



Expert Developer Finder T2309

Visit us at <https://egeergul.github.io/cs490-website/>

Team Members

Ali Eren Günaltılı - 21801897 - eren.gunaltili@ug.bilkent.edu.tr
Bora Altınok - 21902206 - bora.altinok@ug.bilkent.edu.tr
Ceyda Şahin - 21903448 - ceyda.sahin@ug.bilkent.edu.tr
Ege Ergül - 21902240 - ege.ergul@ug.bilkent.edu.tr
Tuna Dalbeler - 21802539 - tuna.dalbeler@ug.bilkent.edu.tr

Supervised by Eray Tüzün

Innovation Expert Eray Tüzün

Course Instructed by Erhan Dolak and Tağmaç Topal

1. INTRODUCTION	2
1.1 Description	4
1.1.1. Target Users	4
1.1.2 Innovation Targets	4
1.2 Similar Works	5
1.3 Academic Research	8
1.3.1 Considering Effectiveness of Exploring Historical Commits for Developer Recommendation	8
1.3.2 Identifying Source Code File Experts	9
1.3.3 Enhancing Developer Recommendation with Supplementary Information	11
1.3.4 DREX: Developer Recommendation with K-Nearest-Neighbor Search and Expertise Ranking	11
1.3.5 DevRec: Multi-Relationship Embedded Software Developer Recommendation	12
1.4 Constraints	13
1.3.1 Implementation Constraints	13
1.3.2 Economic Constraints	13
1.3.3 Sustainability Constraints	13
1.3.4 Language Constraints	13
1.3.5 Social Constraints	13
1.4 Professional and Ethical Issues	14
1.4.1 Professional Issues	14
1.4.2 Ethical Issues	14
2. REQUIREMENTS	14
2.1 Functional Requirements:	14
2.2 Non-Functional Requirements:	17
2.2.1 Effectiveness	17
2.2.2 Performance	18
2.2.3 Maintainability	18
2.2.4 Portability	19
2.3 Functional Priority	19
References	20

1. INTRODUCTION

Software development is a crucial and core aspect of engineering. Millions of software are being developed each day with different sizes and complexities. From 1 to 100+ developers can work on the same project [1]. As the size and complexity of the software increase, the development process gets more challenging. Making the best use of limited resources (i.e. time) is a primary concern for software practitioners. Many practices have proven that with simple ideas and some tools, software development can be facilitated (i.e. version control systems). Thus, it is safe to say that investigating the software development life-cycle and seeking solutions to some common software development problems is a worthy topic to be chosen as our senior year project.

With the COVID-19 pandemic, the entire world shifted its infrastructures towards a more remote environment and adjusted its policies accordingly. IT jobs got affected the most by this transition and many IT companies are planning on going entirely remote working even after the end of the pandemic [1]. This entire remote working strategy causes late adaptation of a junior developer to the working environment and colleagues. In a physical office, junior developers are more likely to communicate with senior developers and can get help from them when they are having difficulties with the code base which is new for them.

On the other hand, knocking on the doors of senior developers may not be easy in remote environments for junior developers since the process takes a long time and the response ratio is expected to be less. Response time to their questions is crucial for junior developers as it accelerates the onboarding process. Thus, the time that companies spend on their fresh starters in the onboarding process, is likely to decline. From a company's point of view, minimizing the onboarding time is a great opportunity to save money and make more profit, however, it may become problematic over the years. According to Business Insider [1] average years of working as a software developer in Uber, Dropbox and Tesla are respectively 1.8, 2.1, and 2.1 years. According to another survey [2] that is conducted among the big tech companies in Silicon Valley (even though the conditions are considerably great) working years, on average, as a developer is 2.3. Half of the fresh developers leave their companies in two years. This is a real problem for companies as they spend money on their developers and train them as an investment for the future, but before that future comes they have to face newly hired developers. Regarding the study of Zhou and Mockus in 2010 [3], the average onboarding process takes almost 12 months. To optimize the cost of a company, the onboarding process had to be minimized since even best conditions are provided for developers, they do not stay as much as companies plan for them. Thus, we believe developing a system that will pinpoint the best fit for an experienced developer for a given code base can help to minimize this onboarding process. This way developers who seek help won't have to mail plenty of developers to get help being redirected to others constantly.

With the introduction of an expert code recommender system, getting help for a given codebase process will be faster. Furthermore, the user target is not limited to junior developers or fresh starters. Even senior developers can use the system as well. Since there is a rapid turnover of proficiency (i.e. every file is constantly modified and the expert of a given codebase changes rapidly), even senior developers can have difficulty adjusting to a

codebase and may need help to get used to it faster. Last, of all, such a system is vital for the reusability of the software as well. Consulting people who are not experts in the given code base can result in even more harm than it causes good. Thus, we believe that finding the correct expert for a code base is vital. In this report, the proposed product regards these concerns and comes up with a renovative and innovative solution.

1.1 Description

Expert developer finder is a product targeting anyone involved in open-source or proprietary software. The offered solution aims to mine every available data and artifact regarding the target software; and by examining those resources, recommend the appropriate participant for a specific part of the code or the given error. This task is important for two reasons: accuracy and time efficiency. The system aims to accurately find the most knowledgeable participant so that better quality information would be provided to the person who is seeking help. Accuracy is important for preventing the spread of wrong information and further wrong implementations. Time efficiency is another core aspect of this project. Since time is the most limited resource in most software, minimizing the time spent in the struggle of finding the appropriate person of interest for getting help is crucial for most of the OSSs and closed software projects.

1.1.1. Target Users

From members of QA to the testing team, from junior developers to senior developers, anyone involved in software development can use this system. Yet, this software is primarily proposed to be used by newly joined developers.

1.1.2 Innovation Targets

Our product “Expert Developer Finder” aims to support software development processes and we can categorize it as process innovation. This product is intended to improve the software development process. This is an appropriate example of making “superior methods for doing your work” [4]. Hence it can be categorized under “process innovation”. We plan to develop a better method to improve consultation on code and authorship attribution topics.

We will be practicing “incremental innovation” in the context of the proposed problem of adapting to the codebase for newly joined developers. Incremental innovation is making an improvement in an existing field. It does not create new business opportunities, but it is an improvement people may need or want [5]. The field of this product is not new. There are a few products roughly doing similar jobs and a lot of research discussing techniques and purposes of this field, which will be discussed in Section 1.2 Similar Works and Section 1.3 Academic Research. We plan to increment some used methods when searching for someone to help with their code. Our product might not include any cutting-edge technology and the impact on the market is dependent on how successful the project will be implemented. The degree of improvement we bring to the table, after its success, will affect its impact. This is why we are thinking if we were to categorize our product, it will be on the incremental innovation side rather than sustained, disruptive or radical.

Optimization type of innovations in technology is an improvement on existing processes and business models [5]. Our system aims to improve on one of the many subprocesses of software development. To clarify, the aim is not to assist the software/code development itself, but to a process that at some point all developers went through during development. While our specific aim cannot be definitively categorized, we may say it is in the Authorship Attribution and Mentorship Process. This product aims to optimize this process and can be categorized under optimization innovation.

1.2 Similar Works

For relevant works, we looked at expert recommendation systems and software. There are some works done in this field of research but none of them is equivalent to our purpose in this project. What we encountered mostly is academic research that focuses on recommending an expert in the bug assignment and code review processes. Very few of the software in the reviewed literature is offered as a product that serves customers.

One of the closest tools available and in use is Git's "blame" command (For example usage see Figure 1) and GitLens. Like our project, GitLens aims to facilitate the adaptation process of developers to the code base; yet it does not recommend an expert developer. GitLens shows the latest people who edited that line after the line as shown in Figure 2 (line-based) and also shows the latest developers who touched that code block on top of the code block (block-based) as shown in Figure 3. Furthermore, if the latest version of a code-block was written/edited by multiple developers, all of those authors are shown. Git blame also does the same job in a more basic way. It shows the authors of each change next to each line. It runs on a terminal or a command line interface (CLI) as shown in Figure 1. Git's blame and GitLens are not expert recommendation systems. However, since they also help developers to see who edited a file / code block, they help developers to find an expert developer to whom they can consult. Thus, both of them can be considered as similar work.

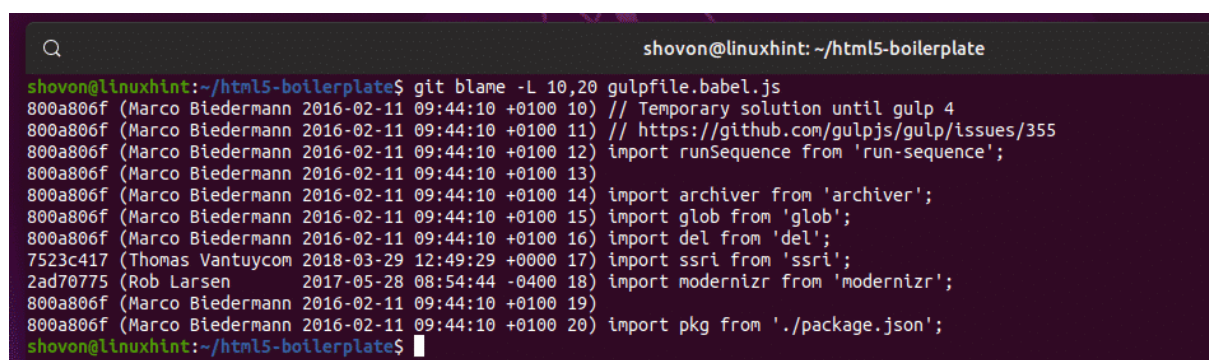
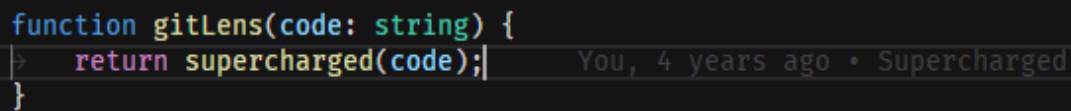
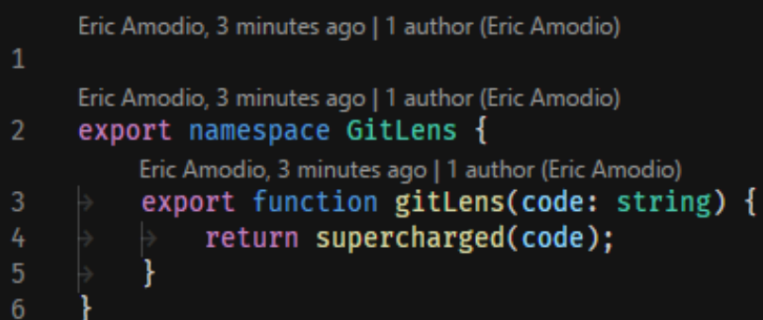


Figure 1 - Example Git Blame output [6]

A screenshot of a code editor showing a function definition. The function is named 'gitLens' and takes a 'code' string as an argument. It returns the result of 'supercharged(code)'. A vertical line on the left side of the code indicates the blame annotations. The text 'You, 4 years ago • Supercharged' is visible on the right side of the code block.

```
function gitLens(code: string) {  
  return supercharged(code);  
}
```

Figure 2 - GitLens's Blame Annotations [7]

A screenshot of a code editor showing a code block with line numbers 1 through 6. The code block is for a namespace 'GitLens' containing a function 'gitLens'. The function takes a 'code' string and returns 'supercharged(code)'. The code is annotated with the author 'Eric Amodio, 3 minutes ago | 1 author (Eric Amodio)' at the top of each line. The code is as follows:

```
1 Eric Amodio, 3 minutes ago | 1 author (Eric Amodio)  
2 export namespace GitLens {  
3   Eric Amodio, 3 minutes ago | 1 author (Eric Amodio)  
4   export function gitLens(code: string) {  
5     return supercharged(code);  
6   }
```

Figure 3 - GitLens showing the last edited authors on top of the code blocks [7]

Another similar work is UnReview. As it can be seen in Table 1, UnReview is a code reviewer-finding tool made by GitLab. It uses Commit History, and Merge Requests to make Machine Learning based expert suggestions [9]. This product is not a direct competitor to our Expert Developer Finder. Expert Developer finder does not aim to recommend recommendations. Instead, it aims to find an expert for specified code units. Furthermore, UnReview recommends only one developer whereas Expert Developer Finder will recommend a list of expert developers. Yet, since UnReview also uses artifacts to make recommendations, it can be considered as a similar work.

Table 1
Comparison of similar works and related academic literature.

Tools	Interface	Heuristic	Algorithms	Data Source	Purpose
Git Blame	CLI	Latest Code Edit	Finds each line's author and latest revision by checking the commit history	Git Commit History	"Show which author last modified each line of a file, and how [6]."
Git Lens	IDE Extension	Latest Code Edit	Uses Git Blame	Git Commit History	Provides various visualization Tools for Git. Shows Git blame annotations which can be interpreted as recommendations.
Expert Developer Finder	IDE Extension and Web UI	Smart Recommendation	To be determined	Git Records, bug reports, and other artifacts.	Finding the most available and knowledgeable people on specific code units.
UnReview	Web	Reviewer Experience	Machine Learning	GitLab Records	Recommend code reviewers for code review .

In the Expert Developer Finder row of Table 1, some of the product specifications are written. This product is designed as an IDE Extension for VSCode, with an additional Web UI for non-VSCode users. At this stage, the "Smart Recommendation" algorithm is not fully decided. Our aim is to examine various data sources, and statistics (see section 1.3.2. for the examples), calculate scores and make recommendations based on these scores. Most of these details will be decided at the Requirement Analysis stage. A more detailed explanation is under section 2.1 Functional Requirements.

Another low-similarity work is Hipikat. Hipkat is an Eclipse plugin that returns the related artifacts when a developer asks for it, in a specific code unit [11]. It uses artifacts such as source code, documentation, bug reports, e-mail, newsgroup articles, and version information for its recommendations. We find this relevant because it analyzes the artifacts

and the source code for the recommendation. It is an IDE plug-in like our product. Developers can use Hipikat on a code unit they specify, similarly to our project.

1.3 Academic Research

1.3.1 Considering Effectiveness of Exploring Historical Commits for Developer Recommendation

Sun et al. [12] conducted a study for enhancing and increasing the accuracy of recommending a developer for an issue. The study investigated the relevance of meaningful words (nouns and verbs) in commit descriptions, developers' tendency to change the files that they have changed many times before (frequency), and the tendency to change the files that they have changed recently (recency).

- For the approach of considering the commit description to recommend a candidate (CDC), the study compared the meaningful messages in the commit description and words in the change request.
- For the approach of considering relevant files to recommend candidates (RFC), the study compared the meaningful messages in the relevant files and words in the change request.
- For the developers' tendency of changing the files that they have changed frequently, all developers and relevant files from the control set were extracted and the files were grouped based on their total change count (the number of times a file was changed by a developer). Next, all the developers and the relevant files from the experimental set were extracted. For each developer, the change count of all relevant files grouped as in the first step was counted. A similarity in the distribution pattern of files was observed in the control set.
- For investigating the developer's tendency of changing the files that they have changed recently, some of the latest commits were extracted as the control set and the other commits were used as the experimental set.

The study found that for some subjects, the commit description reflected the developer's experience. Thus, more meaningful words in the commit description had an impact on the commit's quality (the more meaningful words are better quality). Moreover, the study found the approach of applying CDC to be generally superior to the approach of applying RFC. The study also concluded that the majority of developers have the tendency to change the files that they have changed before (the more times they changed before resulted in higher probability) also the majority of the developers are more likely to change the files that they have changed recently.

Expert Developer Finder has some similarities with this research. In our project, our recommendation algorithm will also be considering how frequently a developer interacts with the given code unit and how recently did he/she interact with it. Commit descriptions can be also taken into consideration while developing the algorithm.

1.3.2 Identifying Source Code File Experts

This study works on a methodology to predict whether a developer is an expert on a given file. Cury et al. [13] have investigated the performance of machine-learning-based source code file expert identification models, compared to a linear repository-mining-based approach. It was found that machine-learning classifiers outperformed the linear technique in F-Measures specifically in public datasets, which contained significantly more data points than private datasets. The main difference was pointed out by a higher precision, implying more valid results, in ML-based techniques, whereas a higher recall, implying more complete results, in linear techniques. Another important finding of the research was that “file creators tend to have high knowledge of the files they created; however, this knowledge decreases with time” which is explained in the chapter named “How do repository-based metrics correlate with developer’s knowledge?”. Statistically, First Authorship (FA) was found to have the highest positive correlation with knowledge in source code files. The recency of Modification was found to have the highest negative correlation, among the variables depicted in Table 1 and Table 2:

Variable	Meaning
Adds	Number of lines added by a developer d on a file f
Dels	Number of lines deleted by a developer d on a file f
Mods	Number of lines modified by a developer d on a file f
Conds	Number of conditional statements added by a developer d on a file f
Amount	Sum of number of added and deleted lines of a developer d on a file
FA	Binary variable that indicates whether a developer d added the file f to the repository
Blame	Number of lines authored by a developer d that are in a file f
NumCommits	Number of commits made by a developer d on a file f
NumDays	Number of days since the last commit of a developer d on a file f
NumModDevs	Number of developers that committed on the file f since the last commit of a developer d
Size	Number of lines of code (LOC) of the file f
AvgDaysCommits	Average number of days between the commits of a developer d on a file f

Table 2: Variables extracted from the development history. The variables description are given considering a developer d and a file f in its last version [3].

Variable	Corr. with Knowledge	p-value
NumDays	- 0.24	<0.001
Size	- 0.21	<0.001
NumModDevs	- 0.21	<0.001
Mods	0.01	0.515
Dels	0.02	0.369
Cond	0.19	<0.001
NumCommits	0.20	<0.001
AvgDaysCommits	0.21	<0.001
Amount	0.28	<0.001
Blame	0.29	<0.001
Adds	0.31	<0.001
FA	0.33	<0.001

Table 3: Correlation of extracted variables with Knowledge

As far as the methodology of the research is concerned, initially, the target repositories were identified. Next, a list of sample pairs (developer, file) was created for each repository. Each developer was sent a survey via email, asking for their expertise level in their file matches out of 5. This data was used to assess the accuracy of both models' results. This methodology might be useful to us, since stakeholder engagement is an important aspect to consider in innovation.

It was found that ML-based and repository-mining-based approaches both have a suitable use case, according to the target user's tolerance to false recommendations. It was suggested that linear techniques were used in scenarios where a complex task needs to be supervised by as many developers as possible.

This study is important since it highlights the correlation of variables (see Table 1.) with the developer's knowledge of the source file. These findings may be useful to us at the stage of developing our own weighted criteria for determining experts.

1.3.3 Enhancing Developer Recommendation with Supplementary Information

Xiaobing Suna, Hui Yanga, Xin Xia and Bin Li [14] have investigated how existing developer recommendation methods for issue requests and bugs can be improved by considering developers' personalized experiences.

The authors came up with CTM (Collaborative Topic Modeling) which ranks the relevancy of a developer with the created issue request or bug by considering their historical data such as their commit history. CTM collects the files and issues that are related to current issues and then creates a ranking system by considering the developer's historical data. The technique that is used in CTM ranking creation is a word-based analysis of commits and opened and

closed issues. It uses natural language processing while preprocessing the historical commits for CTM ranking.

The study has used personalized historical commit analysis in its recommendation process in order to decrease the burden of senior developers. It is more likely that senior developers are dealing with more serious and urgent jobs in a company. Thus, it will increase efficiency if the system doesn't always recommend a senior. But, as they have mentioned, if the existing methods are used which mainly consider the level of experience instead of analyzing every developer individually, then the system will recommend a senior. On the other hand, individual developing experience analysis gives a chance to junior developers to be recommended.

The authors conducted a survey with classified 200 issues and four groups to test the accuracy of their algorithm. They distinguished senior and junior developers by the number of commits that were submitted by a person. Their accuracy with the group which considerably has a fewer number of commits is better than the group which has more commits on average. In conclusion, they proved that their personalized history analysis accurately recommends junior developers and is able to avoid recommending senior developers all the time.

1.3.4 DREX: Developer Recommendation with K-Nearest-Neighbor Search and Expertise Ranking

The DREX approach, proposed by Wu et al. [8], is for recommending a group of developers that might have interest or knowledge for a bug report. This approach is a developer recommendation method for finding who may have knowledge about a newly generated bug report. It is explicitly stated in the paper that this approach is not aiming to find any developer who can work on the problem but finding knowledgeable people to discuss the issue. Another thing that is worth mentioning about DREX is that publicly available code or executables don't exist for this project.

The DREX has two components. The first component goes through historical bug reports and uses Natural Language Processing (NLP) to numerical vectors. Next, it uses K-Nearest-Neighbor (KNN) algorithm to find similar bug reports to new coming bug reports. The second component measures developers' experience using their activity in similar bug reports. How much they participated (frequency) in similar bug reports and Social Network Analysis are crucial when measuring the developer experience. Social Network Analysis analyzes complex relationships through graphs and networking. Stemming from graph analysis, DREX compares a few different metrics like closeness, centrality, betweenness, in-degree, out-degree, PageRank, and ML-KNN.

DREX has some similarities to Expert Developer Finder. Both projects use artifacts to recommend developers and aim to recommend more than one developer. DREX similarly uses a graph to model developers' expertise and the similarity of bug reports.

On the other hand, very distinct differences between these projects appear. For instance, the researchers did not examine the code to find which people are knowledgeable about a specific code unit. Furthermore, DREX works on bug reports and is designed to be executed when a new bug report arrives. On the other hand, our project functions differently. Developers will always be able to search for the experts on any of the code units. Also, the purpose is different. DREX finds knowledgeable people about bugs and our solution finds knowledgeable people about code.

1.3.5 DevRec: Multi-Relationship Embedded Software Developer Recommendation

The study [15] proposes a methodology for a developer recommendation system. Although the study suggests that the same methodology can be used for different recommendation purposes (i.e bug fixer recommendation, issue assigner recommendation), the study focuses on recommending code reviewers. Different from the previous studies so far, DevRec considers collaboration relationship (i.e. the developers who have previously collaborated with each other has a higher probability of accepting their review requests), interaction relationship (i.e. developers who has a higher interaction with the given tasks has a higher probability of accepting the review request) and similarity relationship (i.e. developers who have previously worked on similar tasks has a higher probability of accepting the review request) as their metrics in the recommendation.

DevRec uses the convolutional graph networks and attention mechanism to dedicate the weights of relationships in the multi-relationship graph. These weights are used to find the strong connections in the graph and result in accurate recommendations.

The findings reveal that explicitly encoding the collaboration relationships, interaction relationships, and similarity relationships into learning has many benefits: it alleviates the sparsity issue (the lack of historical data for developers who are new to the codebase, e.g. junior developers, interns, etc.), an elevation in the performance of the recommendation system (when compared to the other art-of-state approaches) and results in better interpretability of the recommendations (the reasoning behind why that developer was recommended is clearer).

The findings state that response time and response quality are also preferred metrics among developers when they are manually selecting reviewers. The study leaves the investigation of these two metrics to future works. Another interesting finding is that considering the workload of developers may reduce the recommendation accuracy unless the workload of developers is “large enough”.

1.3.6 Fuzzy set and cache-based approach for bug triaging

Bugzie, another similar project, is a Fuzzy set-based automatic bug-triaging tool made for an academic paper at Iowa State University [10]. As a data source, Bugzie collects the solved bug reports and who solved them. Next, it makes categories for reports using text classification. When a new bug report comes in, it suggests someone who can solve it. Currently, the tool is not available publicly. However, Bugzie has a similar concept to our

product. The difference is that, Bugzie is a bug-assigning tool. Our purpose is different. However, collecting artifacts, passing them through an algorithm, and offering based on this result is what we will be doing in this project too.

1.4 Constraints

1.3.1 Implementation Constraints

- The system will be developed as an extension to one of the widely used text editors: Visual Studio Code. Furthermore, the same services will be available for the user who doesn't use VS Code as a web application.
- The system will be compatible with public GitHub repositories. It will mine GitHub repositories only (for the initial version) to provide a recommendation. The upcoming releases aim to be compatible with GitLab repositories and BitBucket repositories as well.
- During the development of the systems, Git and GitHub will be used as the version control system.
- The web application will be developed by using Javascript, specifically React library.
- Vercel will be used to host the web application.

1.3.2 Economic Constraints

- Publishing extensions to Visual Studio Code is free of charge
- Deploying a web application will be free of charge with Vercel
- All of the developer tools to be used in the development period are free tools (Git, GitHub, etc.)

1.3.3 Sustainability Constraints

- VSCode uses Azure DevOps for its Marketplace services. This means that authentication, hosting, and management of extensions are provided through Azure DevOps Personal Access Tokens (PATs). PATs have a certain expiration period, which requires maintenance [16].
- A feedback system will be used to periodically update our system to improve the accuracy and the automated recommendation

1.3.4 Language Constraints

- The extension and the web application will be served in English

1.3.5 Social Constraints

- The system will provide a better inclusion of the junior developers who are not familiar with the repository

- Communication between developers with different experience levels will be less socially exhausting/intimidating

1.4 Professional and Ethical Issues

1.4.1 Professional Issues

- The source code of the system will be closed-source since it is harder to maintain open-source projects with an increasing number of contributors.
- Weekly meetings will be held with the team members and bi-weekly meetings will be held with the supervisor.
- The end product will be sent to real developers in the industry and their feedback will be collected to further improve our system.

1.4.2 Ethical Issues

- No personal sensitive data (gender, age, etc.) will be used in our metric in order to prevent discrimination against certain groups of people.
- The data mined from version control systems will not be published on any platforms under any circumstances. The produced system will not store any personal data of the developers/participants or the history of the repository
- VSCode ensures that any user must deliberately agree to give the extension access to the workspace they are working on.
- The extension's expert developer finder takes contributors' workload into account, preventing the same potential experts from being excessively assigned.
- Rating after the recommendation process can lead to conflicts between coworkers as taking low rates constantly from your coworkers may harm your reputation.
- The recommendation system may cause hard feelings between co-workers. For example, if developer A believes he/she is the expert for a given file Z and our algorithm recommends developer B instead of developer A, this may cause developer A to hold a grudge against developer B.

2. REQUIREMENTS

2.1 Functional Requirements:

Our system will have the following core functionalities:

- Specifying the related code part for looking up an expert for and specifying what kind of help is needed
- Automatic detection of the experts to be recommended and displaying them,
- Starting communication with the recommended experts
- Rating the performances of the experts (i.e. how helpful were they, how fastly did they reply) and rating our recommendation algorithm (i.e how accurate the suggested experts were).
- Additional functionalities

Besides these core functionalities that are available to all developers, repository owners (or those responsible who are integrating our system into their repositories) will also be able to edit our configuration file. By editing the configuration file they will be able to edit a part of our system according to their needs.

The system will be provided in two forms: An extension for Visual Studio Code, A web application /GitHub tool for users who don't use VS Code.

Interacting With Code Units

For the two platforms, there will be slightly differing functionalities due to the nature of differing platforms. Thus, for both forms, functionalities will be specified separately.

- VS Code Extension
 - Users can select a section of a file with their cursors to search for an expert
 - Users can select a method/function/class or other varying code blocks to search for an expert
 - Users can select a whole document to search for an expert
 - Users can select a folder to search for an expert
 - When users compile their program and receive compilation errors, they can search for an expert regarding that compile error from the terminal screen
- Web Application / GitHub Tool
 - For web application/ GitHub tool, searching for an expert for a compilation error will not be implemented
- For both of the platforms, users can seek help about the context of the code (the logic) or the syntax of the language. The experts regarding the syntax-related issues will be specified manually in the configuration file. The recommendation for syntax-related problems won't be achieved manually by mining the repository history (see section Configuration Opportunities).

Automatic Detection of the Experts and Displaying a List

- The system will recommend a list of 3-5 expert developers sorted by score instead of a single expert. By recommending multiple expert developers, we aim to reduce the time users receive help from the experts (if only a single expert were to be recommended, if that expert is currently occupied, this will prevent users from receiving help on time)
- Our algorithm will be recommending experts according to varying metrics including how recently they worked on the given artifacts, how much workload they have (e.g integration with Jira), what their ratings are (more information in the next section) and more
- A detailed information about the recommended expert (including their rating), the reason that expert was recommended and the confidence level of the recommendation will be available for users

Starting Communication With Experts

- For the selected expert developer, users can view their contact information directly.
- For the selected expert developer, users can start a Slack conversation immediately by selecting the regarding option
- For the selected expert developer, users can start a Zoom conversation immediately by selecting the regarding option
- For the selected expert developer, users can send an email immediately by selecting the regarding option

For all of the above options, a default message explaining the question/error will be generated for the developer who wishes to get help so that they can send it to the experts. They will choose between using the default message or typing a message manually before they send any of the above-mentioned invitations/emails/messages.

For all of the above options, developers who are getting help can select a single expert developer or more.

Providing Feedback

- After each recommendation, users will be asked to rate the expert who helped them. They will be rating the expert according to:
 - how fastly did they return to their claim for help
 - how much effort they put in
 - how much knowledge they were in that particular problem
- A free text area will be provided for the users to make comments regardless of criteria

Our algorithm will be continuously updating its recommendations according to the provided feedback about the experts. For each project, there will be a webpage showing all the reviews for each software practitioner. This will enable every practitioner of the project to be able to see each feedback given to each software practitioner.

- After each recommendation, users will be asked to rate our algorithm based on
 - how much of our recommended experts was accurate
 - was the number of recommended developers was enough
 - did the algorithm worked fast enough
- A free text area will be provided for the users to make comments regardless of criteria

Our team will be continuously updating the system, algorithm and its metric according to the provided feedback about our services.

Additional Functionalities to help developers

- Users can “bookmark” a specified unit of code (block, method, file, etc.) of interest for future accessibility

- Within a specific file, other files with similar content or similar file names are suggested for the developer to look at
- Methods/ Functions with similar names or content are suggested for the developer to look at
- For methods/ functions and classes/structs (i.e. code blocks), the files that are declaring the specified code blocks will be identified. Also, the code blocks that are used in the declaration file will be identified. This process will recursively continue until the topmost declaration files are found. Then, the parent tree will be visualized for the specified code block (i.e all the files for understanding the implementation of a code block will be visualized as a tree growing upwards)

Configuration Opportunities

- Configuration files will be held in the GitHub repository. Only the authorized developers should be able to edit them
- In the configuration file, authorized developers will be able to assign importance to different artifacts and different software practitioners (quality assurance specialists, testers, developers, designers)
- In the configuration file, authorized developers will be providing the emails of all the developers they wish to be recommended by our system
- In the configuration file, authorized developers will be able to provide Slack usernames if the project uses Slack as well. This will enable us to direct users to Slack chat immediately.
- In the configuration file, authorized developers will be able to provide Zoom usernames if the project uses Zoom as well. This will enable us to direct users to Slack chat immediately.
- In the configuration file, authorized developers will be able to provide Jira usernames if the project uses Jira as well. This will enable us to recommend experts bearing in mind their workloads.
- In the configuration file, authorized developers will be able to specify the expert on the languages used in the project. This information will be used when users want to ask questions regarding the syntax instead of the context of the code (i.e. the logic).

2.2 Non-Functional Requirements:

2.2.1 Effectiveness/Efficacy

Effectiveness is the most important requirement for our project. The value that we offer to users is that we recommend the true experts for a specified code part/artifact. If our recommendations are not accurate enough, we cannot keep the promised value of our project. Thus, the project becomes useless. Furthermore, recommending the wrong experts may result in damage to the project's health as well. With wrong experts, wrong information may be passed among developers, resulting in a slowdown at the productivity of the development.

- The initial release should aim for a success rate of 0.70 (i.e. the confidence levels of our recommendations should match with the feedback we received from the users in a ratio of 0.70). See the note **Success Rate** below for further explanation

- Each new release should not decrease the success rate of the previous release. In case of a reduction in the success rate, the release should be reverted to its previously stable version
- The success rate needs to be tried to be improved periodically for better results by looking at the logs and feedback of the users. This process should be repeated at least twice a year

Success Rate:

Consider a case where our algorithm runs and finds an expert. Our algorithm also suggests a confidence level for its suggestion. The confidence level is the indicator of how knowledgeable we think that the expert is. For example, we recommend developer A with a confidence level of 60%. This means that we believe the most suitable expert for your query is developer A but we also believe that developer A has an understanding of 60 on a scale of 0 to 100. Yet, since 60% is better than all the other candidates, we recommend developer A for your query. When the user provides feedback about how knowledgeable the expert was, if they state that the expert was 55% knowledgeable, this means that our recommendation was 95% successful (there is an error of 5%).

2.2.2 Performance

Performance is a crucial requirement for our project. As we offer to fasten the process of finding the correct expert, the project would lose its meaning if our projects performed poorly in terms of speed.

- For VS Code extension, the installation should take less than 30 seconds with a moderate internet connection (12- 25 mbps) [17].
- The algorithm should recommend a list of experts in less than 30 seconds
- The extension should save the responses to the feedback questions (see **Providing Feedback** under **Section 2.1 Functional Requirements**) in less than 3 seconds

2.2.3 Maintainability

Maintainability is a key requirement for our project since we promise to update our algorithm metric continuously to give better performances (see **Providing Feedback** under the **Section 2.1 Functional Requirements**). Thus, we need our project to be easily modifiable.

- High cohesion and loose coupling among the project files/ modules
- Use of encapsulation (needed for facilitated debugging)
- Using a consistent commenting method (like Javadoc or JSdoc) for each class (methods, properties, etc.). This will make code more understandable and will save lots of time for further bug-fixing / feature adding
- Providing documentation for each version of the project and storing them in a publicly accessible way (documentation of each version of the project can facilitate developers to understand that version of the code better and can be useful when a reversion to the old version is needed)

- The project should be modified at least twice a year to meet up the effectiveness requirements (see **Section 2.2.1 Effectiveness**)

2.2.4 Portability

Portability is an important non-functional requirement for our project since our target users (i.e. developers) tend to work on numerous environments (e.g. they use Linux) unlike many of the non-developers (who mainly use Windows and MacOS only).

- Since Visual Studio Code supports all OS types (Windows, macOS, and Linux), our VS Code extension should also work on all operating systems.
- The web application should work on any browser that has the following versions or higher: Chrome v. 106.0.5249.103, Safari v. 12.1.2, Mozilla Firefox 105.0, Edge v. 106.0.1370.34.

2.3 Functional Priority

Although we have decided what our priority functionalities are since this is a product that will be used by developers, we are planning on sending a survey to junior and experienced developers before finalizing our functional priorities. Below, we will be declaring our decisions for the functional priorities (the final functional priorities decided regarding the survey results will be given in the **Analysis and Requirements Report**)

- Recommendation of a list of expert developers for a specified code block (i.e. methods/functions, classes/structs)
- Recommendation of a list of expert developers for a specific part of code in a file with the cursor
- Recommendation of a list of expert developers for a specified file
- Recommendation of a list of expert developers for a specified folder
- Recommendation of a list of expert developers for compilation errors
- Sending automatic emails for asking for help
- Zoom integration for sending automatic Zoom invites
- Slack integration for sending automatic Slack messages
- Providing a 1 minute survey that asks for an evaluation about the performance of our product and the accuracy of our recommendation algorithm
- Providing a 1 minute survey that asks for an evaluation about the expert developer
 - However, we are concerned that this feature may not be desired by the developers. Thus, we will make the final call according to the survey answers

References

- [1] A. Ç. Eray Tüzün, “Analyzing Software Projects using Artifact Traceability Graphs,” *TÜBİTAK*.
- [2] “HackerLife,” *hackerlife*. <https://hackerlife.co> (accessed Oct. 14, 2022).
- [3] M. Zhou and A. Mockus, “Developer fluency: achieving true mastery in software projects,” in *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, New York, NY, USA, Nov. 2010, pp. 137–146. doi: 10.1145/1882291.1882313.
- [4] J. Desjardins, “10 Types of Innovation: The Art of Discovering a Breakthrough Product,” *Visual Capitalist*, Jul. 01, 2020.
<https://www.visualcapitalist.com/10-types-of-innovation-the-art-of-discovering-a-breakthrough-product/> (accessed Oct. 14, 2022).
- [5] A. Verma, “Digital Business – Transformation vs Optimization,” *Linkedin*.
<https://www.linkedin.com/pulse/digital-business-transformation-vs-optimization-anoop-verma/> (accessed Oct. 14, 2022).
- [6] “Git - git-blame Documentation.” <https://git-scm.com/docs/git-blame> (accessed Oct. 14, 2022).
- [7] “Core Features | GitLens,” Sep. 07, 2022.
<https://help.gitkraken.com/gitlens/gitlens-features/> (accessed Oct. 14, 2022).
- [8] W. Wu, W. Zhang, Y. Yang, and Q. Wang, “DREX: Developer Recommendation with K-Nearest-Neighbor Search and Expertise Ranking,” in *2011 18th Asia-Pacific Software Engineering Conference*, Dec. 2011, pp. 389–396. doi: 10.1109/APSEC.2011.15.
- [9] “UnReview Overview | GitLab.”
<https://about.gitlab.com/handbook/engineering/development/data-science/appliedml/projects/unreview/> (accessed Oct. 14, 2022).
- [10] A. Tamrawi, T. T. Nguyen, J. M. Al-Kofahi, and T. N. Nguyen, “Fuzzy set and

cache-based approach for bug triaging,” in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, New York, NY, USA, Sep. 2011, pp. 365–375. doi: 10.1145/2025113.2025163.

- [11] “Hipikat: Recommending Useful Software Artifacts.”
<https://www.cs.ubc.ca/labs/spl/projects/hipikat/> (accessed Oct. 14, 2022).
- [12] “Effectiveness of exploring historical commits for developer recommendation: an empirical study | SpringerLink.”
<https://link.springer.com/article/10.1007/s11704-016-6023-3> (accessed Oct. 14, 2022).
- [13] O. Cury, G. Avelino, P. Santos Neto, R. Britto, and M. Túlio Valente, “Identifying Source Code File Experts,” in *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Helsinki Finland, Sep. 2022, pp. 125–136. doi: 10.1145/3544902.3546243.
- [14] X. Sun, H. Yang, X. Xia, and B. Li, “Enhancing developer recommendation with supplementary information via mining historical commits,” *Journal of Systems and Software*, vol. 134, pp. 355–368, Dec. 2017, doi: 10.1016/j.jss.2017.09.021.
- [15] Xinqiang Xie, Xiaochun Yang, Bin Wang, and Qiang He, “DevRec: Multi-Relationship Embedded Software Developer Recommendation,” *IEEE Transactions on Software Engineering*, pp. 1–1, Oct. 2021, doi: 10.1109/TSE.2021.3117590.
- [16] chcomley, “Use personal access tokens - Azure DevOps.”
<https://learn.microsoft.com/en-us/azure/devops/organizations/accounts/use-personal-access-tokens-to-authenticate> (accessed Oct. 14, 2022).
- [17] Kristin Shaw, Onjeinika Brooks, and Rachel Hisle, “What Is a Good Internet Speed?,” *U.S. News*, May 23, 2022. [Online]. Available:
<https://www.usnews.com/360-reviews/services/internet-providers/what-is-a-good-internet-speed>