بازبینی کد گذشته و فعلی
سیستم‌های توصیه

پالاک هالوادیا
لیسانس مهندسی کامپیوتر، دانشگاه فناوری گجرات، 2017

پایان‌نامه ارائه شده
در تحقق‌بخشی از الزامات برای درجه

کارشناسی ارشد

که در

علوم کامپیوتر

گروه ریاضیات و علوم کامپیوتر
دانشگاه لت‌بریج
لت بریج، آلبرتا، کانادا

ج Palak Halvadia ، 2021
تحقیق در مورد سیستم‌های توصیه شده بازبینی کننده گذشته و کد فعلی

تاریخ دفاع: 14 آوریل 2021

چکیده:

زمینه :انتخاب مرورگر کد یکی از جنبه‌های مهم توسعه نرم‌افزار است و به عوامل مختلفی بستگی دارد.
اهداف :هدف ، درک راه حل های موجود برای سیستم های توصیه بازبینی کد (CRRS)، عواملی است که هنگام ساخت آنها باید در نظر گرفته شود و ابعاد مختلفی که بر اساس آنها می‌توان آنها را طبقه‌بندی کرد .هدف ما درک ویژگی‌های مهم CRRS و آنچه می‌توان در CRRS های موجود بهبود بخشید است.
روش ها :مطالعه مروری بر ادبیات برای درک CRRS های موجود انجام شد .نظرسنجی از اعضای پروژه توسعه نرم‌افزار برای درک ویژگی‌های مهم و مفقود شده در CRRS انجام شد.
نتایج :ما مقالات انتخاب شده را به دودسته طبقه‌بندی کردیم: بر اساس نوع داده مورداستفاده برای ارائه توصیه‌ها و نوع پروژه مورداستفاده برای ارزیابی. این نظرسنجی به ما کمک کرد ویژگی‌های موجود در CRRS را درک کرده و برخی از روندها و الگوها را مشاهده کنیم.

فصل 1

تعاریف

مرور کد یک بررسی سیستماتیک از کد منبع رایانه است و اغلب به‌عنوان یک بررسی همکار انجام می‌شود .هدف بازبینی کد شناسایی و اصلاح اشتباهات در کد منبع و همچنین بهبود کیفیت کد و مهارت توسعه دهندگان نرم افزار است .همچنین ، این هدف فقط بهبود کیفیت کد یا یافتن نقص در کد منبع نیست .همچنین باعث افزایش آگاهی تیم و همچنین کمک به توزیع دانش می شود .همچنین مالکیت کد مشترک را تشویق می کند.

چهار نوع بررسی کد وجود دارد:

.1برنامه نویسی جفت :در این نوع بازبینی کد، دو توسعه دهنده به طور همزمان کد منبع را تولید می‌کنند و به طور همزمان مرور می‌کنند .

.2مرور کد به کمک ابزار :برای این نوع مرور کد، نویسندگان و توسعه دهندگان از ابزارهای بررسی کد همتا استفاده می کنند .

.3بازبینی کد مرور :در اینجا ، توسعه دهنده مرورگر را از طریق مجموعه ای از تغییرات کد راهنمایی می کند .

.4بازبینی رسمی کد :این نوع بررسی کد شامل یک دقت.

بازرسی دقیق کد با مشارکت تعدادی از شرکت‌کنندگان و در چند مرحله .این یک روش سنتی برای مرور کد است که شامل شرکت در تعدادی از جلسات و مرور خط به خط است.

مرور کد را می توان به عنوان بازرسی دستی تغییرات در کد منبع دانست .تعدادی ابزار و سیستم توصیه ای وجود دارد که به منظور بررسی کد توسط تعدادی از سازمانهای مختلف توسعه یافته است .

زمینه‌های متعددی وجود دارد که در آنها مشارکت سیستم‌های توصیه شده برای اعضای پروژه توسعه نرم افزار مفید بوده است .برای کمک به کار بررسی کد، تحقیقات قابل توجهی در مورد سیستم‌های توصیه ای انجام شده است که هدف آنها ارائه توصیه‌های بازبینان کد بر اساس جنبه های مختلف است .دلایل مختلفی وجود دارد که چرا علاوه بر یافتن عیوب کد، به مرورگر کد نیز نیاز است .این امر به این دلیل است که مرورگران کد بر بهبود کد، یافتن راه حل های جایگزین برای یک مشکل، ارائه دانش، بداهه پردازی در فرایند توسعه، اجتناب از وقفه‌های ساختاری تمرکز می‌کنند، اشتراک مالکیت کد، و همچنین ارزیابی تیم .به عنوان مثال رحمان ، روی و کالینز، یک سیستم بازبینی کد را پیشنهاد کردند که در آن تخصص یک مرورگر کد بر اساس اطلاعات بدست آمده از یک پروژه متقابل است.

سابقه و همچنین تخصص یک بازبینان کد در یک زمینه خاص بر اساس درخواست های کشش آنها.

نمونه هایی از سیستم های توصیه ای کلی در مهندسی نرم افزار

1. سیستم‌های توصیه گر کد گرافیکی
2. سیستم هیپی کات

لی و کانگ مطالعه‌ای بر روی "سیستم‌های توصیه گر کد گرافیکی" انجام دادند تا بفهمند ابزارهای تجسم نرم‌افزار تا چه اندازه به توسعه دهندگان در درک کد کمک کرده است .نویسندگان دریافتند که توسعه دهندگان زمان قابل‌توجهی را صرف درک مبانی کد می‌کنند .برای سهولت این کار ، تعدادی از توصیه کنندگان کد گرافیکی برای آنها ایجاد شد .این سیستم های توصیه گر از دو چکیده استفاده کردند:

1. طراحی و توضیح کد و مستندات سیستم نرم‌افزاری
2. تجزیه‌وتحلیل کد.

یک سیستم توصیه گر به نام Hipikatتوسعه داده شد که دسترسی توسعه دهندگان را به حافظه گروهی که شامل آثار مرتبط با پروژه است که در طول توسعه پروژه ایجاد شده است ، فراهم می کند [D. Cubranic, G. C. Murphy, Singer.jو همکاران2005].
این به توسعه دهندگان کمک می‌کند تا در غلبه بر مشکلات فنی و جامعه‌شناختی وقت صرفه جویی کنند .این ابزار با تغییرات بسیار اندک یا بدون تغییر در شیوه های کاری موجود ، حافظه گروه را به طور خودکار ایجاد می کند .این سیستم پیشنهادی به اشتراک گذاری اطلاعات مربوط به یک پروژه از هر منظر به همه اعضای تیم توسعه کمک کرد ، در نتیجه در توضیح مفاهیم برای اعضای موجود و جدید شرکت صرفه جویی کرد.

1.1 مروری بر کار

1-1-1 انواع مرور کد:

این کار بر روی کد پشتیبانی شده از ابزار تمرکز می‌کند. هدف تحقیق ما دوگانه است: یافتن پاسخ سؤالات "گذشته" با انجام "مرور ادبیات سیستماتیک" و دوم یافتن پاسخ سؤالات "حال" با انجام نظرسنجی از اعضای پروژه نرم‌افزاری. "مرور ادبیات سیستماتیک" در یافتن جزئیات در مورد سیستم‌های توصیه بازبینی کننده کد موجود کمک می‌کند، درحالی‌که این نظرسنجی به یافتن تغییراتی که مهندسان نرم‌افزار فکر می‌کنند نیاز دارند یا آنچه در سیستم‌های توصیه بازبینی کد موجود وجود ندارد، کمک می‌کند.

1-1-2 چرا این کار موردنیاز است

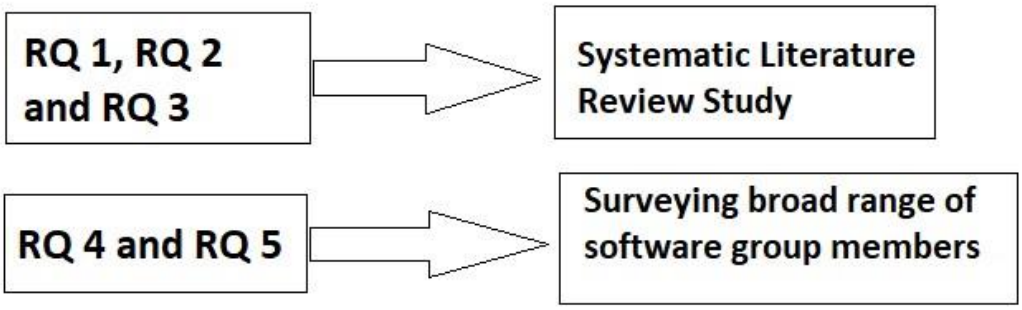
هدف از این تحقیق مستندسازی آموخته‌های به‌دست‌آمده از"مطالعه مروری بر ادبیات سیستماتیک" و همچنین "بررسی" انجام شده بر روی طیف وسیعی از اعضای پروژه نرم‌افزار. این کار به این دلیل انجام شده است که تحقیقات بسیار کمی در مورد سیستم‌های توصیه کد مرورگر انجام شده است. (CRRS)و تحقیقات بیشتری در مورد شیوه ها و رویه های بازبینی کد انجام شده است. اهدافی که در یافتن پاسخ به نتیجه مورد نیاز به ما کمک می کند در زیر ذکر شده است:

- •تجزیه‌وتحلیل راه‌حل‌های ارائه شده توسط "مرورگر کد موجود سیستم‌های توصیه "
- • برای درک راه حل‌ها/ویژگی‌های موجود در "سیستم‌های توصیه مرورگر کد" موجود.
- • تجزیه‌وتحلیل وسایل مختلف انشعاب سیستم‌های پیشنهادی مرورگر کد بر اساس نحوه پیاده‌سازی آنها.

1-2 سؤالات و روش تحقیق

سؤالات تحقیق این کار به شرح زیر است:

1. راهکارهای موجود برای سیستم های توصیه برای مرورگران کد چیست؟
2. هنگام ایجاد یک سیستم توصیه برای مرورگران کد، چه عواملی باید در نظر گرفته شوند؟
3. چگونه می‌توان سیستم‌های توصیه‌ای موجود برای مرورگران کد در ادبیات را دسته‌بندی کرد؟
4. ویژگی‌های مهم سیستم توصیه برای مرورگران کد چیست؟
5. چگونه می‌توان سیستم‌های پیشنهادی موجود برای مرورگران کد را بهبود بخشید؟ به‌عبارت‌دیگر، چه ویژگی‌هایی در پیاده‌سازی‌های موجود برای سیستم‌های توصیه بازبینی کننده کد وجود ندارد؟



شکل 1.1: سؤالات و روش تحقیق

سؤال 1 و 2 و 3 با استفاده از مرور ادبیات سیستماتیک پاسخ داده می‌شوند درحالی‌که سؤال 4 و 5 با بررسی طیف وسیعی از اعضای پروژه نرم‌افزاری پاسخ داده می‌شوند.

3-1 مشارکت

این تحقیق مشارکت‌های زیر را انجام می‌دهد:

1. تعدادی از ویژگی‌های موجود در سیستم‌های پیشنهادی بازبینی کد (CRRS) را شناسایی کرده و آن ویژگی‌ها را بر اساس مفید بودن آنها.
2. CRRSهای موجود را بر اساس ابعاد مختلف طبقه‌بندی کردیم.
3. ویژگی‌هایی را که می‌توان هنگام انتخاب مرورگر کد مهم تلقی کرد، شناسایی کردیم.
4. بهبودهای احتمالی CRRS های موجود را برای تسهیل یافتن مرورگران کد مناسب شناسایی کردیم.

در فصل 2 کارهای مرتبط ارائه شده است .نتایج مطالعه مرور ادبیات در فصل 3 ارائه شده است. فصل 4 بررسی نتایج است .فصل 5 شامل بحث است .پایان‌نامه در فصل 6 به پایان رسیده است.

فصل 2

کار مرتبط

این فصل خلاصه‌ای از مرور ادبیات قبلی انجام شده در مهندسی نرم‌افزار، داده‌کاوی و سیستم‌های توصیه‌گر را ارائه می‌دهد. این مطالعات ارائه شده است تا نشان دهد که چگونه مطالعات مرور ادبیات درگذشته انجام شده و برای راهنمایی روش بررسی ادبیات ما مورداستفاده قرار گرفته است.
بررسی ادبیات انجام شده در مورد سیستم‌های توصیه‌گر شامل سیستم‌های توصیه‌گر است که هدف آنها استخراج اطلاعات مربوطه از حجم زیادی از دانش و سیستم‌های توصیه‌ای برای مهندسی نرم‌افزار است که ویژگی‌های سیستم‌های موجود، شکاف‌های تحقیقاتی و کارهای احتمالی آینده را ارائه می‌دهد. به طور مشابه، یک مطالعه مرور ادبیات در زمینه مهندسی نرم‌افزار در مورد مطالعات پیش‌بینی خطا و روش توسعه نرم‌افزار چابک انجام شد. بررسی ادبیات انجام شده درداده کاوی دو مدل پرکاربرد برای داده کاوی در CRM (مشتری مدیریت روابط) را کشف کرد.

2-1مروری بر ادبیات سیستم ها

مرور ادبیات توسط HARUNA ، Ismail، SUHENDROYONO و همکاران انجام شد در مورد سیستم‌های توصیه‌کننده آگاه به زمینه (CARS) که هدف آنها استخراج اطلاعات مربوطه از حجم زیادی از دانش است. این سیستم‌های توصیه‌گر، ارائه اطلاعات زمینه‌ای و مرتبط بر اساس "جستجوهای کاربران" و ارائه توصیه‌های شخصی‌تر کاربر است شامل سه مرحله اصلی است. اولین مرحله بررسی عمیق و طبقه‌بندی ادبیات بر اساس حوزه‌های مختلف مدل‌های کاربردی، فیلترینگ، استخراج و همچنین رویکردهای ارزیابی است. دومین ارائه نتایج بررسی با مزایا و معایب مرور است. سومین برجسته‌کردن چالش‌ها/فرصت‌های احتمالی یا کار یا تحقیقات آینده است که می‌توان انجام داد. این شامل کمک به مبتدیان و محققان جدید برای درک پیش نیازهای توسعه CARS و همچنین ارائه این بررسی به عنوان معیاری برای توسعه CARS برای کاربران متخصص است,[2017 K. HarunaM. AIsmail, S. Suhendroyono].
سیستم توصیه نوعی نرم‌افزار کاربردی است که هدف آن ارائه/توصیه اطلاعات مربوط به کاربران بر اساس نیاز کاربران است. در این زمینه، مطالعه مروری سیستماتیک ادبیات مشابه Gasparic و Janes اانجام شد، که نتایج عملکرد RSSEهای موجود (سیستم توصیه برای نرم افزار مهندسی) شکاف‌های تحقیق و همچنین جهت‌های احتمالی تحقیق را ارائه می‌دهد. آنها از رویکرد روش شناختی پیروی کردند که شامل فیلتر کردن مقالات تحقیقاتی مرتبط و جمع آوری شده بر اساس معیارهای مختلف بود. معیارهای خروج آنها شامل افرادی بود که به حوزه تحقیق بی ربط بودند، مقالاتی که راه حل های اجرا نشده را توصیف می‌کردند یا مقاله‌هایی که کاملاً در دسترس نبودند. برای استخراج مقالات مربوطه، مقاله‌ها بر اساس محتوای توصیف شده در چکیده مقاله یا گاهی عنوان، فیلتر و تقسیم می‌شوند. نویسندگان پس از پیروی از رویکرد روش شناختی خود، به چهار سوال تحقیقاتی خود پاسخ دادند که عبارتند از: خروجی ارائه شده توسط RSSEهای موجود ، مزایایی که این RSSEها برای مهندسان نرم افزار فراهم می‌کند، انواع ورودی مورد نیاز این RSSEها و چه تلاشهایی یک مهندس نرم افزار باید از این RSSEها استفاده کند. مشاهده شد که برخی از خروجی های 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. RSSEهای موجود عمدتا از استفاده مجدد، اشکال زدایی، پیاده سازی، مراحل/فعالیت‌های نگهداری و پشتیبانی از کیفیت سیستم برای مناسب بودن مهندسان نرم افزار پشتیبانی می‌کنند. برخی از ورودی هایی که این RSSEهای فعلی از آنها استفاده می‌کنند

شامل فایل‌های گزارش، ارتباط بین مهندسین نرم افزار، کد منبع، ورودی کاربر (به عنوان مثال، عبارت‌های جستجو، پرس و جو، تنظیمات، ترجیحات) ، مصنوعات آزمایشی و فرایند توسعه نرم افزار است همچنین، تلاش‌هایی که یک مهندس نرم افزار باید برای استفاده از RSSEهای موجود انجام دهد به عنوان تلاش‌های گسترده، تلاش‌های کم و بدون تلاش طبقه بندی می شود. [M. Gasparic and A. Janesوهمکاران]

مرور ادبیات دیگری برای RSSEها توسط پارک، کیم، چوی و همکاران انجام شد. [8] جایی که نویسندگان مقاله‌های تحقیق را بر اساس هشت زمینه کاربردی و هشت تکنیک داده‌کاوی دسته‌بندی کردند. هدف نویسندگان ارائه اطلاعات در مورد روندهای تحقیق در مورد سیستم‌های توصیه گر و همچنین تعیین جهت تحقیقات احتمالی آینده در مورد سیستم‌های توصیه‌گر بود.

2.2 بررسی ادبیات در مهندسی نرم‌افزار

پیش‌بینی دقیق خطاهای کد می‌تواند هزینه آزمایش را تا حد زیادی کاهش دهد و همچنین کیفیت محصول نرم‌افزاری را افزایش دهد. برای این منظور، یک مطالعه مروری بر ادبیات توسط هال، بو و، و همکاران انجام شد که بر مطالعات پیش‌بینی خطا متمرکز شده است. نویسندگان از رویکرد مرور سیستماتیک ادبیات که توسط Kitchenham و Chartersپیشنهاد شده است پیروی کردند، جایی که مراحل اولیه شامل مقاله‌ها و مطالعات مربوطه و حذف مطالعات مکرر است. هنگام حذف و شامل مقالات، جنبه‌های مختلفی در نظر گرفته می‌شود، مانند مقاله‌هایی که از منابع مختلف مانند مجلات، کنفرانس‌ها و پایگاه‌های داده استخراج شده و بر اساس محتوای عنوان و چکیده آنها مرتب شده‌اند. نویسندگان با حدود 208 مقاله به پایان رسیدند. یافته‌ها نشان می‌دهد که اکثر مطالعات اطلاعات زمینه‌ای و روش‌شناختی کافی را برای درک کامل یک مدل گزارش نمی‌دهند. نویسندگان مجموعه‌ای از معیارها را ارائه می‌دهند که مجموعه‌ای از جزئیات اساسی زمینه‌ای و روش‌شناختی را که مطالعات پیش‌بینی خطا باید گزارش کنند، مشخص می‌کند.

روش توسعه نرم افزار Agile یک متدولوژی توسعه نرم افزاری متداول است که توسط بسیاری از پروژه‌های توسعه نرم افزار استفاده می‌شود. هدف این روش اطمینان از تحویل خوب محصول مطابق نیاز کاربر و تجربه کاربری مناسب (UX)ست. به منظور ارائه یک محصول با کیفیت، مشارکت ذینفعان و کاربران، همراه با حلقه‌های بازخورد از هر دو طرف ضروری است. یک مطالعه مروری بر روی متدولوژی چابک توسط Schon، Thomaschewski و Escalona was به منظور بررسی وضعیت فعلی کار در این زمینه و پیشرفت‌های احتمالی آینده برای بررسی جنبه‌هایی که در وضعیت فعلی وجود ندارد، انجام شد. آنها مطالعه را در سه مرحله اصلی انجام دادند: برنامه ریزی، انجام و گزارش. مرحله "برنامه ریزی" شامل یافتن نیاز شناسایی برای مرور، چارچوب بندی سوالات تحقیق و توسعه و ارزیابی پروتکل بازبینی بود. مرحله "هدایت" با هدف جستجوی مقالات تحقیق، انتخاب مقالات مربوط به مطالعه، ارزیابی کیفی و استخراج و تجزیه و تحلیل داده‌ها انجام شد. مرحله آخر، "گزارش" با هدف استخراج و بحث درباره نتایج به دست آمده از مرحله قبل و سپس نوشتن، ارزیابی و قالب بندی گزارش نهایی برای مطالعه انجام شد. مشابه سایر مطالعات، نویسندگان از روش ارائه شده توسط Kitchenham و Charters پیروی کردند.

2-3 بررسی ادبیات درداده کاوی

مرور ادبیات انجام شده توسط نگای، شیو و چائو نمونه دیگری از روش‌شناسی برای مرور ادبیات سیستماتیک در زمینه داده‌کاوی را ارائه می‌دهد.

تکنیک‌های داده‌کاوی در مدیریت ارتباط با مشتری (CRM)اعمال می شود و Ngai، Xiu و Chauبه کمک مرور ادبیاتی که انجام داده اند، بینش کاملی در این زمینه ارائه می دهد. نویسندگان حدود 87 مقاله تحقیقاتی مرتبط را برای این منظور جمع آوری کردند که بر اساس چهار بعد CRM تقسیم شده اند که شامل توسعه مشتری ، شناسایی مشتری ، جذب مشتری

و حفظ مشتری و هفت تکنیک داده کاوی است. ارتباط ، طبقه بندی ، خوشه بندی ، پیش بینی ، رگرسیون ، کشف دنباله و تجسم. جدا از این ، برای وضوح بیشتر ، ابعاد CRM بیشتر به 9 زیر گروه از عناصر CRM طبقه بندی می شود که تحت تکنیک های داده کاوی قرار می گیرند. بر اساس مطالعه ، مشخص شد که طبقه بندی و ارتباط دو مدل پرکاربرد برای داده کاوی در CRM هستند. همچنین ، از چهار بعد CRM ، مشتری حفظ شده ترین مورد تحقیق است ، اگرچه اکثر آنها مربوط به برنامه های بازاریابی وفاداری یک به یک بود.

4-2 خلاصه

مطالعات مرور ادبیات برای سیستم‌های توصیه گر و از زمینه‌های مهندسی نرم‌افزار و داده‌کاوی ارائه شد. در مورد سیستم‌های توصیه، بررسی ادبیات بر سیستم‌های توصیه‌کننده آگاه از زمینه و سیستم‌های توصیه در مهندسی نرم‌افزار متمرکز شده است. در زمینه مهندسی نرم‌افزار، مطالعه ارائه شده در زمینه پیش‌بینی خطا و همچنین روش Agile انجام شده است. در زمینه داده‌کاوی، یک مطالعه مروری بر ادبیات انجام شد که در آن مفهوم داده‌کاوی اعمال شده برای مدیریت ارتباط با مشتری (CRM) مورد هدف قرار گرفت.

فصل 3

شیوه‌ها و ابزارهای بازبینی کد

درگذشته تعدادی سیستم/ابزار توصیه بازبینی کد وجود داشته است. به‌عنوان‌مثال ، ابزاری به نام Review Bot یکی دیگر از ابزارهای توسعه یافته توسط Balachandran است که در پروژه VMware استفاده شد. Review Bot متشکل از یک الگوریتم بود که تغییرات کد انجام شده در یک خط کد را به شکلی که کاملاً شبیه به دستور git fault است ، بررسی می کند. به هر نویسنده ای که روی تغییر کد در کد منبع کار کرده است امتیاز تعلق می گیرد اما نویسندگان با تغییرات اخیر نسبت به نویسندگان تغییرات قدیمی امتیاز بیشتری کسب می کنند. در پایان ، از جمع بندی هر نویسنده جداگانه برای تصمیم گیری نویسندگان برتر k استفاده می شود و سپس به آنها توصیه می شود که مرورگر کد شوند.
ما یک مطالعه سیستماتیک مرور ادبیات را انجام دادیم تا به سه سؤال تحقیقاتی اولیه خود پاسخ پیدا کنیم. ابتدا ما روش‌شناسی خود را قبل از ارائه نتایج مطالعه خود تعریف می‌کنیم.

3.1 روش‌شناسی

مطالعه مروری بر ادبیات سیستماتیک روشی است که در آن ادبیات موجود مربوط به تحقیق تعیین می‌شود، سپس ارزیابی می‌شود و در نهایت درک می‌شود. برای تحقیقات خود، ما رویکردی را که Kitchenham و Chartersاتخاذ کرده است دنبال می‌کنیم که شامل مراحل زیر است:
1. تمام کلمات کلیدی احتمالی مربوط به تحقیق مشخص شده است. ما کلمات زیر را شناسایی کردیم: کد، مرورگر، توصیه، سیستم‌ها، ابزارها و توصیه‌کننده. این کلمات کلیدی بر اساس موضوع تحقیق ما مشخص شد.
2. از کلمات کلیدی مشخص شده برای تشکیل رشته‌های جستجو استفاده کنید. رشته‌های جستجو برای به‌دست‌آوردن مقالات تحقیق از پایگاه‌های داده آنلاین استفاده می‌شود. ما از یک‌رشته جستجو برای کلمات کلیدی احتمالی و مترادف آنها استفاده کردیم. ما دو رشته جستجو ایجاد کردیم. برای یافتن رشته‌های جستجو به جدول 3.1 مراجعه کنید.
3- مقالات تحقیقاتی به‌دست‌آمده سپس بر اساس معیارهای مختلف حذف و ورود فیلتر می‌شوند. مقالات ابتدا با خواندن عنوان‌ها و چکیده مقالات تحقیق فیلتر می‌شوند.
فیلترکردن نتایج جستجو در سه مرحله اصلی انجام شد:
الف) فیلترکردن مقالات تحقیق بر اساس خواندن عنوان مقاله
(ب) فیلترکردن مقالات تحقیق بر اساس چکیده مقاله
ج) فیلترکردن مقالات تحقیق بر اساس خواندن متن کامل
4. مقالات فیلتر شده به طور کامل خوانده می‌شوند، ارزیابی و تفسیر می‌شوند تا اطلاعات مربوطه را به دست آورند.

1	(کد) و (داوران) و (توصیه‌کننده) یا (توصیه) و(سیستم‌ها) یا (ابزارها)
2	(توصیه)و (توصیه‌کننده) و ((سیستم‌ها)) یا ابزارها و (کد) و (بازدیدها) یا (داوران)

جدول 3.1: رشته‌های جستجو

در هر مرحله از فیلتراسیون، تعدادی مقاله تحقیقاتی فیلتر شد. پس از خواندن عنوان مقاله، 19 مقاله فیلتر شد. از بین این مقالات به‌دست‌آمده، 21 مقاله پس از مطالعه چکیده مقالات فیلتر شد. در نهایت، 7 مقاله پس از خواندن متن کامل مقالات تحقیق فیلتر شدند. در پایان 14 مقاله به دست آمد.

3-2 نتایج

پس از خواندن متن کامل مقاله‌های تحقیقاتی فیلتر شده، نه سیستم توصیه‌کننده مرور کد شناسایی شد. جدول 3.2 این سیستم‌ها و داده‌های مورداستفاده سیستم را برای ارائه توصیه می‌کند. بقیه این بخش شرح هر یک از این سیستم‌ها را ارائه می‌دهد.

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	بر اساس تجربه GitHub توصیه مرورگر کد در :correct 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

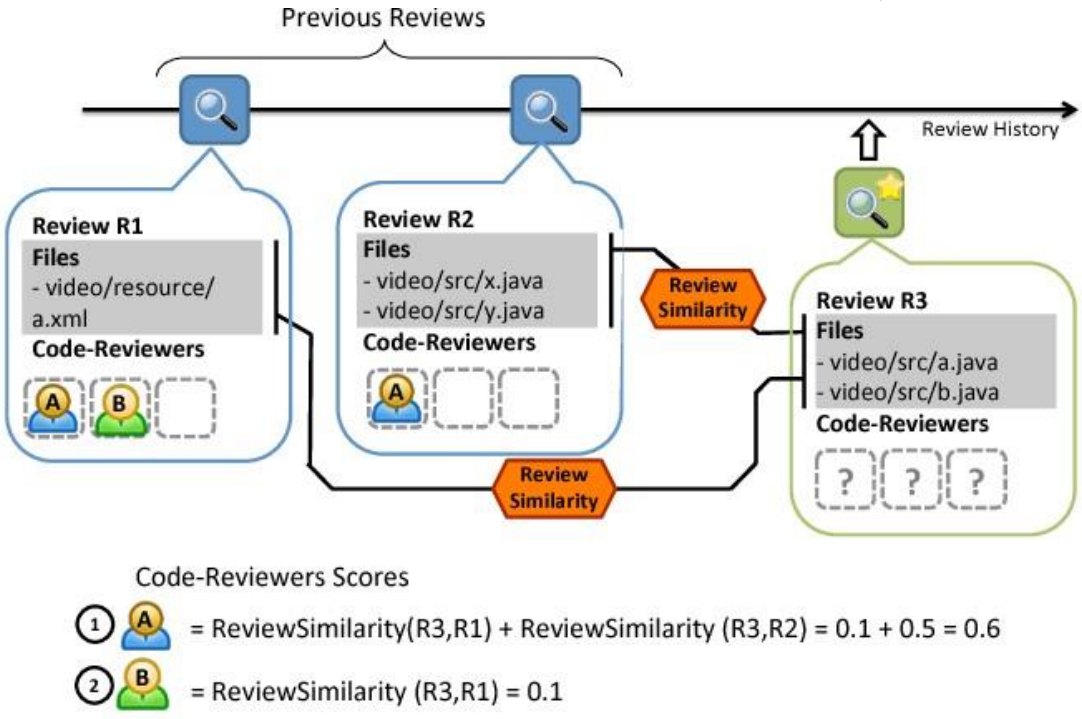
جدول3-2: مقاله‌های استخراج شده و داده‌های مورداستفاده

1-3-3REVFINDER

تعدادی CRRS براساس سیستم های مسیر فایل (FPS)یا رویکرد مبتنی بر مکان فایل ارائه شده است. Thongtanunam، Tantithamthavorn، Kula ، پیشنهاد REVFINDER که از روش توصیه مرورگر مبتنی بر مکان فایل پیروی می‌کند. شهود پشت این رویکرد این است که چندین فایل با یک مکان/مسیر فایل مشابه توسط مرورگران کد مجرب مشابه بررسی و مدیریت می شوند.

Thongtanunam، Tantithamthavorn، Kula ، همچنین یک مطالعه اکتشافی در مورد نحوه تأثیر تکلیف مرورگر کد بر زمان بازبینی انجام داد. این مطالعه اکتشافی نشان داد که حدود 4 تا 30 درصد از بررسی‌های کد با مشکل تعیین مرورگر صحیح کد مواجه هستند و تأیید تغییر کد حدود 12 روز طول می‌کشد. بر اساس نتایج این مطالعه، نویسندگان REVFINDER را پیشنهاد کردند.

REVFINDER شامل دو بخش است: الگوریتم رتبه‌بندی مرورگران کد و تکنیک ترکیبی. نویسندگان از الگوریتم رتبه‌بندی مرورگران کد (همان‌طور که در شکل 3-1 نشان‌داده‌شده است) برای ارزیابی نمرات مرورگران کد بر اساس شباهت مسیرهای پرونده‌هایی که قبلاً بررسی شده بودند، استفاده کردند. باتوجه‌به بررسی جدید R3 و دو بررسی قبلی R1 و R2 ، الگوریتم با مقایسه مسیرهای پرونده با بررسی جدید R3 ، نمره شباهت مرور را برای هر یک از بررسی های گذشته (R1، R2) محاسبه می کند. از این رو، دو نمره شباهت مرور اندازه دو وجود داشت: (R3، R1)و (R3، R2)از شکل می توان دریافت که مرور R3 و R2 در مقایسه با R3 و R1 دارای کلید واژه‌های مشترک بیشتری هستند، به این معنی که Reviewer A را می توان به عنوان داور بالقوه برای بررسی R3 در نظر گرفت. به منظور محاسبه شباهت مسیر فایل، نویسندگان از چهار تکنیک مقایسه رشته ای پیشرفته استفاده کردند:



شکل 3-1: مثال محاسبه الگوریتم رتبه بندی Code-Reviewers

1.طولانی‌ترین پیشوند مشترک(LCP)

LCP اجزای مسیر فایل رایج را که در هر دو مسیر فایل از ابتدا تا انتها ظاهر می‌شود ، محاسبه می‌کند.
2 .طولانی‌ترین پسوند مشترک(LCS)
LCS اجزای مسیر فایل مشترکی را که در هر دو مسیر فایل از انتهای هر دو مسیر فایل ظاهر می‌شوند، محاسبه می‌کند.
3 .طولانی‌ترین زیر رشته مشترک(LCSubstr)
LCSubstr اجزای مسیر پرونده رایج را که در هر دو مسیر پرونده به‌صورت متوالی ظاهر می‌شوند، اما در هر موقعیتی در مسیرهای فایل ظاهر می‌شوند، محاسبه می‌کند.
4 .طولانی‌ترین متعاقب مشترک(LCSubseq)

LCSubseqاجزای مسیر فایل مشترکی را که در هر دو مسیر پرونده‌ها به ترتیب نسبی ظاهر می‌شوند، محاسبه می کند . D. Gusfield],2007[

cHR حال ، برای محاسبه شباهت مسیر فایل بین فایل fn و فایل fp ، تابع filePathS شباهت (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 علامت اسلش ("/") به‌عنوان محدودکننده به نشانه تقسیم می شود. سپس تابع (StringComparison fn، fp)رای مقایسه اجزای مسیر فایل fn و fp استفاده می شود که اجزای رایج فایل را که در هر دو مسیر فایل ظاهر می شوند ، برمی گرداند.

2-3-3cHRev

تعدادی CRRS بر اساس بررسی‌های گذشته ساخته شده‌اند و Zanjani, Kagdi وBird [17]یک چنین سیستم توصیه ای به نام cHRev ایجاد کرده‌اند. cHRev به طور خودکار مرورگران کد را براساس مشارکت‌های قبلی خود در بررسی‌های قبلی خود توصیه می‌کند. cHRev مخفف عبارت review code Histories over other types of previous information برای توصیه به Reviewers است.
این سیستم توصیه دارای دو ویژگی اصلی است:
1 .بررسی‌کنندگان کد توصیه شده توسط cHRev ممکن است لزوماً در توسعه‌بخشی از کد منبع که در حال بررسی آن هستند مشارکت نداشته باشند، اما ممکن است بر روی کد منبع کارکرده باشند که به طور غیرمستقیم به منبع، منبع مورد بررسی بستگی دارد.
2. تخصص در طول زمان تغییر می‌کند و بنابراین تکرار و تکرار باید هنگام جستجوی مناسب‌ترین مرورگر کد در نظر گرفته شود.
فرایند مورد استفاده cHRev شامل سه مرحله است:

1 .کد منبع را که باید بازبینی شود استخراج کنید.
2 .تخصص بازبینی را براساس جزئیات مختلف مانند اینکه چه کسی، چه تعداد و چه زمانی در گذشته انجام شده است، تدوین کنید.
3. یک لیست رتبه‌بندی از داوران داوطلب بر اساس فایل‌های کد منبع در مرحله 1 و مشارکت تجمعی داوران در مرحله 2 به دست آورید و سپس با استفاده از یک پارامتر تعریف شده توسط کاربر، تعداد مترهای برتر کاندیداها را از لیست به‌دست‌آمده توصیه کنید.

به‌منظور آزمایش اثربخشی رویکرد آنها، زنجانی، کادی روش آنها را با REVFINDER، xFinderو RevCom مقایسه کرد. مشخص شد که cHRev از نظر دقت و فراخوانی توصیه‌های دقیق‌تری را ارائه می‌دهد. همچنین، مشاهده شد که cHRev عملکرد بهتری نسبت به REVFINDER دارد، از نظر مرورگرها بر اساس پرونده‌های دارای نام و مسیر مشابه و xFinder که به داده‌های مخزن کد منبع بستگی دارد. مشخص شد که cHRev از نظر آماری معادل RevCom است که نیاز به بررسی و تعهدات قبلی دارد. [M. B. Zanjani, H. H. Kagdi,2016]

CoRReCT 3-3-3

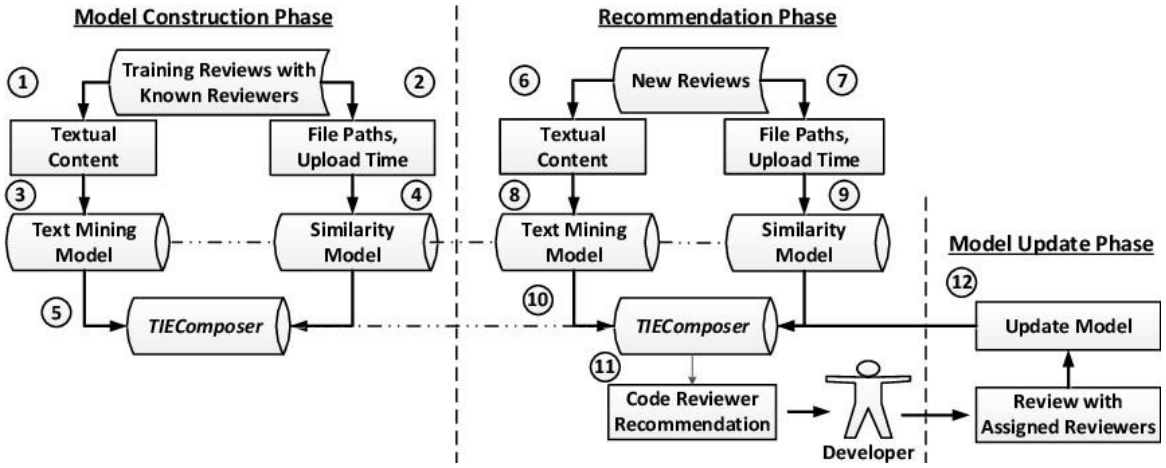
Rahman, Roy, Collins پیشنهاد یک سیستم توصیه بازبینی کد به نام CoRReCT (توصیه بازبینی‌کننده کد بر اساس تجربه پروژه و پروژه) که هدف آن توصیه بازبینی‌کننده‌های کدنویسی بر اساس سابقه کاری مرتبط با پروژه و همچنین تجربه توسعه‌دهندگان در یک فناوری تخصصی خاص مرتبط با درخواست کشش این دو منبع اطلاعاتی برای تعیین تجربه برنامه‌نویس برای بررسی کد استفاده شد. ایده اساسی پشت CRRS پیشنهادی آنها این است که اگر درخواست های pull قبلی دارای کتابخانه‌ها یا فناوری‌های تخصصی مشابه با درخواست های pull فعلی باشند، پس مرورکنندگان کد که آن درخواست های pullرا بررسی کرده‌اند می‌توانند به‌عنوان مرورگرهای احتمالی کد برای درخواست‌های pull فعلی در نظر گرفته شوند.

باتوجه‌به ایده پیشنهادی نویسندگان، توسعه‌دهندگان باتجربه بیشتر در کتابخانه‌های خارجی و فناوری‌های تخصصی پذیرفته شده در پرونده‌های تغییر در مجموعه نشانه‌های درخواست‌های فعلی کشش، نسبت به مواردی که تجربه کمتری دارند، انتخاب مناسب‌تری برای انجام بازبینی کد محسوب می‌شوند.

TIE 4-3-3

شیا، لو، وانگ. یک روش ترکیبی و افزایشی به نام TIE(Text mIning و رویکرد مبتنی بر مکان filE)پیشنهاد شده است که از مزایای استخراج متن و رویکرد مبتنی بر مکان فایل برای توصیه مرورگر کد استفاده می‌کند. ایده پشت این رویکرد تجزیه‌وتحلیل محتوای متنی در یک درخواست بازبینی با استفاده از یک مدل کاوی متنی افزایشی و محاسبه شباهت بین مسیرهای فایل بررسی شده و مسیرهای فایل تغییریافته با استفاده از یک مدل شباهت است.

معماری کلی TIE به سه مرحله تقسیم می‌شود: ساخت مدل، توصیه و به‌روزرسانی مدل همان‌طور که در شکل 3.2 نشان‌داده‌شده است.



شکل 3.2: معماری TIE

1. فاز ساخت مدل

مرحله ساخت مدل شامل یک مدل ترکیبی به نامTIECOMPOSER که با استفاده از بررسی‌های تاریخی داوران شناخته شده ساخته شده است. در این مرحله ، سیستم TIE ابتدا مرورهای آموزشی داوران شناخته شده را از محتوای متنی بررسی‌های گذشته و مسیرهای فایل و همچنین زمان بارگذاری جمع آوری می‌کند. در مرحله بعد ، TIE یک مدل استخراج متن را بر اساس داده‌های متنی پردازش شده با استفاده از تکنیک طبقه بندی متن ایجاد می‌کند. شهود پشت حالت داده کاوی این است که احتمالاً همان بازبین‌ها تغییرات را با اصطلاحات یا کلمات مشابه مرور می‌کنند.

TIE همچنین از یک رویکرد مبتنی بر مکان برای آگاهی از زمان استفاده می‌کند که هدف آن محاسبه شباهت بین بررسی‌های جدید و تاریخی است. این شباهت بین مسیرهای تغییریافته فایل (یعنی مسیرهایی از پرونده‌هایی که در درخواست بازبینی جدید تغییر کرده یا اصلاح شده‌اند) و مسیرهای فایل مرور شده (یعنی مسیرهای فایل‌هایی که در بررسی‌های تاریخی بررسی شده‌اند) محاسبه می‌شود. شهودی که در پشت رویکرد مبتنی بر مکان قرار دارد این است که همان مرورگران تمایل دارند فایل‌ها یا پرونده‌های مشابه را با مسیرهای مشابه مرور کنند. این دو مدل برای ساختن مدل TIECOMPOSER با هم ترکیب شده‌اند.

2. مرحله توصیه

در این مرحله ، TIE برای توصیه بازبینی کنندگان کد برای درخواست بازبینی جدید تعیین نشده استفاده می شود. TIE ابتدا توضیحات تغییر ، مسیرهای فایل و زمان بارگذاری را برای بررسیهای تاریخی در "مرحله ساخت مدل" انجام می دهد. برای مرحله بعدی ، داده های متنی از توضیحات استخراج شده و به عنوان ورودی در مدل داده کاوی ساخته شده در "مرحله ساخت مدل" استفاده می شود. به طور مشابه ، سیستم مسیرهای فایل و زمان بارگذاری را در مدل تشابه ایجاد شده در "مرحله ساخت مدل" وارد می کند.

این دو مدل سپس لیستی از مرورگران کد را ارائه می‌دهند و این دو لیست سپس با استفاده از مدل TIECOMPOSER ساخته شده در "مرحله ساخت مدل" ترکیب می‌شوند.

3. مرحله به‌روزرسانی مدل

در مرحله به روزرسانی مدل ، سیستم TIE با استفاده از مرورگران کد اختصاص داده شده به روز می شود. در عمل ، توسعه دهندگان به طور معمول لیست بازبین های احتمالی را بررسی می کنند و سپس یک درخواست کشش جدید را به گروهی از بررسی کنندگان اختصاص می دهند.

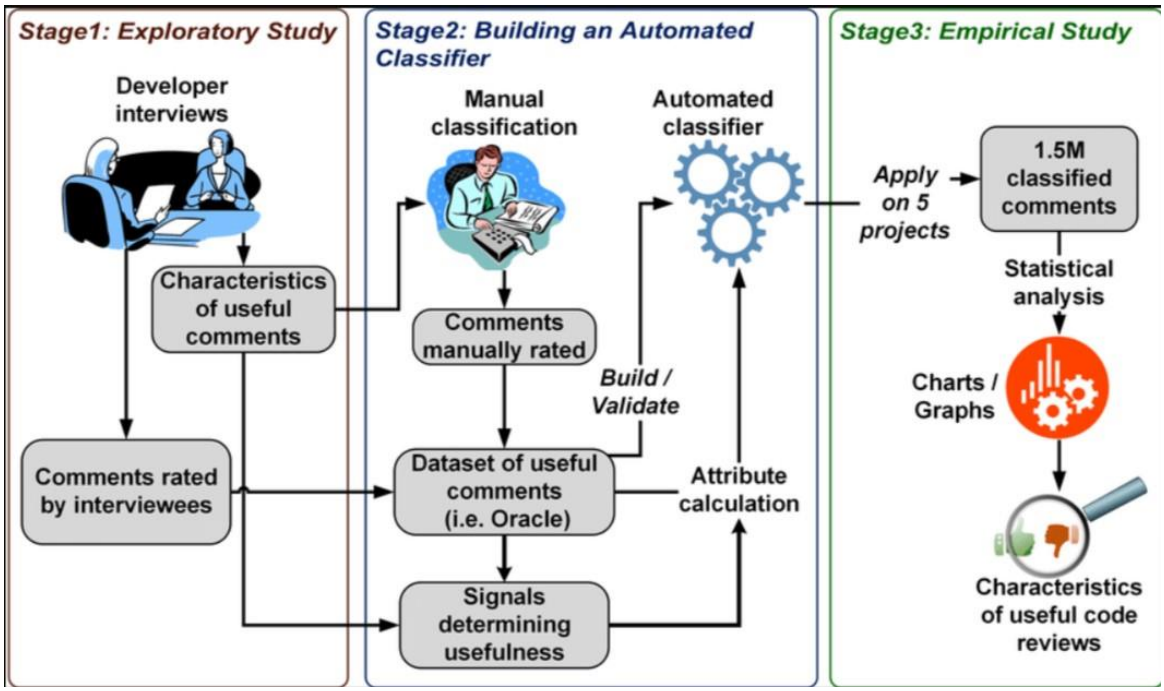
به منظور ارزیابی عملکرد TIE ، نویسندگان از مجموعه داده های ارائه شده توسط Thongtanunam، Tantithamthavorn ، Kula ، استفاده کردند. حاوی 42،045 بررسی و عملکرد TIE را با RevFinder مقایسه کرد. هر یک از بررسی ها در این مجموعه داده ها "ادغام" یا "رها شده" و حداقل یک مسیر فایل را نشان می دهد. مشاهده شد که به طور متوسط در 4 پروژه

منبع باز ، TIE به دقت های پیش بینی بالا 1 ، 3 ، 5 و 10 برتر 0.52 ، 0.73 ، 0.79 و 0.85 و میانگین رتبه متقابل 0.64 (MRR) دست یافت. نتایج RevFinder را به ترتیب 61، ، 33، ، 23، ، 8 and و 37 beat شکست داد.

CodeFlow 5-3-3

Bosu، Greiler ، با استفاده از مصاحبه با توسعه دهندگان و همچنین تجزیه‌وتحلیل نظرات بازبینی پنج پروژه مایکروسافت که با استفاده از CodeFlow CRRS انجام شده بود ، در مایکروسافت ویژگی تجزیه و تحلیل کد مفید را بررسی کرد. مطالعه در سه مرحله انجام شد. ابتدا، آنها با انجام مصاحبه ای با توسعه دهندگان، یک مطالعه اکتشافی انجام دادند تا تفسیر آنها از "مفید" را در زمینه بررسی کد درک کنند. ثانیاً، آنها طبقه بندی کننده ای برای تفکیک نظرات "مفید" و "مفید" با استفاده از داده‌های مصاحبه‌ها ایجاد نمی‌کنند. در نهایت، آنها طبقه بندی خود را برای پنج پروژه مایکروسافت اعمال کردند تا نظرات "مفید" و "مفید" را از هم متمایز کنند. [Bosu M. Greiler, and C. Bird,2015]

شکل: 3.3 روش تحقیق سه‌مرحله‌ای را نشان می‌دهد.



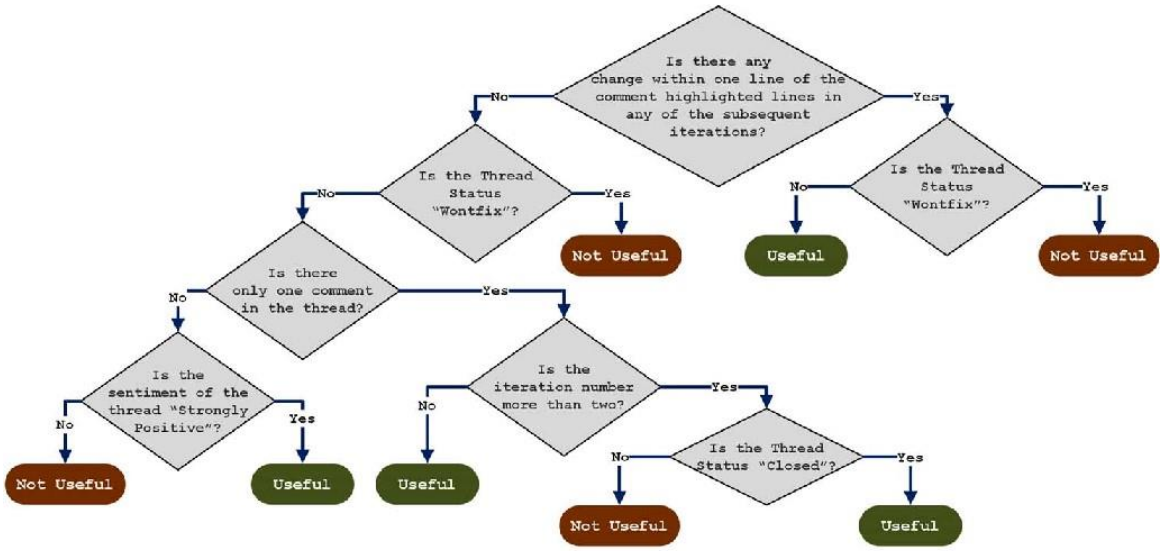
شکل 3-3: روش تحقیق سه‌مرحله‌ای

گردش کار CodeFlow نسبتاً ساده است. ابتدا، نویسنده تغییر نظر را ارسال می‌کند و درخواستی از طریق ایمیل به داور اطلاع داده می‌شود. سپس بازبین می‌تواند تغییر در خود ابزار را مرور کند. هنگامی که یک داور می‌خواهد در مورد یک خط یا بلوک کد نظر دهد، منتقد آن قسمت از کد را برجسته و اضافه می‌کند. این نظرات به عنوان موضوعاتی که در آن بحث شروع می‌شود و همچنین نقاط تعامل برای افرادی که در بررسی مشارکت دارند، ظاهر می‌شود. هر یک از این موضوعات دارای وضعیتی هستند که شرکت کنندگان می‌توانند در طول دوره بررسی تغییر دهند. این وضعیت در ابتدا "فعال" است و با گذشت زمان می‌تواند به "در انتظار" ، "حل شده" ، "برطرف نمی‌شود" و "بسته" تغییر کند. در CodeFlow ، هر به روزرسانی "تکرار" نامیده می‌شود و چرخه بازبینی دیگری را تشکیل می‌دهد. بنابراین، قبل از ادغام تغییر کد در مخزن کد منبع، ممکن است تکرارهای متعددی وجود داشته باشد.

همان‌طور که قبلاً ذکر شد، مطالعه تحقیقاتی در سه مرحله انجام شد که در آن اولین گام به تشخیص نظرات بازبینی کد مفید و غیرمفید بر اساس مصاحبه با توسعه دهندگان کمک کرد. مصاحبه‌های فردی نیمه‌ساختاریافته با توسعه‌دهندگانی که دارای سطوح مختلف تجربه در بازبینی کد و توسعه کد از چهار پروژه مختلف مایکروسافت بودند، انجام شد. از مصاحبه‌شوندگان خواسته شد تا نظرات را از مقیاس 1- 3 (1- مفید، 2- تا حدودی مفید و 3- مفید) امتیاز دهند. نتایج مصاحبه نشان داد که 69 درصد از نظرات مرور یا "مفید" یا "تا حدودی مفید" بودند. نظرات مروری که نشان‌دهنده نقص عملکردی بود، به‌عنوان

نظرات مفیدی در نظر گرفته شد. از سوی دیگر، نظراتی که به دسته‌های: اسناد موجود در کد، نمایش بصری کد (به‌عنوان‌مثال خط خالی یا تورفتگی) ، سازماندهی کد (به‌عنوان‌مثال نحوه تقسیم عملکرد به روش‌ها) و رویکرد راه‌حل توجه شده است. تا حدی مفید است همه نظراتی که یا مثبت کاذب بوده‌اند (به‌عنوان‌مثال به دلیل عدم تخصص هنگامی که داور مشکلی را در کد نشان می‌دهد) یا در هیچ گروهی که قبلاً ذکر شد قرار نگرفتند، به‌عنوان نظرات غیرمفید طبقه‌بندی شدند.

در مرحله دوم، نویسندگان یک طبقه‌بندی خودکار با استفاده از یافته‌های به‌دست‌آمده از مرحله اول ایجاد کردند. به‌منظور ایجاد طبقه‌بندی، نویسندگان نظرات بازبینی را به‌صورت دستی به دودسته مفید و غیرمفید طبقه‌بندی کردند. نظراتی که در مطالعه اکتشافی تا حدی مفید طبقه‌بندی‌شده‌اند، در این مرحله دوم در دسته مفید قرار گرفتند. بر اساس مصاحبه و تجزیه‌وتحلیل دستی، 8 ویژگی بعدی نظرات مشخص شد. بر اساس این ویژگی‌ها و دسته‌ها، "مدل درخت تصمیم برای طبقه‌بندی نظرات مفید" مطابق شکل زیر ساخته شد.



شکل 3-4: مدل درخت تصمیم‌گیری برای طبقه‌بندی نظرات مفید

بر اساس گره‌های تصمیم‌گیری، نظرات به‌عنوان مفید یا مفید طبقه‌بندی می‌شوند. به‌منظور ارزیابی روش پیشنهادی، نویسندگان از نظرات پنج پروژه بزرگ مایکروسافت که شامل Azure، Bing، Visual Studio، Exchange و Office هستند ، استفاده کردند. بر اساس نتایج، نویسندگان موارد زیر را نتیجه گرفتند:

1. توسعه‌دهندگانی که درگذشته تغییراتی را انجام داده یا یک قطعه کد یا یک مصنوع را بررسی کرده‌اند، نظرات مفیدتری ارائه می‌دهند.

2. تفاوت قابل‌توجهی در سودمندی بین نظرات وجود دارد (یعنی آن نظراتی که کلماتی مانند "ثابت" ، "اشکال" یا "حذف" به‌عنوان نظرات "مفید" در نظر گرفته شده‌اند) که توسط بازبینان در همان تیم ساخته شده است و نظرات نویسنده و منتقد از تیم‌های مختلف.

3. تعداد نظرات مفید در طول زمان برای چهار مورد از پنج پروژه افزایش یافت و دلیل این امر افزایش تجربه مرورگران باگذشت زمان در نظر گرفته شد.

در زیر پیامدهای نتایج برای شرکت‌کنندگان در مرور کد و همچنین برای محققان آمده است:

1. این مطالعه نشان داد که تعداد مفید بودن نظرات مرور کد باتجربه توسعه‌دهنده کد افزایش‌یافته است پایه.

2. این مطالعه همچنین نشان داد که با افزایش تعداد فایل‌ها، اثربخشی بررسی‌ها کاهش می‌یابد. پیشنهاد شد که توسعه دهندگان باید تغییرات کوچک‌تر را با تعداد بیشتری فایل برای بررسی ارسال کنند.

3. تراکم مفید بودن نظر می‌تواند توسط تیمی از توسعه دهندگان برای شناسایی مناطقی که بررسی کد در آنها کمتر مؤثر است استفاده شود.

3-3-6 نقد

سادوفسکی، سربرگ، کلیسا و همکاران یک مطالعه موردی انجام داد که در آن آنها یک مطالعه اکتشافی از شیوه‌های مرور کد مدرن در Google انجام دادند.

مطالعه اکتشافی آنها بر 3 جنبه بررسی کد تمرکز داشت:

1) انگیزه‌های بررسی کد

2) شیوه‌های فعلی

3) تفسیر توسعه دهندگان بررسی کد.

به‌منظور ایجاد ساختار بیشتر در بازبینی کد، چندین ابزار در نرم افزار منبع باز (OSS)و تنظیمات صنعتی ظاهر شد. برای این منظور، نویسندگان برخی از رویکردهای مرور مبتنی بر ابزار را مطالعه کردند. این ابزارها شامل CodeFlowمورد استفاده مایکروسافت ، Gerritاستفاده شده توسط Google Chromium، ReviewBoardتوسعه یافته توسط VMware و Phabricatorمورد استفاده فیس بوک است. در زیر مروری کوتاه بر هر یک از این سیستم‌های توصیه مرورگر کد است.

1. CodeFlow: CodeFlow وضعیت هر شخص (توسعه‌دهنده یا بازبینی کننده) و موقعیت آنها در این مرحله (یعنی انتظار، بررسی، امضا) را ردیابی کرد. CodeFlow نویسنده را از ارسال هیچ‌گونه تغییری بدون تأیید منع نمی‌کند و همچنین از گپ ها در موضوعات نظر پشتیبانی می‌کند.

2. Gerrit: Google Chromium از سیستم توصیه بازبینی کد موجود به نام Gerrit استفاده می‌کند که در آن تغییرات تنها پس از تأیید داوران و تأیید خودکار مبنی بر اینکه تغییر ساختار را خراب نمی‌کند، در شاخه اصلی ادغام می‌شوند.

3. ReviewBoard: ReviewBoard توسعه یافته توسط VMware است و هدف آن این است ادغام تجزیه و تحلیل استاتیک در فرایند بررسی این ادغام متکی است

در مورد تغییراتی که نویسندگان به‌صورت دستی درخواست تجزیه‌وتحلیل می‌کنند و در نتیجه کیفیت مرور کد بهبودیافته است.

4. Phabricator: Phabricator ، که توسط فیس بوک استفاده می‌شود، به یک بازبین اجازه می‌دهد تا تغییر را "به عهده بگیرد" و خود آن را انجام دهد. همچنین، این سیستم برای تجزیه و تحلیل استاتیک خودکار یا خطاهای ادغام مداوم رفع می‌کند.

به‌منظور درک روند بررسی کد در Google ، نویسندگان بر دو جنبه اصلی تمرکز کردند: فرایند بازبینی که توسعه دهندگان در طول بررسی‌های خاص تجربه می‌کنند و اینکه آیا توسعه دهندگان با وجود چالش‌ها از بررسی‌های ارائه شده راضی هستند یا خیر. برای بررسی کد در Google ، آنها از CRITIQUE استفاده کردند ، یک ابزار مرور کد داخلی مبتنی بر وب، توسعه یافته داخلی. در این ابزار، یک بازبین می‌تواند تفاوت برجسته تغییرات پیشنهادی را ببیند و همچنین یک بحث موضوعی در مورد خطوط کد با توسعه دهندگان یا سایر بازبینان آغاز کند. CRITIQUE همچنین نمایی از همه عملکردهای ورود به سیستم یک توسعه دهنده، و همچنین تعامل آن با ابزار را شامل بازکردن ابزار، ایجاد تغییرات، مشاهده تفاوت و تأیید تغییرات ارائه می‌دهد. به منظور درک انگیزه توسعه دهندگان برای بررسی کد در Google با استفاده از CRITIQUE و درک توسعه دهندگان در مورد همین، نویسندگان از مصاحبه به عنوان ابزاری برای جمع آوری داده‌ها استفاده کردند.

بر اساس داده‌های جمع‌آوری‌شده از مصاحبه‌های انجام شده، یافته‌های زیر به‌دست‌آمده است.

یافتن 1: بررسی کدهای انجام شده در Google نه‌تنها باهدف تصحیح خطاها یا مشکلات است، بلکه همچنین برای اطمینان از خوانایی و قابلیت نگهداری کد که به‌عنوان جنبه آموزشی در نظر گرفته شد، انجام می‌شود.

یافته 2: انتظارات در مورد بررسی کد خاص بستگی به رابطه مشترک توسعه‌دهنده/نویسنده و مرورگر کد دارد (شکل 3-5)را ببینید.

وقتی صحبت از توسعه‌دهنده و سرپرست پروژه و همچنین اعضای جدید تیم می‌شود، آنها آموزش (آموزش یا یادگیری از طریق مرور کد) را در زمینه مرور کد به اشتراک می‌گذارند. برای توسعه دهندگان و سایر تیم‌ها، آنها در بازبینی دروازه (ایجاد و نگهداری مرزها در اطراف کد منبع) در بررسی کد اشتراک دارند. به طور مشابه، برای توسعه دهندگان و بازبینان خوانایی، آنها هنجارهای حفظ قوانین سازمانی مانند قالب‌بندی یا الگوهای استفاده از APIرا در بررسی کد به اشتراک می‌گذارند. سرانجام، برای توسعه دهندگان و سایر اعضای تیم، آنها پیشگیری از تصادف آموزشی (آموزش اشکالات، نقص‌ها یا سایر مسائل مرتبط با کیفیت) را در بررسی کد به اشتراک می‌گذارند.

شکل 3.5: نمودار رابطه که مضامین انتظارات بازبینی را توصیف می‌کند که عمدتاً در یک زمینه نویسنده/مرور خاص ظاهر می‌شود

تمرین سبک بودن و انعطاف‌پذیری آن فرایند بررسی کد کاملاً با CRITIQUE ترکیب شده است که به شرح زیر عمل می‌کند:
ایجاد نویسندگان شروع به ایجاد، اضافه یا ویرایش یک کد می‌کنند.
پیش نمایش با کمک CRITIQUE ، نویسندگان تفاوت تغییرات و نتایج تجزیه و تحلیل کد خودکار را مشاهده می‌کنند.
نظر دادن نویسندگان/داوران تفاوت رابط کاربری CRITIQUE را می‌بینند و سپس با تغییر از تغییر به نظر دیگر، اظهارنظر می‌کنند.
پرداختن به بازخورد بر اساس نظرات مراحل قبل، نویسندگان یا شروع به پاسخ‌دادن به نظرات می‌کنند و یا بر اساس درخواست‌های ذکر شده در نظرات تغییراتی را آغاز می‌کنند.
تصویب پس از پرداختن به همه نظرات، بازبین‌ها تغییرات را تأیید می‌کنند و آن را با عنوان ") "LGTMبرای من خوب به نظر می‌رسد) علامت‌گذاری می‌کنند.
یافتن 4: بررسی کد در گوگل به نقطه‌ای رسیده است که در مقایسه با پروژه‌های قدیمی، روند بررسی با تغییرات کوچک‌تر سریع‌تر می‌شود. همچنین، یک داور کافی است، در مقایسه با دو داور موردنیاز برای پروژه‌های قدیمی.
یافته 5: علی‌رغم سال‌ها پیشرفت، تعدادی خرابی در برنامه‌نویسی در گوگل رخ‌داده است که بیشتر به پیچیدگی تعاملاتی که حول بررسی‌های کد می‌چرخد ، مربوط می‌شود.
مشاهده شد که در طول یک هفته، تقریباً 70 درصد از تغییرات در کمتر از 24 ساعت پس از ارسال پیام برای بررسی اولیه انجام شد. بر اساس مصاحبه‌ها، همچنین مشخص شد که توسعه دهندگان از الزامات مرور کد راضی هستند، اکثر تغییرات کوچک بوده، نظرات دارای یک مرورگر است و هیچ نظری به جز مجوز تعهد ندارد. این ویژگی‌ها باعث شده است که روند بررسی کد سریع‌تر و سبک‌تر در مقایسه با سایر پروژه‌هایی باشد که از فرایند مشابهی استفاده می‌کنند.

CRRS 7-3-3 مبتنی بر مشخصات

Fejzer، Przymus و Stencelیک سیستم توصیه بازبینی کد بر اساس مشخصات را پیشنهاد کردند. در مدل پیشنهادی داور پیشنهادی، مشخصات بازبینی شامل سابقه بازبینی و تعهدات یک داور بالقوه است.

در مدل توصیه بازبینی آنها، هنگامی که یک درخواست تعهد جدید به مخزن می‌رسد، با نمایش چند مجموعه‌ای از تعهدات (مجموعه‌های متعددی از دنباله کلمات موجود در یک مسیر فایل اصلاح شده در یک تعهد) و همچنین پروفایل بازبین‌ها مقایسه می‌شود. به شباهت بین نمایندگی چند مجموعه‌ای از تعهدات و نمایه‌های بازبینی کنندگان محاسبه می شود و n بازنگری برتر انتخاب می‌شوند. در اینجا، به روز رسانی مشخصات بازبینی کننده یکی از مهم‌ترین و مکررترین عملیات است. هرگاه نظر جدیدی توسط یک بازبین انجام شود، تعهد به نمایه وی اضافه می‌شود. همچنین، وقتی صحبت از مشخصات بالقوه یک داور می‌شود، زمان یکی از عوامل مهمی است که باید مورد توجه قرار گیرد. کاندیدایی که بررسی‌های اخیر یا تعهداتی در نمایه خود داشته باشد، بعنوان کاندیدای محتمل تری برای بررسی درخواست تعهد در نظر گرفته می‌شود.

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

به شرح زیر است:

1. تعداد نظرات به‌ازای هر داور: اکثر داوران کمتر از 20 نظر برای Android و LibreOffice و کمتر از 60 نظر برای OpenStack و Qt ایجاد کردند.

2. مدت فعالیت تک‌نفره داوران: در مورد Android و LibreOffice ، داوران در مقایسه با نظر دهندگان Qt و OpenStack زمان بیشتری صرف کردند. علت احتمالی این نتیجه، نگهدارندگان تعیین شده ای است که برای شرکت‌های مشارکت کننده در این پروژه‌ها کار می‌کنند.

3. مدت بررسی‌های فردی: اکثر بررسی‌ها طی سه روز برای پروژه‌های LibreOffice و OpenStack ، حداکثر دو روز برای Qt و حداکثر شش روز برای پروژه‌های Android تکمیل شد.

3-3-8 طبقه‌بندی سیستم‌ها

طبقه‌بندی سیستم‌ها بر اساس داده‌های مورداستفاده برای ارائه توصیه مرورگر کد انجام شد. داده‌ها شامل مسیرهای مشابه فایل، سابقه مرور 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 3.3: منابع داده برای توصیه‌های بررسی کد در سیستم‌ها

3-3-8-1پروژه مورد استفاده برای ارزیابی

جدا از منبع داده به‌عنوان یکی از عوامل طبقه‌بندی مقالات پژوهشی، نوع پروژه‌ای که این سیستم‌های توصیه مرورگر کد بر روی آن آزمایش شده‌اند را می‌توان به‌عنوان یکی دیگر از عوامل طبقه‌بندی مقالات در نظر گرفت. این پروژه‌ها شامل پروژه‌های منبع‌باز 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

1. پروژه های منبع باز: موارد زیر لیستی از سیستم هایی است که فقط در پروژه های منبع باز ارزیابی شده اند. به منظور ارزیابی ، RevFinder، TIE و CRRS بر اساس Profile از 42،045 بررسی پروژه های منبع باز استفاده کردند که شامل Android Open Source Project (AOSP)، OpenStack، Qt و LibreOffice بود. دلایل متعددی در انتخاب این سیستم ها وجود داشت. ابتدا ، این سیستم ها از سیستم Gerrit به عنوان ابزار بررسی کد استفاده می کنند. دوم ، این سیستم

ها پروژه های نرم افزاری فعال ، بزرگ و واقعی هستند. سرانجام ، هر یک از این سیستم ها یک سیستم بررسی کد خوب دارند که به ایجاد یک مجموعه داده اوراکل خوب برای ارزیابی سیستم توصیه کننده کمک می کند. در زیر به نتایجی که از طریق آزمایش بدست آوردیم اشاره می شود:

(الف) RevFinderبه 10 دقت بالا دست یافت (دقت Top-k درصد بازبینی هایی را محاسبه می کند که یک رویکرد می تواند به درستی مرورگران کد را در کل تعداد بازبینی ها توصیه کند) به ترتیب 86٪ ، 87٪ ، 69٪ و 74٪ برای Android، OpenStack، Qt و .LibreOfficeبه طور متوسط ، برای the 79 از بررسی ها ، RevFinder به مرورگران کد CoRReCT با 10 توصیه برتر توصیه کرد.

ب) مشاهده شد که به طور متوسط در 4 پروژه TIEبه بهترین 1 ، top-3، top-5 و top-10 دقت پیش بینی و مقادیر MRR 0.52، 0.73 ، 0.79 ، 0.85 و 0.64 که به ترتیب 61، ، 33، ، 23، ، 8 and و 37 results از نتایج RevFinder بهتر است.

ج) مشابه TIE و RevFinder، CRRS بر اساس پروفایل با استفاده از 4 سیستم منبع باز یعنی Android، OpenStack، Qt و LibreOffice آزمایش شد. مشاهده شد که برای LibreOffice و OpenStack ، اکثر بررسی‌ها در عرض سه روز انجام شد، برای Android تا 6 روز و برای Qt تا 2 روز به طول انجامید.

2.پروژه های تجاری: [11] CodeFlowبا استفاده از پنج پروژه مایکروسافت آزمایش شده است که شامل Azure، Bing ، Visual Studio، Exchange و Office است. مشاهده شد که نظرات مفید دریافت شده از نظر دهندگان در Azure از 60 to به 66، افزایش یافته است ، 62 to به 67٪ در بینگ ، 60 تا 70 درصد در ویژوال استودیو، 60 تا 68 درصد در آفیس و 60 تا 65 درصد در مبادله.

3. **پروژه‌های منبع‌باز و تجاری:** برخی از سیستم‌ها که با استفاده از پروژه‌های تجاری و پروژه‌های منبع‌باز مورد آزمایش قرار گرفته‌اند که در زیر با نتایج به‌دست‌آمده از آزمایشات انجام شده ذکر شده است.

CoRReCT (a)با استفاده از 17115 درخواست کشش از ده پروژه تجاری و شش پروژه منبع‌باز آزمایش شد. معیارهای عملکردی که نویسندگان در اینجا استفاده کرده اند شامل Top-K Accuracy،میانگین رتبه متقابل (MRR)، میانگین دقت (MP)و میانگین فراخوانی(MRR).

•هنگام آزمایش روی پروژه های منبع باز ، مشخص شد که CoRReCT دارای دقت Top-k 85.20 است در حالی که برای پروژه های تجاری دقت Top-K 92.15 was به دست آمد.

• CoRReCT نتیجه prec 85.93 برای پروژه‌های تجاری و دقت for 84.76 برای پروژه‌های منبع‌باز به دست آورد.

•برای پروژه های تجاری ، CoRReCT فراخوانی از شرکت را برگرداندwhere 81.39 در حالی که برای پروژه منبع باز سیستم به دست آورد 78.73 درصد فراخوان.

• CoRReCT مقدار MRR 0.62 را برای تجارت به دست آورد درحالی‌که برای پروژه‌های منبع باز ، نویسندگان ارزش MRR را ذکر نکرده اند ، اما نسبتاً بیشتر است.

cHRev (b)در 3 پروژه منبع باز (Android، Mylyn و Eclipse)و یک پروژه تجاری (MS Office)مورد ارزیابی قرار گرفت. مشاهده شد که فراخوان و دستاوردهای دقیق به دست آمده برای MS Office بهتر از مواردی بود که در Android، Mylyn و Eclipse برای cHRev به دست آمد.

4. بدون پروژه (مصاحبه)

CRITIQUEکه به عنوان سیستم توصیه بازبینی کد در Google استفاده می‌شود از روش مصاحبه برای ارزیابی سیستم خود استفاده می‌کند. مشاهده شد که توسعه دهندگان به طور متوسط 2.6 ساعت در هفته به بررسی تغییرات پرداختند که در مقایسه با 6.4 ساعت/هفته زمان خودکار برای پروژه‌های منبع باز کم بود.

3-4 خلاصه

بر اساس مطالعه مروری بر ادبیات انجام شده، ما هفت سیستم توصیه‌کننده مرورگر کدگذار را یافتیم: cHREv، CoRReCT، CRRS مبتنی بر مشخصات، RevFinder، CodeFlow، TIE وCRITIQUE.

این سیستم‌ها بر اساس دو بعد تقسیم می‌شوند: منبع داده مورداستفاده برای ساخت سیستم و نوع پروژه مورداستفاده برای ارزیابی سیستم.

فصل 4
نیازهای اطلاعاتی مرورگران کد

4-1 روش‌شناسی

به‌منظور انجام نظرسنجی از مهندسان نرم‌افزار برای تعیین نیازهای اطلاعاتی برای مرورگران کد، ما از مراحل زیر برای اطمینان از نتایج صحیح، غیر مغرضانه و دقیق استفاده کردیم.

4-1-1 بررسی غربالگری

نظرسنجی ما برای مهندسان نرم‌افزار به دو قسمت تقسیم می‌شود که قسمت اول آن یک بررسی غربالگری است. ما از نظرسنجی غربالگری استفاده کردیم تا مطمئن شویم از اعضای توسعه محصول نرم‌افزاری که در سیستم‌های توصیه بازبینی کد تجربه دارند و بنابراین می‌توانند اطلاعات دقیق و بی‌طرفانه ارائه دهند، پاسخ می‌دهیم. در زیر سؤالاتی است که ما برای بررسی غربالگری خود گنجانده‌ایم.

1-لطفاً آدرس ایمیل خود را وارد کنید.
2 - چند سال/سال تجربه توسعه نرم‌افزار دارید؟
الف) کمتر از 1 سال
ب) 1-2 سال/ثانیه
ج) 3-5 سال
د) 6-10 سال
ه) 11+ سال
3-آیا شما 20 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 1 سال
ب) 1-2 سال/ثانیه
ج) 3-5 سال
د) 6-10 سال
ه) 11+ 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؟

4-1-2 سؤالات نظرسنجی سیستم‌ها و ابزارهای توصیه توصیف‌کننده اطلاعات جمعیتی و ابزار

یک نظرسنجی جمعیت شناختی و تجربه CRRS به افرادی که مرحله غربالگری را پشت سر گذاشته اند داده شد، یعنی شرکت کنندگان حداقل دو سال سابقه کار و تجربه استفاده از CRRSها را داشتند. این سوالات به ما کمک کرد تا نیازهای اطلاعاتی مرورگران کد را درک کنیم، آنها چه ویژگی‌هایی را در سیستم‌های توصیه کننده مرورگر کد مهم می‌دانند و چه ویژگی‌هایی را در سیستم‌های موجود از دست داده 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
2. گروه سنی شما چیست؟
(الف) 20-25
(ب) 26-35
(ج) 36-45
(د) 46-55
(ه) 56-60
(و) بالای 60
3. موقعیت جغرافیایی شما چیست؟
الف) اروپا
ب) آفریقا
ج) آمریکای جنوبی
(د) آمریکای شمالی
ه) آسیا
(و) استرالیا
(ز) نیوزلند
ح) جزایر اقیانوس آرام
4. تیم پروژه شما در چه اندازه‌ای است؟
الف) من به‌صورت جداگانه روی پروژه‌هایم کار می‌کنم
(ب) 2-7 نفر
(ج) 8-12 نفر

(د) 13-20 نفر

(ه) 21-40 نفر

(و) بیش از 40 نفر

5. 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

6. با کدام یک از سیستم‌های توصیه کننده مرورگر کد (CRRS)زیر 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

7. از کدام CRRS استفاده کرده‌اید؟

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

9. کدام ویژگی های 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

10. 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

11. چه معیارهایی در لیست بالا وجود نداشت؟

چه اهمیتی به آنها می‌دهید؟

12. کدام یک از ویژگی های رابط کاربری ((UIزیر باعث می شود تجربه کاربری ((UXتعاملی تر ، نزدیک تر و راحت تر 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.
13. 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؟
14- در مرور کد به دنبال چه موارد خاصی هستید (مانند تعداد سال‌ها کار)

تجربه، زمینه تخصص و غیره؟
15. 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
16. 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)

2-4 نتایج

ما 27 پرس‌وجو در مورد نظرسنجی خود به دست آوردیم، اما تنها 15 مورد از آنها به جلو حرکت کردند و به بررسی غربالگری ما پاسخ دادند. از بین این 15 پاسخ، ما 4 پاسخ را فیلتر کردیم که با استفاده از نظرسنجی غربالگری حداقل معیارها را نداشتند. در زیر نتایج به‌دست‌آمده برای نظرسنجی سیستم‌ها و ابزارهای توصیف مرورگر جمعیت‌شناختی و کد ارائه شده است.

1.نقش شغلی شرکت کنندگان

بر اساس نتایج به‌دست‌آمده، مشخص شد که اکثر شرکت‌کنندگان یا توسعه دهندگان، یا مهندسان 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 4.1: نقش‌های شغلی شرکت‌کنندگان

2.موقعیت جغرافیایی شرکت‌کنندگان

اکثر شرکت‌کنندگان از آسیا بودند و درصد معادل باقی‌مانده افراد از آمریکای شمالی و آمریکای جنوبی بودند.

درصد	رشته
9.09	آمریکای جنوبی
9.09	شمالی عامری CA
81.82	آسیا

جدول 4.2: موقعیت جغرافیایی شرکت‌کنندگان

3 .اندازه تیم پروژه

از همه شرکت‌کنندگانی که در این مطالعه شرکت کردند، 45.45٪ آنها به ترتیب در تیمی از 2-7 نفر و 8-12 نفر کارکرده‌اند/در حال کار هستند درحالی‌که 9.09٪ از افراد کارکرده‌اند/در گروهی بیش از 40 نفر.

درصد	رشته
45.45	2-7نفر
45.45	8-12نفر
9.09	بیش از 40 نفر

جدول 4.3: اندازه تیم پروژه

4.توزیع تیم

همچنین در پایان‌نامه مورد بررسی اطلاعاتی در مورد نحوه توزیع تیم‌ها جمع‌آوری کردند، به این معنی که آنها در تیم‌هایی کار می‌کنند که در محل مشترک یا توزیع شده‌اند. مشاهده شد که 45.45 of از تیم‌ها در محل مشترک ، 18.18 were توزیع شده و 36.36 were ترکیبی از هر دو مکان مشترک و توزیع شده بود.

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

جدول 4.4: توزیع تیم

5 .آشنایی با CRRS در بین شرکت‌کنندگان

اکثر شرکت‌کنندگان در نظرسنجی پایان نامه مورد بررسی با GitHub/GitLab آشنا بودند. از سوی دیگر ، هیچ کس با Gerrit Code Review System (Chromium)و ReviewBoard آشنا نبود. مشاهده شد که 47.62٪ از شرکت کنندگان با GitHub/GitLab ، ٪23.81 از شرکت کنندگان با BitBucket ، ٪14.29 از شرکت کنندگان با Code Flow Review Tool(مایکروسافت) ، ٪9.52 با سایر ابزارهای بازبین(که شامل SVN) است آشنا بودند.و 4.76 of از شرکت کنندگان با Phabricator آشنا بودند.

درصد انتخاب ها	CRRS
٪0	سیستم بازبینی کد(Gerrit (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 4.5: آشنایی با CRRS در بین شرکت‌کنندگان

6 .مفید بودن ویژگی های CRRS

ما تعدادی از ویژگی‌های CRRS را برای شرکت‌کنندگان خود لیست کردیم و از آنها خواستیم همه آن ویژگی‌هایی را که برای آنها مفید بوده است انتخاب کنند. موارد زیر ویژگی‌هایی است که ما برای شرکت‌کنندگان در نظر گرفتیم و درصد مشارکت‌کنندگان آن را مفید دانستند.

(الف) بحث کد با نسخه‌های قدیمی و جدید که برای نشان‌دادن تغییر کد مشخص شده است: ٪25.53 از شرکت‌کنندگان آن را مفید دانستند.

(ب) ادغام نرم‌افزارهای ردیابی پروژه (مانند Trello، Jira و غیره): ٪20.59 از شرکت‌کنندگان آن را مفید می‌دانند.

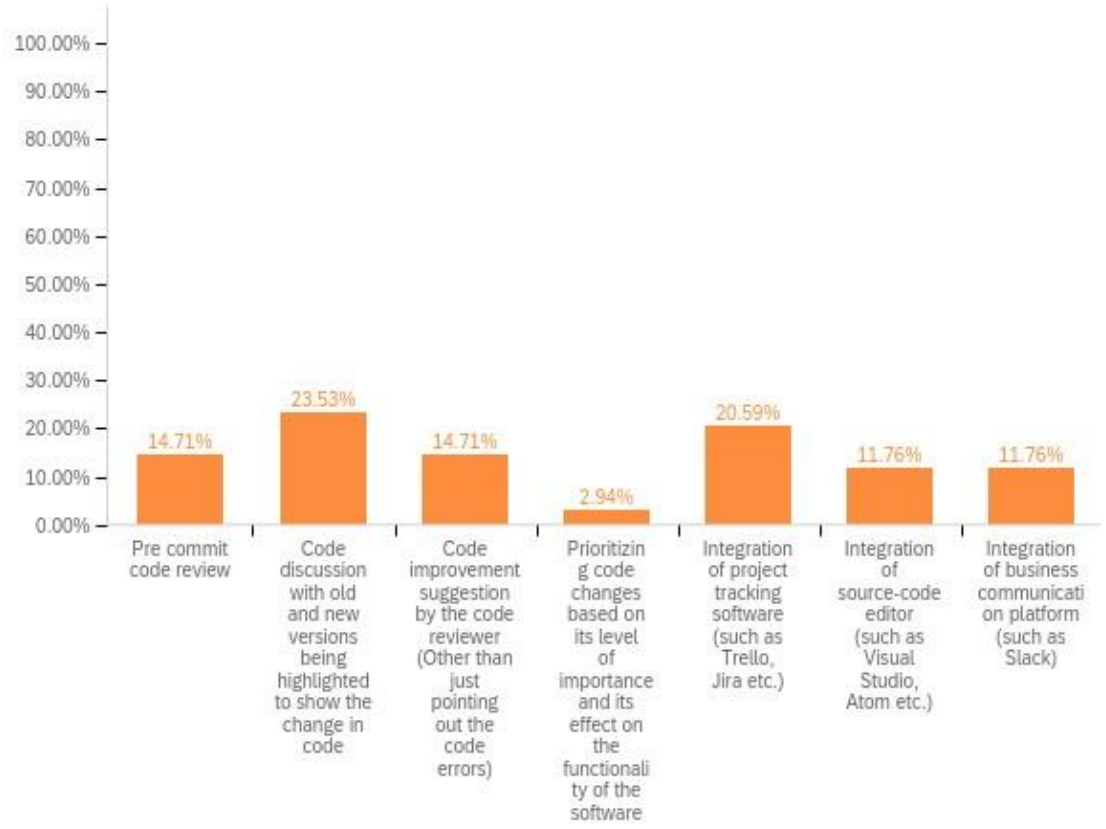
(ج) بررسی کد قبل از انجام: 14.71 of از شرکت‌کنندگان آن را مفید دانستند.

(د) پیشنهاد بهبود کد توسط مرورگر کد (غیر از فقط اشاره به خطاهای کد): 14.71 of از شرکت‌کنندگان دریافتند مفید است

ه) ادغام بستر ارتباطات تجاری (مانند 11.76 :(Slack٪ از شرکت کنندگان آن را مفید دانستند.

(و) ادغام ویرایشگر کد منبع (مانند Visual Studio، Atom و غیره): 11.76 of از شرکت‌کنندگان آن را مفید دانستند.

(ز) اولویت‌بندی تغییرات بر اساس میزان اهمیت و تأثیر آن بر عملکرد نرم‌افزار: ٪2.94 از شرکت‌کنندگان آن را مفید می‌دانند.



شکل 4.1: سودمندی گزارش شده از ویژگی های CRRS

.7. معیارهای انتخاب مرورگر کد

هنگام انتخاب مرورگر کد عوامل متعددی باید موردتوجه قرار گیرد. ما چندین مورد از آنها را فهرست کردیم که شرکت‌کنندگان اهمیت هر یک از آنها را انتخاب کردند. شرکت‌کنندگان اهمیت هر یک از این ویژگی ها را در مقیاس Likert اعم از بسیار محتمل تا فوق العاده بعید 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 4.2: معیارهای انتخاب مرورگر کد

الف) تعدادی سال سابقه کار

• از همه شرکت کنندگان 9.09 them از آنها این ویژگی را بسیار محتمل می‌دانند ، 72.73 them آنها آن را تا حدی محتمل می‌دانند ، 9.09 them از آنها آن را بعید یا غیر محتمل می‌دانند ، 9.09 them آنها را تا حدودی بعید می‌دانند در حالی که هیچ کس آن را در نظر نگرفته است به عنوان بسیار بعید

(ب) تخصص بازبینی کنندگان کد در زبان برنامه‌نویسی

• مشاهده شد که 54.55 of از شرکت‌کنندگان این ویژگی را بسیار محتمل می‌دانند ، 36.36 them آنها آن را تا حدی محتمل ، 9.09 them از آنها آن را بعید یا غیر محتمل می‌دانند در حالی که هیچ کس آن را تا حدی بعید یا بسیار بعید نمی‌داند.

(ج) تخصص مرورگر کد در یک حوزه به‌عنوان‌مثال: مهندس نرم‌افزار-

هوش مصنوعی و غیره

• برای این معیارها ، 45.45 of از شرکت کنندگان این ویژگی را بسیار محتمل می دانند ، 45.45 them آنها آن را تا حدی محتمل ، 9.09 them از آنها آن را بعید یا غیر محتمل می دانند در حالی که هیچ کس آن را تا حدی بعید یا بسیار بعید نمی داند.

(د) زبان ارتباط بین مرورگر کد و توسعه‌دهنده نرم‌افزار

• از بین همه شرکت‌کنندگان 54.55 درصد آنها این ویژگی را بسیار محتمل، 18.18 درصد آنها آن را تا حدودی محتمل، 27.27 درصد از آنها آن را بعید یا غیرمحتمل در نظر گرفتند، درحالی‌که هیچ‌کس آن را تا حدی بعید یا بسیار بعید نمی‌دانست.

(ه) نقش مرورگر کد

• برای این معیارها، 18/18 درصد از شرکت‌کنندگان این ویژگی را بسیار محتمل می‌دانند، 45/45 درصد آنها آن را تا حدی محتمل، 36،36 درصد از آنها آن را بعید یا غیرمحتمل می‌دانند درحالی‌که هیچ‌کس آن را تا حدی بعید یا بسیار بعید نمی‌داند.

(و) تعداد پروژه‌هایی که روی آنها کار شده است

• همچنین مشاهده شد که 9.09 of از شرکت‌کنندگان این ویژگی را بسیار محتمل می‌دانند ، 63.64 them از آنها تا حدی محتمل ، 27.27 them آنها را تا حدودی بعید می‌دانند در حالی که هیچ کس آن را بسیار بعید و نه محتمل و نه بعید نمی‌داند.

(ز) تعداد بررسی کد انجام شده

• برای این معیارها ، 27.27 of از شرکت کنندگان این ویژگی را بسیار محتمل می دانند ، 45.45 them از آنها تا حدی محتمل ، 9.09 them آنها را تا حدودی بعید می دانند ، 18.18 them از آنها آن را بعید یا غیر محتمل می دانند در حالی که هیچ کس در نظر نگرفته است بسیار بعید است

ما همچنین از شرکت‌کنندگان خواسته‌ایم تا برخی از ویژگی‌های دیگر را به‌غیراز موارد ارائه شده ارائه دهند که هنگام انتخاب مرورگر کد، آنها را به‌عنوان ویژگی‌های مهم در نظر گرفته‌اند. ویژگی‌های گزارش شده عبارت‌اند از:

• اطمینان از اینکه داور خود را با فناوری و زبان برنامه‌نویسی به‌روز می‌کند (شرکت‌کننده این ویژگی را بسیار محتمل می‌داند)

• تعداد درخواست‌های کشش اختصاص‌داده‌شده به داوران.

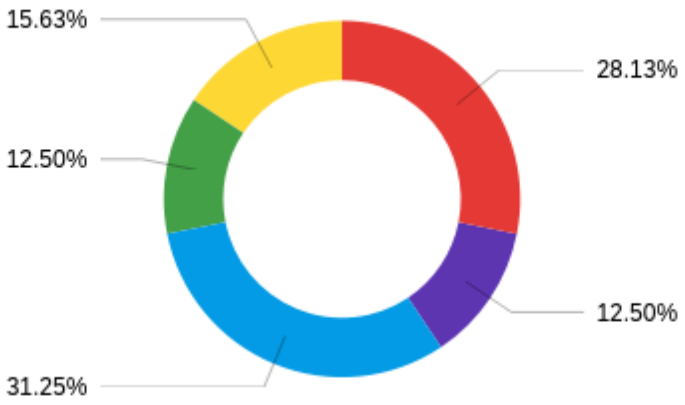
8. رابط کاربری CRRS (UI):

به غیر از ویژگی های CRRS ، ما همچنین در مورد ویژگی های مختلف UI که می توان برای CRRS مفید دانست ، با تعامل بیشتر ، تجربه کاربری و دسترسی راحت ، برای CRRS مفید واقع شد. درصد افرادی که ویژگی های زیر را مفید دانسته اند به شرح زیر با توزیع نمودار پای در زیر نشان داده شده است:

(الف) ارائه خط لوله‌ای که نشان می‌دهد پروژه در کدام مرحله است، یعنی ساخت، آزمایش، بازبینی کد، استقرار و غیره: 31.25٪ از شرکت کنندگان این ویژگی UI را مفید و راحت‌تر برای پیگیری پروژه می‌دانند.

(ب) وجود داشبورد برای هر فرد که داده‌های آماری کلیه اقدامات انجام شده را نشان می‌دهد (مانند تعداد تعهدات، تعداد بررسی کد انجام شده، تعداد خطاها/هشدارهای کد در پروژه فعلی و غیره): 28.13 of از شرکت‌کنندگان این ویژگی برای جستجوی جزئیات و اقدامات انجام شده توسط هر فرد مفید بود.

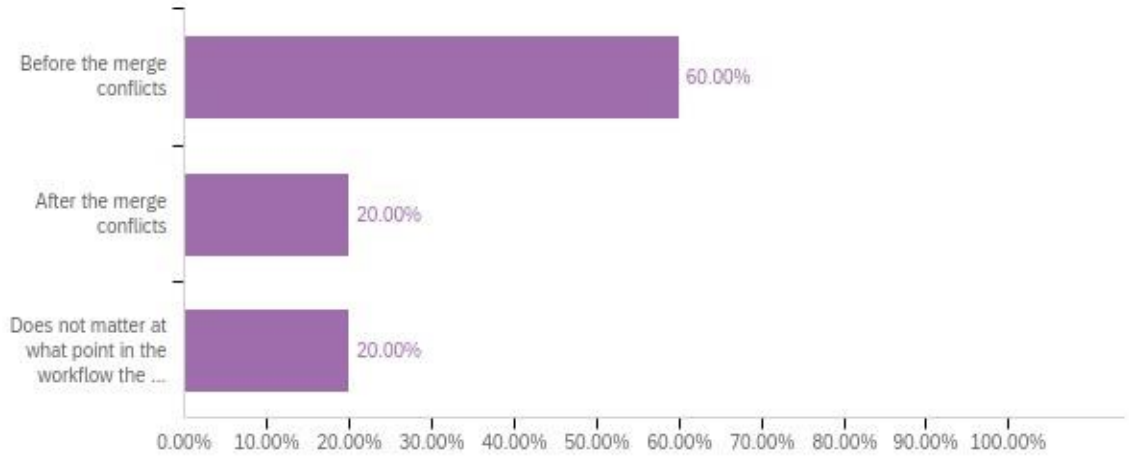
ج) هنگام تغییر کد، بحث کد جدید و قدیمی در مورد کد: 15.63 of از شرکت‌کنندگان این ویژگی را مفید می‌دانند تا روند مرور کد را برای مرورگر و توسعه‌دهنده آسان‌تر کند.
(د) انتقال کد با استفاده از طرح رنگی با نام برنامه‌نویس/برنامه: 12.50 the از شرکت‌کنندگان این ویژگی را مفید دانستند.
ه) گزینه‌ای برای انتخاب یک "شاخه/پرونده" خاص در یک پروژه برای حفظ گردشکار سیستماتیک و یک روش بازبینی کد سازمان‌یافته: 12.50٪ از شرکت‌کنندگان با سازماندهی بیشتر فرایند بررسی کد، این ویژگی را مفید دانستند.



شکل 4.3: ویژگی های UI CRRS

9. ترجیح زمان توصیه برای بررسی کد:

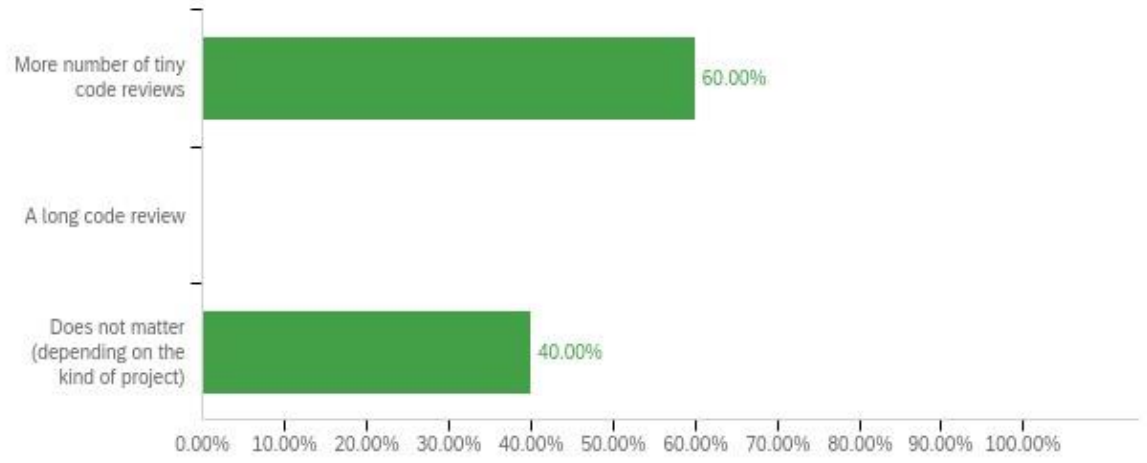
توصیه بازبینی کد می‌تواند در مراحل مختلف فرایند توسعه نرم‌افزار انجام شود که شامل قبل از درگیری ادغام، پس از ناسازگاری ادغام یا در هر نقطه از گردشکار می‌شود. مشاهده شد که 60 of از شرکت‌کنندگان معتقد بودند که توصیه کد باید قبل از درگیری ادغام انجام شود ، 20.00 them از آنها معتقد بودند که باید پس از درگیری ادغام انجام شود در حالی که تا 20.00 them از آنها مهم نیست که در کدام نقطه از گردش کار بررسی کد انجام شده است.



شکل 4.4: اولویت زمانی که توصیه بازبینی کد وجود دارد

10. نوع مرور کد

از شرکت‌کنندگان خواسته شد که چه نوع بازبینی کد را ترجیح دهند که شامل بسیاری از بررسی‌های کد کوچک‌تر یا مرور طولانی کد یا بستگی به نوع پروژه دارد. مشاهده شد که 60.00 of از شرکت‌کنندگان ترجیح می‌دهند بسیاری از بررسی‌های کد کوچک‌تر را انجام دهند درحالی‌که بقیه (40.00٪) از شرکت‌کنندگان احساس می‌کردند که مهم نیست. هیچ یک از شرکت‌کنندگان ترجیح می‌دهند که بررسی طولانی کد را انجام دهند.



شکل 4.5: نوع مرور کد

4-3 برخی از روندها و الگوهای مشاهده شده

بر اساس نتایج به‌دست‌آمده، برخی از الگوها و روندها بین نتایج دو یا چند س surveyال نظرسنجی یافت شد. برخی از روندهایی که مشاهده کردیم به شرح زیر است:

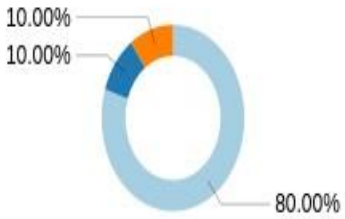
(الف) نوع CRRS مورداستفاده و نقش شغلی

بر اساس مشاهدات ما، مشخص شد که توسعه‌دهنده تقریباً از تمام سیستم های CRRS که در سوال ذکر کرده بودیم مطلع بود در حالی که DevOps Engineer فقط از یک CRRS مطلع بود ، سرپرست تیم از 2 CRRS و محصول مطلع بود. مالک در مورد 4 CRRS می‌دانست.

Gerrit Code Review System (Chromium)

NO DATA

GitHub/ GitLab

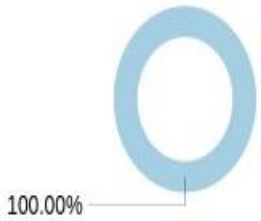


Code Flow Review Tool (Microsoft)

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

شکل 4-6 : نوع CRRS و نقش شغلی

ما نتایج زیر را برای هر یک ازCRRS های زیر به همراه توزیع نمودار پای در شکل4.4 به دست آوردیم.

- **Github/GitLab**:از شرکت کنندگانی که با این CRRS آشنا بودند ، 10٪ از شرکت کنندگان سرپرست تیم بودند ، 10٪ دیگر صاحبان محصولات و بقیه (80٪) توسعه دهندگان/برنامه نویسان/مهندسان نرم افزار بودند.
- **دوم ابزار بازبینی کد جریان (مایکروسافت):** از شرکت کنندگان که با این CRRS آشنا بودند ، 33.33٪ توسعه دهندگان/برنامه نویسان، مهندسان نرم افزار، DevOps Engineer/Infrastructure Developerودیگران بودند
- **Phabricator :**توسعه دهندگان/برنامه‌نویسان/مهندسان نرم‌افزار تنها افرادی بودند که با Phabricator آشنا بودند
- **BitBucket:**از شرکت کنندگانی که با این CRRS آشنا بودند،60.00٪ آنها توسعه دهندگان/برنامه نویسان/مهندسان نرم افزار بودند، 20.00٪ آنها رهبران تیم و دیگران بودند.
- **دیگران**: تعدادی CRRS دیگر در بازار وجود دارد و از شرکت‌کنندگان خواسته شد در صورت اطلاع از CRRSهای دیگری که در نظرسنجی ذکر نشده‌اند، آن گزینه را انتخاب کنند. زیرا، ما 50.00 respons پاسخ از توسعه دهندگان و دیگران دریافت کردیم.

از بین شرکت‌کنندگانی که در نظرسنجی ما شرکت کردند، هیچ یک از آنها با دو مورد از سیستم‌ها/ابزارهای توصیه بازبینی کد آشنا نبودند ReviewBoard (VMware) و Gerrit Code System Review (Chromium)

(ب) ویژگی های CRRS و نقش شغلی

ما تعدادی ویژگی CRRS را برای شرکت‌کنندگان مطرح کردیم که آنها معتقد بودند استفاده از CRRSها راحت‌تر و آسان‌تر است. شکل 4.7 نشان می‌دهد که نقش شغلی کدام ویژگی 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 4-6 : ویژگی های CRRS و نقش شغلی

1) بررسی کد قبل از انجام:
مشاهده شد که از همه شرکت‌کنندگانی که موافقت کردند که این ویژگی را مهم بدانند 60٪ آنها توسعه‌دهنده بودند ، 20٪ مهندس/DevOps توسعه دهنده زیرساخت و دیگران (یعنی صاحب محصول).
2) بحث کد با نسخه‌های قدیمی و جدید که برای نشان‌دادن تغییر کد مشخص می‌شوند:
از بین همه شرکت کنندگان ، 75 them از آنها توسعه دهنده ، 12.50 them از آنها رهبران تیم و مهندس/توسعه دهنده زیرساخت DevOps بودند که این ویژگی CRRS را مهم می دانستند.
3) پیشنهاد بهبود کد توسط مرورگر کد (به جز اشاره به خطاهای کد): از همه شرکت‌کنندگانی که این ویژگی را مفید دانستند، 40.00٪ آنها توسعه‌دهنده، 20.00٪ آنها رهبران تیم ، مهندس/DevOps توسعه دهنده زیرساخت و دیگران (صاحب محصول).
4) اولویت‌بندی تغییرات کد بر اساس میزان اهمیت آن و تأثیر آن بر عملکرد نرم‌افزار: از بین همه شرکت‌کنندگان، فقط صاحب محصول این ویژگی را مفید دانست.
5) ادغام نرم‌افزارهای ردیابی پروژه (مانند Trello، Jira و غیره): برای این ویژگی، 71.43٪ توسعه دهندگان، 14.29٪ رهبران تیم و مدیر پروژه مفید دانستند.
6) ادغام ویرایشگر کد منبع (مانند Visual Studio، Atom و غیره): فقط توسعه دهندگان از همه شرکت‌کنندگان بودند که این ویژگی را به‌عنوان یک ویژگی مهم در نظر گرفتند.
7) ادغام بستر ارتباطات تجاری مانند (Slack) :این ویژگی فقط توسط توسعه دهندگان همه شرکت‌کنندگان که در نظرسنجی ما شرکت کردند مفید واقع شد.

4.4 خلاصه

ما نتایج به‌دست‌آمده را با بررسی طیف وسیعی از اعضای پروژه نرم‌افزار در اینجا ارائه کردیم که در آن متوجه شدیم کدام ویژگی CRRS مفیدتر است ، چه ویژگیهایی در سیستم موجود وجود ندارد و چه عواملی هنگام انتخاب یک مرورگر کد مربوط مهم است. ما همچنین ترجیح توسعه دهندگان/داوران را نسبت به نوع بررسی کد (بررسی طولانی یا کوتاه) و در چه مرحله ای از گردش کار باید بدست آوریم. به غیر از این ، ما برخی از گرایش ها و الگوها را بین استفاده از سیستم CRRS و ارتباط آن با اطلاعات جمعیت شناختی منتقد/توسعه دهنده پیدا کردیم.
در فصل زیر به سؤالات تحقیق خود پاسخ داده و بحث می‌کنیم

فصل 5

5-1 بحث

این بخش به سؤالات تحقیق ما پاسخ می‌دهد و بحث می‌کند. با انجام مرور ادبیات سیستماتیک، ما پاسخ به سه سؤال اول تحقیق را پیدا کردیم و با استفاده از نظرسنجی، پاسخ دو سؤال اخیر تحقیق را پیدا کردیم.
با انجام یک مرور ادبیات سیستماتیک (SLR)ما برخی از راه‌حل‌های موجود برای سیستم‌های توصیه برای مرورگران کد را شناسایی کردیم، عواملی که هنگام ایجاد CRRS و دسته‌بندی CRRSهای موجود باید در نظر گرفته شوند. از سوی دیگر، با انجام نظرسنجی، ما ویژگی‌هایی را که برای یک سیستم توصیه برای مرورگران کد مهم هستند و چه پیشرفت‌هایی می‌توان در CRRSهای موجود انجام داد، دریافتیم.
سؤال 1: راه‌حل‌های موجود برای سیستم‌های توصیه برای مرورگران کد چیست؟
پاسخ: ما تعدادی مقاله را بررسی کردیم و تعدادی از سیستم‌های توصیه بازبینی کد موجود را پیدا کردیم. این سیستم‌ها/ابزارها شامل cHRev، REVFINDER، CoRReCT، TIE، CodeFlow، ReviewBoard، Phabricator، Gerrit، rDevX و CRRS مبتنی بر پروفایل است که بر اساس عوامل متعددی/انواع داده توصیه‌هایی را ارائه می‌دهند. این نوع داده‌ها شامل سابقه مرور کد، شباهت مسیر پرونده، تجربه پروژه و فناوری مربوطه، استخراج متن و مکان‌یابی فایل و وضعیت ردیابی هر داور یا نویسنده است.
سؤال 2: هنگام ایجاد یک سیستم توصیه برای مرورگران کد، چه عواملی باید در نظر گرفته شوند؟ پاسخ: فاکتور اصلی که هنگام ایجاد یک سیستم توصیه برای مرورگران کد باید موردتوجه قرار گیرد، معیارهای ارزیابی منبع داده یا نوع پروژه‌ای است که سیستم روی آن آزمایش شده است (یعنی منبع‌باز یا تجاری یا هر دو). وقتی نوبت به توصیه بازبینی کننده کد بر اساس مشخصات مرورگر می‌رسد، لازم است مشخصات مرورگر که شامل مرور قبلی و سابقه تعهد است به‌روز شود. به طور مشابه، وقتی صحبت از سابقه بازبینی گذشته می‌شود، مهم است که مخزن/مجموعه‌داده‌های بررسی‌های گذشته را به‌روز کنید تا در آینده بر اساس مرورهای گذشته، مرورگران کد مربوطه را توصیه کنید.
سؤال 3: چگونه می‌توان سیستم‌های توصیه‌ای موجود برای مرورگران کد را دسته‌بندی کرد؟
پاسخ: سیستم‌های توصیه‌ای موجود بر اساس نوع داده طبقه‌بندی‌شده‌اند که شامل سابقه مرور کد، شباهت مسیر پرونده، تجربه پروژه و فناوری مرتبط، استخراج متن است.

و موقعیت فایل، وضعیت ردیابی هر داور یا نویسنده. cHRev یک سیستم توصیه بازبینی کد بود که بر اساس سابقه مرور کد، مرورگران کد را توصیه می کرد. REVFINDER CRRS دیگری بود که مرورگران کد را بر اساس شباهت مسیر فایل توصیه می‌کرد. به طور مشابه ، ما یک CRRS به نام CoRReCT پیدا کردیم که هدف آن توصیه بازبینی کنندگان کد بر اساس پروژه و فناوری مرتبط بود. TIE (Text mIning and a filE) همانطور که از نامش مشخص است، مرورگرهای کد را با کمک متن کاوی و فایل توصیه می‌کند.
محل.
سؤال 4:ویژگی‌های مهم سیستم توصیه برای مرورگران کد چیست؟
پاسخ: بر اساس نظرسنجی اعضای پروژه نرم‌افزاری که انجام شد، تعدادی ویژگی پیدا کردیم که برای یک سیستم توصیه برای مرورگران کد مهم تلقی می‌شد که شامل موارد زیر است:
پاسخ: بر اساس نظرسنجی اعضای پروژه نرم‌افزاری که انجام شد، ما تعدادی ویژگی پیدا کردیم که برای یک سیستم توصیه برای مرورگران کد مهم تلقی می‌شد که شامل موارد زیر است:

1. بحث کد با نسخه‌های قدیمی و جدید که برای نشان‌دادن تغییر کد مشخص می‌شوند.
2. ادغام با یک نرم افزار ردیابی مسئله مانند JIRA ،Trello
و غیره.
3. بررسی کد را از قبل مرتکب شوید.
4. پیشنهادات بهبود کد توسط مرورگر کد، فراتر از اشاره به خطاهای کد.
5. ادغام با ویرایشگر کد منبع ، مانند Visual Studio یا .Atom
6. ادغام با یک بستر ارتباطی تجاری ، مانند Slack یا .MS Teams
7. اولویت بندی تغییرات بر اساس میزان اهمیت و تأثیر آن بر عملکرد نرم افزار.
8. ارائه خط لوله‌ای که نشان می‌دهد پروژه در کدام مرحله توسعه شامل ساخت، آزمایش، بررسی کد و استقرار است.
9. وجود داشبورد برای همه اعضای پروژه که داده‌های آماری کلیه اقدامات انجام شده را نشان می‌دهد، مانند تعداد تعهدات، تعداد بررسی کد انجام شده و تعداد خطاها/هشدارهای کد در پروژه فعلی.
10. بحث کد جدید و قدیمی در صورت تغییر در کد.
11. گزینه‌ای برای انتخاب یک شاخه یا پرونده خاص در یک پروژه برای حفظ گردشکار سیستماتیک و روش بازبینی کد سازمان‌یافته.
12. تشخیص کد را با استفاده از یک طرح کدگذاری رنگی با نام توسعه‌دهنده منتقل کرد.
[[.RQ5چگونه می توان سیستم های پیشنهادی موجود برای مرورگران کد را بهبود بخشید؟ به عبارت دیگر ، چه ویژگی هایی در پیاده سازی های موجود برای سیستم های توصیه بازبینی کننده کد وجود ندارد؟
پاسخ: بر اساس نتایج نظرسنجی، موارد زیر برخی از ویژگی‌هایی است که می‌توان در سیستم‌های توصیه‌ای مرورگر کد یا هنگام جستجوی بازبین کد مربوطه، بهبود بخشید یا وجود ندارد:
1. وقتی نوبت به انتخاب یک مرورگر کد می‌رسد، شرکت‌کنندگان معتقد بودند که تخصص بازبینان در زبان برنامه‌نویسی پروژه، زمینه تخصص، سال‌ها تجربه کاری، تخصص کیفیت کد و درک معماری پروژه عوامل مهمی هستند.
2. برخی از شرکت‌کنندگان معتقد بودند که چندین سال سابقه کار و همچنین زمینه تخصص، هر دو مهم هستند. استدلال این بود که این عوامل هنگام یافتن رویکرد بهینه شده برای یک مشکل و ارائه پیشنهادات برای) LLDطراحی سطح پایین) می‌توانند مفید واقع شوند. همچنین، این عوامل در نوشتن یک استاندارد از تجربه و تخصص مفید است.

5-2 پیشنهاد برای بهبود سیستم توصیه‌کننده مرور کد

بر اساس یافته‌های تحقیق ما، ما یک سیستم توصیه بازبینی کد را پیشنهاد می‌کنیم که دارای تمام ویژگی‌های لازم باشد که در همه سیستم‌ها وجود ندارد یا ویژگی‌هایی که در حال حاضر وجود ندارد-

سیستم‌های ورودی پیشنهادی دارای موارد زیر است
امکانات:
1. شفاف‌تر از نظر شرایطی که همه جزئیات مربوط به مرورگر کد در داشبورد قابل‌مشاهده باشد. این جزئیات شامل تعداد پروژه‌هایی است که آنها روی آن کارکرده‌اند (یعنی تجربه کاری آنها) ، زمینه کاربردی خاصی که در آن تخصص دارند، تعداد بررسی کد انجام شده توسط آنها و حجم کار آنها (یعنی تعداد بررسی‌هایی که داور در حال حاضر دارد) مرور) برای اطمینان از بارگیری بیش از حد داور با بررسی کد جدید. اعتقاد بر این است که ارائه این جزئیات در نتیجه روند بررسی کد را تسریع می‌کند.
2. ترکیبی وسیع‌تر از داده‌ها برای آموزش توصیه گر. این مجموعه‌داده شامل نظرات مرور قبلی و پیام‌های متعهد و تجربه متقابل پروژه و فناوری خواهد بود. این داده‌ها به شما کمک می‌کند تا بدانید آیا کدی که باید بازبینی شود باتجربه پروژه‌ای که

یک بازبین دارد مطابقت دارد یا خیر. به طور مشابه، تاریخچه گذشته نظرات و تعهدات بازبینی بر اساس تعداد مرتکبین و بررسی‌هایی که قبلاً توسط آنها انجام شده بود و اینکه چگونه مرورهای کد قبلی برای توسعه دهندگان مفید بوده است، در انتخاب یک مرورگر مربوطه کمک خواهد کرد. این امر به شما اطمینان می‌دهد که نظرات مرور آینده آنها برای بررسی کد مفید خواهد بود.

3. بررسی کد قبل از ادغام کد و درگیری احتمالی کد اتفاق می‌افتد؛ بنابراین، سیستم پیشنهادی قبل از وقوع درگیری ادغام، مرورگران کد را توصیه می‌کند. بااین‌حال، این سیستم همچنین اجازه می‌دهد تا پس از ادغام درگیری‌ها، انتخاب بازبینی کد انجام شود تا در صورت لزوم از تأخیر جلوگیری شود و درعین‌حال از ایجاد یک محصول نرم‌افزاری باکیفیت خوب اطمینان حاصل شود.

فصل 6

نتیجه

در این تحقیق ما تعدادی از سیستم های توصیه بازبینی کد (CRRS)موجود در ادبیات ، روش های مختلف طبقه بندی این سیستم ها، ویژگی های مهم یک سیستم توصیه برای مرورگران کد و چگونگی سیستم های موجود را مشخص کردیم. بهبود یافته یا کدام ویژگی ها در CRRS های موجود وجود ندارد. یک مطالعه مروری سیستماتیک برای شناسایی سیستمهای پیشنهادی مرورگر کد و درک جزئیات مربوط به این سیستمها ، که شامل ویژگیها و عوامل مهم برای CRRS است، انجام شد. سپس ما یک نظرسنجی برای درک نیازهای اعضای پروژه نرم افزار در مورد سیستم های توصیه بازبینی کد انجام دادیم که مشخص می کند آنها در CRRS چه ویژگیهایی مهم هستند و چه چیزی را می توان در CRRS های موجود بهبود بخشید.

2-6 مشارکت:

این تحقیق مشارکت‌های زیر را انجام می‌دهد:

C1:رتبه بندی ویژگی های موجود در سیستم های توصیه کننده مرورگر کد موجود در مورد اینکه کدام یک از این موارد مفیدتر است.

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

C3:بهبودهایی که می توان در سیستم های توصیه مرورگر کد موجود ایجاد کرد.

C4:ویژگی هایی که هنگام انتخاب مرورگر کد مهم هستند.

6.2 کار آینده

جهت‌های احتمالی آینده بر اساس این کار عبارت‌اند از:

مروری گسترده‌تر بر ادبیات سیستماتیک: می‌توان مطالعه ادبیات سیستماتیک گسترده‌ای را انجام داد که نه‌تنها سیستم‌های توصیه مرورگر کد، بلکه شیوه‌ها و رویه‌های بازبینی کد را نیز بررسی می‌کند. این می‌تواند به ارائه تصویر بهتر در مورد نیازهای اعضای پروژه نرم‌افزار در مورد بررسی کد در ارتباط با استفاده از CRRS کمک کند.

ایجاد سیستم توصیه بازبینی کد: برای کارهای آینده، هدف ما ایجاد سیستمی است که تمام جزئیات مرورگر در داشبورد سیستم قابل‌مشاهده باشد (یعنی تجربه کار، تجربه فناوری، تعداد بررسی کد انجام شده و غیره). همچنین، این سیستم داده‌های بیشتری برای آموزش سیستم توصیه گر دارد که شامل نظرات مرور قبلی و پیام‌های متعهد، پروژه‌های مرتبط و تجربه فناوری است. بر اساس بازخورد به‌دست‌آمده از نظرسنجی، بررسی‌ها قبل از وقوع درگیری‌های ادغام انجام می‌شود.

مراجع

- D. Gusfield, Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology. 1997.
- D. Cubranic, G. C. Murphy, J. Singer, and K. S. Booth, “Hipikat: A project memory for software development,” IEEE Trans. Software Eng., vol. 31, no. 6, pp. 446–465, 2005. DOI: 10.1109/TSE.2005.71.
- VMware, Reviewboard, 2006. [Online]. Available: https://www.reviewboard.org/.
- B. Kitchenham and S. Charters, “Guidelines for performing systematic literature reviews in software engineering,” Journal of Software Engineering and Applications, 2007.
- H. H. Kagdi, M. Hammad, and J. I. Maletic, “Who can help me with this source code change?,” pp. 157–166, 2008. DOI: 10.1109/ICSM.2008.4658064. [Online]. Available: https://doi.org/10.1109/ICSM.2008.4658064.
- E. W. T. Ngai, L. Xiu, and D. C. K. Chau, “Application of data mining techniques in customer relationship management: A literature review and classification,” Expert Syst. Appl., vol. 36, no. 2, pp. 2592–2602, 2009. DOI: 10.1016/j.eswa.2008.02.021. [Online]. Available: https://doi.org/10.1016/j.eswa.2008.02.021
- T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, “A systematic literature review on fault prediction performance in software engineering,” IEEE Trans. Software Eng., vol. 38, no. 6, pp. 1276–1304, 2012. DOI: 10.1109/TSE.2011.103. [Online]. Available: https://doi.org/10.1109/TSE.2011.103.
- D. H. Park, H. K. Kim, I. Y. Choi, and J. K. Kim, “A literature review and classification of recommender systems research,” Expert Syst. Appl., vol. 39, no. 11, pp. 10059–10072, 2012.
- A. Bacchelli and C. Bird, “Expectations, outcomes, and challenges of modern code review,” D. Notkin, B. H. C. Cheng, and K. Pohl, Eds., pp. 712–721, 2013.
- V. Balachandran, “Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation,” D. Notkin, B. H. C. Cheng, and K. Pohl, Eds., pp. 931–940, 2013.
- Bosu, M. Greiler, and C. Bird, “Characteristics of useful code reviews: An empirical study at microsoft,” M. D. Penta, M. Pinzger, and R. Robbes, Eds., pp. 146–156, 2015. DOI: 10.1109/MSR.2015.21.
- P. Thongtanunam, C. Tantithamthavorn, R. G. Kula, N. Yoshida, H. Iida, and K. Matsumoto, “Who should review my code? A file location-based code-reviewer recommendation approach for modern code review,” Y. Gueh´ eneuc, B. Adams, and´ A. Serebrenik, Eds., pp. 141–150, 2015.
- X. Xia, D. Lo, X. Wang, and X. Yang, “Who should review this change?: Putting text and file location analyses together for more accurate recommendations,” R. Koschke, J. Krinke, and M. P. Robillard, Eds., pp. 261–270, 2015.
- M. Gasparic and A. Janes, “What recommendation systems for software engineering recommend: A systematic literature review,” *J. Syst. Softw.*, vol. 113, pp. 101–113, 2016.
- S. Lee and S. Kang, “What situational information would help developers when using a graphical code recommender?” *J. Syst. Softw.*, vol. 117, pp. 199–217, 2016. DOI: 10.1016/j.jss.2016.02.050.

- M. M. Rahman, C. K. Roy, and J. A. Collins, “Correct: Code reviewer recommendation in github based on cross-project and technology experience,” L. K. Dillon, W. Visser, and L. Williams, Eds., pp. 222–231, 2016.
- M. B. Zanjani, H. H. Kagdi, and C. Bird, “Automatically recommending peer reviewers in modern code review,” *IEEE Trans. Software Eng.*, vol. 42, no. 6, pp. 530–543, 2016.
- K. Haruna, M. A. Ismail, S. Suhendroyono, D. Damiasih, A. C. Pierewan, H. Chiroma, and T. Herawan, “Context-aware recommender system: A review of recent developmental process and future research direction,” 2017.
- E. Schon, J. Thomaschewski, and M. J. Escalona, “Agile requirements engineering: A systematic literature review,” *Comput. Stand. Interfaces*, vol. 49, pp. 79–91, 2017.
- M. Fejzer, P. Przymus, and K. Stencel, “Profile based recommendation of code reviewers,” *J. Intell. Inf. Syst.*, vol. 50, no. 3, pp. 597–619, 2018.
- V. Kovalenko, N. Tintarev, E. Pasynkov, C. Bird, and A. Bacchelli, “Does reviewer recommendation help developers?,” 2018.
- C. Sadowski, E. Soderberg, L. Church, M. Sipko, and A. Bacchelli, “Modern code review: A case study at google,” F. Paulisch and J. Bosch, Eds., pp. 181–190,
- Gerrit, *Gerrit*, 2021. [Online]. Available: https://www.gerritcodereview.com/.
- phacility, *Phabricator*, 2021.