نسخه‌ی ابتدایی پرپوزال
ایمان شفیعی علویجه

ادبیات پژوهش:

همین‌طور که نرم‌افزار در بحث توسعه و نگهداری جلو می‌رود، توسعه‌دهندگان همیشه مانند ابتدای روند توسعه نمی‌توانند کد جدیدی به برنامه اضافه کنند. در سال ۱۹۹۲، وارد کانینگهام برای اولین بار مفهوم "کدی که دقیقا درست نیست" و یا در معادل انگلیسی "no-quite-right code" را به عنوان شکلی از بدهی ارائه کرد (کانینگهام ۱۹۹۲). از این بدهی به عنوان بدهی فنی نیز یاد می‌شود. وقتی از بدهی فنی استفاده می‌شود که یک راه حل ناقص، موقت و یا با بهینگی ناقص ارائه شود. بنابراین تیم‌های توسعه برای رسیدن به هدف‌های کوتاه مدت بدهی‌های فنی پشت سر هم ایجاد می‌کنند و باید در زمان طولانی تر و هزینه بیشتر این بدهی‌ها را پرداخت کنند. توسعه دهندگان اتفاقات متفاوتی را تجربه می‌کنند که آن‌ها را به ایجاد بدهی فنی در کد هدایت می‌کند، مانند فشار ناشی از نزدیک شده به پایان مهلت کارها، وجود کد با کیفیت پایین در کد، روند ناقص توسعه‌ی نرم افزارو یا موارد مربوط به تجارت و بازار (لیم و همکاران ۲۰۱۲). بدهی فنی می‌تواند خودآگاه و یا ناخودآگاه به سیستم اضافه شود و با توجه به تحقیاتی که صورت گرفته است برنامه نویسان بازپرداخت این بدهی‌های فنی را دست گرفته و موجب مشکلات بیشتری می‌شود (بلومی و همکاران ۲۰۱۶). با توجه به اینکه تشخیص و مدیریت بدهی‌های فنی تاثیر زیادی در روند توسعه‌ی نرم‌افزار و کیفیت نرم‌افزار دارد، پژوهش‌های بسیاری روی این موضوع شده است (لی و همکاران ۲۰۱۵; آلوس و همکاران ۲۰۱۶).

پوتدار و شیهاب (۲۰۱۴) یک پژوهش را روی کامنت‌های موجود در کد منبع نرم‌افزارها برای پیدا کردن نمونه‌های بدهی شروع کردند. نویسندگان از این پدیده به عنوان بدهی فنی خود پذیرفته (Self-Admitted Technical Debt) عنوان کردند. در واقع این نوع از بدهی فنی زمانی رخ می دهد که توسعه‌دهنده در کامنتی که در کد می‌گذارد به این موضوع که بخشی از کد کاملا درست نیست و بدهی وجود دارد اعتراف می‌کند.

ضرورت انجام تحقیق، چالش‌ها و راهبردّ ها:

شرکت‌ها برای افزایش بهره‌وری نیاز به انجام کارهای متفاوتی دارند یکی از این کارها افزایش بهره‌وری تیم‌های برنامه‌نویسی و به صورت کلی تیم‌های برنامه‌نویسی است.

بدهی فنی در کدهای نوشته‌شده در نرم‌افزارها قابل مشاهده است و باعث کاهش انعطاف‌پذیری نرم‌افزار می‌شود به این صورت نرم‌افزار به تغییر مقاومت می‌کند و همان‌طور که می‌دانیم تغییر بخش اصلی توسعه‌ی نرم‌افزار است و در نتیجه هنگامی که بدهی‌های فنی یک نرم‌افزار بالا می‌رود تغییر بخشی از کد برای افزودن ویژگی‌های جدید

به سیستم سخت‌تر می‌شود و این سخت‌تر شدن می‌تواند موجب دوباره‌نوسی کد شود و در نتیجه تیم برنامه‌نویسی باید زمان بسیار زیادی را برای افزودن یه تغییر کوچک به سیستم صرف کند که در نتیجه‌ی آن باعث کاهش بهره‌وری تیم و شرکت خواهد شد.
پس پیدا کردن به موقع بدهی‌های فنی توسط تیم برنامه‌نویسی می‌تواند باعث شود تا تیم به موقع و در زمانی که محصول در بمباران ویژگی‌های جدید که توسط تیم بازاریابی و مدیران محصول ارائه می‌شود، قرار ندارد به حل و فصل این بدهی‌های فنی بپردازد.
بخشی از این بدهی‌های فنی از دید برنامه‌نویس خارج هستند اما بخشی از آن‌ها را خود برنامه‌نویس آشکارا در سیستم ایجاد می‌کند و در مواردی به وسیله‌ی قرار دادن کامنت در کد به آن‌ها اقرار می‌کند. به این نوع از بدهی‌های فنی بدهی‌های فنی خود پذیرفته نیز گفته می‌شود. به طور مثال برنامه نویسی یک متد در کلاسی تعریف کرده است که نیاز به نوشتن تست دارد اما به دلیل شرایط زمانی باید به سرعت این ویژگی را پیاده‌سازی کند و از نوشتن تست برای متد صرف نظر می‌کند و کامنتی تحت عنوان TO DO: Should write a unit test for this method می‌نویسد، به این کامنت یک بدهی فنی خود پذیرفته و یا اقرار شده توسط خود برنامه‌نویس می‌گوییم.
حال اگر بتوانیم با استفاده از تکنیک‌های یادگیری ماشین و پردازش طبیعی متن و روش‌های نو ظهوری مانند BERT الگوهای این نوع از کامنت‌ها را در پروژه‌های متن‌باز استخراج کنیم، می‌توانیم روند پیدا کردن این نوع از بدهی‌های فنی را هوشمند کرده تا تیم‌ها راحت تر بتوانند بخش‌هایی از کد را که نیار به اصلاح دارند را پیدا کنند. همچنین اگر بتوانیم با حل مسئله‌ی طبقه‌بندی این نوع از بدهی‌های فنی را در گروه‌های مختلف دسته‌بندی کنیم می‌تواینم منبع مشکل تیم‌ها را به دست آورده به حل ریشه‌ای مشکلات بپردازیم. به طور مثال می توان چند نوع دسته‌بندی زیر را ارائه نمود:

- بدهی فنی مربوط به تست
- بدهی فنی مربوط به مستندسازی
- بدهی فنی مربوط به طراحی و تحلیل
- بدهی فنی مربوط به پیاده‌سازی

در شرکت‌های بزرگ که از معماری‌های سرویس‌گرا و یا میکرو سرویس استفاده می‌کنند هر تیم برنامه‌نویسی روی یک سرویس خاص کار می‌کند و از یکسری تکنولوژی خاص برای توسعه استفاده می‌شود. مهم‌ترین بخش تفاوت زبان برنامه‌نویسی سرویس مورد نظر است. اگر بتوانیم به استفاده از روش‌های مبتنی بر یادگیری ماشین بگوییم که آیا بدهی‌های فنی خود پذیرفته وابسته به زبان هستندیا خیر می‌تواند به مدیران فنی جهت پیگیری بهتر این بدهی‌ها کمک کندو در مجموع باعث شود تا برنامه‌نویس‌ها حتی اگر می‌دانند که کد و روش کدنویسی اشتباه است اما به گذاشتن این کامنت‌ها بپردازند تا در آینده این بدهی‌های فنی به سادگی پیدا شوند.

استخراج کامیت‌ها و مشکلات موجود (Issues) در پروژه‌های متن باز در سایت‌هایی مانند GitHub و یا GitLab نیز می‌تواند اطلاعات مهمی را به ما ارائه دهد. این اطلاعات می‌تواند توسط الگوریتم‌های پردازش متن به تیم‌ها کمک کند تا نه تنها از طریق کامنت‌های موجود در کد بلکه از متن کامیت (Commit) های موجود در پروژه به پیدا کردن بدهی‌های فنی خود پذیرفته بپردازد.
همچنین در این پلتفرم‌های مدیریت ورژن کد مواردی مانند توسعه‌دهنده‌ایی که بدهی فنی را ایجاد کرده، توسعه‌دهنده‌ایی که بدهی فنی را بر طرف کرده و زمان مربوط به هر دو اتفاق مشخص و معلوم است و با استفاده از این داده‌های می‌توان نتیجه گرفت آیا کسی که بدهی فنی را ایجاد کرده خودش همان را حل می‌کندو یا این بدهی فنی قابل انتقال است زیرا همانطور که می‌دانیم انتقال کد از یک نفر به نفر دیگر زمان‌بر بوده ممکن است باعث کاهش بهروری افراد گردد. همچنین می‌توان فهمید در چه بازه‌هایی از زمان بدهی‌های فنی حل می‌شوند، آیا همه‌ی آن‌ها در زمان‌های ابتدایی ماه حل می‌گردنند و یا در طول هر هفته مقداری از آن‌ها حل می‌شود. با بررسی این داده می‌توانیم گزارش‌های مفیدی به مدیران فنی ارائه کنیم تا با تصمیمات درست باعث افزایش بهره‌وری تیم‌ها بشوند.

پیشینه پژوهش

پوتدار و شیهاب (۲۰۱۴) اولین کسانی بودند که مفهوم بدهی‌های فنی خود پذیرفته را مطرح کردند و تلاش کردندتا بتوانند یکسری الگو برای تشخیص این نوع از بدهی فنی به دست بیاورند. آن‌ها به صورت دستی ۱۰۱۷۶۲ کامنت را در کدهای موجود در پروژه‌های متن باز بررسی کردند و ۶۲ الگوی متفاوت برای تشخیص بدهی‌های فنی خود پذیرفته رسیدند. همچنین آن‌ها در این مقاله راهکارهای زیر ارائه شد:

- به دست آوردن مقدار بدهی فنی موجود
- دلیل به وجود آمدن بدهی فنی
- مقدار رفع شدن این بدهی‌های فنی بعد از به وجود آمدنشان

ماریو اندره و همکاران (۲۰۲۰) به بررسی و استخراج بدهی فنی خود پذیرفته به وسیله‌ی آنالیز کامنت‌های گذاشته شده در کد پرداختند و در این راه از روش مجموعه واژگان زمینه‌دار (contextualized vocabulary) استفاده شده است. در این مقاله محققان سه تحقیق تجربی را اجرا کردند تا بتوانند بدهی‌های فنی خود پذیرفته را در کامنت تشخیص دهند و همچنین بتوانند آن‌ها را در دسته‌بندی‌ها مخصوص به خودشان قرار دهند. در نتیجه‌ی این مقاله محققان توانستند نشان دهند روش‌های الگو محور (pattern-based) برای

آنالیز کامنت‌های موجود در کد می‌تواند باعث بهبود روش‌های موجود برای شناسایی خودکار بدهی‌های فنی خود پذیرفته بشود.

منساه و همکاران (۲۰۱۸) به بررسی بدهی فنی خود پذیرفته پرداختند و توانستند یک طرح اولویت بندی برای پیدا کردن این بدهی های فنی ارائه کنند. همچنین توانستند تخمین بزنند که برای از بین بردن هر کدام از این بدهی ها چند خط کد نیاز است نوشته شود. الگوریتمی نیز برای تشخیص و اولویت بندی بدهی فنی خود پذیرفته ارائه شد.

زمپتی و همکاران (۲۰۲۱) به ایجاد یک مطالعه‌ی تجربی دو مرحله‌ای پرداختند که در مرحله ی اول یکسری مصاحبه با برنامه نویسان موجود در صنعت و برنامه‌نویسان متن باز انجام دادند و از نتیجه مرحله‌ی اول یکسری سوال برای تولید یک نظرسنجی برای مرحله‌ی دوم استفاده کردند. هدف از این تحقیق پاسخ به سوالاتی نظیر:

- توسعه‌دهندگان تا چه حد به بدهی فنی خود پذیرفته اعتراف می‌کنند؟
- چرا توسعه‌دهندگان به بدهی فنی خود پذیرفته اقرار نمی‌کنند؟
- از طریق چه ابزار و کانال‌هایی توسعه دهندگان به بدهی فنی خود پذیرفته اقرار می‌کنند؟
- محتوای الگوهای بدهی فنی خود پذیرفته چیست؟ و ارتباطش با دسته‌بندی های بدهی فنی چیست؟
- توسعه‌دهندگان با کامنت‌هایی که نشان‌دهنده‌ی بدهی‌های فنی خود پذیرفته هستند چگونه برخورد می‌کنند؟

مالدونادو و همکاران (۲۰۱۷) از تکنیک‌های پردازش طبیعی متن برای تشخیص بدهی‌های فنی در دسته‌بندی طراحی و نیازمندی از کامنت‌های موجود در کد منبع برنامه استفاده کردند. ۶۲۵۶۶ کامنت را از ۱۰ پروژه‌ی متفاوت بررسی کردند و با استفاده از Stanford Classifier به طبقه‌بندی در دو دسته‌ی مثبت (حاوی بدهی فنی) و منفی (عدم وجود بدهی فنی) پرداختند.

منابع:

Cunningham, W., 1992. The wycash portfolio management system. Proc. Obj. Ori- ented Programm. Syst. Lang. Appl. 4 (2), 29–30.

Lim, E., Taksande, N., Seaman, C., 2012. A balancing act: what software practitioners have to say about technical debt. IEEE Softw. 29 (6), 22–27.

Bellomo, S., Nord, R.L., Ozkaya, I., Popeck, M., 2016. Got technical debt? Surfacing elusive technical debt in issue trackers. In: Proceedings of the 13th International Conference on Mining Software Repositories. IEEE, pp. 327–338.

Alves, N.S., Mendes, T.S., de Mendonça, M.G., Spínola, R.O., Shull, F., Seaman, C., 2016. Identification and management of technical debt: a systematic mapping study. Inf. Softw. Technol. 70, 100–121.

Li, Z., Avgeriou, P., Liang, P., 2015. A systematic mapping study on technical debt and its management. J. Syst. Softw. 101, 193–220.

Potdar, A., Shihab, E., 2014. An exploratory study on self-admitted technical debt. In: Proceedings of the 30th International Conference on Software Maintenance and Evolution. IEEE, pp. 91–100.

de Freitas Farias, M. A., de Mendonça Neto, M. G., Kalinowski, M., & Spínola, R. O. (2020). Identifying self-admitted technical debt through code comment analysis with a contextualized vocabulary. *Information and Software Technology*, *121*, 106270.

Mensah, S., Keung, J., Svajlenko, J., Bennin, K. E., & Mi, Q. (2018). On the value of a prioritization scheme for resolving Self-admitted technical debt. *Journal of Systems and Software*, *135*, 37-54.

Zampetti, F., Fucci, G., Serebrenik, A., & Di, M. (2021). Self-Admitted Technical Debt Practices: A Comparison Between Industry and Open-Source. *Empirical Software Engineering*.

Maldonado, E., Shihab, E., Tsantalis, N., 2017. Using natural language processing to automatically detect self-admitted technical debt. IEEE Trans. Softw. Eng. 43 (11), 1044–1062.