بررسی سیستم‌های پیشنهادی بازبینی کننده کدهای گذشته و حال

پالاک هالوادیا

لیسانس مهندسی کامپیوتر، دانشگاه فنی گجرات، ۲۰۱۷

پایان‌نامه ارسال شده

در تحقق‌بخشی از الزامات برای درجه

کارشناسی ارشد

که در

علوم کامپیوتر

گروه ریاضی و علوم کامپیوتر

دانشگاه لثبریج

لثبریج، آلبرتا، کانادا

خلاصه

زمینه: انتخاب یک بررسی‌کننده کد جنبه مهمی از توسعه نرم‌افزار است و به عوامل مختلفی بستگی دارد.

اهداف: هدف درک راه‌حل‌های موجود برای سیستم‌های توصیه بازنگری کد (CRRS)، عواملی که باید در هنگام ساخت آن‌ها در نظر گرفته شوند و ابعاد مختلفی که بر اساس آن می‌توان آنها را طبقه‌بندی کرد، است. هدف ما درک ویژگی‌های مهم CRRS و آنچه می‌توان در CRRSهای موجود بهبود داد، است.

روش‌ها: مطالعه مروری ادبیات برای درک CRRSهای موجود انجام شد. یک نظرسنجی از اعضای پروژه توسعه نرم افزار برای درک ویژگی های مهم و گمشده در CRRS انجام شد.

یافته‌ها: مقالات انتخاب شده را به دودسته دسته‌بندی کردیم: بر اساس نوع داده‌های مورداستفاده برای ارائه توصیه‌ها و نوع پروژه مورداستفاده برای ارزیابی. این نظرسنجی به ما کمک کرد تا ویژگی‌های گمشده در CRRS را درک کنیم و برخی روندها و الگوها را مشاهده کنیم.

فصل ۱

معرفی

بررسی کد یک بررسی سیستماتیک از کد منبع کامپیوتر است و اغلب به‌عنوان یک بررسی همتا انجام می‌شود. هدف بررسی کد شناسایی و اصلاح اشتباهات در کد منبع و همچنین بهبود کیفیت کد و مهارت‌های توسعه‌دهنده نرم‌افزار است. همچنین، هدف آن تنها بهبود کیفیت کد یا یافتن نقص در کد منبع نیست. همچنین باعث افزایش آگاهی تیم و همچنین کمک به توزیع دانش می‌شود. همچنین مالکیت کد مشترک را تشویق می‌کند.

چهار نوع بررسی کد وجود دارد:

۱. برنامه‌نویسی جفتی: در این نوع بررسی کد، دو توسعه‌دهنده به طور هم‌زمان کد منبع تولید می‌کنند و همچنین به طور هم‌زمان بازبینی می‌کنند.

۲. بررسی کد به کمک ابزار: برای این نوع بررسی کد، نویسندگان و توسعه دهندگان از ابزارهایی برای بررسی کدهای همتا استفاده می‌کنند.

۳. مرور کد: در اینجا، توسعه‌دهنده مرورگر را از طریق مجموعه‌ای از تغییرات کد راهنمایی می‌کند.

٤. بازبینی کد رسمی: این نوع بررسی کد شامل بازرسی دقیق و دقیق کد با مشارکت تعداد زیادی از شرکت‌کنندگان و در چند مرحله است. این یک روش سنتی بازبینی کد است که شامل شرکت در تعدادی جلسات و بررسی خط به خط کد است.

بررسی کد را می‌توان به‌عنوان یک بازرسی دستی از تغییرات در کد منبع در نظر گرفت. تعدادی ابزار و سیستم‌های پیشنهادی برای بررسی کد توسط تعدادی از سازمان‌های مختلف توسعه‌یافته‌اند.

تعدادی زمینه وجود دارد که کمک سیستم‌های توصیه برای اعضای پروژه توسعه نرم‌افزار مفید بوده است. برای کمک به کار بازبینی کد، تحقیقات قابل‌توجهی در مورد سیستم‌های توصیه‌ای انجام شده است که هدف آن ارائه توصیه‌های بازبینان کد بر اساس جنبه‌های مختلف است. دلایل مختلفی وجود دارد که چرا علاوه بر یافتن عیوب کد، به بازبینی کد نیاز است. دلیل این امر این است که بازبینی کنندگان کد نیز بر بهبود کد، یافتن راه‌حل‌های جایگزین برای یک مشکل، انتقال دانش، بداهه‌سازی در فرایند توسعه، اجتناب از شکست‌های ساخت، اشتراک‌گذاری مالکیت کد و همچنین ارزیابی تیم تمرکز می‌کنند.

به‌عنوان‌مثال رحمان، روی و کالینز، یک سیستم بازبینی کد را پیشنهاد کردند که در آن تخصص یک بازبینی کننده کد بر اساس اطلاعات به‌دست‌آمده از سابقه کاری بین پروژه‌ای و همچنین تخصص یک بازنگری کد در یک زمینه خاص است. در مورد درخواست‌های کشش آنها

نمونه‌هایی از سیستم‌های توصیه عمومی‌تر در مهندسی نرم‌افزار شامل سیستم‌های توصیه گر کد گرافیکی و سیستم Hipikat است. لی و کانگ مطالعه‌ای بر روی «سیستم‌های توصیه‌گر کد گرافیکی» انجام دادند تا بفهمند ابزارهای تجسم نرم‌افزار تا چه اندازه به توسعه‌دهندگان در درک کد کمک کرده‌اند. نویسندگان دریافتند که توسعه دهندگان زمان قابل‌توجهی را صرف درک مبانی کد می‌کنند. برای سهولت این کار، تعدادی توصیه گر کد گرافیکی برای آنها

ایجاد شد. این سیستم‌های توصیه گر از دو انتزاع استفاده می‌کردند: طراحی و توضیح کد و مستندسازی سیستم نرم‌افزار و همچنین تجزیه‌وتحلیل آن.

یک سیستم توصیه‌کننده به نام Hipikat توسعه داده شد که به توسعه‌دهندگان دسترسی به یک حافظه گروهی را که حاوی مصنوعات مرتبط با پروژه ایجاد شده در طول توسعه پروژه است، فراهم می‌کند [۲]. این به توسعه دهندگان کمک می‌کند تا در زمان غلبه بر مشکلات فنی و جامعه‌شناختی صرفه‌جویی کنند. این ابزار حافظه گروه را به طور خودکار با تغییرات بسیار کم یا بدون تغییر در شیوه‌های کاری موجود ایجاد می‌کند. این سیستم توصیه به‌اشتراک‌گذاری اطلاعات مربوط به یک پروژه از هر منظر به همه اعضای تیم توسعه کمک کرد و در نتیجه در زمان توضیح مفاهیم برای اعضای موجود و جدید تیم توسعه صرفه‌جویی کرد.

۱,۱ مروری بر کار

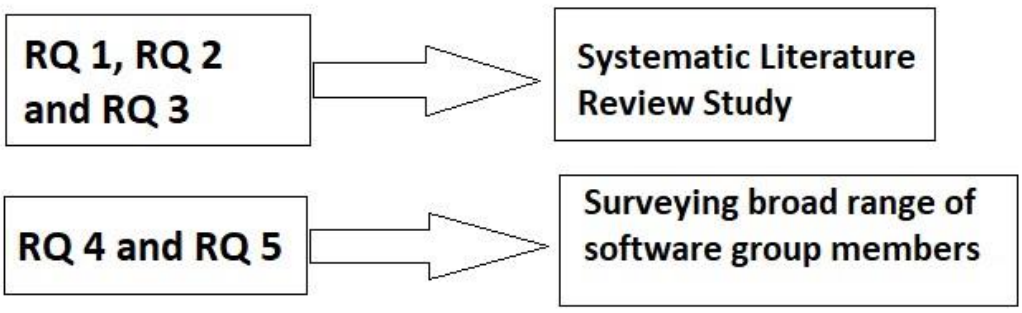
۱. انواع بررسی کد. این کار بر بررسی کدهای پشتیبانی شده از ابزار متمرکز خواهد بود.

هدف تحقیق ما دو مورد است: اول، یافتن پاسخ به سؤالات «گذشته» با انجام «مرور ادبیات سیستماتیک» و دوم یافتن پاسخ سؤالات «حال» با انجام یک نظرسنجی از اعضای پروژه نرم‌افزاری. یک «مرور ادبیات سیستماتیک» به یافتن جزئیات در مورد سیستم‌های پیشنهادی بازبین کد موجود کمک می‌کند، درحالی‌که این نظرسنجی به یافتن تغییراتی که مهندسان نرم‌افزار فکر می‌کنند موردنیاز است یا آنچه در سیستم‌های پیشنهادی بازبینی کد موجود وجود ندارد، کمک می‌کند.

۱,۲ چرا این کار موردنیاز است

هدف این تحقیق مستندسازی آموخته‌های به‌دست‌آمده از «مطالعه مروری نظام‌مند ادبیات» و همچنین «نظرسنجی» انجام‌شده بر روی طیف وسیعی از اعضای پروژه نرم‌افزاری است. این کار به این دلیل انجام شد که تحقیقات بسیار کمتری در مورد سیستم‌های توصیه بازبین کد (CRRS) و تحقیقات بیشتری در مورد رویه‌ها و رویه‌های بررسی کد انجام شده است. اهدافی که به ما در یافتن پاسخ برای نتیجه موردنیاز کمک می‌کند در زیر ذکر شده است:

۱. تجزیه‌وتحلیل راه‌حل‌های ارائه شده توسط "سیستم‌های پیشنهادی بازنگری کد" موجود در حال حاضر.
۲. برای درک راه‌حل‌ها/ویژگی‌هایی که در «سیستم‌های پیشنهادی بازنگری کد» موجود وجود ندارد.
۳. تجزیه‌وتحلیل ابزارهای مختلف انشعاب "سیستم‌های پیشنهادی بازنگری کد" موجود بر اساس روش اجرای آنها.



شکل ۱٫۱: سؤالات و روش تحقیق

سؤالات تحقیق برای این کار به شرح زیر است:

۱) راه‌حل‌های موجود برای سیستم‌های پیشنهادی برای بازبینی کنندگان کد چیست؟
۲) چه عواملی باید در هنگام ایجاد یک سیستم توصیه برای بازبینان کد در نظر گرفته شوند؟
۳) چگونه می‌توان سیستم‌های پیشنهادی موجود برای بازبینی کنندگان کد در ادبیات را دسته‌بندی کرد؟
٤) ویژگی‌های مهم یک سیستم توصیه برای بازبینان کد چیست؟
٥) چگونه می‌توان سیستم‌های پیشنهادی موجود برای بازبینان کد را بهبود بخشید؟ به‌عبارت‌دیگر، چه ویژگی‌هایی در پیاده‌سازی‌های موجود برای سیستم‌های توصیه بازدیدکنندگان کد وجود ندارد؟

۱ و ۲ و ۳ با استفاده از مرور ادبیات سیستماتیک پاسخ داده می‌شوند، درحالی‌که ٤ و ٥ با بررسی طیف گسترده‌ای از اعضای پروژه نرم‌افزار پاسخ داده می‌شوند.

۱,۳ مشارکت

این تحقیق مشارکت‌های زیر را انجام می‌دهد:

۱. ما تعدادی از ویژگی‌های موجود در سیستم‌های پیشنهادی بازبینی کد موجود ((CRRSرا شناسایی کردیم و آن ویژگی‌ها را بر اساس مفید بودنشان رتبه‌بندی کردیم.
۲. ما CRRSهای موجود را بر اساس ابعاد مختلف دسته‌بندی کردیم.
۳. ما ویژگی‌هایی را شناسایی کردیم که می‌توان هنگام انتخاب یک بازبین کد مهم در نظر گرفت.
٤. ما بهبودهای احتمالی را در CRRSهای موجود شناسایی کردیم تا یافتن بازبینی کننده‌های کد مناسب را تسهیل کنیم.

فصل ۲ کار مرتبط را ارائه می‌دهد. نتایج مطالعه مرور ادبیات در فصل ۳ ارائه شده است. فصل ٤ بررسی نتایج است. فصل پنجم شامل بحث است. پایان‌نامه در فصل ٦ به پایان رسیده است.

فصل ۲
کار مرتبط

این فصل خلاصه‌ای از بررسی‌های ادبیات قبلی انجام شده در مهندسی نرم‌افزار، داده‌کاوی و سیستم‌های توصیه‌کننده را ارائه می‌کند. این مطالعات برای نشان‌دادن اینکه چگونه مطالعات مرور ادبیات درگذشته انجام شده و برای هدایت روش‌شناسی مرور ادبیات ما استفاده شده است، ارائه شده است.

بررسی ادبیات انجام شده در مورد سیستم‌های توصیه‌گر شامل سیستم‌های توصیه‌کننده است که هدف آن استخراج اطلاعات مرتبط از حجم عظیمی از دانش و سیستم‌های توصیه‌ای برای مهندسی نرم‌افزار است که ویژگی‌های موجود در سیستم‌های موجود، شکاف‌های تحقیقاتی و کارهای احتمالی آینده را ارائه می‌دهد. به طور مشابه، مطالعه مروری بر ادبیات در زمینه مهندسی نرم‌افزار دررابطه‌با مطالعات پیش‌بینی خطا و روش توسعه نرم‌افزار چابک انجام شد. بررسی ادبیات انجام شده درداده کاوی دو مدل پرکاربرد برای داده‌کاوی در CRM(مدیریت ارتباط با مشتری) را کشف کرد.

۲,۱ بررسی‌های ادبیات سیستم‌های توصیه‌کننده

یک مرور ادبیات توسط هارونا، اسماعیل، سوهندرویونو و همکاران انجام شد. [۱۸] در مورد سیستم‌های توصیه‌کننده آگاه از زمینه (CARS)که هدف آن استخراج اطلاعات مربوطه موردنیاز از حجم عظیمی از دانش است. هدف این نوع سیستم‌های توصیه‌گر ارائه اطلاعات متنی و مرتبط بر اساس «جستجوهای کاربر» و ارائه توصیه‌های شخصی شده‌تر برای کاربران است. رویکرد هارونا، اسماعیل، سوهندرویونو و همکاران [۱۸] از سه مرحله اصلی تشکیل شده است. مرحله اول شامل بررسی عمیق و طبقه بندی ادبیات بر اساس حوزه‌های مختلف مدل‌های کاربردی، فیلترینگ، استخراج و همچنین رویکردهای ارزیابی است. مرحله دوم شامل ارائه نتایج بررسی با مزایا و معایب بررسی است. مرحله سوم و نهایی شامل برجسته کردن چالش‌ها/فرصت‌های احتمالی یا کار یا تحقیقات آینده است که می‌توان انجام داد. این شامل کمک به محققان تازه کار و جدید در درک پیش نیازهای توسعه CARS و همچنین ارائه این بررسی به عنوان معیاری برای توسعه CARS برای کاربران متخصص است.

سیستم توصیه نوعی نرم افزار کاربردی است که هدف آن ارائه/توصیه اطلاعات مرتبط به کاربران بر اساس نیازهای کاربر است. برای این حوزه، یک مطالعه مروری نظام‌مند ادبیات مشابه با Gasparic و Janes انجام شد که نتایج عملکردی را که RSSEهای موجود (سیستم توصیه‌ای برای مهندسی نرم‌افزار) ارائه می‌دهند، شکاف‌های تحقیقاتی و همچنین جهت‌های تحقیقاتی ممکن را بررسی می‌کند. آنها یک رویکرد روش‌شناختی را دنبال کردند که شامل فیلتر کردن مقالات پژوهشی جمع‌آوری‌شده و مرتبط بر اساس معیارهای مختلف بود. معیارهای خروج آنها شامل مقالاتی بود که به زمینه تحقیق بی ربط بودند، مقالاتی که راه حل های اجرا نشده را توصیف می کردند یا مقالاتی که به طور کامل در دسترس نبودند. برای استخراج مقالات مربوطه، مقالات بر اساس محتوای شرح داده شده در چکیده مقاله یا گاهی عنوان، فیلتر و تقسیم شدند.

نویسندگان پس از پیروی از رویکرد روش‌شناختی خود، به چهار سؤال تحقیقاتی خود پاسخ دادند که عبارتنداز:

- خروجی ارائه شده توسط RSSEهای موجود

- مزایایی که این RSSEها برای مهندسان نرم‌افزار ارائه می‌دهند
- انواع ورودی‌هایی که این RSSEها نیاز دارند و تلاش‌هایی که یک RSSE انجام می‌دهد. مهندس نرم افزار باید برای استفاده از این RSSEها قرار دهد. مشاهده شد که برخی از خروجی‌های ارائه شده توسط RSSEهای موجود شامل فایل‌های کد منبع باینری، تغییرات در محیط استقرار، الگوهای طراحی و اسناد دیجیتالی است که ممکن است برای مهندس نرم افزار جالب باشد.

RSSE های موجود عمدتاً از استفاده مجدد، اشکال زدایی، پیاده سازی، مراحل/فعالیت های نگهداری و پشتیبانی از بهبود کیفیت سیستم به نفع مهندسان نرم افزار. برخی از ورودی‌هایی که این RSSEهای موجود استفاده می‌کنند شامل فایل‌های گزارش، ارتباط بین مهندسان نرم‌افزار، کد منبع، ورودی کاربر (به‌عنوان مثال، عبارت‌های جستجو، پرس و جو، تنظیمات، اولویت‌ها)، مصنوعات آزمایشی و فرآیند توسعه نرم‌افزار است. همچنین تلاش‌هایی که یک مهندس نرم‌افزار برای استفاده از RSSEهای موجود می‌خواهد به‌عنوان تلاش گسترده، تلاش کم و بدون تلاش دسته‌بندی می‌شود.

بررسی ادبیات دیگری برای RSSEها توسط پارک، کیم، چوی و همکاران انجام شد.[۸] که در آن نویسندگان مقالات تحقیقاتی را بر اساس هشت زمینه کاربردی و هشت تکنیک داده کاوی دسته بندی کردند. هدف نویسندگان ارائه اطلاعاتی در مورد روندهای حوزه تحقیق برای سیستم‌های توصیه گر و همچنین تعریف جهت گیری احتمالی تحقیقات آینده در مورد سیستم‌های توصیه گر بود.

۲,۲ بررسی ادبیات در مهندسی نرم افزار

پیش بینی دقیق عیوب در کد می‌تواند هزینه تست را تا حد زیادی کاهش دهد و همچنین کیفیت محصول نرم افزار را افزایش دهد. برای این منظور، مطالعه مروری بر ادبیات توسط هال، بیچم، بووز و همکاران انجام شد. [۷] که بر مطالعات پیش‌بینی خطا متمرکز بود. نویسندگان رویکرد مرور سیستماتیک ادبیات را همانطور که توسط کیچنهام و چارترز [٤] پیشنهاد شده بود، دنبال کردند که در آن مراحل اولیه شامل مقالات و مطالعات مربوطه و حذف مطالعات مکرر است. در حذف و شمول مقالات به جنبه‌های مختلفی توجه می‌شود، مانند مقالاتی که از منابع مختلف مانند مجلات، کنفرانس‌ها و پایگاه‌های اطلاعاتی استخراج شده و بر اساس محتوای عنوان و چکیده مرتب شده اند. نویسندگان در نهایت حدود ۲۰۸ مقاله داشتند. یافته‌ها نشان می‌دهد که اکثر مطالعات اطلاعات زمینه‌ای و روش‌شناختی کافی برای درک کامل یک مدل را گزارش نمی‌کنند. نویسندگان مجموعه‌ای از معیارها را ارائه می‌کنند که مجموعه‌ای از جزئیات زمینه‌ای و روش‌شناختی ضروری را که مطالعات پیش‌بینی خطا باید گزارش کنند، شناسایی می‌کنند.

متدولوژی توسعه نرم افزار Agile یک متدولوژی توسعه نرم افزار رایج است که توسط بسیاری از پروژه‌های توسعه نرم افزار استفاده می‌شود. هدف این روش اطمینان از تحویل خوب محصول مطابق با نیازهای کاربر و تجربه کاربری مناسب (UX)است. برای ارائه محصولی با کیفیت، مشارکت ذینفعان و کاربران به همراه حلقه‌های بازخورد از هر دو طرف ضروری است. یک مطالعه مروری بر روی روش چابک توسط شون، توماسچوسکی و اسکالونا [۱۹]¨ انجام شد تا وضعیت فعلی کار در این زمینه و پیشرفت‌های احتمالی آینده برای پرداختن به هر جنبه‌ای که در وضعیت فعلی وجود ندارد، نشان داده شود. آنها مطالعه را در سه مرحله اصلی انجام دادند: برنامه ریزی، انجام و گزارش. مرحله "برنامه ریزی" شامل یافتن نیاز شناسایی برای بازبینی، چارچوب بندی سوالات تحقیق و توسعه و ارزیابی پروتکل

بازبینی بود. مرحله «انجام» با هدف جستجوی مقالات پژوهشی، انتخاب مقالات مربوطه برای مطالعه، ارزیابی کیفی و استخراج و تجزیه و تحلیل داده‌ها است. مرحله آخر، "گزارش" با هدف استخراج و بحث در مورد نتایج به دست آمده از مرحله قبل و سپس نوشتن، ارزیابی و قالب بندی گزارش نهایی برای مطالعه است. مشابه سایر مطالعات، نویسندگان از روش ارائه شده توسط کیچنهام و چارترز [٤] پیروی کردند.

۲,۳ بررسی ادبیات در داده کاوی

بررسی ادبیات انجام شده توسط Ngai، Xiu و Chau [6]نمونه دیگری از روش شناسی برای مرور ادبیات سیستماتیک در حوزه داده کاوی را ارائه می‌دهد.

تکنیک‌های داده کاوی برای مدیریت ارتباط با مشتری (CRM)اعمال می شود و نگای، ژیو و چاو [٦] با کمک بررسی ادبیاتی که انجام داده اند، بینش کاملی در این مورد ارائه می‌دهند. نویسندگان حدود ۸۷ مقاله تحقیقاتی مرتبط را برای این منظور جمع آوری کردند که بر اساس چهار بعد CRM تقسیم شدند که شامل توسعه مشتری، شناسایی مشتری، جذب مشتری و حفظ مشتری و هفت تکنیک داده کاوی می‌شود. ارتباط، طبقه بندی، خوشه بندی، پیش بینی، رگرسیون، کشف توالی و تجسم. جدای از این، برای وضوح بیشتر، ابعاد CRM بیشتر به ۹ زیر شاخه از عناصر CRM که تحت تکنیک‌های داده کاوی قرار می‌گیرند، طبقه بندی شدند. بر اساس مطالعه، مشخص شد که طبقه بندی و انجمن دو مدل پرکاربرد برای داده کاوی در CRM هستند. همچنین، از چهار بعد CRM، حفظ مشتری بیشترین تحقیق را دارد، اگرچه بیشتر آنها مربوط به برنامه‌های بازاریابی و وفاداری یک به یک بودند.

۲,٤ خلاصه

مطالعات مرور ادبیات برای سیستم‌های توصیه‌گر، و از زمینه‌های مهندسی نرم‌افزار و داده‌کاوی ارائه شد. برای سیستم‌های توصیه، مرور ادبیات بر روی سیستم‌های توصیه‌گر آگاه از زمینه و سیستم‌های توصیه در مهندسی نرم‌افزار متمرکز است. در زمینه مهندسی نرم افزار، مطالعه ارائه شده بر روی حوزه پیش بینی خطا و همچنین روش چابک انجام شده است. در زمینه داده کاوی، مطالعه مروری بر ادبیات انجام شد که در آن مفهوم داده کاوی اعمال شده در مدیریت ارتباط با مشتری (CRM)مورد هدف قرار گرفت.

فصل ۳

روش‌ها و ابزارهای مرور کد

تعدادی سیستم/ابزار توصیه بازبینی کد در گذشته توسعه یافته است. به عنوان مثال، ابزاری به نام Review Bot یکی دیگر از ابزارهای توسعه یافته توسط Balachandran [10] است که در پروژه VMware مورد استفاده قرار گرفت. Review Bot از الگوریتمی تشکیل شده بود که تغییرات کد انجام شده در یک خط کد را به روشی کاملاً شبیه به دستور git blame بررسی می‌کند. به هر نویسنده‌ای که روی تغییر کد در کد منبع کار کرده باشد، امتیاز تعلق می‌گیرد، اما نویسندگانی که تغییرات اخیر دارند امتیاز بیشتری نسبت به نویسندگانی که تغییرات قدیمی‌تر دارند، کسب می‌کنند. در پایان، از جمع‌بندی هر نویسنده برای تعیین نویسندگان برتر استفاده می‌شود و سپس به آن‌ها توصیه می‌شود تا مرورگر کد شوند.

ما یک مطالعه مروری نظام مند ادبیات را به منظور یافتن پاسخ به سه سوال تحقیق اول خود انجام دادیم. ابتدا، قبل از ارائه نتایج مطالعه خود، روش شناسی خود را تعریف می‌کنیم.

۳,۱ روش شناسی

مطالعه مروری نظام مند ادبیات روشی است که در آن ادبیات موجود مرتبط با تحقیق تعیین، سپس ارزیابی و در نهایت درک می‌شود. برای تحقیق خود، ما رویکرد اتخاذ شده توسط کیچنهام و چارترز [٤] را دنبال کرده ایم که شامل مراحل زیر است:

۱. کلیه کلیدواژه‌های ممکن مرتبط با تحقیق مشخص شده است. ما کلمات زیر را شناسایی کردیم: کد، بازبینی، توصیه، سیستم، ابزار و توصیه‌کننده. این کلمات کلیدی بر اساس موضوع تحقیق ما شناسایی شدند.

۲. از کلمات کلیدی شناسایی شده برای تشکیل رشته‌های جستجو استفاده کنید. رشته‌های جستجو برای به دست آوردن مقالات تحقیقاتی از پایگاه‌های داده آنلاین استفاده می شود. ما از یک رشته جستجو از همه کلمات کلیدی ممکن و مترادف آنها استفاده کردیم. ما دو رشته جستجو ایجاد کردیم. برای رشته‌های جستجو به جدول ۳,۱ مراجعه کنید.

۳. مقالات پژوهشی به دست آمده بر اساس معیارهای مختلف خروج و ورود فیلتر می‌شوند. مقاله‌ها ابتدا با مطالعه عناوین و چکیده مقالات پژوهشی فیلتر می‌شوند.

فیلتر کردن نتایج جستجو در سه مرحله اصلی انجام شد:

الف) فیلتر کردن مقالات پژوهشی بر اساس مطالعه عنوان مقاله.

ب) فیلتر کردن مقالات پژوهشی بر اساس چکیده مقاله.

ج) فیلتر کردن مقالات پژوهشی بر اساس خواندن متن کامل.

٤. مقالات فیلتر شده به طور کامل خوانده، ارزیابی و تفسیر می‌شوند تا اطلاعات مربوطه به دست آید.

در هر مرحله از فیلتراسیون، تعدادی از مقالات تحقیقاتی فیلتر شدند. پس از مطالعه عناوین مقاله، ۱۹ مقاله فیلتر شدند. از این مقالات به دست آمده، ۲۱ مقاله پس از مطالعه چکیده مقالات فیلتر شدند. در نهایت ۷ مقاله پس از مطالعه متن کامل مقالات پژوهشی فیلتر شدند. در نهایت ١٤ مقاله به دست آمد.

۳,۲ نتایج

پس از مطالعه متن کامل مقالات پژوهشی فیلتر شده، ۹ سیستم توصیه‌کننده مرور کد شناسایی شدند. جدول ۳,۲ این سیستم‌ها و داده‌های مورد استفاده توسط سیستم را برای ارائه یک توصیه فهرست می‌کند. در ادامه این بخش توضیحاتی در مورد هر یک از این سیستم‌ها ارائه می‌شود.

جدول ۳,۲: مقاله‌های استخراج شده و داده‌های مورداستفاده

نوع داده	عنوان مقاله پژوهشی
تاریخچه مرور کد	توصیه خودکار داوران در بررسی کد مدرن
متعهد می‌شود	مرور کد مدرن: مطالعه موردی در گوگل
وضعیت هر منتقد یا نویسنده را دنبال می‌کند	An : ویژگی های بررسی کد مفید مطالعه تجربی در مایکروسافت
شباهت مسیر پرونده (FPS)	یک مطالعه در مقیاس وسیع در مورد توصیه مرورگر کد منبع
پروژه متقابل مرتبط و تجربه فناوری	correct:توصیه مرورگر کد در GitHub بر اساس تجربه پروژه و فناوری
استخراج متن و محل فایل	چه کسی باید این تغییر را مرور کند؟
بر اساس مشخصات	توصیه مبتنی بر مشخصات مرورگران کد

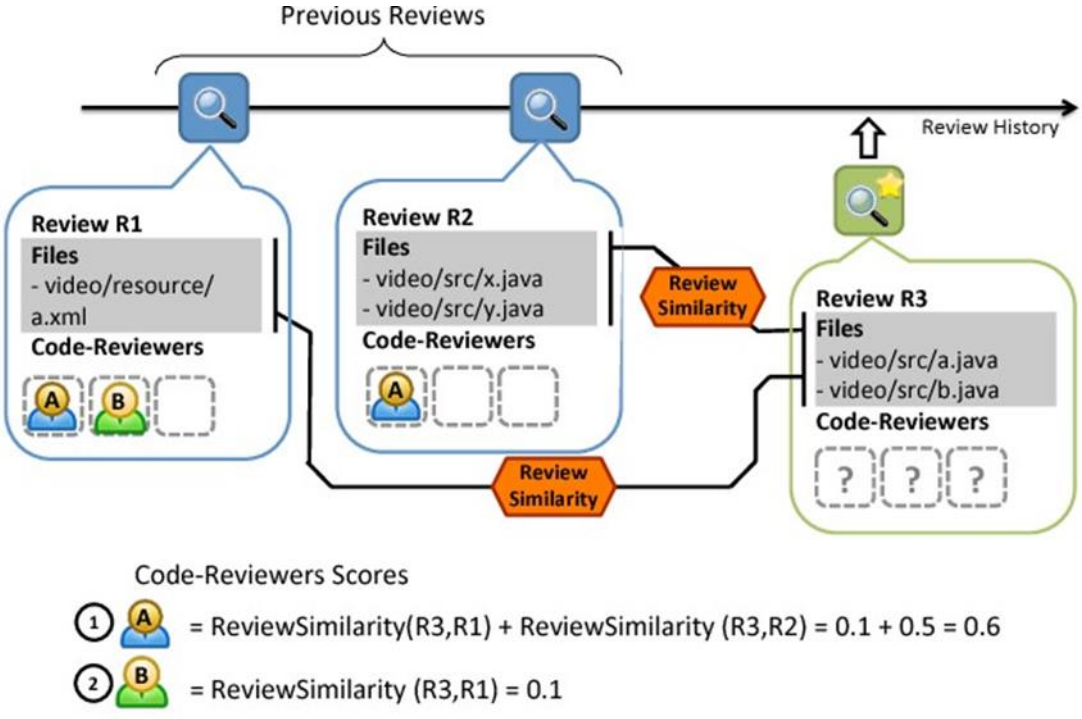
۳,۳ سیستم پیدا شد

REVFINDER ۳,۳,۱

تعدادی CRRS بر اساس سیستم‌های مسیر فایل (FPS)یا رویکرد مبتنی بر مکان فایل پیشنهاد شده است. Thongtanunam، Tantithamthavorn، Kula، و همکاران.[۱۲] REVFINDER پیشنهاد شده است که از رویکرد بازبینی کد مبتنی بر مکان فایل پیروی می‌کند. شهود پشت این رویکرد این است که چندین فایل با یک مکان/مسیر فایل مشابه توسط بازبینی‌کنندگان کد با تجربه مشابه بررسی و مدیریت می‌شوند.

Thongtanunam، Tantithamthavorn، Kula، و همکاران.[۱۲] همچنین یک مطالعه اکتشافی در مورد اینکه چگونه تخصیص مرورگر کد بر زمان بازبینی تأثیر می‌گذارد، انجام داد. مطالعه اکتشافی نشان داد که حدود ۳۰-٤ درصد از بررسی‌های کد با مشکل تعیین بازنگری کد صحیح روبرو هستند و حدود ۱۲ روز بیشتر طول می‌کشد تا تغییر کد تأیید شود. بر اساس نتایج این مطالعه، نویسندگان REVFINDER را پیشنهاد کردند.

REVFINDER از دو بخش تشکیل شده است: الگوریتم رتبه بندی بازبینان کد و تکنیک ترکیبی. نویسندگان از الگوریتم رتبه‌بندی بازبینان کد (همانطور که در شکل ۳,۱ نشان داده شده است) استفاده کردند تا نمرات بازبینی‌کنندگان کد را بر اساس شباهت مسیرهای فایل‌هایی که قبلاً بررسی شده‌اند ارزیابی کنند. با توجه به یک بررسی جدید R3 و دو بررسی قبلی R1 و R2، الگوریتم امتیاز شباهت مرور را برای هر یک از بررسی‌های گذشته ((R1,R2با مقایسه مسیرهای فایل با بررسی جدید R3 محاسبه می‌کند. از این رو، دو نمره شباهت مرور در اندازه دو وجود داشت: (R3، (R1و (R3، .(R2از شکل مشاهده می شود که بررسی R3 و R2 در مقایسه با R3 و R1 کلمات کلیدی رایج تری را به اشتراک می گذارند که به این معنی است که داور A می تواند به عنوان بازبینی کننده بالقوه برای بررسی R3 در نظر گرفته شود. به منظور محاسبه شباهت مسیر فایل، نویسندگان از چهار تکنیک مقایسه رشته ای پیشرفته [۱] استفاده کردند:



شکل ۳,۱: مثال محاسبه الگوریتم رتبه بندی Code-Reviewers

۱. طولانی ترین پیشوند مشترک (LCP)

LCP اجزای مسیر فایل مشترک را محاسبه می‌کند که در هر دو مسیر فایل از ابتدا تا انتها ظاهر می‌شوند.

۲. طولانی‌ترین پسوند مشترک (LCS)

LCS اجزای مسیر فایل مشترک را که در هر دو مسیر فایل از انتهای هر دو مسیر فایل ظاهر می‌شوند محاسبه می‌کند.

۳. طولانی‌ترین زیر رشته مشترک (LCSubstr)

LCSubstr اجزای مسیر فایل مشترک را محاسبه می‌کند که در هر دو مسیر فایل به طور متوالی ظاهر می‌شوند، اما همچنین در هر موقعیتی در مسیرهای فایل ظاهر می‌شوند.

٤. طولانی‌ترین دنباله متداول (LCSubseq)

LCSubseq اجزای مسیر فایل مشترک را که در هر دو مسیر فایل به یک ترتیب نسبی ظاهر می‌شوند محاسبه می‌کند.

حال برای محاسبه شباهت مسیر فایل بین فایل fn و فایل fp تابع filePathSimilarity(fn,fp) به صورت زیر محاسبه می شود:

$$\text{filePathSimilarity}(f_n, f_p) = \frac{\text{StringComparison}(f_n, f_p)}{\max(\text{Length}(f_n), \text{Length}(f_p))}$$

مسیر فایل با استفاده از کاراکتر اسلش ("/") به عنوان جداکننده به نشانه‌ها تقسیم می‌شود. سپس از تابع StringComparison(fn, fp) برای مقایسه مولفه های مسیر فایل fn و fp استفاده می‌شود که اجزای فایل مشترکی را که در هر دو مسیر فایل ظاهر می‌شوند، برمی گرداند.

cHRev 3.3.2

تعدادی از CRRS بر اساس بررسی‌های گذشته ساخته شده اند و Zanjani, Kagdiو Bird [۱۷] یکی از این سیستم‌های توصیه به نام cHRev را ساخته اند. cHRev به طور خودکار بازبینان کد را بر اساس مشارکت‌های قبلی آنها در بررسی‌های قبلی خود توصیه می‌کند. cHRev مخفف عبارت Code Review Histories به جای دیگر انواع اطلاعات گذشته برای توصیه Reviewers است. این سیستم توصیه دو ویژگی کلیدی دارد:

۱. بازبینان کد توصیه شده توسط cHRev ممکن است لزوماً در توسعه بخشی از کد منبع که در حال بررسی هستند دخالت نداشته باشند، اما ممکن است روی کد منبعی کار کرده باشند که به طور غیرمستقیم به کد منبعی که بررسی می‌کنند وابسته است.

۲. تخصص در طول زمان تغییر می‌کند و از این رو باید به تازگی و فراوانی در هنگام جستجوی مناسب‌ترین بازبینی کننده کد در نظر گرفته شود.

فرآیند استفاده شده توسط cHRev شامل سه مرحله است:

۱. کد منبعی که نیاز به بررسی دارد را استخراج کنید.

۲. تخصص بازبین را بر اساس جزئیات مختلف از جمله اینکه چه کسی، چه تعداد و چه زمانی در گذشته بررسی شده است، تدوین کنید.

۳. فهرست رتبه‌بندی شده بازبینان نامزد را بر اساس فایل‌های کد منبع در مرحله ۱ و مشارکت‌های انباشته بازبینان از مرحله ۲ به دست آورید و سپس با استفاده از یک پارامتر تعریف‌شده توسط کاربر، تعداد بالای m از نامزدها را از لیست به‌دست‌آمده توصیه کنید.

به منظور آزمایش اثربخشی رویکرد خود، زنجانی، کاگدی و برد [۱۷] رویکرد خود را با REVFINDER[12]، xFinder [5]و RevCom مقایسه کردند. مشخص شد که cHRev توصیه‌های دقیق تری را از نظر دقت و یادآوری ارائه می‌دهد. همچنین، مشاهده شد که cHRev از نظر بررسی کننده‌ها بر اساس فایل‌هایی که نام‌ها و مسیرهای مشابه

دارند و xFinder که به داده‌های مخزن کد منبع بستگی دارد، بهتر از REVFINDER عمل می‌کند. مشخص شد که cHRev از نظر آماری معادل RevCom [17]است که هم نیاز به بررسی و هم تعهدات گذشته دارد.

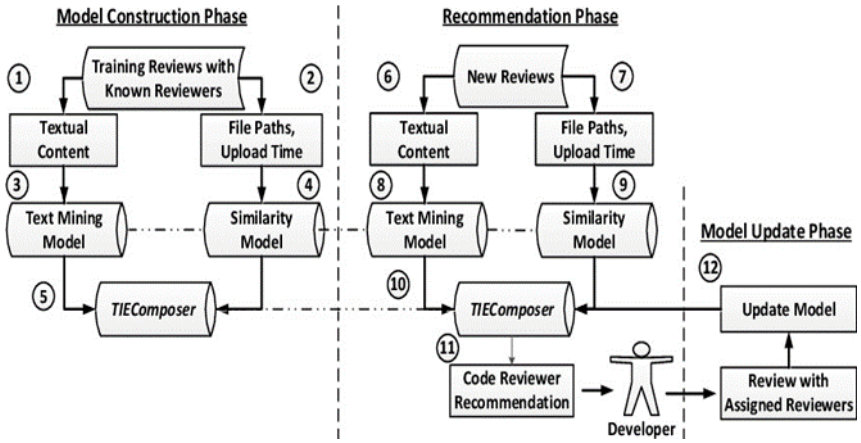
3.3.3 CoRReCT

رحمان، روی و کالینز [۱٦] یک سیستم توصیه بازبینی کد به نام CORReCTتوصیه بازبینی کد مبتنی بر تجربه بین پروژه و فناوری) را پیشنهاد کردند که هدف آن توصیه بازبینان کد بر اساس سابقه کاری بین پروژه‌ای مرتبط و همچنین تجربه بود. توسعه دهندگان در یک فناوری تخصصی خاص مرتبط با درخواست کشش. این دو منبع اطلاعاتی برای تعیین تجربه توسعه دهنده برای بازبینی کد مورد استفاده قرار گرفتند. ایده اصلی پشت CRRS پیشنهادی آن‌ها این است که اگر درخواست‌های کشش گذشته کتابخانه‌ها یا فناوری‌های تخصصی مشابه درخواست‌های کشش فعلی داشته باشند، بازبینی‌کنندگان کدی که این درخواست‌های کشش را بررسی کرده‌اند می‌توانند به‌عنوان بازبینی‌کننده‌های کد بالقوه برای درخواست‌های کشش فعلی در نظر گرفته شوند. با توجه به ایده پیشنهادی نویسندگان، توسعه‌دهندگانی که تجربه بیشتری در کتابخانه‌های خارجی دارند و فناوری‌های تخصصی اتخاذ شده در فایل‌های تغییر در مجموعه نشانه‌های درخواست‌های کشش فعلی، انتخاب‌های مناسب‌تری برای انجام بازبینی کد نسبت به آنهایی با تجربه کمتر در نظر گرفته می‌شوند. . نویسندگان یک مطالعه اکتشافی با پروژه های تجاری و ۱۰ کتابخانه خارجی با فناوری های تخصصی موجود در آنها انجام دادند. نویسندگان فرض می کنند که دو درخواست کششی با کتابخانه های مشترک و فناوری های رایج در فایل هایی که تغییر می کنند مشابه هستند. بر اساس این فرض، آنها شباهت کسینوس را با استفاده از نام کتابخانه یا فناوری به عنوان کیسه ای از نشانه ها محاسبه کردند. کیسه توکن ها به دو دسته توکن تقسیم می شود، یکی برای درخواست کشش فعلی و دیگری برای درخواست کشش گذشته. مقدار شباهت کسینوس از ۰ تا ۱ است که ۰ عدم تشابه کامل کتابخانه ها و فناوری ها و ۱ شباهت کامل کتابخانه ها و فناوری ها است. سپس نویسندگان اقدام به محاسبه تخمین‌های شباهت (به عنوان پروکسی برای بررسی تخصص) به بازبینی‌کنندگان کد مربوطه درخواست‌های کشش گذشته کردند.

TIE ۳,۳,٤

Xia، Lo، Wang، و همکاران. [۱۳] یک رویکرد ترکیبی و افزایشی به نام Text mining(TIE و رویکرد مبتنی بر مکان فایل) را پیشنهاد کرد که از مزایای متن کاوی و رویکرد مبتنی بر مکان فایل برای توصیه بازبین کد استفاده می‌کند. ایده پشت این رویکرد تجزیه و تحلیل محتوای متنی در یک درخواست بازبینی با استفاده از یک مدل متن کاوی افزایشی و محاسبه شباهت بین مسیرهای فایل بررسی شده و مسیرهای فایل تغییر یافته با استفاده از مدل مشابهت است.

معماری کلی TIE به سه مرحله تقسیم می‌شود: ساخت مدل، توصیه و به روز رسانی مدل همانطور که در شکل ۳,۲ نشان داده شده است.



شکل ۳,۲: معماری TIE

۱. فاز ساخت مدل

فاز ساخت مدل شامل یک مدل ترکیبی به نامTIECOMPOSER که با استفاده از بررسی های تاریخی بازبینان شناخته شده ساخته شده است. در این مرحله، سیستم TIE ابتدا بررسی های آموزشی بازبینان شناخته شده را از محتوای متنی بررسی های گذشته و مسیرهای فایل و همچنین زمان آپلود جمع آوری می کند. در مرحله بعد، TIE یک مدل متن کاوی را بر اساس داده های متنی پردازش شده با استفاده از تکنیک طبقه بندی متن می سازد. شهود پشت حالت داده کاوی این است که همان بازبینان احتمال بیشتری دارد که تغییرات را با اصطلاحات یا کلمات مشابه بررسی کنند.

TIE همچنین از یک رویکرد مبتنی بر مکان فایل آگاه به زمان استفاده می‌کند که هدف آن محاسبه شباهت بین بررسی‌های جدید و تاریخی است. این شباهت بین مسیرهای فایل تغییر یافته (یعنی مسیرهای فایل‌هایی که در درخواست بررسی جدید تغییر یا اصلاح شده‌اند) و مسیرهای فایل بررسی شده (یعنی مسیرهای فایل‌هایی که در بررسی‌های تاریخی بررسی شده‌اند) محاسبه می‌شود. شهود پشت رویکرد مبتنی بر مکان فایل این است که همان بازبینان تمایل دارند فایل‌ها یا فایل‌هایی را با مسیرهای مشابه بررسی کنند. این دو مدل برای ساخت مدل TIECOMPOSER با هم ترکیب شده اند.

۲. مرحله توصیه

برای این مرحله، از TIE برای توصیه بازبینی‌کنندگان کد برای درخواست بازبینی اختصاص‌نشده جدید استفاده می‌شود. TIE ابتدا توضیحات تغییر، مسیرهای فایل و زمان آپلود را همانطور که برای بررسی‌های تاریخی در «مرحله ساخت مدل» انجام شد، استخراج می‌کند. برای مرحله بعدی، داده‌های متنی از توضیحات استخراج شده و به عنوان ورودی در مدل داده کاوی ساخته شده در "مرحله ساخت مدل" استفاده می‌شود. به طور مشابه، سیستم همچنین مسیرهای فایل و زمان آپلود را در مدل مشابه ساخته شده در "مدل ساخت فاز" وارد می‌کند.

سپس این دو مدل فهرستی از بازبینی‌کننده‌های کد را تولید می‌کنند و سپس این دو فهرست با استفاده از مدل TIECOMPOSER ساخته‌شده در «مرحله ساخت مدل» ترکیب می‌شوند.

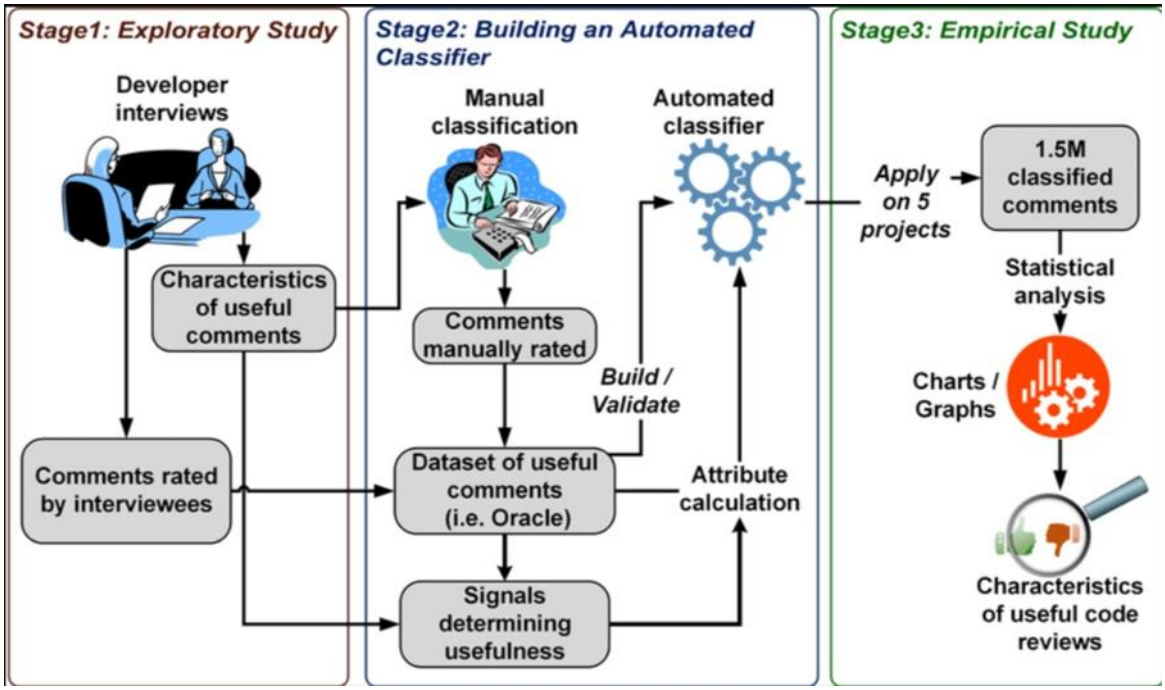
۳. فاز به روز رسانی مدل

در مرحله به روز رسانی مدل، سیستم TIE با استفاده از بازبینی کننده‌های کد جدید اختصاص داده شده به روز می‌شود. در عمل، توسعه دهندگان به طور معمول لیست بازبینان بالقوه را بررسی می‌کنند و سپس یک درخواست کشش جدید را به گروهی از بازبینان اختصاص می‌دهند.

به منظور ارزیابی عملکرد TIE، نویسندگان از مجموعه داده‌های ارائه شده توسط Thongtanunam، Tantithamthavorn، Kula و همکاران استفاده کردند.[۱۲] شامل ٤٢٠٤٥ بررسی و مقایسه عملکرد TIE با RevFinder [12].هر یک از بررسی‌ها در این مجموعه داده‌ها دارای برچسب «ادغام شده» یا «رها شده» بودند و حداقل یک مسیر فایل را شامل می‌شدند. مشاهده شد که به طور متوسط در میان ٤ پروژه منبع باز، TIE به دقت پیش‌بینی برتر ۱، ۳، ۵ و ۱۰ برتر ۰,۵۲، ۰,۷۳، ۰,۷۹ و ۰,۸۵ و میانگین رتبه متقابل 0.64 (MRR) دست یافت. نتایج RevFinder را به ترتیب ٦١، ٣٣، ٢٣، ٨ و ٣٧ درصد شکست داد.

CodeFlow ۳,۳,۵

Bosu، Greiler و Bird [11]یک مطالعه تجربی در مایکروسافت در مورد ویژگی‌های بررسی کد مفید با انجام مصاحبه با توسعه‌دهندگان، و همچنین تجزیه و تحلیل نظرات بررسی پنج پروژه مایکروسافت با استفاده از CodeFlow CRRS انجام دادند. مطالعه در سه مرحله انجام شد. ابتدا، آنها یک مطالعه اکتشافی را با انجام مصاحبه با توسعه دهندگان انجام دادند تا تفسیر آنها از "مفید" را در زمینه بررسی کدها درک کنند. ثانیاً، آنها یک طبقه‌بندی برای تفکیک نظرات «مفید» و «غیر مفید» با استفاده از داده‌های مصاحبه می‌سازند. در نهایت، آنها طبقه‌بندی‌کننده خود را در پنج پروژه مایکروسافت اعمال کردند تا نظرات «مفید» و «غیر مفید» را تشخیص دهند. شکل: ۳,۳ [۱۱] روش تحقیق سه مرحله ای را نشان می دهد.

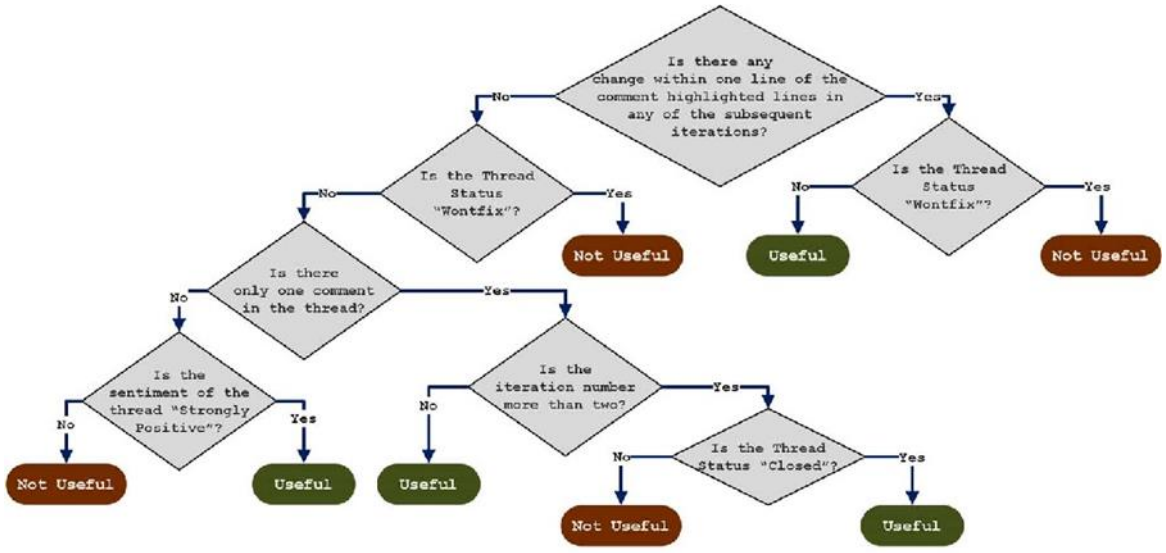


گردش کار CodeFlow نسبتاً ساده است. ابتدا، یک نویسنده یک تغییر بررسی را ارسال می کند و از طریق ایمیل به داور در مورد درخواست بررسی مطلع می‌شود. سپس بازبینی کننده می‌تواند تغییر در خود ابزار را بررسی کند. هنگامی که یک بازبین می‌خواهد در مورد یک خط یا بلوک کد نظر بدهد، بازبین آن قسمت از کد را برجسته می‌کند و نظراتی

را اضافه می‌کند. این نظرات به عنوان موضوعاتی که در آن بحث شروع می‌شود و همچنین به عنوان نقاط تعامل برای افرادی که در بررسی شرکت دارند ظاهر می شوند. هر یک از این موضوعات دارای وضعیتی هستند که شرکت کنندگان می توانند در طول دوره بررسی آن را تغییر دهند. این وضعیت در ابتدا "فعال" است و با گذشت زمان می توان آن را به "در انتظار"، "حل شده" تغییر داد، "رفع نمی‌شود" و "بسته است". در CodeFlow، هر به‌روزرسانی یک «تکرار» نامیده می‌شود و چرخه بررسی دیگری را تشکیل می‌دهد. بنابراین، ممکن است قبل از اینکه تغییر در کد در نهایت در مخزن کد منبع ادغام شود، چندین بار تکرار شود.

همانطور که قبلاً ذکر شد، مطالعه تحقیقاتی در سه مرحله انجام شد که در مرحله اول به تشخیص نظرات مفید و غیر مفید مرور کد بر اساس مصاحبه با توسعه دهندگان کمک کرد. یک مصاحبه انفرادی نیمه ساختاریافته از توسعه دهندگانی که دارای سطوح مختلف تجربه در بررسی کد و توسعه کد از چهار پروژه مختلف مایکروسافت بودند، انجام شد. از مصاحبه شوندگان خواسته شد تا نظرات را از مقیاس ۱ – ۳ (۱– مفید نیست، ۲– تا حدودی مفید و ۳– مفید) ارزیابی کنند. نتایج مصاحبه نشان داد که ٦٩٪ از نظرات مرور یا "مفید" یا "تا حدودی مفید" بودند. نظرات مروری که حاکی از نقص عملکردی بود به عنوان نظرات مفید در نظر گرفته شد. از سوی دیگر، نظراتی که متعلق به دسته‌های: مستندسازی در کد، نمایش تصویری کد (مثلاً خط خالی یا تورفتگی)، سازمان‌دهی کد (مثلاً نحوه تقسیم کارکرد به روش‌ها) و رویکرد راه‌حل در نظر گرفته شد. به عنوان تا حدودی مفید است. همه نظراتی که یا مثبت کاذب بودند (مثلاً به دلیل عدم تخصص زمانی که یک بازبین به اشتباه به مشکلی در کد اشاره می‌کند) یا در هیچ دسته‌ای که قبلاً ذکر شد قرار نمی‌گرفتند، به عنوان نظرات غیر مفید طبقه‌بندی شدند.

در مرحله دوم، نویسندگان با استفاده از یافته‌های به دست آمده از مرحله اول، یک طبقه بندی خودکار ساختند. برای ساخت طبقه‌بندی‌کننده، نویسندگان نظرات بررسی را به صورت دستی به دو دسته مفید و غیر مفید طبقه‌بندی کردند. نظراتی که در مطالعه اکتشافی به‌عنوان تا حدودی مفید طبقه‌بندی شدند، برای این مرحله دوم در دسته مفید قرار گرفتند. بر اساس مصاحبه و تحلیل دستی، ۸ ویژگی بعدی نظرات مشخص شد. بر اساس این ویژگی‌ها و دسته بندی‌ها، یک "مدل درخت تصمیم برای طبقه بندی نظرات مفید" مطابق شکل زیر ساخته شد.



بر اساس گره‌های تصمیم، نظرات به عنوان مفید یا غیر مفید طبقه بندی می‌شوند. به منظور ارزیابی روش پیشنهادی، نویسندگان از نظرات پنج پروژه بزرگ مایکروسافت که شامل Azure، Bing، Visual Studio، Exchange و Office بودند، استفاده کردند. بر اساس نتایج، نویسندگان به این نتیجه رسیدند:

۱. توسعه دهندگانی که در گذشته تغییراتی ایجاد کرده اند یا یک قطعه کد یا یک مصنوع را بررسی کرده اند نظرات مفیدتری ارائه می‌دهند.

۲. تفاوت قابل توجهی در مفید بودن نظرات وجود دارد (یعنی آن دسته از نظراتی که دارای کلماتی مانند "اصلاح"، "اشکال" یا "حذف" هستند به عنوان نظرات "مفید" در نظر گرفته می‌شوند) که توسط بازبینان در همان تیم ارائه می‌شود. و نظرات ارائه شده توسط نویسنده و داور از تیم‌های مختلف.

۳. تعداد نظرات مفید در طول زمان برای چهار پروژه از پنج پروژه افزایش یافت و دلیل این امر افزایش تجربه بازبینان با گذشت زمان در نظر گرفته شد.

در زیر پیامدهای نتایج برای شرکت کنندگان در بررسی کد و همچنین برای محققان آمده است:

۱. این مطالعه نشان داد که تعداد سودمندی نظرات مرور کد با تجربه توسعه دهنده یک کد افزایش یافته است. پایه.

۲. این مطالعه همچنین پیشنهاد کرد که اثربخشی بررسی‌ها با افزایش تعداد پرونده‌ها کاهش می‌یابد. پیشنهاد شد که توسعه دهندگان باید تغییرات کوچک‌تر را با تعداد فایل‌های بیشتر برای بررسی ارسال کنند.

۳. تراکم سودمندی نظرات می‌تواند توسط تیمی از توسعه دهندگان برای شناسایی مناطقی که بررسی کد در آنها مؤثرتر است استفاده شود.

۳,۳,٦ نقد

Sadowski, Soderberg, Church, ¨ et al.[22]یک مطالعه موردی انجام دادند که در آن یک مطالعه اکتشافی در مورد شیوه‌های بازبینی کد مدرن در Google انجام دادند.

مطالعه اکتشافی آنها بر ۳ جنبه بازبینی کد متمرکز بود: ۱) انگیزه‌های محرک بازبینی کد، ۲) شیوه‌های فعلی و ۳) تفسیر توسعه دهندگان بازبینی کد.

به منظور ایجاد ساختار بیشتر در بررسی کد، چندین ابزار در نرم افزار منبع باز ((OSSو تنظیمات صنعتی پدید آمدند. برای این کار، نویسندگان برخی از رویکردهای مرور مبتنی بر ابزار را مطالعه کردند. این ابزارها شامل CodeFlow [11]مورد استفاده مایکروسافت، [23] Gerritمورد استفاده توسط Google's Chromium، ReviewBoard [3]توسعه یافته توسط VMware و [24] Phabricatorمورد استفاده فیس بوک است. در زیر مروری کوتاه بر هر یک از این سیستم‌های توصیه بازنگری کد ارائه شده است.

۱. CodeFlow: CodeFlow وضعیت هر شخص (توسعه‌دهنده یا بازبین) و جایگاه آنها در این فرآیند (یعنی انتظار، بررسی، امضاء) را ردیابی می‌کند. CodeFlow نویسنده را از ارسال هیچ گونه تغییری بدون تایید و همچنین پشتیبانی از چت ها در موضوعات نظرات جلوگیری نکرد.

۲. Gerrit: Google's Chromium از سیستم توصیه بازبینی کد خارجی موجود به نام Gerrit استفاده می‌کند که در آن تغییرات تنها پس از تأیید بازبین‌ها و تأیید خودکار مبنی بر عدم ایجاد شکست در این تغییر در شاخه اصلی ادغام می‌شوند.

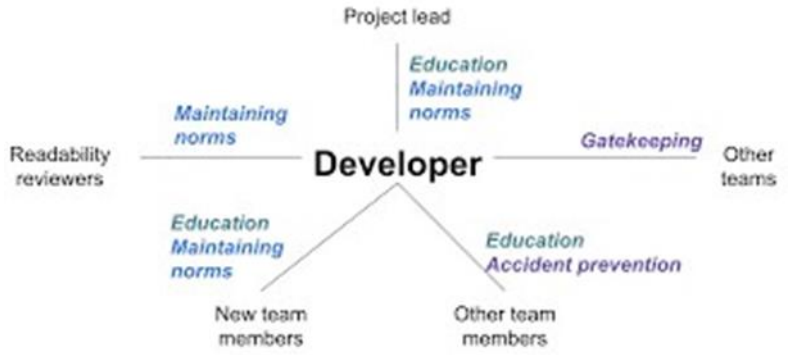
۳. ReviewBoard: ReviewBoard توسط VMware توسعه داده شده است و هدف آن است ادغام تجزیه و تحلیل استاتیک در فرآیند بررسی این ادغام متکی است در مورد تغییراتی که نویسندگان به صورت دستی برای تجزیه و تحلیل درخواست می‌کنند، که منجر به بهبود کیفیت بررسی کد می‌شود.

٤. Phabricator: Phabricator که توسط فیس‌بوک استفاده می‌شود، به یک بازبین اجازه می‌دهد تا تغییری را به عهده بگیرد و خودش آن را انجام دهد. همچنین، سیستم راه‌حلی برای تجزیه و تحلیل استاتیک خودکار یا خطاهای یکپارچه‌سازی مداوم ارائه می‌دهد. برای درک فرآیند بررسی کد در Google، نویسندگان بر دو جنبه اصلی تمرکز کردند: فرآیند بررسی که توسعه‌دهندگان در طول بررسی‌های خاص تجربه می‌کنند و اینکه آیا توسعه‌دهندگان با وجود چالش‌ها از بررسی‌های ارائه شده راضی هستم. برای بررسی کد در گوگل، آنها از CRITIQUE، یک ابزار بررسی کد مبتنی بر وب متمرکز داخلی و توسعه یافته استفاده کردند. در این ابزار، یک بازبین می‌تواند تفاوت برجسته‌شده تغییر پیشنهادی را ببیند و همچنین یک بحث رشته‌ای را روی خطوط کد با توسعه‌دهندگان یا سایر بازبین‌ها شروع کند. CRITIQUE همچنین نمایی از تمام قابلیت‌های گزارش یک توسعه‌دهنده و همچنین تعامل آن با ابزار را ارائه می‌دهد که شامل باز کردن ابزار، ایجاد تغییرات، مشاهده تفاوت و تأیید تغییرات می‌شود. به منظور درک انگیزه توسعه دهندگان برای بررسی کد در Google با استفاده از CRITIQUE و درک توسعه دهندگان در مورد آن، نویسندگان از مصاحبه به عنوان ابزاری برای جمع آوری داده‌ها استفاده کردند.

بر اساس داده‌های جمع آوری شده از مصاحبه‌های انجام شده، یافته‌های زیر بدست آمد.

یافته ۱: بررسی کدهای انجام شده در گوگل نه تنها با هدف تصحیح خطاها یا مشکلات، بلکه برای اطمینان از خوانایی و نگهداری کد که به عنوان یک جنبه آموزشی در نظر گرفته شده است، انجام می‌شود.

یافته ۲: انتظارات در مورد یک بازبینی کد خاص به رابطه مشترک بین توسعه‌دهنده/نویسنده و بازبینی کد بستگی دارد وقتی نوبت به توسعه‌دهنده و سرپرست پروژه و همچنین اعضای جدید تیم می‌رسد، آن‌ها جنبه آموزشی (آموزش یا یادگیری از مرور کد) بررسی کد را به اشتراک می‌گذارند. برای توسعه‌دهندگان و سایر تیم‌ها، آن‌ها جنبه‌ی نگهبانی (ایجاد و حفظ مرزها در اطراف کد منبع) بررسی کد را به اشتراک می‌گذارند. به طور مشابه، برای توسعه‌دهندگان و بازبین‌های خوانایی، آن‌ها هنجارهای حفظ (حفظ قوانین سازمانی مانند قالب‌بندی یا الگوهای استفاده ازAPI)در بررسی کد را به اشتراک می‌گذارند. در نهایت، برای توسعه‌دهنده و سایر اعضای تیم، آنها جنبه پیشگیری تصادفی آموزشی (آموزش اشکالات، نقص‌ها یا سایر مسائل مرتبط با کیفیت) را در بررسی کد به اشتراک می‌گذارند.



شکل ۳,۵: نمودار رابطه ای که موضوعات انتظارات مرور را که عمدتاً در یک زمینه نویسنده/بازبینی خاص ظاهر می‌شوند، توصیف می‌کند[۲۲]

یافته ۳: فرآیند بررسی کد با روش همگرای سبک وزن و انعطاف پذیر بودن آن هماهنگ است. فرآیند بررسی کد به شدت با CRITIQUE ترکیب شده است که به شرح زیر عمل می‌کند:

ایجاد نویسندگان شروع به ایجاد، اضافه کردن یا ویرایش یک کد می‌کنند.

پیش نمایش با کمک CRITIQUE، نویسندگان سپس تفاوت تغییرات و نتایج تحلیلگرهای کد خودکار را مشاهده خواهند کرد.

نظر دادن نویسندگان/بازبینان تفاوت در رابط کاربری CRITIQUE را می‌بینند و سپس در حین رفتن از یک تغییر به تغییر دیگر شروع به اظهار نظر می‌کنند.

پرداختن به بازخورد بر اساس نظرات مراحل قبلی، نویسندگان یا شروع به پاسخ دادن به نظرات می‌کنند یا طبق درخواست‌های ذکر شده در نظرات شروع به ایجاد تغییرات می‌کنند.

تأیید پس از بررسی همه نظرات، بازبینان تغییرات را تأیید می کنند و آن را به عنوان LGTMبه نظرم خوب می رسدعلامت گذاری می کنند.

یافته ٤: بررسی کد در گوگل به نقطه ای رسیده است که در مقایسه با پروژه‌های قدیمی، روند بررسی با تغییرات کوچک‌تر سریع‌تر شده است. همچنین، در مقایسه با دو بررسی کننده برای پروژه‌های قدیمی‌تر، یک داور کافی در نظر گرفته می‌شود.

یافته ۵: علیرغم سال‌ها بهبود، تعدادی از خرابی‌های کدنویسی در Google وجود داشته است که عمدتاً به پیچیدگی تعاملاتی که حول بررسی کدها می‌چرخند مرتبط هستند.

مشاهده شد که در مدت یک هفته، تقریباً ۷۰ درصد تغییرات در کمتر از ٢٤ ساعت پس از ارسال پستی برای بررسی اولیه انجام شد. بر اساس مصاحبه‌ها، همچنین مشاهده شد که توسعه‌دهندگان از نیاز به بازبینی کد راضی بودند، اکثر تغییرات کوچک بودند، بررسی‌ها یک بازبین دارند و هیچ نظری به جز مجوز انجام دادن وجود نداشت. این ویژگی‌ها باعث شده است که روند بررسی کد سریع‌تر و سبک‌تر در مقایسه با سایر پروژه‌هایی که فرآیند مشابهی را اتخاذ می‌کنند، داشته باشد.

۳,۳,۷ CRRS مبتنی بر نمایه

Fejzer، Przymus و [20] Stencelیک سیستم توصیه بازبینی کد مبتنی بر پروفایل را پیشنهاد کردند. در مدل پیشنهادی پیشنهادی، نمایه بازبین شامل تاریخچه بررسی و تعهدات یک بازبین احتمالی است.

در مدل توصیه بازبین آنها، زمانی که یک درخواست commit جدید به مخزن می‌رسد، با نمایش چند مجموعه ای از commitها (مجموعه‌های متعددی از دنباله کلمات موجود در مسیر فایل تغییر یافته در یک (commitو همچنین نمایه های بازبینی کنندگان مقایسه می‌شود.. شباهت بین نمایش چند مجموعه ای از تعهدات و نمایه‌های بازبینان محاسبه می‌شود و n بازبین برتر انتخاب می شوند. در اینجا، به‌روزرسانی نمایه بازبین یکی از مهم‌ترین و پر تکرارترین عملیات است. هر زمان که نظر جدیدی توسط یک بازبین ارائه شود، commit به نمایه وی اضافه می‌شود. همچنین، وقتی صحبت از مشخصات بالقوه یک بازبین می‌شود، زمان یکی از عوامل مهمی است که باید در نظر گرفته شود. کاندیدایی که بررسی‌ها یا تعهدات اخیر بیشتری در نمایه خود دارد، به عنوان کاندیدای محتمل تری برای بررسی درخواست تعهد در نظر گرفته می‌شود.

نویسندگان یک ارزیابی تجربی از روش پیشنهادی خود با استفاده از Android، LibreOffice، OpenStack و Qt انجام دادند. نتایج تجربی به شرح زیر بود:

۱. تعداد نظرات به ازای هر یک بازبین: اکثر بازبینان کمتر از ۲۰ نظر برای Android و LibreOffice و کمتر از ۶۰ نظر برای OpenStack و Qt ایجاد کردند.

۲. مدت زمان فعالیت بازبینان فردی: در مورد Android و LibreOffice، بازبینان در مقایسه با بازبینان برای Qt و OpenStack زمان بیشتری را صرف کردند. دلیل احتمالی پشت این نتیجه، نگهبانان منتخب در نظر گرفته شد که برای شرکت‌های مشارکت کننده در این پروژه‌ها کار می‌کنند.

۳. مدت زمان بررسی‌های فردی: اکثر بررسی‌ها در عرض سه روز برای پروژه‌های LibreOffice و OpenStack، حداکثر تا دو روز برای Qt و حداکثر تا شش روز برای پروژه‌های Android تکمیل شدند.

۳,۳,۸ طبقه بندی سیستم‌ها

طبقه‌بندی سیستم‌ها بر اساس داده‌های مورد استفاده برای ایجاد توصیه بازنگری کد انجام شد. داده‌ها شامل مسیرهای فایل مشابه، تاریخچه بررسی کد، تعهدات و تجربه فناوری است.

منبع اطلاعات	نام سیستم
تاریخچه مرور کد	cHRev
شباهت مسیر فایل	REVFINDER
وضعیت هر منتقد یا نویسنده	CodeFlow

تغییرات پس از تأیید صریح داوران ادغام می‌شوند	گریت
تجزیه و تحلیل استاتیک را در فرایند بررسی ادغام می‌کند	ReviewBoard
تجزیه‌وتحلیل استاتیک خودکار یا پیوسته ساخت/تست ادغام	سازنده
پروژه متقابل مرتبط و تجربه فناوری	درست
استخراج متن و محل فایل	TIE
مشخصات مرورگر کد	CRRSمبتنی بر مشخصات

جدول ۳,۳: منابع داده برای توصیه‌های بررسی کد در سیستم‌ها

جدای از منبع داده به عنوان یکی از عوامل طبقه بندی مقالات پژوهشی، نوع پروژه ای که این سیستم‌های توصیه بازنگری کد بر روی آن آزمایش می‌شوند را می‌توان به عنوان یکی دیگر از عوامل طبقه بندی مقالات در نظر گرفت. این پروژه‌ها شامل پروژه‌های متن باز و پروژه‌های تجاری است.

نوع پروژه	نام سیستم
پروژه های منبع باز	REVFINDER، TIE و بر اساس نمایه CRRS
پروژه های تجاری	CodeFlow
منبع باز و پروژه‌های تجاری	درست و cHRev،
بدون پروژه (مصاحبه)	نقد

۱. پروژه های منبع باز: در زیر لیستی از سیستم هایی است که فقط در پروژه های منبع باز ارزیابی شده اند.
برای هدف ارزیابی، RevFinder، TIE و CRRS مبتنی بر پروفایل از ٤٢٠٤٥ بررسی پروژه‌های منبع باز استفاده کردند که شامل پروژه منبع باز Android (AOSP)، OpenStack، Qt و LibreOffice می‌شد. دلایل زیادی پشت انتخاب این سیستم‌ها وجود داشت. ابتدا این سیستم‌ها از سیستم Gerrit به عنوان ابزار بررسی کد استفاده می کنند. دوم، این سیستم‌ها پروژه‌های نرم افزاری فعال، بزرگ و واقعی هستند. در نهایت، هر یک از این سیستم‌ها یک سیستم بررسی کد خوب را حفظ می‌کنند که به ایجاد مجموعه داده‌های اوراکل خوب برای ارزیابی سیستم توصیه‌گر کمک می‌کند. در زیر به نتایجی اشاره شده است که ما از طریق آزمایش به دست آوردیم: (الف) RevFinder [12]به ۱۰ دقت بالا دست یافت (دقت Top-k درصد بررسی هایی را محاسبه می‌کند که یک رویکرد می‌تواند به درستی بازبینان کد را نسبت به تعداد کل بررسی‌ها توصیه کند) ٨٦٪، ٨٧٪، ٦٩٪ و ٧٤٪ برای Android، OpenStack، Qt و LibreOffice به ترتیب. به طور متوسط، برای ٧٩٪ از بررسی‌ها، RevFinder در نهایت به بازبینان کد CoRReCT با ١٠ توصیه برتر توصیه می‌کند.
(ب) مشاهده شد که به طور متوسط در ٤ پروژه TIE [13]به برتری دست یافت
دقت پیش بینی ١، top-3، top-5 و top-10 و مقادیر MRR 0.52، ٠,٧٣، ٠,٧٩، ٠,٨٥ و ٠,٦٤ که به ترتیب ٦١، ٣٣، ٢٣، ٨ و ٣٧ درصد بهتر از نتایج RevFinder بودند.
(ج) مشابه TIE و RevFinder، CRRS مبتنی بر پروفایل [٢٠] با استفاده از ٤ سیستم منبع باز یعنی Android، OpenStack، Qt و LibreOffice آزمایش شد. مشاهده شد که برای LibreOffice و OpenStack، اکثر بررسی‌ها در عرض سه روز، برای اندروید تا ٦ روز و برای Qt تا ٢ روز طول می‌کشد.
٢. پروژه‌های تجاری: CodeFlow [11]با استفاده از پنج پروژه مایکروسافت شامل Azure، Bing، Visual Studio، Exchange و Office آزمایش شد. مشاهده شد که افزایش از ٦٠٪ به ٦٦٪ در نظرات مفید دریافت شده از بازبینان در Azure، ٦٢٪ به٦٧٪ در Bing، ٦٠٪ تا ٧٠٪ در Visual Studio، ٦٠٪ تا ٦٨٪ در Office و ٦٠٪ تا ٦٥٪ در Exchange.
٣.پروژه های منبع باز و تجاری: سیستم هایی وجود دارند که هم با استفاده از پروژه های تجاری و هم پروژه های متن باز آزمایش شده اند که در زیر با نتایج به دست آمده از آزمایش های انجام شده به آنها اشاره می شود.
(الف) CorreCT [16]با استفاده از ١٧١١٥ درخواست کششی از ده پروژه تجاری و شش پروژه منبع باز آزمایش شد. معیارهای عملکردی که نویسندگان در اینجا استفاده کرده اند عبارتند از: Top-K Accuracy، میانگین رتبه متقابل (MRR)، میانگین دقت (MP) و میانگین فراخوان (MR).
• هنگامی که روی پروژه‌های منبع باز آزمایش شد، مشاهده شد که CoRReCT دارای دقت Top-k 85.20٪ است در حالی که برای پروژه‌های تجاری دقت Top-K 92.15٪ به دست آمده است.
• CorreCT نتیجه ٨٥,٩٣٪ دقت برای پروژه‌های تجاری و دقت ٨٤,٧٦٪ برای پروژه‌های منبع باز را به دست آورد.
• برای پروژه‌های تجاری CoRReCT فراخوانی ٨١,٣٩٪ را برگرداند در حالی که برای پروژه منبع باز سیستم به ٧٨,٧٣٪ فراخوان دست یافت.

• CORReCT مقدار MRR 0.62 را برای تجاری به دست آورد پروژه‌ها در حالی که برای پروژه‌های منبع باز، نویسندگان ارزش MRR را ذکر نکرده‌اند، اما نسبتاً بالاتر بود.

(ب) cHRev [17]در ۳ پروژه منبع باز (Android، Mylyn و Eclipse)و یک پروژه تجاری (MS Office) مورد ارزیابی قرار گرفت. مشاهده شد که دستاوردهای فراخوانی و دقت به‌دست‌آمده برای MS Office بهتر از دستاوردهای Android، Mylyn و Eclipse برای cHRev بود.

٤. بدون پروژه (مصاحبه)

CRITIQUE [22]که به عنوان یک سیستم توصیه بازبینی کد در گوگل استفاده می‌شود، از حالت مصاحبه برای ارزیابی سیستم خود استفاده می کند. مشاهده شد که توسعه دهندگان به طور متوسط ۲,٦ ساعت در هفته را صرف بررسی تغییرات کردند که در مقایسه با ٦,٤ ساعت در هفته زمان گزارش شده توسط خود پروژه‌های منبع باز، کم بود.

۳,٤ خلاصه

بر اساس مطالعه مرور ادبیات انجام شده، ما هفت سیستم توصیه بازبین کدنویس را پیدا کردیم: cHREV، CoRReCT، CRRS مبتنی بر پروفایل، RevFinder، CodeFlow، TIE و .CRITIQUEاین سیستم ها بر اساس دو بعد تقسیم می شوند: منبع داده مورد استفاده برای ساخت سیستم و نوع پروژه مورد استفاده برای ارزیابی سیستم.

فصل ٤

نیازهای اطلاعاتی بازبینان کد

٤,١ روش شناسی

به منظور انجام نظرسنجی از مهندسان نرم افزار برای تعیین نیازهای اطلاعاتی برای بازبینان کد، از مراحل زیر استفاده کردیم تا اطمینان حاصل کنیم که نتایج صحیح، غیر جانبدارانه و دقیق به دست آمده است.

٤,١,١ بررسی غربالگری

نظرسنجی ما برای مهندسین نرم افزار به دو بخش تقسیم می‌شود که بخش اول یک نظرسنجی غربالگری است. ما از نظرسنجی غربالگری استفاده کردیم تا مطمئن شویم از اعضای توسعه محصول نرم افزاری که تجربه کار با سیستم‌های توصیه بازبین کد دارند و بنابراین می‌توانند اطلاعات دقیق و بی طرفانه ارائه دهند، پاسخ دریافت می‌کنیم. در زیر سؤالاتی وجود دارد که برای بررسی غربالگری خود قرار داده ایم.

١-لطفاً آدرس ایمیل خود را وارد کنید.
2 - چند سال/سال تجربه توسعه نرم‌افزار دارید؟
الف) کمتر از ١ سال
ب) ١-٢ سال/ثانیه
ج) ٣-٥ سال
د) ٦-١٠ سال
ه) ١١+ سال
3-آیا شما ٢٠ سال یا بیشتر هستید و قادر به ارائه رضایت آگاهانه هستید؟
الف) بله
ب) خیر
4-چند سال تجربه در استفاده از سیستم/های توصیه مرورگر کد دارید؟
الف) کمتر از ١ سال
ب) ١-٢ سال/ثانیه
ج) ٣-٥ سال
د) ٦-١٠ سال
ه) ١١+ سال
5-با کدام یک از سیستم‌های توصیه کننده مرورگر کد ((CRRSزیر آشنا هستید؟ (سوال چند پاسخ)
الف) سیستم بازبینی کد Gerrit (Chromium)
ب) GitHub/GitLab
ج) ابزار مرور کد جریان (مایکروسافت)
د) صفحه بازبینی(VMware)
ه) سازنده
و) سطل بیت
ز) دیگر
6-از کدام CRRS استفاده کرده‌اید؟
7-اگر از CRRS استفاده کرده‌اید که در لیست نیست، لطفاً نام یا توضیحات سیستم را ذکر کنید؟

۴-۱-۲ سؤالات نظرسنجی سیستم‌ها و ابزارهای توصیه توصیف‌کننده اطلاعات جمعیتی و ابزار

یک نظرسنجی دموگرافیک و تجربه CRRS برای افرادی که مرحله غربالگری را گذرانده بودند، انجام شد، یعنی شرکت کنندگانی که حداقل دو سال تجربه کاری داشته و تجربه استفاده از CRRS را دارند. این سؤالات به ما کمک کرد تا نیازهای اطلاعاتی بازبینان کد را درک کنیم، آنها چه ویژگی‌هایی را در سیستم‌های توصیه بازبین کد مهم می دانند و چه ویژگی‌هایی را در سیستم‌های موجود گم کرده اند.

1.وظیفه شغلی شما صرف نظر از سطح موقعیت در سازمان شما چیست؟

الف) توسعه‌دهنده/برنامه‌نویس/مهندس نرم‌افزار

ب) سرپرستی تیم

ج) DevOps Engineer/Developer Infrastructure

د) معمار

ه) توسعه دهنده UI/UX

(و) پشتیبانی فنی

(ز) تحلیلگر داده/دانشمند داده/مهندس داده

۲. گروه سنی شما چیست؟

(الف) ۲۵-۲۰

(ب) ۳۵-۲۶

(ج) ۴۵-۳۶

(د) ۵۵-۴۶

(ه) ۶۰-۵۶

(و) بالای ۶۰

۳. موقعیت جغرافیایی شما چیست؟

الف) اروپا

ب) آفریقا

ج) آمریکای جنوبی

(د) آمریکای شمالی

ه) آسیا

(و) استرالیا

(ز) نیوزلند

ح) جزایر اقیانوس آرام

۴. تیم پروژه شما در چه اندازه‌ای است؟

الف) من به‌صورت جداگانه روی پروژه‌هایم کار می‌کنم

(ب) ۷-۲ نفر

(ج) ۱۲-۸ نفر

(د) ۲۰-۱۳ نفر

(ه) ۴۰-۲۱ نفر

(و) بیش از ۴۰ نفر

۵. آیا تیم شما در سراسر جهان توزیع شده یا در یک مکان مشترک قرار دارد؟

(الف) توزیع شده

(ب) در محل مشترک

(ج) هر دو

۶. با کدام یک از سیستم‌های توصیه کننده مرورگر کد ((CRRSزیر آشنا هستید؟ (س answerال چند پاسخ)

(الف) سیستم بازبینی کد Gerrit (Chromium)

(ب) GitHub/GitLab

(ج) ابزار مرور کد جریان (مایکروسافت)

(د) صفحه بازبینی ((VMware

ه) سازنده

و) سطل بیت

(ز) دیگر

۷. از کدام CRRS استفاده کرده‌اید؟

۸. اگر از CRRS استفاده کرده‌اید که در لیست نیست، لطفاً نام یا توضیحات سیستم را ذکر کنید؟

۹. کدام ویژگی‌های موجود در CRRS فوق مفید بوده است؟ (س answerال چند پاسخ)

(الف) بررسی کد را از قبل مرتکب شوید

(ب) بحث کد با نسخه‌های قدیمی و جدید که برای نشان‌دادن تغییر کد مشخص شده است

(ج) پیشنهاد بهبود کد توسط مرورگر کد (غیر از فقط اشاره به خطاهای کد)

(د) اولویت‌بندی تغییرات بر اساس میزان اهمیت آن و تأثیر آن بر عملکرد نرم‌افزار

ه) ادغام نرم‌افزارهای ردیابی پروژه (مانند Trello، JIRA و غیره).

(و) ادغام ویرایشگر کد منبع (مانند Visual Studio، Atom و غیره).

(ز) ادغام بستر ارتباطات تجاری (مانند (Slack

۱۰. موارد زیر لیستی از معیارهایی است که می‌توان برای انتخاب مرورگر کد استفاده کرد. لطفاً نشان دهید که چقدر معتقدید که آنها در انتخاب مرورگر کد اهمیت دارند. (مقیاس لیکرت: بسیار محتمل، تا حدی محتمل، نه محتمل و نه بعید، تا حدودی بعید، بسیار بعید)

الف) تعدادی سال سابقه کار

(ب) تخصص مرورگر کد در زبان برنامه‌نویسی

(ج) تخصص مرورگر کد در یک حوزه (مانند مهندسی نرم‌افزار، هوش مصنوعی و غیره)

(د) زبان ارتباط بین مرورگر کد و توسعه‌دهنده نرم‌افزار

(ه) نقش مرورگر کد

(و) تعداد پروژه‌هایی که روی آنها کار شده است

(ز) تعداد بررسی کد انجام شده است

۱۱. چه معیارهایی در لیست بالا وجود نداشت؟

چه اهمیتی به آنها می‌دهید؟

۱۲. کدام یک از ویژگی‌های رابط کاربری ((UIزیر باعث می شود تجربه کاربری ((UXتعاملی تر ، نزدیک تر و راحت‌تر باشد؟ (س answerال چند پاسخ)

الف) وجود داشبورد برای همه که داده‌های آماری کلیه اقدامات انجام شده را نشان می‌دهد مانند به‌عنوان تعداد تعهدات، تعداد بررسی کد انجام شده، تعداد خطاها/هشدارهای کد در پروژه فعلی و غیره

(ب) گزینه‌ای برای انتخاب یک "شاخه/پرونده" خاص در یک پروژه برای حفظ گردشکار سیستماتیک و یک روش بازبینی کد سازمان‌یافته
(ج) ارائه خط لوله‌ای که نشان می‌دهد پروژه در کدام مرحله است یعنی ساخت، آزمایش، بررسی کد، استقرار و غیره.
(د) تشخیص کد را با استفاده از طرح رنگی با نام برنامه‌نویس/برنامه‌ها منتقل کرد.
(ه) هنگام تغییر کد، بحث کد جدید و قدیمی کدگذاری می‌شود.
۱۳. آیا فکر می‌کنید که زمینه تخصص بازبینی کننده کد حتی در صورتی که داور تجربه یا دانش کمی در زمینه‌ای داشته باشد که از او خواسته شود کد را مرور کند، مهم است؟
۱۴- در مرور کد به دنبال چه موارد خاصی هستید (مانند تعداد سال‌ها کار)

تجربه، زمینه تخصص و غیره؟
۱۵. در کدام مرحله از گردشکار خود ترجیح می‌دهید توصیه مرور کد را داشته باشید؟
(الف) قبل از ادغام درگیری
(ب) پس از ادغام درگیری‌ها
(ج) مهم نیست که کد در چه مرحله‌ای از گردشکار انجام می‌شود
۱۶. چه نوع بررسی کد را از کد زیر ترجیح می‌دهید
ذیل؟
(الف) تعداد بیشتری از بررسی‌های کوچک کد
(ب) مرور طولانی کد
(ج) مهم نیست (بسته به نوع آن پروژه)

٤,٢ نتایج

ما ۲۷ پرس و جو در مورد نظرسنجی خود به دست آوردیم، اما تنها ۱۵ مورد از آنها پیش رفتند و به نظرسنجی غربالگری ما پاسخ دادند. از این ۱۵ پاسخ، ما ٤ پاسخ را که حداقل معیارها را برآورده نمی‌کردند، با استفاده از بررسی غربالگری فیلتر کردیم. در زیر نتایج به‌دست‌آمده برای نظرسنجی استفاده از ابزار/سیستم‌ها و توصیه‌های بازبینی کننده کد جمعیتی و کد ارائه شده است.

۱. نقش شغلی شرکت کنندگان

بر اساس نتایج به‌دست‌آمده مشاهده شد که اکثر شرکت‌کنندگان یا توسعه‌دهنده، یا مهندس نرم‌افزار یا برنامه‌نویس بودند.

درصد	رشته
72.73	توسعه دهنده/برنامه نویس/مهندس نرم افزار
9.09	سرپرستی تیم
9.09	مهندس/DevOps مهندس زیرساخت
9.09	مالک محصول

جدول ۴,۱: نقش‌های شغلی شرکت‌کنندگان

همانطور که در جدول ۴,۱ مشاهده می‌شود، ٪۷۲,۷۳ توسعه دهندگان / برنامه نویسان / مهندسان نرم افزار بودند در حالی که ٪۹,۰۹ برای هر یک از تیم‌ها، مهندس DevOps/توسعه دهنده زیرساخت و صاحب محصول وجود داشت.

۲. موقعیت جغرافیایی شرکت کنندگان

همانطور که در جدول ۴,۲ نشان داده شده است، اکثر شرکت کنندگان ما از آسیا بودند و درصد معادل باقیمانده از آمریکای شمالی و آمریکای جنوبی بودند.

جدول ۴,۲: موقعیت جغرافیایی شرکت کنندگان

درصد	رشته
9.09	آمریکای جنوبی
9.09	شمالی عامری CA
81.82	آسیا

۳. اندازه تیم پروژه

از کل شرکت‌کنندگانی که در مطالعه ما شرکت کردند، ۴۵,۴۵ درصد از آنها به ترتیب در یک تیم ۲-۷ نفره و ۸-۱۲ نفر کار کرده اند، در حالی که ۹,۰۹ درصد از افراد در گروهی بیش از بیش از ۴۰ نفر همانطور که در جدول ۴,۳ نشان داده شده است.

درصد	رشته
45.45	2-7نفر
45.45	8-12نفر
9.09	بیش از ۴۰ نفر

جدول ۴,۳: اندازه تیم پروژه

٤. توزیع تیم

ما همچنین اطلاعاتی در مورد نحوه توزیع تیم‌ها جمع‌آوری کردیم، به این معنی که آیا آنها در تیم‌هایی کار می‌کنند که هم‌اکنون یا توزیع شده‌اند. مشاهده شد که ٤٥,٤٥ درصد از تیم‌ها در محل مشترک، ۱۸,۱۸ درصد توزیع شده و ۳٦,۳٦ درصد ترکیبی از هر دو محل مشترک و توزیع شده بودند.

درصد	رشته
45.45	در محل مشترک
٪18.18	توزیع شده است
36.36	هر دو

جدول ۴,۴: توزیع تیم

۵. آشنایی با CRRS در میان شرکت کنندگان

اکثر شرکت کنندگانی که در نظرسنجی ما شرکت کردند با GitHub/GitLab آشنا بودند. از طرف دیگر، هیچکس با سیستم بازبینی کد Gerrit (Chromium) و ReviewBoard آشنایی نداشت. مشاهده شد که ٤٧,٦٢ درصد از شرکت کنندگان با GitHub/GitLab، ٢٣,٨١ درصد از شرکت کنندگان با BitBucket، ١٤,٢٩ درصد از شرکت کنندگان با) Code Flow Review Toolمایکروسافت)، ۹,۵۲ درصد با سایر ابزارهای بررسی آشنا بودند. (که شامل SVN می‌شود) و ٤,٧٦ درصد از شرکت کنندگان با Phabricator آشنا بودند. در زیر نمودار خطی است که توزیع آشنایی با CRRS را در بین شرکت کنندگان نشان می‌دهد.

درصد انتخاب ها	CRRS
٪0	سیستم بازبینی کدGerrit (Chromium)
47.62	GitHub/GitLab
14.29	ابزار مرور کد جریان (مایکروسافت)
٪0	تابلوی بررسی(VMware)
4.76	سازنده
23.81	سطل بیت
9.52	دیگر

جدول ۴,۵: آشنایی با CRRS در بین شرکت‌کنندگان

٦. سودمندی از ویژگی‌های CRRS

ما تعدادی از ویژگی‌های CRRS را برای شرکت‌کنندگان فهرست کردیم و از آن‌ها خواستیم همه آن ویژگی‌هایی را که برایشان مفید است انتخاب کنند. موارد زیر ویژگی‌هایی است که با درصد شرکت‌کنندگانی که آن را مفید می‌دانستند، برای شرکت‌کنندگان ارائه کرده‌ایم.

(الف) بحث کد با برجسته شدن نسخه‌های قدیمی و جدید برای نشان دادن تغییر در کد: ۲۵,۵۳٪ از شرکت کنندگان آن را مفید دانستند.

(ب) ادغام نرم افزار ردیابی پروژه (مانند Trello، Jira و غیره): ۲۰,۵۹٪ از شرکت کنندگان آن را مفید دانستند.

(ج) بررسی کد قبل از انجام تعهد: ١٤,٧١٪ از شرکت کنندگان آن را مفید دانستند.

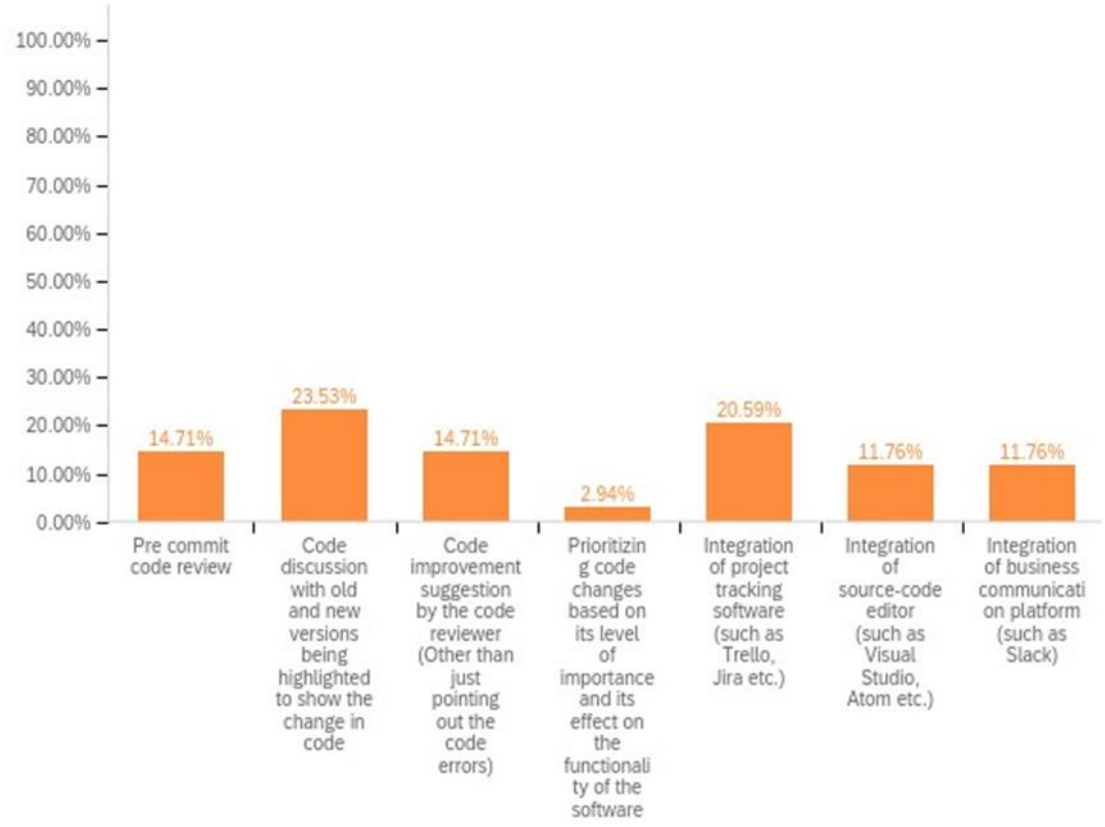
(د) پیشنهاد بهبود کد توسط بازبین کد (غیر از اشاره صرف به اشتباهات کد): ٪١٤,٧١ از شرکت کنندگان پیدا کردند مفید است

(ه) ادغام پلت فرم ارتباط تجاری (مانند Slack): 11.76٪ از شرکت کنندگان آن را مفید دانستند.

(و) ادغام ویرایشگر کد منبع (مانند ویژوال استودیو، اتم و غیره):

١١,٧٦ درصد از شرکت کنندگان آن را مفید دانستند.

(ز) اولویت بندی تغییرات کد بر اساس سطح اهمیت و تأثیر آن بر عملکرد نرم افزار: ۲,۹٤٪ از شرکت کنندگان آن را مفید می‌دانند.



شکل ۴,۱: سودمندی گزارش شده از ویژگی‌های CRRS

۷. معیارهای انتخاب بازنگری کد

تعدادی از عوامل وجود دارد که باید در هنگام انتخاب یک بازنگری کد در نظر گرفته شوند. ما چندین مورد از آنها را فهرست کردیم که شرکت کنندگان اهمیت هر یک از آنها را انتخاب کردند. شرکت‌کنندگان اهمیت هر یک از این ویژگی‌ها را در مقیاس لیکرت از بسیار محتمل تا بسیار بعید، همانطور که در زیر نشان داده شده است، مشخص کردند.

#	Field	Extremely likely	Somewhat likely	Neither likely nor unlikely	Somewhat unlikely	Extremely unlikely	Total
1	Number of years of work experience	9.09%	72.73%	9.09%	9.09%	0.00%	11
2	Code reviewers expertise in programming language	54.55%	36.36%	9.09%	0.00%	0.00%	11
3	Code reviewer's expertise in a domain (eg> software engineering, artificial intelligence etc.)	45.45%	45.45%	9.09%	0.00%	0.00%	11
4	Language of communication between the code reviewer and software developer	54.55%	18.18%	27.27%	0.00%	0.00%	11
5	Role of the code reviewer	18.18%	45.45%	36.36%	0.00%	0.00%	11
6	Count of projects worked on	9.09%	63.64%	0.00%	27.27%	0.00%	11
7	Count of code reviews done	27.27%	45.45%	9.09%	18.18%	0.00%	11

Showing rows 1 - 7 of 7

شکل ۴,۲: معیارهای انتخاب مرورگر کد

الف) تعداد سال سابقه کار

• از کل شرکت کنندگان ۹,۰۹٪ این ویژگی را بسیار محتمل، ۷۲,۷۳٪ تا حدودی محتمل، ۹,۰۹٪ آن را نه محتمل و نه بعید، ۹,۰۹٪ آنها را تا حدودی بعید می‌دانستند در حالی که هیچکس آن را تا حدودی محتمل تلقی نمی‌کرد. بسیار بعید است.

(ب) تخصص بازبینان کد در زبان برنامه نویسی

• مشاهده شد که ٥٤,٥٥ درصد از شرکت کنندگان این ویژگی را بسیار محتمل، ٣٦,٣٦ درصد از آنها تا حدودی محتمل، ۹,۰۹ درصد از آنها نه محتمل و نه بعید می‌دانستند در حالی که هیچکس آن را تا حدودی بعید یا بسیار بعید تلقی می‌کردند.

(ج) تخصص بازبین کد در یک حوزه (مانند: مهندسی نرم افزار، هوش مصنوعی و غیره)

• برای این معیار، ٤٥,٤٥ درصد از شرکت کنندگان این ویژگی را بسیار محتمل، ٤٥,٤٥ درصد از آنها تا حدودی محتمل، ۹,۰۹ درصد از آنها نه محتمل و نه بعید در نظر گرفتند، در حالی که هیچکس آن را تا حدودی بعید یا بسیار بعید تلقی کردند.

(د) زبان ارتباط بین بازبینی کننده کد و توسعه دهنده نرم افزار

• از کل شرکت کنندگان، ٥٤,٥٥ درصد از آنها این ویژگی را بسیار محتمل، ۱۸,۱۸ درصد از آنها تا حدودی محتمل، ۲۷,۲۷ درصد از آنها نه محتمل و نه بعید می‌دانستند، در حالی که هیچکس آن را تا حدودی بعید یا بسیار بعید تلقی می‌کردند.

(ه) نقش بازبینی کد

• برای این معیار، ۱۸,۱۸ درصد از شرکت کنندگان این ویژگی را بسیار محتمل، ٤٥,٤٥ درصد از آنها تا حدودی محتمل، ٣٦,٣٦ درصد از آنها آن را نه محتمل و نه بعید در نظر گرفتند، در حالی که هیچکس آن را تا حدودی بعید یا بسیار بعید تلقی کردند.

(و) تعداد پروژه‌های انجام شده

• همچنین مشاهده شد که ۹,۰۹ درصد از شرکت کنندگان این ویژگی را بسیار محتمل، ٦٣,٦٤ درصد از آنها تا حدودی محتمل، ۲۷,۲۷ درصد آنها را تا حدودی بعید و هیچکس آن را بسیار بعید و نه محتمل و نه بعید می‌دانستند.

(ز) تعداد بررسی‌های کد انجام شده

• برای این معیار، ۲۷,۲۷ درصد از شرکت کنندگان این ویژگی را بسیار محتمل، ٤٥,٤٥ درصد آن را تا حدودی محتمل، ۹,۰۹ درصد آن را تا حدودی بعید، ۱۸,۱۸ درصد از آنها را نه محتمل و نه بعید می‌دانستند در حالی که هیچ کس آن را در نظر نمی‌گرفت. بسیار بعید است.

ما همچنین از شرکت‌کنندگان خواستیم تا ویژگی‌هایی غیر از موارد ارائه شده را که در هنگام انتخاب بازنگری کد در نظر گرفته‌اند، ارائه دهند. ویژگی‌های گزارش شده عبارتند از:

• اطمینان از اینکه بازبین خود را با تکنولوژی و زبان برنامه نویسی به روز نگه می‌دارد (شرکت کننده این ویژگی را بسیار محتمل در نظر گرفت).

• تعداد درخواست‌های کششی که به بازبینان اختصاص داده شده است.

۸. رابط کاربری :CRRS (UI)

جدای از ویژگی‌های CRRS، ما همچنین یک سؤال در مورد ویژگی‌های مختلف UI پرسیدیم که می‌توان با تعاملی‌تر، قابل دسترس‌تر و راحت‌تر کردن تجربه کاربر ((UXبرای یک CRRS مفید بود. درصد افرادی که ویژگی‌های زیر را مفید می‌دانند با توزیع نمودار دایره‌ای که در زیر نشان داده شده است، به شرح زیر است:

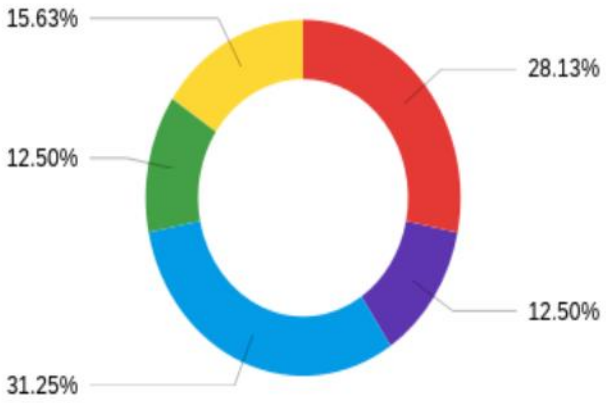
(الف) ارائه خط لوله ای که نشان می‌دهد پروژه در کدام مرحله است یعنی ساخت، آزمایش، بررسی کد، استقرار و غیره: ۳۱,۲۵٪ از شرکت کنندگان این ویژگی UI را برای پیگیری پروژه مفید و راحت‌تر می‌دانستند.

(ب) وجود داشبورد برای هر فرد که داده‌های آماری تمامی اقدامات انجام شده را نشان می‌دهد (مانند تعداد commit ها، تعداد بررسی‌های کد انجام‌شده، تعداد خطاها/اخطارهای کد در پروژه فعلی و غیره): ۲۸,۱۳٪ از شرکت‌کنندگان این ویژگی برای جستجوی جزئیات و اقدامات انجام شده توسط هر فرد مفید است.

(ج) بحث کدهای رنگی جدید و قدیمی در صورت تغییر در کد: ۱۵,۶۳ درصد از شرکت‌کنندگان این ویژگی را مفید دانستند که فرآیند بررسی کد را برای بازبین و توسعه‌دهنده آسان‌تر می‌کند.

(د) تشخیص کد جابجا شده با استفاده از طرح کد رنگی با نام توسعه‌دهنده: ۱۲,۵۰٪ از شرکت‌کنندگان این ویژگی را مفید دانستند.

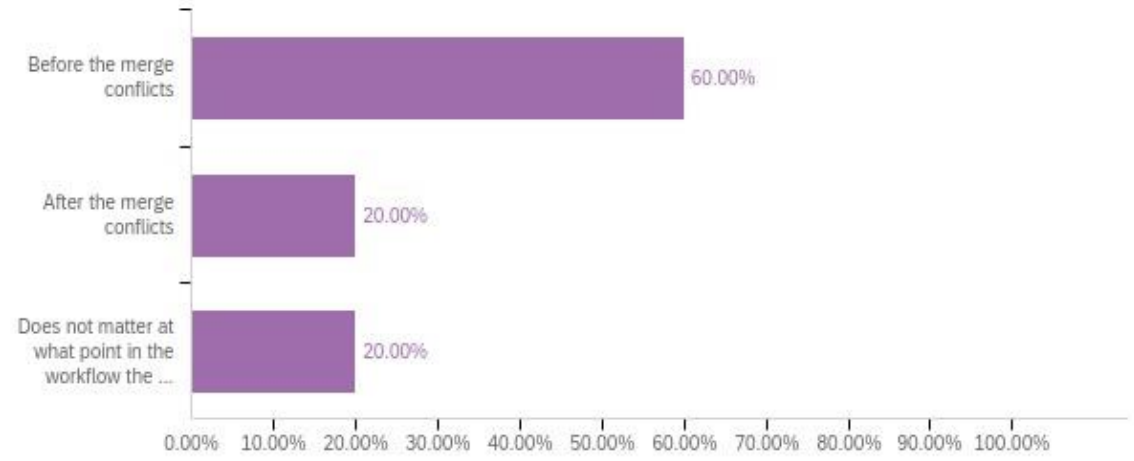
(ه) گزینه ای برای انتخاب یک «شاخه/فایل» خاص در یک پروژه برای حفظ یک گردش کار سیستماتیک و یک روند بررسی کد سازماندهی شده: ۱۲,۵۰٪ از شرکت کنندگان این ویژگی را با سازماندهی بیشتر فرآیند بررسی کد مفید دانستند.



شکل ۴,۳: ویژگی‌های UI CRRS

۹. ترجیح زمان داشتن توصیه بازبینی کد:

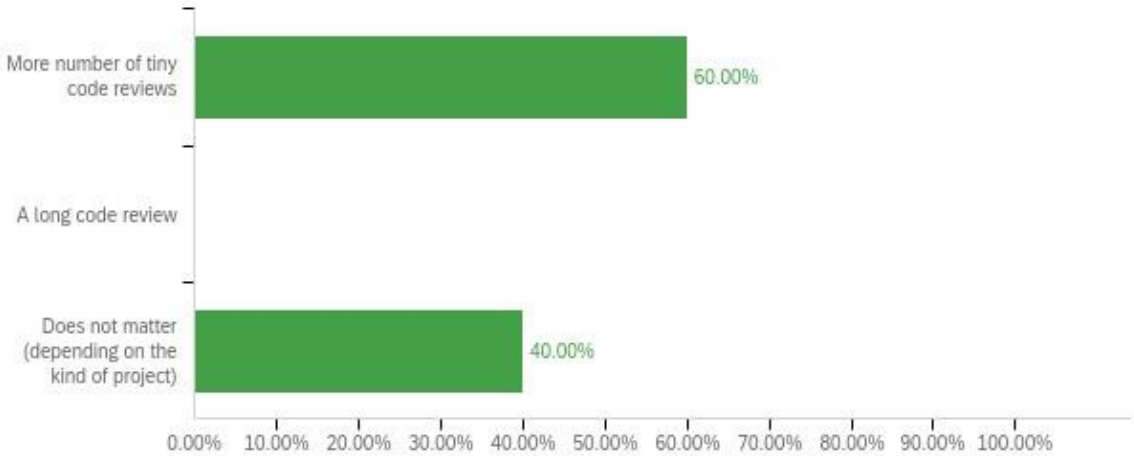
یک توصیه بازبینی کد را می‌توان در مراحل مختلف فرآیند توسعه نرم افزار ارائه کرد که شامل قبل از تضادهای ادغام، پس از تضادهای ادغام یا در هر نقطه از گردش کار می‌شود. مشاهده شد که ۶۰٪ از شرکت کنندگان معتقد بودند که توصیه کد باید قبل از تداخل ادغام انجام شود، ۲۰,۰۰٪ از آنها فکر می‌کردند که باید بعد از تضادهای ادغام انجام شود، در حالی که برای ۲۰,۰۰٪ از آنها اهمیتی نداشت در چه نقطه ای از ادغام. گردش کار بررسی کد انجام شده است.



شکل ۴,۴: اولویت زمانی که توصیه بازبینی کد وجود دارد

۱۰. نوع بررسی کد

از شرکت کنندگان پرسیده شد که چه نوع بازبینی کدی را ترجیح می‌دهند که شامل بازبینی کدهای کوچک‌تر یا بازبینی کد طولانی است یا به نوع پروژه بستگی دارد. مشاهده شد که ٦٠,٠٠٪ از شرکت کنندگان ترجیح می‌دهند بسیاری از بررسی‌های کد کوچک‌تر را انجام دهند در حالی که بقیه (٤٠,٠٠٪) از شرکت کنندگان احساس می‌کردند که این کار مهم نیست. هیچ یک از شرکت کنندگان ترجیح ندادند کدهای طولانی را بررسی کنند.



٤,٣ برخی از روندها و الگوهای مشاهده شده

بر اساس نتایج به‌دست‌آمده، الگوها و روندهایی بین نتایج دو یا چند سؤال نظرسنجی یافتیم. برخی از روندهایی که مشاهده کردیم به شرح زیر است:

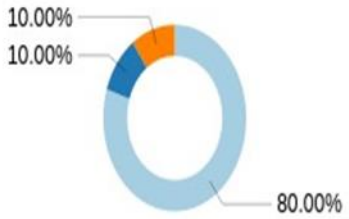
(الف) نوع CRRS مورد استفاده و نقش شغلی

بر اساس مشاهدات ما، مشاهده شد که توسعه دهنده تقریباً از تمام سیستم‌های CRRS که در سؤال ذکر کرده بودیم آگاه بود در حالی که مهندس DevOps فقط از یک CRRS آگاه بود، سرپرست تیم از ۲ مورد از CRRSها و محصول آگاه بود. مالک ٤ مورد از CRRSها را می‌دانست.

Gerrit Code Review System (Chromium)

NO DATA

GitHub/ GitLab

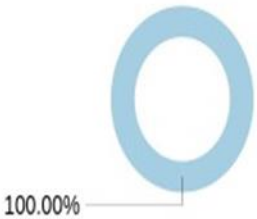


Code Flow Review Tool (Microsoft)

Review Board (VMware)

NO DATA

Phabricator



Bitbucket

Other

Developer/ Programmer/ Software Engineer

Team Lead

DevOps Engineer/ Infrastructure Developer

Architect

UI/UX developer

Technical Support

Data Analyst/ Data Scientist/ Data Engineer

Other

ما نتایج زیر را برای هر یک از CRRS زیر همراه با توزیع نمودار دایره ای در شکل ٤,٦ به دست آوردیم:

i.Github/GitLab: از میان شرکت‌کنندگانی که با این CRRS آشنا بودند، ۱۰ درصد از شرکت‌کنندگان سرپرست تیم، ۱۰ درصد دیگر صاحب محصول و بقیه (۸۰ درصد) توسعه‌دهندگان/برنامه‌نویسان/مهندسین نرم‌افزار بودند.

ii.Code Flow Review Tool (Microsoft: از شرکت کنندگانی که با این CRRS آشنا بودند، ۳۳,۳۳٪ توسعه دهندگان/برنامه نویسان، مهندسان نرم افزار، مهندس /DevOps توسعه دهنده زیرساخت و دیگران بودند.

III. Phabricator: توسعه دهندگان / برنامه نویسان / مهندسان نرم افزار تنها افرادی بودند که با Phabricator آشنا بودند.

iv.BitBucket: از شرکت کنندگانی که با این CRRS آشنایی داشتند، ٦۰,۰۰٪ آنها توسعه دهندگان / برنامه نویسان / مهندسان نرم افزار، ۲۰,۰۰٪ از آنها سرپرست تیم و دیگران بودند.

v.Others: تعدادی CRRS دیگر در بازار وجود دارد و از شرکت کنندگان خواسته شد در صورت اطلاع از CRRSهای دیگری که در نظرسنجی ذکر نشده است، آن گزینه را انتخاب کنند. برای، ما ۵۰,۰۰٪ پاسخ از توسعه دهندگان و همچنین دیگران دریافت کردیم. از میان شرکت کنندگانی که در نظرسنجی ما شرکت کردند، هیچ یک از آنها با دو سیستم/ابزار توصیه بازبینی کد آشنا نبودند: ReviewBoard (VMware) و Gerrit Code Review (.System (Chromium)ب) ویژگی‌های CRRS و نقش شغلی ما تعدادی از ویژگی‌های CRRS را برای شرکت‌کنندگان ارائه کردیم که آنها آن‌ها را برای استفاده راحت‌تر و آسان‌تر از CRRS مهم می‌دانستند. شکل ٤,۷ نشان می‌دهد که چه نقش شغلی چه ویژگی‌هایی مفید است.

#	Field	Pre commit code review	Code discussion with old and new versions being highlighted to show the change in code	Code improvement suggestion by the code reviewer (Other than just pointing out the code errors)	Prioritizing code changes based on its level of importance and its effect on the functionality of the software	Integration of project tracking software (such as Trello, Jira etc.)	Integration of source-code editor (such as Visual Studio, Atom etc.)	Integration of business communication platform (such as Slack)	Total
1	Developer/ Programmer/ Software Engineer	12.50%	25.00%	8.33%	0.00%	20.83%	16.67%	16.67%	24
2	Team Lead	0.00%	33.33%	33.33%	0.00%	33.33%	0.00%	0.00%	3
3	DevOps Engineer/ Infrastructure Developer	33.33%	33.33%	33.33%	0.00%	0.00%	0.00%	0.00%	3
4	Architect	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0
5	UI/UX developer	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0
6	Technical Support	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0
7	Data Analyst/ Data Scientist/ Data Engineer	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0
8	Other	25.00%	0.00%	25.00%	25.00%	25.00%	0.00%	0.00%	4

Showing rows 1 - 8 of 8

شکل ٤,۷: ویژگی‌های CRRS و نقش شغلی

۱. بررسی کد پیش از انجام تعهد:

مشاهده شد که از تمام شرکت کنندگانی که موافقت کردند این ویژگی را مهم در نظر بگیرند، ٦٠٪ آنها توسعه دهنده، ٢٠٪ مهندس / DevOpsتوسعه دهنده زیرساخت و دیگران (یعنی مالک محصول) بودند.

٢. بحث کد با نسخه‌های قدیمی و جدید که برای نشان دادن تغییر کد برجسته شده اند:
از کل شرکت‌کنندگان، ٧٥ درصد از آنها توسعه‌دهنده، ١٢,٥٠ درصد از آن‌ها رهبران تیم و مهندس /DevOpsتوسعه‌دهنده زیرساخت بودند که این ویژگی CRRS را مهم می‌دانستند.

٣. پیشنهاد بهبود کد توسط بازبینی کننده کد (غیر از اشاره صرف به خطاهای کد): از همه شرکت کنندگانی که این ویژگی را مفید دانستند، ٤٠,٠٠٪ از آنها توسعه دهندگان، ٢٠,٠٠٪ از آنها رهبر تیم، مهندس / DevOpsتوسعه دهنده زیرساخت و دیگران (صاحب محصول).

٤. اولویت بندی تغییرات کد بر اساس سطح اهمیت و تأثیر آن بر عملکرد نرم افزار: از بین همه شرکت کنندگان، فقط صاحب محصول این ویژگی را مفید دانسته است.

٥. یکپارچه سازی نرم افزارهای ردیابی پروژه (مانند Trello، Jira و غیره): برای این ویژگی، ٧١,٤٣٪ از توسعه دهندگان، ١٤,٢٩٪ از رهبران تیم و مدیر پروژه آن را مفید دانستند.

٦. ادغام ویرایشگر کد منبع (مانند ویژوال استودیو، اتم و غیره): فقط توسعه دهندگان از همه شرکت کنندگان بودند که این ویژگی را به عنوان یک ویژگی مهم در نظر گرفتند.

٧. یکپارچه سازی پلت فرم ارتباط تجاری (مانند Slack): این ویژگی فقط توسط توسعه دهندگان همه شرکت کنندگانی که در نظرسنجی ما شرکت کرده اند مفید است.

٤,٤ خلاصه

ما نتایج به‌دست‌آمده با بررسی طیف گسترده‌ای از اعضای پروژه نرم‌افزاری را در اینجا ارائه کردیم که در آن متوجه شدیم کدام ویژگی‌های CRRS مفیدتر هستند، کدام ویژگی‌ها در سیستم موجود گم شده‌اند و چه عواملی هنگام انتخاب یک بازنگری کد مربوطه مهم هستند. ما همچنین ترجیحات توسعه‌دهنده/بازبین را نسبت به اینکه چه نوع بررسی کد باید انجام شود (بررسی طولانی یا کوتاه) و در چه مرحله‌ای از گردش کار باید انجام شود، به دست آوردیم. جدای از این، ما همچنین گرایش‌ها و الگوهایی را بین استفاده از سیستم CRRS و ارتباط آن با اطلاعات جمعیت‌شناختی بازبین/توسعه‌دهنده پیدا کردیم.

فصل ۵

بحث

این بخش به سوالات تحقیق ما پاسخ می‌دهد و بحث می‌کند. با انجام مرور ادبیات سیستماتیک، پاسخ‌هایی را برای سه سؤال اول تحقیق یافتیم و با استفاده از نظرسنجی پاسخ‌هایی را برای دو سؤال تحقیقاتی آخر یافتیم. عواملی که باید در هنگام ایجاد یک CRRS و دسته بندی CRRSهای موجود در نظر گرفته شوند. از سوی دیگر، با انجام این نظرسنجی متوجه ویژگی‌هایی شدیم که برای یک سیستم توصیه برای بازبینی‌کنندگان کد مهم هستند و چه پیشرفت‌هایی می‌توان در CRRSهای موجود انجام داد.

با انجام یک مرور ادبیات سیستماتیک ((SLRما برخی از راه‌حل‌های موجود را برای سیستم‌های پیشنهادی برای بازبینی‌کنندگان کد شناسایی کردیم، عواملی که باید هنگام ایجاد یک CRRS و دسته‌بندی CRRSهای موجود در نظر گرفته شوند. از سوی دیگر، با انجام این نظرسنجی متوجه ویژگی‌هایی شدیم که برای یک سیستم توصیه برای بازبینی‌کنندگان کد مهم هستند و چه پیشرفت‌هایی می‌توان در CRRSهای موجود انجام داد.

سوال ۱:راه‌حل‌های موجود برای سیستم‌های توصیه برای بازبینی‌کنندگان کد چیست؟

پاسخ: ما تعدادی از مقالات را بررسی کردیم و تعدادی سیستم پیشنهادی بررسی کد موجود را پیدا کردیم. این سیستم‌ها/ابزارها عبارتند از cHRev، REVFINDER، CorreCT، TIE، CodeFlow، ReviewBoard، Phabricator، Gerrit، rDevX و CRRS مبتنی بر پروفایل که بر اساس تعدادی از فاکتورها/انواع داده توصیه‌هایی را ارائه می‌دهند. این نوع داده‌ها شامل تاریخچه بررسی کد، شباهت مسیر فایل، تجربه بین پروژه و فناوری مرتبط، متن کاوی و مکان فایل و وضعیت ردیابی هر بازبین یا نویسنده است.

سوال ۲:چه عواملی باید در هنگام ایجاد یک سیستم توصیه برای بازبینان کد در نظر گرفته شوند؟ پاسخ: عامل اولیه ای که هنگام ایجاد یک سیستم توصیه برای بازبینان کد باید در نظر گرفته شود، معیارهای ارزیابی منبع داده یا نوع پروژه ای است که سیستم بر روی آن آزمایش شده است (یعنی منبع باز یا تجاری یا هر دو). وقتی نوبت به توصیه یک بازبین کد بر اساس نمایه بازبین می‌شود، باید نمایه بازبین را به‌روزرسانی کنید که شامل بررسی‌های گذشته و سابقه تعهد می‌شود. به طور مشابه، وقتی نوبت به تاریخچه مرور گذشته می‌رسد، مهم است که مجموعه مخزن/داده بررسی‌های گذشته را به‌روزرسانی کنید تا بر اساس بررسی‌های گذشته، بازبینی‌کنندگان کد مربوطه را در آینده توصیه کنید.

سوال ۳:چگونه می‌توان سیستم‌های پیشنهادی موجود برای بازبینان کد را دسته بندی کرد؟

پاسخ: سیستم‌های پیشنهادی موجود بر اساس نوع داده دسته‌بندی شده‌اند که شامل تاریخچه بررسی کد، شباهت مسیر فایل، تجربه بین پروژه‌ای و فناوری مرتبط، متن کاوی و مکان فایل، وضعیت ردیابی هر بازبین یا نویسنده است. cHRev یک سیستم توصیه بازبینی کد بود که بر اساس تاریخچه مرور کد، بازبینان کد را توصیه می‌کرد. REVFINDER CRRS دیگری بود که بازبینان کد را بر اساس شباهت مسیر فایل توصیه می‌کرد. به طور مشابه، ما یک CRRS به نام CoRReCT پیدا کردیم که هدف آن توصیه بازبینان کد بر اساس تجربه بین پروژه و فناوری مربوطه بود. TIE (Text mining and a fileE) همانطور که از نامش می‌گوید، بازبینی‌کنندگان کد را با کمک متن کاوی و مکان فایل توصیه می‌کند.

سوال ٤: ویژگی‌های مهم یک سیستم توصیه برای بازبینان کد چیست؟

پاسخ: بر اساس نظرسنجی اعضای پروژه نرم افزاری که انجام شد، تعدادی ویژگی را یافتیم که برای یک سیستم توصیه برای بازبینان کد مهم در نظر گرفته می‌شد.

شامل:

۱. بحث کد با برجسته شدن نسخه‌های قدیمی و جدید برای نشان دادن تغییر در کد.

۲. ادغام با یک نرم افزار ردیابی مشکل مانند Trello، JIRA

و غیره.

۳. بررسی کد پیش از انجام تعهد.

٤. پیشنهادات بهبود کد توسط بازبین کد، فراتر از اشاره به خطاهای کد.

۵. ادغام با یک ویرایشگر کد منبع، مانند ویژوال استودیو یا اتم.

٦. ادغام با یک پلتفرم ارتباطی تجاری، مانند تیم‌های Slack یا .MS

۷. اولویت بندی تغییرات کد بر اساس میزان اهمیت و تأثیر آن بر عملکرد نرم افزار.

۸. ارائه خط لوله ای که نشان می‌دهد پروژه در کدام مرحله توسعه از جمله ساخت، آزمایش، بررسی کد و استقرار است.

۹. وجود داشبورد برای همه اعضای پروژه که داده‌های آماری تمامی اقدامات انجام شده را نشان می‌دهد، مانند تعداد commit ها، تعداد بازبینی کد انجام شده و تعداد خطاها/اخطارهای کد در پروژه جاری.

۱۰. بحث کدهای جدید و قدیمی با کد رنگی زمانی که تغییر در کد وجود دارد.

۱۱. گزینه ای برای انتخاب یک شاخه یا فایل خاص در یک پروژه برای حفظ یک گردش کار سیستماتیک و یک روند بررسی کد سازمان یافته.

۱۲. شناسایی کد با استفاده از یک طرح رنگی با نام توسعه دهنده/ها منتقل شد.

سوال ۵:چگونه می‌توان سیستم‌های پیشنهادی موجود برای بازبینان کد را بهبود بخشید؟ به عبارت دیگر، چه ویژگی‌هایی در پیاده‌سازی‌های موجود برای سیستم‌های توصیه بازبین‌کنندگان کد وجود ندارد؟

پاسخ: بر اساس نتایج نظرسنجی، موارد زیر برخی از ویژگی‌هایی است که می‌توان آن‌ها را بهبود بخشید یا در سیستم‌های توصیه بازبینی کد یا زمانی که به دنبال یک بازنگری کد مربوطه می‌گردید، وجود ندارد:

۱. وقتی صحبت از انتخاب یک بازبین کد می‌شود، شرکت کنندگان بر این باور بودند که تخصص داوران در زبان برنامه نویسی پروژه، زمینه تخصص، سال‌ها سابقه کار، تخصص کیفیت کد و درک معماری پروژه از عوامل مهم هستند.

۲. برخی از شرکت کنندگان معتقد بودند که تعداد سال سابقه کار و همچنین زمینه تخصص هر دو مهم است. استدلال این بود که این عوامل می‌توانند برای یافتن یک رویکرد بهینه برای یک مشکل و ارائه پیشنهادهایی برای) LLDطراحی سطح پایین) مفید باشند. همچنین، این عوامل در نوشتن یک کد عملی استاندارد که از تجربه و تخصص ناشی می‌شود کمک کننده است.

۵,۱ پیشنهاد برای سیستم پیشنهادی بازنگری کد بهبودیافته

بر اساس یافته‌های تحقیق ما، ما یک سیستم توصیه بازنگری کد بهبودیافته را پیشنهاد می‌کنیم که تمام ویژگی‌های لازم را داشته باشد که در همه سیستم‌ها یا ویژگی‌هایی که در سیستم‌های موجود وجود ندارند وجود ندارد. سیستم توصیه‌گر پیشنهادی دارای ویژگی‌های زیر خواهد بود:

۱. شفاف‌تر در شرایطی که تمام جزئیات مربوط به بازبینی کد روی داشبورد قابل مشاهده باشد. این جزئیات شامل تعداد پروژه‌هایی است که روی آن‌ها کار کرده‌اند (یعنی تجربه کاری آنها)، زمینه کاربردی خاص که در آن تخصص دارند، تعداد بررسی‌های کد انجام‌شده توسط آنها، و حجم کاری آنها (یعنی تعداد بررسی‌هایی که در حال حاضر بازبین انجام می‌شود. بررسی) برای اطمینان از اینکه بازبینی کننده با بررسی کدهای جدید بیش از حد سنگین نیست. اعتقاد بر این است که ارائه این جزئیات در نتیجه روند بررسی کد را سرعت می‌بخشد.

۲. ترکیب گسترده تری از داده‌ها برای آموزش توصیه کننده. این مجموعه داده شامل نظرات مرور گذشته و پیام‌های تعهد، و تجربه بین پروژه‌ای و فناوری مرتبط خواهد بود. این داده‌ها به دانستن اینکه آیا کدی که باید بازبینی شود مطابقت نزدیکی با تجربه پروژه که یک بازبین دارد، کمک می‌کند. به طور مشابه، تاریخچه گذشته نظرات و تعهدات بررسی به انتخاب یک بازبین مربوطه بر اساس تعداد تعهدات و بررسی‌هایی که قبلاً توسط آنها انجام شده و اینکه بررسی‌های کد گذشته چقدر برای توسعه دهندگان مفید بوده است، کمک خواهد کرد. این کمک می‌کند تا اطمینان حاصل شود که نظرات بررسی آینده آنها برای بررسی کد مفید خواهد بود.

۳. بررسی کد قبل از ادغام کد و تداخل کد بالقوه انجام می‌شود. بنابراین، سیستم پیشنهادی پیشنهادی، قبل از وقوع تضادهای ادغام، بازبینان کد را توصیه می‌کند. با این حال، این سیستم همچنین اجازه می‌دهد تا پس از تضادهای ادغام، انتخاب بازبینی کد انجام شود تا در صورت نیاز از تأخیر جلوگیری شود، و در عین حال از ایجاد یک محصول نرم افزاری با کیفیت خوب اطمینان حاصل شود.

فصل ٦

نتیجه

در این تحقیق ما تعدادی از سیستم‌های پیشنهادی بازنگری کد ((CRRSرا که در ادبیات یافت می‌شوند، راه‌های مختلف طبقه‌بندی این سیستم‌ها، چه ویژگی‌هایی برای یک سیستم توصیه‌ای برای بازبینی‌کنندگان کد مهم هستند و سیستم‌های موجود را چگونه می‌توان شناسایی کرد. بهبود یافته یا اینکه کدام ویژگی در CRRSهای موجود وجود ندارد. یک مطالعه مرور ادبیات سیستماتیک برای شناسایی سیستم‌های پیشنهادی بازبینی کد موجود و درک جزئیات مربوط به این سیستم‌ها، که شامل ویژگی‌ها و عواملی است که برای یک CRRS مهم هستند، انجام شد. سپس ما یک نظرسنجی برای درک نیازهای اعضای پروژه نرم افزار در مورد سیستم‌های توصیه بازنگری کد انجام دادیم که مشخص می‌کند کدام ویژگی‌ها در یک CRRS مهم هستند و چه چیزی می‌تواند در CRRSهای موجود بهبود یابد.

٦,١ مشارکت:

این تحقیق مشارکت‌های زیر را انجام می‌دهد:

C1:رتبه‌بندی ویژگی‌های موجود در سیستم‌های پیشنهادی بازنگری کد موجود برای اینکه کدام یک از اینها مفیدتر هستند.

C2:دسته بندی CRRS های موجود از ادبیات به همراه ابعاد مختلف.

C3:بهبودهایی که می توان در سیستم های پیشنهادی بازنگری کد موجود ایجاد کرد.

C4:ویژگی هایی که هنگام انتخاب یک بازبینی کننده کد مهم هستند.

٦,٢ کار آینده

جهت گیری‌های احتمالی آینده بر اساس این کار عبارتند از:

مروری بر ادبیات سیستماتیک گسترده‌تر: یک مطالعه مروری سیستماتیک گسترده‌تر می‌تواند انجام شود که نه تنها به سیستم‌های توصیه بازبین کد، بلکه به شیوه‌ها و رویه‌های بررسی کد نیز می‌پردازد. این می‌تواند به ارائه تصویر بهتری در مورد نیازهای اعضای پروژه نرم افزاری در مورد بررسی کد همراه با استفاده از CRRS کمک کند.

ساختن یک سیستم توصیه بازبینی کد: برای کارهای آینده، هدف ما ساختن سیستمی است که تمام جزئیات بازبین را در داشبورد سیستم قابل مشاهده باشد (یعنی تجربه کاری، تجربه فناوری، تعداد بررسی‌های کد انجام شده و غیره). همچنین، سیستم داده‌های بیشتری برای آموزش سیستم توصیه‌گر خواهد داشت که شامل نظرات مرور گذشته و پیام‌های تعهد، پروژه متقابل مرتبط و تجربه فناوری است. بر اساس بازخورد به‌دست‌آمده از نظرسنجی، بررسی‌ها قبل از وقوع تضادهای ادغام انجام می‌شود.