرفته است. از دیدگاه اهل فن، مدل‌سازی با مجموعه‌ی کوچکی از معیارها بسیار خوشایند است. هنگامی که یک مجموعه از معیارها را در اختیار داریم به نظر می‌رسد تعدادی از آنها در بدست آوردن دانش مورد نیاز در مورد پروژه زائد هستند. برخی از آنها دانش

1 Build
2 Gap size

افزونه‌ای را تولید می‌کنند یا اطلاعات جدیدی را فراهم نمی‌آورند یا در برخی موارد تأثیر مخرّبی بر روی مدل پیش‌بینی خطا دارند. به طور مثال، تحقیقات اخیر در[۲۷] نشان می‌دهد که هنگامی که ویژگی‌های افزونه و غیرمرتبط حذف می‌شوند در کارایی مدل‌های پیش‌بینی خطا بهبود حاصل می‌شود.

فرایند انتخاب ویژگی[۱]، فرآیند انتخاب زیرمجموعه‌ای از ویژگی‌ها می باشد و بطورکلی به دو دسته‌ی زیر فرایند، رتبه‌بندی ویژگی[۲] و انتخاب زیرمجموعه‌ای از ویژگی[۳] تقسیم می‌شود. رتبه‌بندی ویژگی، ویژگی‌ها را بر اساس قدرت پیشگویانه‌ی آنها مرتب می‌کند در حالی که انتخاب زیرمجموعه‌ای از ویژگی، زیرمجموعه‌ای از ویژگی‌ها را که مجموعاً قدرت پیشگویانه‌ی خوبی دارند را پیدا می‌کند. فیلترها، الگوریتم‌های انتخاب ویژگی‌ای هستند که در آنها یک زیرمجموعه از ویژگی‌ها بدون به کار بردن هرگونه الگوریتم یادگیری انتخاب می‌شود. بحث بیشتر در مورد انتخاب ویژگی در حوزه‌ی این پژوهش نمی‌گنجد. روش-های مختلف انتخاب ویژگی در حوزه‌ی کیفیت و قابلیت اطمینان نرم‌افزار خیلی محدود است. در زمینه‌ی پیش‌بینی خطای نرم‌افزار، در [۲۳] از روش انتخاب ویژگی مبتنی بر آستانه ($TBFS$)[۴] استفاده شده است. پنج نسخه‌ی مختلف و مؤثّر از انتخاب ویژگی مبتنی بر آستانه در [۲۳] مورد استفاده قرار گرفته است. زیرمجموعه‌هایی با اندازه‌های مختلف از ویژگی‌ها توسط پنج روش انتخاب ویژگیِ مبتنی بر آستانه انتخاب می‌شوند. بعد از آن توسط سه رده‌بندی[۵]، MLP[۶]، KNN[۷] و LR[۸] مدل‌های رده‌بندی ساخته می‌شوند. در نهایت کارایی یک رده‌بند توسط معیار کارایی بنام AUC[۹] ارزیابی می‌شود. در [۲۳] از نه مجموعه‌ی داده‌ای مربوط به پروژه‌ی إکلیپس استفاده شده است و مدل با ویژگی‌های ذکر شده در بالا بر روی این مجموعه‌های داده‌ای ساخته می‌شود.

۳-۴-۱- رتبه‌بندی ویژگی مبتنی بر آستانه

روش‌های رتبه‌بندی مبتنی بر فیلتر، ویژگی‌ها را مستقل از هرگونه الگوریتم آموزشی[۱۰] مرتّب می‌کنند. سپس بهترین ویژگی‌ها از لیست رتبه‌بندی انتخاب می‌شوند. راه‌های مختلفی برای رتبه‌بندی ویژگی‌ها وجود دارد که در [۲۳] از پنج روش رتبه‌بندی مبتنی بر آستانه استفاده شده است. در این روش ابتدا مقدار هر ویژگی بوسیله‌ی نگاشت F^j به $\hat{F}^j$ بین صفر و یک نرمال می‌شود. مقادیر نرمال شده به صورت احتمالات خلفی[۱۱] عمل می‌کنند. هر ویژگی مستقل که یک متغیر پیش‌بینی کننده‌ی نرم‌افزاری است، با یک ویژگیِ رده[۱۲] جفت می‌شود. این ویژگیِ رده یکی از دو برچسبِ "خطادار " و یا "عاری از خطا " است. مجموعه‌ی داده‌ای کاهش یافته با استفاده از پنج معیار مختلف کارایی و براساس یک مجموعه از احتمالات خلفی ارزیابی می‌شود. در رده‌بندی دوتایی استاندارد، براساس آستانه‌ی تصمیمِ[۱۳] پیش‌فرض که برابر با ۰،۵ است، به هر ویژگی مستقل، رده‌ی پیش‌بینی شده اختصاص داده می‌شود. آستانه‌ی تصمیم پیش‌فرض همیشه بهینه نیست، خصوصاً هنگامی که توزیع رده‌ی مربوطه متوازن نیست. بنابراین، در [۲۳] پیشنهاد شده است که از معیارهای کارایی استفاده کنیم که بتوانند در نقاط مختلفی از دامنه‌ی توزیع $\hat{F}^j$ محاسبه شوند. در هر نقطه‌ی آستانه، مقادیر بالاتر از آستانه به عنوان مقدار مثبت و مقادیر پایین‌تر از آستانه بعنوان مقدار منفی رده‌بندی می‌شوند. پس از آن مثبت و منفی بودن را جابجا می‌کنند، یعنی مقادیر بالاتر از آستانه را منفی در نظر می‌گیرند و برای مقادیر کمتر از آستانه مقدار مثبت را در نظر می‌گیرند. جهتی از نشان‌گذاری مثبت و منفی که منجر به تولید مقادیر بهینه‌ی بیشتری برای ویژگی‌ها شود، مورد استفاده قرار می‌گیرد.

Threshold-Based Feature Selection Algorithm
input :
1. Data set D with features F^j, j = 1, … , m;
2. Each instance $x \in D$ is assigned to one of two classes $c(x) \in \{fp, nfp\}$;
3. The value of attribute F^j for instance x is denoted as $F^j(x)$;
4. Metric $\omega \in \{MI, KS, DV, AUC, PRC\}$;
5. A predefined threshold: number (or percentage) of the features to be selected.
output:
Selected feature subsets.
for F^j, j = 1, . .. , m do
Normalize $F^j \rightarrow \hat{F}^j = F^j - min(F^j)$ / $max(F^j) - min(F^j)$;
Calculate metric ω using attribute $\hat{F}^j$ and class attribute at various
decision threshold in the distribution of $\hat{F}^j$. The optimal ω is used, $\omega(\hat{F}^j)$).
Create feature ranking R using $\omega(\hat{F}^j)$ $\forall j$.
Select features according to feature ranking R and a predefined threshold.

شکل (۳): الگوریتم رتبه‌بندی ویژگی مبتنی بر آستانه

در مسئله‌ی رده‌بندی دوتایی، همچون خطادار بودن و فاقد خطا بودن، چهار نرخ رده‌بندی ممکن وجود دارد: نرخ مثبت

1 Feature selection
2 Feature ranking
3 Feature subset selection
4 Threshold-Based Feature Selection
5 Classifier
6 Multi-Layer Perceptron networks
7 K-Nearest Neighbors
8 Logistic Regression
9 Area Under the Curve
10 Learning
11 Posterior probabilities
12 Class attribute
13 Decision threshold

صادق، نرخ مثبت کاذب، نرخ منفی صادق، نرخ منفی کاذب. این چهار نرخ رده‌بندی می‌توانند در هر آستانهٔ $t \in [0,1]$ نسبت به ویژگی نرمال‌شدهٔ $\hat{F}^j$ به دست آید. تکنیک رتبه‌بندی ویژگی مبتنی بر آستانه شکل (۳)، نرخ‌های رده‌بندی را آن چنان که در ادامه آمده است به کار می‌گیرد.

۱) اطلاعات متقابل (MI)، وابستگی متقابل بین دو متغیر تصادفی را اندازه‌گیری می‌کند. اطلاعات متقابل زیاد، کاهش زیادی در عدم قطعیت را مشخص می‌کند و اطلاعات متقابل برابر با صفر بین دو متغیر تصادفی به معنی این است که متغیرها مستقل هستند.

۲) کُلمُگُروف-اِسمیرنُف (KS)، از فرآیند آماری کُلمُگُروف-اِسمیرنُف استفاده می‌کند تا حداکثر اختلاف بین توابع توزیع تجربی مربوط به مقادیر ویژگی‌ها (معیارهای نرم‌افزاری) نمونه‌ها (ماژول‌های برنامه) در هر رده را اندازه‌گیری کند. این مقدار به طور مؤثری برابر است با حداکثر اختلاف بین منحنی‌هایی که وقتی آستانهٔ تصمیم بین ۰ و ۱ تغییر کند، توسط نرخ‌های مثبت صادق و مثبت کاذب ایجاد می‌شود.

۳) دِوِینس (DV)، برابر است با مجموع باقیمانده از مربعات بر اساس آستانهٔ t. این تکنیک مجموع مربعات خطا را از دستهٔ میانگین که فضا را بر اساس آستانهٔ t افراز می‌کند، اندازه‌گیری می‌کند. ازآنجاکه دِوِیِنس خطاها را ارائه می‌دهد، مقدارِ حداقل برای آن بهینه است.

۴) فضای زیر منحنی راک[1] (ROC)، به طور وسیعی برای اندازه‌گیری کارایی مدل رده‌بندی به کار می‌رود. منحنی راک برای مشخص‌کردن مصالحه[2] بین نرخ مثبت صادق و مثبت کاذب به کار می‌رود. در [۲۳] منحنی‌های راک با تغییر دادن آستانه‌ی تصمیم t ایجاد می‌شوند و برای انتقال مقادیر نرمال‌شده‌ی یک ویژگی به دسته‌ی پیش‌بینی شده برای آن به کار می‌رود.

۵) فضای زیر منحنی فراخوانی مجدد-دقت[3] (PRC)، یک اندازه‌گیری است که تنها دارای یک مقدار است که از فضای بازیابی اطلاعات به دست می‌آید. فضای زیر PRC از ۰ تا ۱ متغیر است. نمودار PRC مصالحه بین فراخوانی مجدد و دقت را نمایش می‌دهد. فراخوانی مجدد و دقت دو روش اندازه‌گیری برای ارزیابی کارایی سیستم‌های بازیابی اطلاعات است

۳-۴-۲- رتبه‌بندی ویژگی با استفاده از رده‌بندها

در مسئله‌ی رده‌بندی[4]، یک مجموعه از رکوردها موجود است که به‌عنوان داده‌های آموزش شناخته می‌شوند و هر رکورد دارای چند ویژگی است. یکی از این ویژگی‌ها که متغیر وابسته نامیده می‌شود، مشخص می‌کند که هر رکورد به کدام یک از دسته‌ها تعلق پیدا می‌کند. هدف رده‌بندی این است که مدلی را بسازد که متغیر وابسته را برحسب ویژگی‌های دیگر که اغلب متغیرهای مستقل نامیده می‌شوند، حدس بزند. هنگامی که این‌چنین مدلی ساخته شد، از این مدل می‌توان برای تعیین دستهٔ رکوردهایی که رده‌بندی نشده‌اند استفاده کرد. سه رده‌بند که در [۲۳] بکار رفته‌اند، عبارتند از: MLP ، KNN و LR. تمامی این سه یادگیرنده هیچ کدام قابلیت انتخاب ویژگی را ندارند و معمولاً در جامعه‌ی نرم‌افزاری مورد استفاده قرار می‌گیرند. تمامی رده‌بندها در ابزار وِکا پیاده‌سازی شده‌اند و از پارامترهای پیش‌فرض مشخص شده در ابزار وِکا برای این سه الگوریتم یادگیرنده استفاده شده است. تنظیمات پارامترها تنها برای افزایش کارایی رده‌بند به میزان قابل توجه، تغییر داده می‌شوند.

- ❖ MLP: سعی می‌کند تا هوشمندانه عملکرد سیستم عصبی را تقلید کند. در واقع یک شبکهٔ عصبی با یک‌لایهٔ پنهان می‌باشد. در [۲۳] این لایه‌ی پنهان با سه گره تعریف شده است و از ۱۰٪ داده‌های آموزش[5] به عنوان مجموعه‌ی داده‌ای برای ارزیابی بکار می‌رود تا مشخص شود که فرآیند تکراری آموزش چه موقع متوقف شود.
- ❖ KNN: در دستهٔ یادگیرنده‌های تنبل[6] قرار می‌گیرد. انتخاب معیار فاصله بسیار مهم است.
- ❖ LR: یک روش آماری است که می‌تواند برای حل مسئله‌ی رده‌بندی دوتایی بکار رود. مدل رگرسیون لجستیک بر اساس داده‌های آموزش ساخته می‌شود و برای تصمیم‌گیری در مورد عضویت در دسته برای نمونه‌های بعدی بکار می‌رود.

۴- معیار پیشنهادی

۴-۱- مقدمه

در این بخش مقاله به ارائه‌ی معیار جدید بر اساس تغییرات و وابستگی در کد می‌پردازیم. در بخش قبل به بررسی کارهای صورت‌گرفته در زمینهٔ تغییرات و وابستگی‌ها پرداختیم. یک نکتهٔ مهم که با ارائه‌ی این معیار به دنبال آن بودیم، بررسی این نکته بود که باتوجه‌به تأثیر وابستگی‌ها و تغییرات در خطادار شدن یک ماژول، این دو عامل در هیچ‌کدام از کارها به‌صورت ترکیبی دیده

1 Receiver Operating Characteristic
2 Trade-off
3 Precision-Recall Curve
4 Classification
5 Training data
6 Lazy learners

نشده بودند. این در حالی است که تغییرات معمولاً از عوامل ایجاد خطا در ماژول‌ها وابسته هستند.

۴-۲- شرح ایده

به طور تجربی این نکته به‌دست‌آمده است که تغییراتی که در کد ایجاد می‌شود، منشأ بسیاری از خطاها در نسخهٔ بعدی نرم‌افزار می‌باشد. به طور مثال در پژوهشی مربوط به گروه تحقیقاتی مایکروسافت است[۱۷]، مدلی برای پیش‌بینی شکست در ماژول‌ها نرم‌افزار ایجاد شده و این مدل در قالب یک ابزار به نام کِرین[1] مورد استفاده قرار گرفته است. در این مقاله از معیار درهم‌ریختگی کد به عنوان یکی از معیارهای ساخت مدل استفاده شده بود. در بخش مروری بر کارهای گذشته به شرح معیار درهم‌ریختگی در کد پرداختیم. پس از مطرح شدن معیارهای مبتنی بر تغییرات در مجامع علمی، در اکثر قریب به اتفاق کارهای پژوهشی جدید این دسته از معیارها در ساخت مدل به کار رفته‌اند که از آن جمله می‌توان به [۷]، [۱۸]، [۱۹]، [۲۰]، [۲۱]، [۲۵] اشاره کرد.

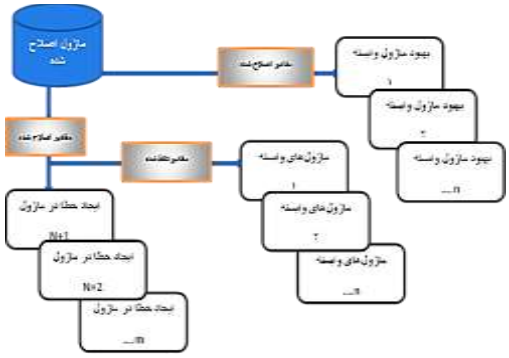
کاربرد وسیع معیارهای مبتنی بر تغییرات، باعث شد تمرکز ما بر روی تغییرات در کد قرار گیرد. فرآیند توسعهٔ یک نرم‌افزار را می‌توان به‌صورت دنباله‌ای از تغییرات در نظر گرفت، فعالیت‌هایی که ویژگی خاصی را به نرم‌افزار اضافه می‌کنند، تغییراتی را برای سازگاری با محیط‌های جدید انجام می‌دهند و یا ویژگی‌هایی را حذف می‌کنند. تمامی این فعالیت‌ها توسط انسان صورت می‌پذیرد و از آن جا که انسان دارای خطا است، این اجتناب‌ناپذیر است که این تغییرات خطایی را در سیستم سبب نشوند.

یک نکتهٔ بسیار مهم این است که تغییر در یک ماژول علاوه بر این که ممکن است منجر به بروز خطا در همان ماژول شود، احتمال بروز خطا را در ماژول‌ها وابسته افزایش می‌دهد.

در مطالعاتی که صورت گرفت، مشاهده شد که وابستگی‌ها در کد نیز به‌عنوان یک عامل مؤثر در ایجاد خطا می‌باشد. در [۱۸] کاری که روی وابستگی صورت گرفته است به این صورت است که برای هر ماژول، مقادیر معیارهای مختلف را برای ماژول‌ها دارای وابستگی به آن محاسبه کرده و تأثیر آن را در خطادار شدن آن ماژول بررسی کرده است. نتیجه‌ای که در این تحقیق به دست آمد نشان می دهد که معیارهای همچون اندازه و درهم‌ریختگی یک ماژول در خطادار شدن ماژول‌های وابسته موثراست

تأثیر تغییرات و وابستگی‌ها در کد، ما را بر آن داشت تا تمرکز خود را برای تعریف معیار جدید بر روی این دو عامل متمرکز کنیم. در اولین قدم بر آن شدیم تا به طور تجربی هم‌بستگی معیارهای تغییر در ماژول‌ها و ایجاد خطا در ماژول‌های وابسته را بررسی کنیم. بدین منظور، نیاز است تا برای هر ماژول وابسته، ماژول‌هایی که به آنها وابسته هستند را مشخص کنیم شکل (۴). برای این منظور گراف وابستگی برای ماژول‌ها باید به دست آید.

1 Crane



شکل (۴): نمودار معیار پیشنهادی

۴-۲-۱- وابستگی بین ماژول‌ها

وابستگی‌ها در بین ماژول‌ها می‌تواند متفاوت باشد. به‌عنوان‌مثال در سطح کلاس، وابستگی عبارت است از:

- فراخوانی تابع از کلاس دیگر
- فراخوانی تابع سازندهٔ کلاس دیگر
- استفاده از متغیری از کلاس دیگر
- و مواردی ازاین‌قبیل

در واقع برای این مبحث، هر نوع ارجاعی که بین دو ماژول ایجاد وابستگی کند، مدنظر می‌باشد. نکته قابل‌ذکر این است که یک ماژول نرم‌افزاری در تمامی مطالعات پیش‌بینی خطا برابر با یک بسته و یا یک فایل هست. به‌عنوان‌مثال در کاربردهایی که بر روی پروژهٔ إکلیپس مطالعه کرده‌اند، ماژول را بسته که مجموعه‌ای از چند فایل و یا فایل که معمولاً یک کلاس را شامل می‌شود، در نظر گرفته‌اند. در این مقاله، ماژول را فایل در نظر گرفتیم. برای به دست آوردن ارجاعات بین ماژول‌ها که ایجاد وابستگی کرده‌اند، از ابزار آندِرستَند[2][۲۶] استفاده شد.

۱- پس از تعیین وابستگی‌ها، نیاز است تا برای هر ماژول تعیین کنیم آیا ماژول یا ماژول‌هایی که به آنها وابسته است، در نسخه‌ی قبل تغییراتی داشته‌اند یا نه؟ پس از آن تعداد خطا در هر ماژول را مشخص کرده و به بررسی میزان همبستگی تغییرات در ماژول و ایجاد خطا در ماژول‌های وابسته پرداخت می‌شود.

۲- پس از حصول اطمینان از همبستگی بین دو عامل ذکر شده، حال معیاری را مبنی بر این دو عامل، یعنی تغییر در ماژول و ایجاد خطا در ماژول وابسته ارائه می‌کنیم.

۴-۲-۲- مجموعه‌ی داده‌ای مورد استفاده

در ادامه معرفی و تشریح نیاز است تا به شرح پروژه‌ی نرم‌افزاری بکار برده شده در این مطالعه بپردازیم ، اکثر مطالعات در زمینه‌ی پیش‌بینی خطا بر روی یکی از دو پروژه‌ی بزرگ إکلیپس و نسخه‌های مختلف ویندوز صورت گرفته است. در مورد پروژه‌ی إکلیپس لازم به ذکر است که طی مقاله‌ای در سال ۲۰۰۷، مجموعه داده‌ای متشکل از مقادیر اندازه‌گیری شده برای ۲۰۰ معیار تعریف شده در حوزه‌ی پیش‌بینی خطا برای پروژه‌ی إکلیپس جمع‌آوری شد[۲۷]. این مجموعه‌ی جمع‌آوری شده پرومایز[3] نام گرفت. این

2 Understand

3 Promise

مجموعه مبنای کار بسیاری از تحقیقات [۶]، [۱۸]، [۲۳] شد و در [۲۸] در دسترس عموم قرار گرفت. در این بررسی نیز از این مجموعه داده‌ای استفاده شد. علاوه بر این برای استخراج وابستگی‌ها، به صورتی که در بخش قبل گفته شد، نیاز است تا کد منبع را در اختیار داشت تا بتوان وابستگی‌ها را با ابزار آندرِستَند استخراج کرد. پروژۀ إکلیپس کد منبع است و به‌صورت متن‌باز و در اختیار است؛ بنابراین انتخاب ما در این مقاله، پروژۀ إکلیپس است.

E	D	C	B	A	
To	From	References	To File	From File	1
3	4	6	org.eclipse.ant.core\src\org\eclipse\ant\internal\core\AntCorePrefe	org.eclipse.ant.core\src\org\eclipse\ant\core\AntCorePlugin.ja	2
1	2	2	org.eclipse.ant.core\src\org\eclipse\ant\internal\core\IAntCoreCons	org.eclipse.ant.core\src\org\eclipse\ant\core\AntCorePlugin.ja	3
1	2	2	org.eclipse.core.runtime\src\org\eclipse\core\runtime\CoreExcepti	org.eclipse.ant.core\src\org\eclipse\ant\core\AntCorePlugin.ja	4
2	2	2	org.eclipse.core.runtime\src\org\eclipse\core\runtime\IConfiguratio	org.eclipse.ant.core\src\org\eclipse\ant\core\AntCorePlugin.ja	5
2	2	2	org.eclipse.core.runtime\src\org\eclipse\core\runtime\IExtensionPo	org.eclipse.ant.core\src\org\eclipse\ant\core\AntCorePlugin.ja	6
2	2	2	org.eclipse.core.runtime\src\org\eclipse\core\runtime\IPluginDescr	org.eclipse.ant.core\src\org\eclipse\ant\core\AntCorePlugin.ja	7
4	4	4	org.eclipse.core.runtime\src\org\eclipse\core\runtime\Plugin.java	org.eclipse.ant.core\src\org\eclipse\ant\core\AntCorePlugin.ja	8
2	1	2	org.eclipse.ant.core\src\org\eclipse\ant\core\AntCorePlugin.java	org.eclipse.ant.core\src\org\eclipse\ant\core\AntRunner.java	9
2	2	3	org.eclipse.ant.core\src\org\eclipse\ant\internal\core\AntClassLoad	org.eclipse.ant.core\src\org\eclipse\ant\core\AntRunner.java	10
3	3	4	org.eclipse.ant.core\src\org\eclipse\ant\internal\core\AntCorePrefe	org.eclipse.ant.core\src\org\eclipse\ant\core\AntRunner.java	11
4	3	11	org.eclipse.ant.core\src\org\eclipse\ant\internal\core\IAntCoreCons	org.eclipse.ant.core\src\org\eclipse\ant\core\AntRunner.java	12
1	1	4	org.eclipse.ant.core\src\org\eclipse\ant\internal\core\Policy.java	org.eclipse.ant.core\src\org\eclipse\ant\core\AntRunner.java	13
2	2	2	org.eclipse.core.boot\src\org\eclipse\core\boot\BootLoader.java	org.eclipse.ant.core\src\org\eclipse\ant\core\AntRunner.java	14
1	2	2	org.eclipse.core.boot\src\org\eclipse\core\boot\IPlatformRunnable.	org.eclipse.ant.core\src\org\eclipse\ant\core\AntRunner.java	15
2	2	10	org.eclipse.core.runtime\src\org\eclipse\core\runtime\CoreExcepti	org.eclipse.ant.core\src\org\eclipse\ant\core\AntRunner.java	16
1	1	1	org.eclipse.core.runtime\src\org\eclipse\core\runtime\IProgressMo	org.eclipse.ant.core\src\org\eclipse\ant\core\AntRunner.java	17
1	1	4	org.eclipse.core.runtime\src\org\eclipse\core\runtime\IStatus.java	org.eclipse.ant.core\src\org\eclipse\ant\core\AntRunner.java	18
2	1	8	org.eclipse.core.runtime\src\org\eclipse\core\runtime\Status.java	org.eclipse.ant.core\src\org\eclipse\ant\core\AntRunner.java	19

شکل (۵): فایل وابستگی‌های استخراج شده برای إکلیپس ۲,۰

۴-۲-۳- روش محاسبه معیار جدید

در این بخش به شرح مرحله به مرحلۀ روند به‌دست‌آمدن معیار پیشنهادی می‌پردازیم. پردازش‌هایی بر روی مجموعه‌داده‌های موجود برای إکلیپس صورت می‌گیرد تا بتوان مقدار معیار پیشنهادی خود را برای هر ماژول تعیین کرد. در ادامه به‌تفصیل به شرح آنها پرداخته می‌شود.

۴-۲-۴- استخراج وابستگی‌ها

همان‌طور که ذکر شد، برای استخراج وابستگی‌ها از ابزار آندرستند استفاده کردیم. این ابزار قادر است وابستگی‌ها را در سه سطح بسته، فایل، کلاس و موجودیت استخراج کند. هرگونه ارجاع در بین این سطوح به‌عنوان وابستگی محاسبه می‌شود. موارد ارجاع عبارت‌اند از: فراخوانی توابع، واسط‌ها، متغیرها، ایجاد شیء جدید. وابستگی‌های ذکر شده علاوه بر قابلیت مشاهده در محیط ابزار آندرِستند، قابل‌استخراج هم هستند. به‌عنوان نمونه برای نسخۀ ۲,۰ از پروژۀ إکلیپس وابستگی‌ها را در قالب یک فایل اکسل استخراج کردیم. فرمت فایل خروجی را می‌توان در شکل (۵) دید. ماژول در این راهکار برابر با فایل درنظر گرفته شده است. نسخه‌ی ۲,۰ حاوی ۶۷۲۸ فایل است. قسمتی از فایل وابستگی‌های استخراج شده را شکل (۵) مشخص شده است.

C	B	A
References	To File	From File
6	\org.eclipse.ant.core\src\org\eclipse\ant\internal\core\AntCorePreferences.java	\org.eclipse.ant.core\src\org\eclipse\ant\core\AntCorePlugin.java
2	\org.eclipse.ant.core\src\org\eclipse\ant\internal\core\IAntCoreConstants.java	\org.eclipse.ant.core\src\org\eclipse\ant\core\AntCorePlugin.java
2	\org.eclipse.core.runtime\src\org\eclipse\core\runtime\CoreException.java	\org.eclipse.ant.core\src\org\eclipse\ant\core\AntCorePlugin.java
2	\org.eclipse.core.runtime\src\org\eclipse\core\runtime\IConfigurationElement.ja	\org.eclipse.ant.core\src\org\eclipse\ant\core\AntCorePlugin.java
2	\org.eclipse.core.runtime\src\org\eclipse\core\runtime\IExtensionPoint.java	\org.eclipse.ant.core\src\org\eclipse\ant\core\AntCorePlugin.java
2	\org.eclipse.core.runtime\src\org\eclipse\core\runtime\IPluginDescriptor.java	\org.eclipse.ant.core\src\org\eclipse\ant\core\AntCorePlugin.java
4	\org.eclipse.core.runtime\src\org\eclipse\core\runtime\Plugin.java	\org.eclipse.ant.core\src\org\eclipse\ant\core\AntCorePlugin.java
		7 \org.eclipse.ant.core\src\org\eclipse\ant\core\AntCorePlugin.j

شکل (۶): گروه بندی ماژول‌ها بر حسب ماژول‌های که به آن‌ها وابسته هستند.

در شکل (۵) ستون اول نام فایلی را مشخص می‌کند که به فایل متناظر ذکر شده در ستون دوم دارای وابستگی است. ستون سوم تعداد ارجاع‌ها از فایل ستون اول به فایل ستون دوم را نشان می‌دهد. ستون چهارم تعداد موجودیت‌هایی از فایل ستون اول را که به فایل ستون دوم وابسته است را مشخص می‌کند. بر یک موجودیت ممکن است بیش از یک ارجاع وجود داشته باشد. در واقع ستون سوم مجموع ارجاعات برای موجودیت‌ها است. در نهایت ستون آخر، تعداد موجودیت‌های فایل ستون دوم که به

فایل ستون اول وابسته هستند را مشخص می‌کند. البته از فایل خروجی تنها سه ستون اول را نیاز داریم، چون ماژول را فایل در نظر گرفتیم و نیاز است تا تعداد ارجاعات بین فایل‌ها را داشته باشیم.

C	B	A
ChangeFo	To File	From File
7	\org.eclipse.ant.core\src\org\eclipse\ant\internal\core\AntCorePreferen	\org.eclipse.ant.core\src\org\eclipse\ant\core\AntCorePlugin.
4	\org.eclipse.ant.core\src\org\eclipse\ant\internal\core\IAntCoreConstan	\org.eclipse.ant.core\src\org\eclipse\ant\core\AntCorePlugin.
6	\org.eclipse.core.runtime\src\org\eclipse\core\runtime\CoreException.j	\org.eclipse.ant.core\src\org\eclipse\ant\core\AntCorePlugin.
4	\org.eclipse.core.runtime\src\org\eclipse\core\runtime\IConfigurationEl	\org.eclipse.ant.core\src\org\eclipse\ant\core\AntCorePlugin.
4	\org.eclipse.core.runtime\src\org\eclipse\core\runtime\IExtensionPoint_	\org.eclipse.ant.core\src\org\eclipse\ant\core\AntCorePlugin.
6	\org.eclipse.core.runtime\src\org\eclipse\core\runtime\IPluginDescripto	\org.eclipse.ant.core\src\org\eclipse\ant\core\AntCorePlugin.
25	\org.eclipse.core.runtime\src\org\eclipse\core\runtime\Plugin.java	\org.eclipse.ant.core\src\org\eclipse\ant\core\AntCorePlugin.
8		\org.eclipse.ant.core\src\org\eclipse\ant\core\AntCoreP

شکل (۷): افزودن معیار تغییرات به اعضای گروه و محاسبه‌ی میانگین این تغییرات برای هر گروه

۴-۲-۵- گروه‌بندی ماژول‌ها

برای هر ماژول (فایل)، ماژول‌ها وابسته به آن را به دست می‌آوریم. برای این کار از ویژگی گروه‌بندی در نرم‌افزار اکسل بهره بردیم. برای وضوح تصویر تنها اطلاعات مربوط به یک فایل را به‌عنوان نمونه در شکل (۶) آورده‌ایم. ستون اول نام فایل وابسته، ستون دوم نام فایلی که فایل ذکر شده در ستون اول به آن وابسته است و ستون سوم تعداد ارجاعات را از فایل ستون اول به فایل ستون دوم مشخص می‌کند. برای فایل *AntCorePlugin.java* مجموعه فایل‌هایی که به آنها وابستگی دارد را مشاهده می‌کنیم که تعداد آنها ۷ فایل می‌باشد.

۴-۲-۶- افزودن معیار تغییرات به مجموعه‌ی داده‌ای

در مجموعه‌ی داده‌های پرومایز، معیار تغییرات موجود نمی‌باشد. در راهکار پیشنهادی، به تعداد تغییرات برای هر فایل در طی فرآیند توسعه نیاز است. تعداد تغییرات یکی از معیارهایی بود که جمع‌آوری شده بود. پروژه‌ای که در [۲۰] معیارهای توالی تغییرات برای آن محاسبه شد، إکلیپس بود. معیارهای جمع‌آوری شده در قالب یک مجموعه‌ی داده‌ها در [۲۹] در دسترس است. در این مجموعه‌ی داده‌ها تعداد تغییرات برای هر فایل ذکر شده است. ما نیاز داشتیم تا این معیار را در مجموعه داده‌های گروه‌بندی شده که در بخش قبل آن را توضیح دادیم برای هر فایل وارد کنیم. به دلیل حجم زیاد داده‌ها، این کار توسط ماژولی به نام ماکرو[1] و به زبان ویژوال بیسیک[2] انجام شد. خروجی این ماژول به صورت مجموعه‌ای از داده‌ها مورد استفاده قرار گرفت. علاوه بر این، میانگین تغییرات برای هر گروه داده‌ها (در واقع میانگین تغییرات فایل‌های موجود در گروه مربوط به هر فایل) برای آن محاسبه شده است. البته لازم به ذکر است برای معیار پیشنهادی، سه مقدار میانگین، مقدار بیشینه و مجموع تغییرات برای هر گروه محاسبه شد. در شکل (۷) مشخص شده است. در واقع این کار به این دلیل بود که پس از به دست آوردن مقادیر تغییرات برای هر گروه، نیاز است تا برآوردی از آنها به ماژول شاخص آن گروه به عنوان مقداری برای معیار پیشنهادی تخصیص داده شود. هر سه مقدار میانگین، مقدار بیشینه و مجموع محاسبه و مدل براساس آنها ساخته شد تا بتوان میزان قوت آنها را نسبت به یکدیگر و نیز نسبت به معیارهای دیگر تعیین کرد. در ادامه توضیحات را براساس میزان میانگین ارائه می‌کنیم.

۴-۲-۷- محاسبه‌ی معیار جدید برای هر فایل

میانگین (مجموع، مقدار بیشینه) تغییرات به‌دست‌آمده در مرحلهٔ قبل، معیاری است که برای هر فایل محاسبه شده است. این معیار جدیدی است که قرار است به مجموعهٔ داده‌ای پرومایز اضافه شود و مدل بر اساس آن ساخته شود. نام معیار پیشنهادی *ChForDependees* می‌باشد. باز با استفاده از ماکرو در محیط برنامه‌نویسی اکسل، ستونی ایجاد کردیم که مقدار میانگین تولید شده در هر گروه را به نام آن گروه (که یک فایل خاص است) نسبت دادیم. پس از آن ستون ایجاد شده را به مجموعه‌دادهٔ پرومایز اضافه کردیم و مجموعهٔ دادهٔ جدید به همراه معیار *ChForDependees* به دست آمد.

۴-۲-۸- ساخت مدل

روش‌های مختلفی در مقالات مختلف برای ایجاد مدل پیش‌بینی خطا وجود دارد. در [۵] این روش‌ها به دسته‌های زیر تقسیم شده است:

۱. آماری
۲. یادگیری ماشین و آمار

1 Macro
2 Visual Basic

۳. رأی خبره[1] و آمار

الگوریتم‌های یادگیری ماشین به طور عمده در پیش‌بینی خطای نرم‌افزار به کار می‌روند و از جمله الگوریتم‌های قدرتمند در این زمینه عبارت‌اند از: نِیو بِیز[2]، جنگل‌های تصادفی. برخی محققان از روش‌های آماری همچون رگرسیون لجستیک[3] دوتایی تک‌متغیره و چندمتغیره برای پیش‌بینی خطاها استفاده می‌کنند. مدل‌های آماری راه‌حل‌های جعبه سیاه هستند، چون ارتباط بین ورودی‌ها و متغیرهای پاسخ به‌راحتی دیده نمی‌شود.

۴-۲-۸- رده‌بندها

در مسئله‌ی رده‌بندی، با مجموعه‌ای از داده‌ها روبرو هستیم که داده‌های برازش[4] نامیده می‌شوند. داده‌های برازش شامل یک سری رکورد هستند. این رکوردها دارای ویژگی‌هایی هستند. یکی از این ویژگی‌ها به‌عنوان متغیر وابسته در نظر گرفته می‌شود. این متغیر وابسته یک ویژگی دارد و آن این است که رده‌ای که هر رکورد به آن تعلق دارد را مشخص می‌کند. بقیهٔ ویژگی‌ها متغیر مستقل نامیده می‌شوند. هدف روش‌های رده‌بند[5] این است که مدلی را بسازند که متغیر وابسته را براساس مقادیر متغیرهای مستقل پیش‌بینی کند. پس از آن که این مدل ساخته شد، می‌توان آن را برای تعیین دسته‌ی مربوط به رکوردهای رده‌بندی نشده به کار برد[۴۰].

هدف ما در این مقاله پیش‌بینی خطادار بودن و یا عاری از خطا بودن برای ماژول‌های نرم‌افزاری است. ماژول‌هایی که به‌عنوان خطادار معرفی می‌شوند، به‌عنوان ماژول‌های مستعد خطا در مرحلهٔ آزمون مورد بررسی قرار می‌گیرند؛ بنابراین مسئله‌ی ما یک مسئله‌ی رده‌بندی است. متغیر وابسته در اینجا خطادار بودن و یا عاری از خطا بودن است. در مجموعهٔ داده‌ای پرومایز، معیار *post* تعداد خطاهای رخداده پس از انتشار نسخهٔ نرم‌افزاری را مشخص می‌کند. برای به‌کاربردن مسئله‌ی رده‌بندی، لازم است این معیار به دسته‌های خطادار و عاری از خطا تبدیل کنیم. بدین منظور همان‌طور که در [۲۳] نیز مطرح شده است، مقادیر غیر صفر را که نشان دهنده‌ی ماژول‌های خطادار می‌باشند، به کلاس *yes* و مقادیر کلاس صفر را که نشان دهنده‌ی ماژول‌های عاری از خطا می‌باشند به کلاس *no* نگاشت می‌کنیم. از جمله روش‌های رده‌بند می‌توان به رگرسیون لجستیک و شبکه‌ی بِیزین اشاره کرد[۴۷].

۴-۲-۱۰-شبکه‌ی بیزین

در این قسمت بنا به استفاده‌ای که از شبکهٔ بیزین در ساخت مدل پیش‌بینی خطا در این پژوهش صورت‌گرفته است، به معرفی آن می‌پردازیم.

مجموعه متغیرهای U را به‌صورت

$$U = \{x_1, \dots, x_n\}, n \geq 1$$

در نظر بگیرید. شبکه‌ی بیزین بر روی مجموعه متغیرهای U از دو قسمت تشکیل می‌شود: ساختار شبکه و جدول احتمالات. ساختار شبکه، B_S، یک گراف جهت‌دار بدون دور (DAG) براساس مجموعه‌ی U است. جدول احتمالات، B_P، به صورت $B_P = \{p(u|pa(u)) \,|u \in U\}$ تعریف می‌شود که در آن $pa(u)$ مجموعه‌ی پدران مربوط به متغیر u در گراف ایجاد شده می‌باشد. شبکه‌ی بیزین توزیع احتمالی را برای مجموعه‌ی U به این صورت تشکیل می‌دهد:

$P(U) = \Pi_{u \in U}\, p(u|pa(u))$. همان‌طور که ذکر شد، در روش‌های رده‌بند یک متغیر وابسته و تعدادی متغیر مستقل داریم. روش‌های رده‌بند از داده‌هایی برای یادگیری استفاده می‌کنند که هر رکورد در این داده‌ها دارای مقداری برای متغیر پیشگو و متغیرهای مستقل است. مقدار متغیر وابسته کلاس مربوط به هر داده را مشخص می‌کند. در کاربرد شبکه‌ی بیزین به عنوان رده‌بند، فرآیند یادگیری عبارت خواهد بود از یافتن شبکه‌ی بیزین مناسب برای داده‌های ورودی.

در این مقاله پس از ساخت داده‌ها که حاوی معیار اضافه شده می‌باشد، از روش بیزین و نِیْوبِیز برای ساخت مدل استفاده شده است. ساخت مدل توسط ابزار وِکا[6] صورت‌گرفته است. از جمله نکاتی که در اینجا باید ذکر کنیم، این است که روش اعتبارسنجی یک مدل به این صورت است که صحت یک مدل از طریق تخمین پارامترهای مدل توسط مجموعه‌داده‌ای آموزش، پیش‌بینی متغیر وابستهٔ هر نمونه در مجموعهٔ داده‌ای آزمون و سپس ارزیابی نتایج بررسی می‌شود. در این مقاله از روش اعتبارسنجی متقاطع برای ساخت دادهٔ آزمون و دادهٔ آموزش بهره برده‌ایم.

برای آشنایی با نحوهٔ عملکرد اعتبارسنجی متقاطع، فرض کنید n رکورد در دسترس باشد. یک رکورد را به‌عنوان مجموعه‌دادهٔ آزمون فعلی و بقیهٔ رکوردها را به‌عنوان مجموعه داده‌ای آموزش در نظر بگیرید. مدل را بر اساس مجموعه آموزش فعلی می‌سازیم و آن را توسط مجموعه‌داده آزمون مورد ارزیابی قرار می‌دهیم. این عمل را برای هر رکورد تکرار می‌کنیم و به‌این‌ترتیب n مدل ایجاد می‌شود. نتیجهٔ نهایی بر اساس نتایج مدل‌های ایجاد شده به دست می‌آید

۴-۲-۱۱- معیار ارزیابی رده‌بند

در ابتدا نیاز است تا با چهار مفهوم اولیهٔ مثبت صادق (TP)، مثبت کاذب (FP)، منفی صادق (TN) و منفی کاذب (FN)

1 Expert opinion
2 Naive Bayes
3 Logistic Regression
4 Fit data
5 Classifier
6 Weka

آشنا شویم. این مفاهیم در مسائل مربوط به رده‌بندی برای تحلیل نتایج به کار می‌روند.

۱. منفی صادق: ماژول‌ها عاری از خطایی که به‌درستی رده‌بندی شده‌اند.
۲. منفی کاذب: ماژول‌ها خطاداری که به‌عنوان ماژول‌ها عاری از خطا برچسب خورده‌اند.
۳. مثبت کاذب: ماژول‌ها عاری از خطایی که به‌اشتباه به‌عنوان ماژول‌ها خطادار برچسب خورده‌اند.
۴. مثبت صادق: ماژول‌های که به‌درستی برچسب خطادار خورده‌اند.

معیارهای ارزیابی روش‌های رده‌بندی بر اساس این چهار ویژگی تعریف می‌شوند. احتمال تشخیص (PD) و احتمال هشدار اشتباه (PF) برای ارزیابی نتایج حاصل از یک روش خاص به کار می‌روند. احتمال تشخیص به‌صورت احتمال رده‌بندی صحیحِ ماژول‌های خطادار تعریف می‌شود.

رابطه (۱) $$PD = \frac{TP}{TP+FN}$$

احتمال هشدار اشتباه نیز به‌صورت نسبت مثبت‌های کاذب به کل ماژول‌های عاری از خطا تعریف می‌شود.

رابطه (۲) $$PF = \frac{FP}{FP+TN}$$

احتمال هشدار اشتباه باید مقدار کمی داشته باشد که نمایانگر رده‌بندی صحیح ماژول‌ها به ماژول‌ها خطادار و عاری از خطا است. در مقابل احتمال تشخیص باید دارای مقدار زیادی باشد تا شانس رده‌بندی صحیح ماژول‌ها خطادار افزایش یابد که می‌تواند کیفیت نرم‌افزار را افزایش دهد.

روش‌های اندازه‌گیری کارایی قدیمی همچون *F-measure*، دقت کلی رده‌بندی[1] و نرخ رده‌بندی اشتباه[2] برای کاربردهایی که داده‌های نامتوازن را رده‌بندی می‌کنند مناسب نمی‌باشند. در حوزه‌هایی همچون پیش‌بینی کیفیت نرم‌افزار، تعداد ماژول‌های دارای خطا خیلی کمتر از تعداد ماژول‌های عاری از خطا هستند. در [۳۱] نشان داده شده است که معیار فضای زیر منحنی راک انحراف کمتری دارد و نسبت به معیارهای دیگر برای پیش‌بینی خطای نرم‌افزار مطمئن‌تر است.

فضای زیر منحنی راک تنها دارای یک مقدار است که مقدار آن از ۰ تا ۱ متغیر است. منحنی راک برای مشخص کردن مصالحه بین احتمال تشخیص و احتمال اعلام اشتباه به کار می‌رود. رده‌بندی که فضای بزرگی را زیر منحنی ایجاد کند نسبت به رده‌بند با فضای زیر منحنی کمتر ارجحیت دارد.

۴-۳- نتایج و ارزیابی

در این بخش، نشان‌داده‌شده که معیار پیشنهادی تا چه میزانی در تعیین خطرخیزی کد تأثیرگذار است. برای این منظور، در ابتدا میزان تأثیرهای هر یک از معیارهای شناخته شده در مقایسه با معیار پیشنهادی را مشخص می‌نماییم. در گام دوم، میزان دقت مدل یادگیری را در مقایسه با دو مدل موجود؛ شبکه بیزین و نِیْوبِیز؛ مشخص می‌نماییم. برای این منظور، در ابتدا میزان همبستگی مجموعه معیار پیشنهادی با تعداد خطاها در یک ماژول را ذکر کرده و سپس به مقایسه بین این مقدار با مقادیر مشابه برای معیارهای دیگر می‌پردازیم. پس از آن باتوجه‌به معیار ارزیابی رده‌بند، فضای زیر منحنی راک، به بررسی میزان قوت مدل پیش‌بینی طراحی شده می‌پردازیم. در نهایت با استفاده از روش انتخاب ویژگی، معیار خود را با دیگر معیارهای موجود از نظر قدرت پیش‌بینی مقایسه می‌کنیم. همچنین مقایسه‌ای بین کار خود و دیگر کارها ارائه می‌کنیم.

۴-۳-۱- میزان هم‌بستگی

یکی از روش‌های ارزیابی معیارهای نرم‌افزاری تعیین میزان هم‌بستگی آنها با تعداد خطاهای شناخته شده موجود در یک ماژول است. این روش در [۲۵][۲۷] استفاده شده است. در جدول(۱) به طور نمونه مقدار هم‌بستگی را برای چند نمونه از معیارهای موجود در مدل پیشنهادی این مقاله که پرومایز است مشخص نموده‌ایم. تعداد ۲۰۰ معیار در پرومایز وجود دارد که در اینجا ما تنها برای نشان دادن بیش‌ترین مقدار هم‌بستگی و کم‌ترین مقدار هم‌بستگی و نیز تعیین جایگاه معیار پیشنهادی نسبت به آنها دسته‌ای از معیارها را انتخاب و در جدول(۱) آورده‌ایم.

جدول (۱): مقادیر هم‌بستگی برای معیارهای پیشنهادی و نمونه‌ای از معیارهای موجود در مجموعه داده‌ای پرومایز

نام معیار	میزان هم‌بستگی با تعداد خطا
ACD	۰.۱۱۹۲۵۲
FOUT_avg	۰.۱۸۲۸۴۸
FOUT_max	۰.۲۸۴۳۴۱
FOUT_sum	۰.۴۱۹۱۷
MLOC_avg	۰.۱۹۹۵۳۳
NORM_VariableDeclarationFragment	۰.۰۱
NORM_VariableDeclarationStatement	۰.۱۸۸۳۴۵
NORM_Modifier	-۰.۱۲۲۱۵
AVGCHFORDEPENDEES (معیار پیشنهادی)	۰.۱۳۰۸۷۱
SUMCHFORDEPENDEES (معیار پیشنهادی)	۰.۳۴۱۰۱۴
MAXCHFORDEPENDEES (معیار پیشنهادی)	۰.۱۸۷۵۱۸

در جدول (۱) سه معیار مشخص شده است که نشانگر مجموع، میانگین و مقدار حداکثر تغییرات در ماژول‌هایی که یک ماژول به آنها وابسته است، می‌باشد. همان‌طور که در این جدول

1 Overall classification accuracy

2 Misclassification rate

مشاهده می‌کنیم به ترتیب معیار مربوط به مجموع مقادیر بیش‌ترین هم‌بستگی را با خطادار شدن ماژول دارد. پس از آن، معیار مربوط به مقدار حداکثر هم‌بستگی بیشتری با خطادار شدن ماژول دارد و در آخر هم معیار مربوط به میانگین مقادیر قرار گرفته است. جدول نشان می‌دهد برای ماژول مورد نظر، هرچه مجموع تغییرات در ماژول‌هایی که آن ماژول به آنها وابسته است بیشتر باشد، احتمال خطادار شدن آن ماژول به میزان زیادی افزایش می‌یابد.

۴-۳-۲- ارزیابی مدل

همان‌طور که در بخش قبل ذکر شد، برای ساخت مدل دو روش شبکهٔ بیزین و نِیوبِیز را توسط ابزار وِکا به کار بردیم. مدل پیش‌بینی خطا بر اساس این دو روش و نیز با مجموعه‌های داده‌ای زیر ساخته شد:

- ✓ مجموعهٔ داده‌ای پرومایز
- ✓ مجموعهٔ داده‌ای متشکل از معیارهای پرومایز به همراه معیار میانگین تغییرات برای موردِ وابستگی‌ها.
- ✓ مجموعهٔ داده‌ای متشکل از معیارهای پرومایز به همراه معیار مجموع تغییرات برای موردِ وابستگی‌ها.
- ✓ مجموعهٔ داده‌ای متشکل از معیارهای پرومایز به همراه معیار مقدار حداکثری تغییرات برای موردِ وابستگی‌ها.
- ✓ مجموعهٔ داده‌ای متشکل از معیارهای پرومایز به همراه هر سه معیار ارائه شده.

در مجموع ده مدل ساخته شده است که در جدول‌های (۲) و (۳) نتایج مربوط به ارزیابی آنها آمده است. همانگونه که مشاهده می شود، معیارهای متفاوت در مقایسه با یکدیگر تفاوت قابل ملاحظه‌ای ندارند، اما معیار پیشنهادی در این مقاله در مقایسه با سایر معیارها به نحو بهتری در شناسایی خطاخیزی تاثیر گذار بوده است.

جدول (۲): مدل‌های پیش‌بینی خطای ایجاد شده بر اساس روش شبکه‌ی بِیزین

نام مجموعه‌داده	TP	FP	Precision	Recall	F-Meature	ROC Area
پرومایز	۰,۷۲	۰,۲۶۶	۰,۸۴۹	۰,۷۲	۰,۷۵۹	۰,۷۹۶
پرومایز+ معیار *AVGCHForDependees*	۰,۷۲۱	۰,۲۶۴	۰,۸۵	۰,۷۲۱	۰,۷۶	۰,۷۹۶
پرومایز+ معیار *SUMCHForDependees*	۰,۷۲۲	۰,۲۶۴	۰,۸۵	۰,۷۲۲	۰,۷۶۱	۰,۷۹۶
پرومایز+ معیار *MAXCHForDependees*	۰,۷۲۳	۰,۲۵۷	۰,۸۵۲	۰,۷۲۳	۰,۷۶۱	۰,۷۹۶
پرومایز+ معیارهای *AVGCHForDependees, MAXCHForDependees, SUMCHForDependees*	۰,۷۲۴	۰,۲۵۶	۰,۸۵۲	۰,۷۲۴	۰,۷۶۲	۰,۷۹۸

جدول (۳): مدل‌های پیش‌بینی خطای ایجاد شده بر اساس روش نِیوبِیز

نام مجموعه‌داده	TP	FP	Precision	Recall	F-Meature	ROC Area
پرومایز	۰,۸۴۳	۰,۵۲۵	۰,۸۳۵	۰,۸۴۳	۰,۸۳۹	۰,۷۱۵
پرومایز+ معیار *AVGCHForDependees*	۰,۸۴۲	۰,۵۲۵	۰,۸۳۵	۰,۸۴۲	۰,۸۳۸	۰,۷۱۵
پرومایز+ معیار *SUMCHForDependees*	۰,۸۴۱	۰,۵۲۵	۰,۸۳۴	۰,۸۴۱	۰,۸۳۷	۰,۷۱۵
پرومایز+ معیار *MAXCHForDependees*	۰,۸۴۲	۰,۵۲۵	۰,۸۳۵	۰,۸۴۲	۰,۸۳۸	۰,۷۱۵
پرومایز+ معیارهای *AVGCHForDependees, MAXCHForDependees, SUMCHForDependees*	۰,۸۴۱	۰,۵۲۴	۰,۸۳۴	۰,۸۴۱	۰,۸۳۷	**۰,۷۱۶**

۴-۳-۳- انتخاب ویژگی

برای ارزیابی معیارهای پیشنهادی با معیارهای دیگر، از روش انتخاب ویژگی نیز بهره بردیم. در این مقاله از ابزار وِکا برای پیاده‌سازی روش انتخاب ویژگی استفاده شده است. از بین روش‌های انتخاب ویژگی، از روش رتبه‌بندی ویژگی استفاده کردیم. هر یک از سه معیار پیشنهادی را به طور جداگانه به مجموعه داده‌ای پرومایز اضافه کرده و پس از آن روش انتخاب ویژگی رتبه‌بندی را بر روی مجموعه داده‌ای حاصل اعمال کردیم. نتایج به‌دست‌آمده در جدول (۴) به‌نمایش‌درآمده است.

جدول (۴): پایگاه معیارهای پیشنهادی در بین معیارهای موجود در پرومایز از نظر قدرت پیش‌بینی

نام معیار	رتبهٔ معیار در بین ۲۰۱ معیار
AVGCHForDependees	۵۸
SUMCHForDependees	۱۸
MAXCHForDependees	۳۰

نتایجی که از اعمال روش رتبه‌بندی ویژگی به دست آمد، با نتایج حاصل از هم‌بستگی کاملاً هم‌خوانی دارد. در بین سه معیار،

معیار مبتنی بر مجموع تغییرات ماژول‌ها موردِ وابستگی قدرت پیش‌بینی بالاتری داشت و از بین ۲۰۱ معیار به عنوان معیار هجدهم قرار گرفت.

حال که جایگاه سه معیار پیشنهادی را در بین معیارهای دیگر پرومایز مشاهده کردیم، مدل‌های پیش‌بینی را بر اساس مجموعهٔ داده‌ای شامل معیارهای رتبه‌بندی شده تا معیار پیشنهادی می‌سازیم و آن را با مدل‌های ساخته شده در قسمت قبل مقایسه می‌کنیم. نتایج ساخت مدل‌ها در جدول (۵) آمده‌اند.

جدول (۵): مدل‌های پیش‌بینی خطای ایجاد شده بر اساس روش شبکه‌ی بِیزین (معیارهای رتبه‌بندی شده)

نام مجموعه‌داده	تعداد معیارها	TP	FP	Precision	Recall	F-Meature	ROC Area
پرومایز+ معیار AVGCHForDependees	۶۰	۰,۷۲۵	۰,۲۷۱	۰,۸۴۹	۰,۷۲۵	۰,۷۶۲	۰,۷۹۹
پرومایز+ معیار SUMCHForDependees	۲۲	۰,۷۴۸	۰,۳۱۷	۰,۸۴۳	۰,۷۴۸	۰,۷۷۹	**۰,۸۰۸**
پرومایز+ معیار MAXCHForDependees	۳۲	۰,۷۴	۰,۳۰۲	۰,۸۴۵	۰,۷۴	۰,۷۷۴	۰,۸۰۵
پرومایز+ معیارهای AVGCHForDependees, MAXCHForDependees, SUMCHForDependees	۶۲	۰,۷۳	۰,۲۶۵	۰,۸۵۱	۰,۷۳	۰,۷۶۷	۰,۸۰۳

جدول (۶): مدل‌های پیش‌بینی خطای ایجاد شده بر اساس روش شبکه‌ی نِیوبِیز (معیارهای رتبه‌بندی شده)

نام مجموعه‌داده	تعداد معیارها	TP	FP	Precision	Recall	F-Meature	ROC Area
پرومایز+ معیار AVGCHForDependees	۶۱	۰,۸۵۱	۰,۵۴۵	۰,۸۳۷	۰,۸۵۱	۰,۸۴۳	۰,۷۴۲
پرومایز+ معیار SUMCHForDependees	۲۲	۰,۸۵	۰,۵۱۹	۰,۸۴۱	۰,۸۵	۰,۸۴۵	**۰,۸۰۷**
پرومایز+ معیار MAXCHForDependees	۳۲	۰,۸۵۱	۰,۵۳۲	۰,۸۳۹	۰,۸۵۱	۰,۸۴۴	۰,۸۰۱
پرومایز+ معیارهای AVGCHForDependees, MAXCHForDependees, SUMCHForDependees	۶۲	۰,۸۵۱	۰,۵۴	۰,۸۳۸	۰,۸۵۱	۰,۸۴۳	۰,۷۴۲

باز هم مشاهده می‌کنیم بهترین نتیجه در این حالت برای مجموعهٔ داده‌ای حاوی معیار مبتنی بر مجموع تغییرات برای موردهای وابستگی است. علاوه بر این بهترین نتیجه برای کمترین تعداد معیارها به دست آمد. در نتیجه میزان قدرت پیش‌بینی معیار و نیز تعداد معیارهای مورداستفاده دو عامل اساسی در مدل‌های پیش‌بینی خطا هستند.

اگر بخواهیم مقایسه‌ای بین الگوریتم یادگیری مورداستفاده در ساخت مدل بیندازیم، می‌یابیم در حالتی که مدل را با تمامی معیارها ساختیم و نیز در حالتی که دسته‌ای از معیارها را انتخاب و مدل را ایجاد کردیم، مقادیر فضای زیر منحنی راک برای شبکهٔ بیزین بهتر از نِیوبیز بوده است.

۵- نتیجه‌گیری

یک عامل خطا در برنامه‌ها تغییرات در کد است، ممکن است یک تغییر موجب اصلاحات در بخشی از کد بگردد و در بخش وابسته به آن اشکال ایجاد شود؛ بنابراین باتوجه‌به میزان وابستگی ماژول‌ها به یکدیگر می‌توان تغییرات را عاملی جهت انتشار خطا در کد در نظر گرفت. این واقعیت نوآوری عمده مقاله بوده است. وابستگی در خلاف اصل آزمون‌پذیری کد می‌باشند.

هرچه ماژول‌هایی که از یکدیگر مستقل‌تر باشند کار آزمون ساده‌تر خواهد بود چرا که امکان انتشار خطا کمترمی شود.

در این مقاله به‌منظور پیش‌بینی ماژول‌ها مستعد خطا در یک نرم‌افزار، مجموعه معیار جدیدی اضافه نمودیم. دلیل این که ما به سمت ارائه‌ی این معیار رفتیم، این بود که تعداد اندکی از معیارهای نرم‌افزاری موجود بر اساس ویژگی‌هایی از کد که در خطادار شدن مؤثر باشند، ساخته شده بودند. این خلأ ما را بر آن داشت تا بر روی ویژگی‌هایی از ماژول‌ها که در بروز خطا تأثیر دارند، تمرکز کنیم. بدین منظور در ابتدا به بررسی معیارهای نرم‌افزاری در بخش‌های قبل انجام‌گرفته پرداختیم و میزان موفقیت آنها در ساخت مدل پیش‌بینی خطا را عنوان کردیم. تغییرات در کد یکی از معیارهای موفق در زمینهٔ پیش‌بینی خطا است. اما این تغییرات به‌واسطهٔ وابستگی‌ها احتمال خطادار بودن را در ماژول‌های وابسته بالا می‌برند؛ بنابراین ما معیاری ارائه دادیم تا با آن به اندازه‌گیری تأثیر تغییرات و وابستگی‌ها بر روی خطادار شدن کد بپردازیم. پس از آن که وابستگی‌ها و تغییرات را مشخص کردیم، نیاز است تا مقداری را به‌عنوان اندازهٔ تغییرات موردهای وابستگیِ یک ماژول به آن ماژول نسبت دهیم. این مقدار را برای مقادیر میانگین، مجموع و مقدار حداکثری به دست آوردیم. این سه معیار را در مدل پیش‌بینی خطا وارد کردیم. نتایج به‌دست‌آمده به شرح زیر است:

- ✓ برای یک ماژول نرم‌افزاری، هرچه مجموع تغییرات در ماژول-های که به آنها وابسته است، بیشتر شود، آن ماژول برای بروز خطا مستعدتر می‌شود.
- ✓ در ساخت یک مدل پیش‌بینی خطا سه عامل بسیار مهم هستند: انتخاب معیارهای مناسب و قوی، تعداد معیارهای انتخابی و روش ساخت مدل.
- ✓ با افزودن هر سه معیار به مجموعهٔ دادهٔ پرومایز (که در این مقاله بر روی آن کار کردیم) به مدلی بادقت بالاتر رسیدیم؛ بنابراین بین تعداد معیارها و انواع آنها باید مصالحه‌ای برقرار گردد.

۷. مراجع

[1] G. Krishnan Rajbahadur, S. Wang, G. A. Oliva, Y.Kamei, and A.E.Hassan "The impact of feature importance methods on the interpretation of defect classifiers",IEEE Transactions on Software Engineering TSE 2021.3056941

[2] S. Kumar Pandey, R. Bhushan Mishra, A. Kumar Tripathi "Machine learning based methods for software fault prediction: A survey" 0957-4174, 2021 Elsevier

[3] J.P.D.Wijesekara, P.G.T.P. Gunawardhana" A Review on Mining Software Engineering Data for Software Defect Prediction" ITRU RESEARCH SYMPOSIUM 2020, FACULTY OF INFORMATION TECHNOLOGY, UNIVERSITY OF MORATUWA,

[4] R. Moussaa, D. Azara" A PSO-GA Approach Targeting Fault-Prone Software Modules" Preprint submitted to Journal of Systems and Software June 19, 2017

[5] E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," Journal of Systems and Software, vol. 83, no. 1, pp. 2 – 17, 2010.

[6] T. Gilb and S. Finzi, Principles of software engineering management, vol. 4. Addison-Wesley Wokingham, 1988.

[7] T. DeMarco, Controlling software projects: management, measurement & estimation. Yourdon Press, 1982.

[8] N. E. Fenton and S. L. Pfleeger, Software metrics: a rigorous and practical approach. PWS Publishing Co., 1998.

[9] C. Catal and B. Diri, "A systematic review of software fault prediction studies," Expert Systems with Applications, vol. 36, no. 4, pp. 7346–7354, 2009.

[10] K. Pan, S. Kim, and E. J. Whitehead Jr, "Bug classification using program slicing metrics," in Source Code Analysis and Manipulation, 2006. SCAM'06. Sixth IEEE International Workshop on, 2006, pp. 31–42.

[11] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy, "Predicting fault incidence using software change history," Software Engineering, IEEE Transactions on, vol. 26, no. 7, pp. 653–661, 2000.

[12] 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.

[13] N. Nagappan, B. Murphy, and V. Basili, "The influence of organizational structure on software quality," in Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on, 2008, pp. 521–530.

[14] A. Mockus, "Organizational volatility and its effects on software defects," in Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering, 2010, pp. 117–126.

[15] M. Pinzger, N. Nagappan, and B. Murphy, "Can developer-module networks predict failures?," in Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, 2008, pp. 2–12.

[16] [A. Meneely, L. Williams, W. Snipes, and J. Osborne, "Predicting failures with developer networks and social network analysis," in Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, 2008, pp. 13–23.

[17] 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 Symposium on the Foundations of Software Engineering and the 13rd European Software Engineering Conference, 2011, pp. 4–14.

[18] F. Rahman and P. Devanbu, "Ownership, experience and defects: a fine-grained study of authorship," in Proceedings of the 33rd International Conference on Software Engineering, 2011, pp. 491–500.

[19] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy, "Does distributed development affect software quality?: an empirical case study of windows vista," Communications of the ACM, vol. 52, no. 8, pp. 85–93, 2009.

[20] J. Kumar Chhabra and V. Gupta, "A survey of dynamic software metrics," Journal of Computer Science and Technology, vol. 25, no. 5, pp. 1016–1029, 2010.

[21] J. Czerwonka, R. Das, N. Nagappan, A. Tarvo, and A. Teterev, "CRANE: Failure Prediction, Change Analysis and Test Prioritization in Practice -- Experiences from Windows," in Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on, 2011, pp. 357–366.

[22] T. Zimmerman, N. Nagappan, K. Herzig, R. Premraj, and L. Williams, "An Empirical Study on the Relation between Dependency Neighborhoods and Failures," in Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on, 2011, pp. 347–356.

[23] 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, 2005, pp. 284– 292.

[24] N. Nagappan, A. Zeller, T. Zimmermann, K. Herzig, and B. Murphy, "Change bursts as defect predictors," Changes, vol. 2, no. 4, p. 3, 2010.

[25] N. Nagappan and T. Ball, "Using Software Dependencies and Churn Metrics to Predict Field Failures: An Empirical Case Study," in Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on, 2007, pp. 364–373.

[26] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on, 2008, pp. 531–540.

[27] H. Wang, T. M. Khoshgoftaar, and N. Seliya, "How Many Software Metrics Should be Selected for Defect Prediction?," in Twenty-Fourth International FLAIRS Conference, 2011.

[28] K. Gao, T. M. Khoshgoftaar, and H. Wang, "An empirical investigation of filter attribute selection techniques for software quality classification," in Information Reuse & Integration, 2009. IRI'09. IEEE International Conference on, 2009, pp. 272–277.

[29] H. Hata, O. Mizuno, and T. Kikuno, "Bug prediction based on fine-grained module histories," in Software Engineering (ICSE), 2012 34th International Conference on, 2012, pp. 200–210.

[30] "Understand Your Code." [Online]. Available: http://www.scitools.com/index.php. [Accessed: 02-Feb-2013].

[31] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on, 2007, pp. 9–9.

[32] "Eclipse Bug Data! - Software Engineering Chair (Prof. Zeller) - Saarland University." [Online]. Available: http://www.st.cs.uni-saarland.de/softevo/bug-data/eclipse/. [Accessed: 02-Feb-2013].

[33] "Eclipse Burst Data! - Software Engineering Chair (Prof. Zeller) - Saarland University." [Online]. Available: http://www.st.cs.uni-saarland.de/softevo/burst-data/eclipse/. [Accessed: 02-Feb-2013].

[34] T. M. Khoshgoftaar and N. Seliya, "Software quality classification modeling using the SPRINT decision tree algorithm," in Tools with Artificial Intelligence, 2002.(ICTAI 2002). Proceedings. 14th IEEE International Conference on, 2002, pp. 365–374.

[35] Y. Jiang, J. Lin, B. Cukic, and T. Menzies, "Variance analysis in software fault prediction models," in Software Reliability Engineering, 2009. ISSRE'09. 20th International Symposium on, 2009, pp. 99–108.

[36] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," Software Engineering, IEEE Transactions on, vol. 20, no. 6, pp. 476–493, 1994.

[37] F. B. Abreu and R. Carapuça, "Object-oriented software engineering: Measuring and controlling the development process," in proceedings of the 4th International Conference on Software Quality, 1994.

[38] M. Lorenz and J. Kidd, Object-oriented software metrics: a practical guide. Prentice-Hall, Inc., 1994.

[39] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," Software Engineering, IEEE Transactions on, vol. 28, no. 1, pp. 4–17, 2002.

[40] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Predicting the location and number of faults in large software systems," Software Engineering, IEEE Transactions on, vol. 31, no. 4, pp. 340–355, 2005.

[41] S. Bibi, G. Tsoumakas, I. Stamelos, and I. Vlahavas, "Software defect prediction using regression via classification," in IEEE International Conference on, 2006, pp. 330–336.

[42] T. M. Khoshgoftaar and N. Seliya, "Fault Prediction Modeling for Software Quality Estimation: Comparing Commonly Used Techniques," Empirical Software Engineering, vol. 8, no. 3, pp. 255–283, 2003.

[43] B. Selic, G. Gullekson, and P. Ward, "Real-time object oriented modeling and design," 1994.

[44] E. Arisholm, L. C. Briand, and A. Foyen, "Dynamic coupling measurement for object-oriented software," Software Engineering, IEEE Transactions on, vol. 30, no. 8, pp. 491–506, 2004.

[45] Á. Mitchell and J. F. Power, "A study of the influence of coverage on the relationship between static and dynamic coupling metrics," Science of Computer Programming, vol. 59, no. 1, pp. 4–25, 2006.

[46] Y. Hassoun, S. Counsell, and R. Johnson, "Dynamic coupling metric: proof of concept," in Software, IEE Proceedings-, 2005, vol. 152, pp. 273–279.

[47] Y. Hassoun, R. Johnson, and S. Counsell, "A dynamic runtime coupling metric for meta-level architectures," in Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings. Eighth European Conference on, 2004, pp. 339–346.