DECISION SUPPORT SYSTEM FOR PULL REQUESTS REVIEW USING PATH-BASED NETWORK PORTRAIT DIVERGENCE AND VISUALIZATION

A Thesis

IN

Computer Science

Presented to the Faculty of the University
of Missouri–Kansas City in partial fulfillment of
the requirements for the degree

MASTER OF SCIENCE

by
VASIM SHAIKH

M.S., University of Missouri - Kansas City, USA, 2022

Kansas City, Missouri
2022

DECISION SUPPORT SYSTEM FOR PULL REQUESTS REVIEW USING PATH-BASED NETWORK PORTRAIT DIVERGENCE AND VISUALIZATION

Vasim Shaikh, Candidate for the Master of Science Degree

University of Missouri–Kansas City, 2022

ABSTRACT

Pull requests are widely used in open-source and industrial environments to contribute and assess contributions. Unlike the typical code review process, pull requests provide a more lightweight approach for committing, reviewing, and managing code changes. Pull request code reviews also serve multiple objectives, including detecting problems in code, giving a venue to discuss code contributions, and supporting the easy integration of external contributions by project maintainers. The code changes and tests are written for a specific work item are contained in a pull request. Previous studies have reported that pull request review is crucial for software development and that reviewers do not spend more time on test files than on code files. At the same time, code reviewers are concerned that the tests that accompany the code modifications are adequate and cover all possible paths. The purpose of this research is to determine whether the test changes that go along with the code changes match the structural changes made in the Pull Request. The structural changes are determined using recent network comparison breakthroughs in prior work with GraphEvo. We also determine whether or not the visual representation and software metrics can support the software review process. We conducted a case study of 14 Java open-source projects, analyzing thousands of lines of code quality issues in 627 pull requests. We calculated the class level metrics, including network portrait divergence for each Pull request with and without change. In addition, for each pull request, we counted the number of existing test cases that failed due to the modification. Furthermore, correlations were investigated between class-level metrics, including network portrait divergence and tests that failed in pull requests.

APPROVAL PAGE

The faculty listed below, appointed by the Dean of the School of Computing and Engineering, have examined a thesis titled "Decision Support System for Pull Requests Review Using Path-based Network Portrait Divergence and Visualization," presented by Vasim Shaikh, candidate for the Master of Science degree, and hereby certify that in their opinion it is worthy of acceptance.

<u>Supervisory Committee</u>

Yugyung Lee, Ph.D., Committee Chair
Department of Computer Science Electrical Engineering, UMKC

Sejun Song, Ph.D.
Department of Computer Science Electrical Engineering, UMKC

Ye Wang, Ph.D.
Department of Communication & Journalism, UMKC

Gharib Gharibi, Ph.D.
TripleBlind, Inc., Kansas City, MO

CONTENTS

ILLUSTRATIONS

TABLES

ACKNOWLEDGEMENTS

This thesis would not have been possible without the extensive support, care, and advice provided from those around me. I would like to start by thanking my academic advisor Dr. Yugyung Lee for introducing me to the world of Machine Learning, and welcoming me into their research group with open arms. Through them, I have learned an immense amount, and have been able to build my skill set practically from the ground up. Thank you for your patience, knowledge, and direction. For answering every question I have thrown your way, and pushing me to be a better researcher. Through you, I have found a passion for Deep Learning, and look forward to becoming more and more well-versed in it as my career progresses.

Next, I would like to thank my support system: my wife and daughter. I would be utterly lost without all of your encouragement, kindness, and care for my well being. Your support has held me up through thick and thin, and I deeply appreciate how each and every one of you has opened yourselves to me for advice, assistance, and emotional support. Last, I want to thank Vijay Walunj for allowing me to be part of this research and guiding me on every single step.

CHAPTER 1

INTRODUCTION

1.1 Overview

A variety of code review methodologies have been put forth in the past and are frequently used by open-source and for-profit enterprises. Code reviews, which involve manual code inspection by various developers, assist businesses in lowering the amount of errors and raising the caliber of their software. When developers used to plan review sessions to examine the code line by line, code reviews were frequently conducted in that manner. Today, both the business community and academics concur that code inspection helps to lower the amount of flaws. The time and effort needed to carry out code inspections, however, occasionally prevents their widespread use. Companies have been able to adopt more lenient code review practices thanks to the blessing of new tools and procedures. Several companies, including Facebook, Google, and Microsoft, perform code reviews employing tools such as Gerrit[1] or the pull request mechanism provided by Git.

We concentrate on pull requests in the framework of this thesis. Pull recommendations give developers a straightforward way to contribute to projects, and many well-known projects, both open-source and for-profit, use them to evaluate the efforts of various developers. Researchers have concentrated on mechanisms for pull requests [1, 2], examining several areas, such as the review procedure, the impact of code reviews on

[1]https://www.gerritcodereview.com/

continuous integration builds, the assignment of pull requests to different reviewers, and the circumstances under which they are accepted. Few studies have looked into whether developers accept pull requests based on quality considerations. The reputation of the developer who submitted the pull request is reportedly one of the most crucial acceptance factors, according to many studies. To our knowledge, however, no studies have looked into whether the caliber of the code sent in a pull request affects whether this pull request is accepted [3]. As a crucial component of pull requests, code reviews, we firmly believe that pull requests containing subpar code should typically not be allowed.

This research aims to determine whether the test changes that go along with the code changes match the structural changes made in the Pull Request. The structural changes are determined using recent network comparison breakthroughs in prior work with GraphEvo [4]. We conducted a case study of 14 Java open-source projects, analyzing thousands of lines of code quality issues in 627 pull requests. We calculated the class-level metrics, including network portrait divergence for each Pull request with and without change. In addition, for each pull request, we counted the number of existing test cases that failed due to the modification. Furthermore, correlations were investigated between class-level metrics, including network portrait divergence and tests that failed in pull requests [5].

1.2 Motivation

A well-liked method of contemporary collaborative software development that is housed on GitHub is known as pull-based development. A distributed development paradigm can review a pull request before being merged into the main branch or repository. A pull request takes into account new functionality, bug fixes, and maintenance requests from both the integrator and the contributor.

It appears that numerous empirical research is carried out to ascertain the process of pull request evaluation. Furthermore, little research has been done to our knowledge to evaluate the quality and preparedness of pull requests. Numerous pull requests made at once, according to a study, can cause software systems to include problems. Thus, a rigorous examination of the pull requests is necessary [6]. So, the topic of whether the code review process improves the quality of the code is consequently one that merits further investigation.

1.3 Problem Statements

The pull request review process is impacted by a variety of factors, and open pull requests are frequently ignored.

- **Lack of Time -** Study shows that one of the top 5 complaints that engineering leaders have reported is that their developers can't review the code because lack of time. Developers may be busy with their own tasks and sometime needs multiple follow ups to get the code reviewed. If it is a simple change then review can be faster. However, if it's a complex change them it may take minutes to few hours to review the code change and it's impacts.

- **Lack of Skills -** The experienced reviewers are the ones who are seniors on the team. Usually, in a team, a pull request raised by a new developer will be reviewed by the most experienced team member, whereas vice versa is not valid. Nowadays, code review skill is as essential as the quality of code a developer can write.

- **Lack of Context -** Some developers have a propensity to automatically disqualify anyone who shows even the slightest initial unfamiliarity with the modifications they are presenting. Ignorance of code base or features will have a direct impact on confidence. A developer may be reluctant to review a pull request for various reasons, but one of them is that they are concerned about publishing or speaking inappropriately.

1.4 Summary of Contributions

This section describes the goal of the thesis and the non-functional and functional requirements for creating a tool that combines the information of code changes and test coverage changes. Also, it generates the call graph visualization and software metrics to support the pull request review process.

1.4.1 GraphEVOPR

The tool GraphEVOPR is what this thesis is made of. A pull request review aid tool GraphEVOPR helps developers become more aware of changes to the code and changes to test coverage simultaneously. By merging data into a single report, it may be utilized to track the co-evolution of code modifications and test code changes.

Open-source projects should have the means to verify the pull requests submitted for their projects in light of GitHub's rising popularity. The best tool for use in version control systems is GraphEVOPR. This thesis aims to describe the functionality of such a tool and the structure of the generated reports. As a result, the tool will be used to implement and analyze real-world open-source projects.

1.4.2 Non-Functional Requirements

A desired execution of the tool has a few non-functional constraints. The instrument must be simple to use, first and foremost. This means running it shouldn't require any particular setup or additional dependencies. The program must be able to be configured to run for several projects and folders; therefore, a setup should be possible. The configuration of the two project versions that should be compared and the location of the generated reports should be covered by these variables. The tool must support both versions to compare them. These versions must build without errors, and a test suite must be run consistently.

1.4.3 Functional Requirement

In this section functional requirements of the tool are described, which are within the scope of this thesis :

- We should be able to Compare source code for changes in a pull request.
- We should be able to identify the role tests play in Pull Request review.
- We should be able to check the influence of path-based visualization aid on a software Pull request review process.
- We should be able to construct a report of software changes and respective tests.

1.5 Structure of Thesis

1.5.1 Chapter 2: Related Work

In dispersed software teams, the pull based development model which is made possible by git and made popular by collaborative coding platforms is extensively utilized like BitBucket[2], Gitorius[3], and GitHub[4]. Since anyone can raise a pull request to any repository, this model lowers the entry barrier for potential contributors. Still, it also burdens integrators, who must review and incorporate proposed changes into the main development line while also trying to keep up with the influx of pull requests. The modern pull request visualization means that the code or software change and its impact on the software are illustrated graphically. The pull request visualization provides additional code information and helps a faster review process. This visualization helps reduce the cycle time and reveals the software structure or architectural issues before merging the code. This chapter is to elaborate more on how the software code structure is and understand more about the interaction of software code components with the help of the GraphEVO tool.

To compare networks utilizing their portraits, portrait divergence was recently developed. Portrait divergence is based on information theory, contrary to the prior ad hoc comparison measures, and offers a coherent interpretation of the divergence metric. As a result, comparing the networks based on their different topology structures is possible rather than assuming that they are built on the same set of nodes. Portrait divergence is

[2]https://bitbucket.org/
[3]https://gitorious.org/
[4]https://github.com/

also network invariant, making it computationally more efficient than the present pricey node matching optimization techniques. It should be listed that this method can handle both directed and undirected networks equally.

1.5.2 Chapter 3: Methodology

We follow the study design in this chapter to determine the quality of pull requests before merging and the potential of using the static analysis technique. First, we selected the open-source java project repository, including multiple pull requests with code and test changes. We then pulled the code using two different methods 1) using jar files prePR and post PR, 2) an API to pull the source code, for further processing in the GraphEvoPR tool. We are mainly interested in the software code change, how it is impacting the structure of software, and respective test changes. Second, we run the source code in the GraphEvoPR tool and generate metadata related to class paths, software metrics, and network portrait divergence. Third, we then apply static analysis techniques to evaluate the code change and its impact along with respective test changes. Finally, based on the evaluation, we determine the level of pull requests. Is it reasonable to accept or reject, and what's the impact of the code change. And can it support the pull request review process.

1.5.3 Chapter 4: Result and Evaluation

GraphEvo examines the call graphs created in the preceding phase for each software release to quantify and illustrate the program structure. To identify changes in the structure, we first examine meaningful graph metrics, such as the number of nodes and edges, the average degree of the network, the clustering coefficient, the dimension of the graph, and its modularity. Second, we use the Portrait Divergence approach to compare software versions based on their portraits. The main objective is to show that software call graphs may be used to measure and visually portray software evolution using graph metrics and portrait divergence.

1.5.4 Chapter 5: Conclusion

We have studied and analyzed 14 open source java projects. From these 14 projects, we deep-dived into 627 plus pull requests and compared different software metric data. Primarily, we compared the network portrait divergence data from previous and current pull requests to understand the impact of code change. We further analyzed the behavior of tests related to the code change.In this chapter, we elaborate on the conclusion and future work.

CHAPTER 2

RELATED WORK

2.1 Overview

Users with read access can review and comment on the changes proposed in a pull request once it has been open. Additionally, developers can make precise changes to code lines that the author can implement right from the pull request. In public repositories, users can, by default, write reviews that accept or demand changes to a pull request. Owners of organizations and repository administrators can restrict who can submit and approve pull requests or make change requests. The review of a pull request can be requested from a specific individual by repository owners and collaborators. Members of the organization may also ask a team with read access to the repository to review a pull request [7].

Developers can designate a portion of the team to be automatically assigned instead of the entire squad. Reviews enable discussion of suggested modifications and ensure that the changes adhere to other quality requirements and the repository's contributing guidelines. In a CODEOWNERS file, developers can specify which teams or individuals control particular categories or sections of code. A person or team will automatically be asked to evaluate code with a stated owner when a pull request alters it [8].

Fig. 1 shows the end to end process of a pull request. The code contributor creates a copy from the source code, works on changes, commits the changes in the local branch,

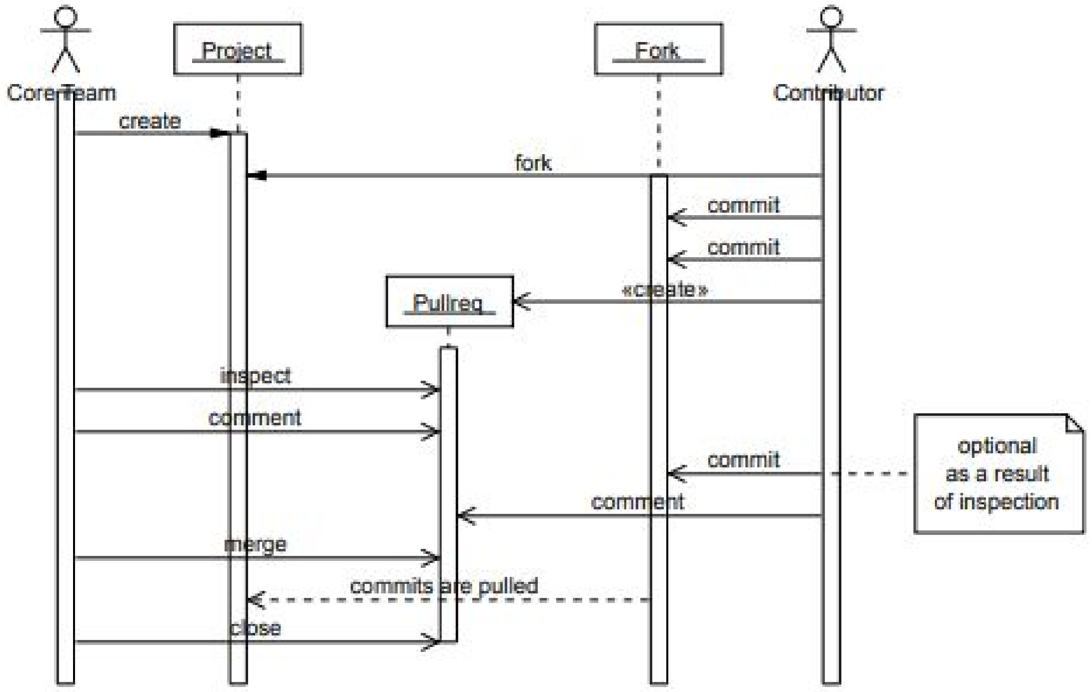


Figure 1: Pull Request Process

and submits the pull request. If the changes are approved, those get pulled into the main version of the code base. Further details are in Section 2.2.3.

2.2 Pull Request and Test Coverage

2.2.1 Development turnover

Fast development turnover, or the interval between the submission of a pull request and its approval in the project's central repository, is one of the promises made by the pull request paradigm. The time to submit 50 of the contributions to the central project repository spans from a few hours to less than three days, according to research on the patch submission process in projects like Apache and Mozilla. According to these figures, pull-based development via pull requests might be more effective than conventional email-based patches. Additionally, rather than the qualities of the pull request itself, project-related factors influence the turnaround time [9]. This means that the project must primarily fine-tune its methods (namely, testing transparency and coverage) for quicker turnover[1], see Fig. 2.

2.2.2 Managing pull requests

In of the study researches found out that interviewers rank managing pull requests as the most crucial project task. According to researcher Dabbish, project managers inferred information about the caliber of a code contribution based on the contribution's style, effectiveness, thoroughness (e.g., was testing included?), and the submitter's prior work. However, they do not appear to have as big of an impact on the merging selection process as other inspection points cited by project managers (testing code in pull requests, track record). To process pull requests quickly, the developer's track record is crucial [10]. Furthermore, they discovered that roughly 53 percent of rejected pull requests are denied due to the distributed nature of pull-based development.

2.2.3 How pull request work

A pull request enables developers to add new functionality or fix bugs without changing the main project code or the user interface. They can write and test code modifications locally in this way without worrying that they'll break the entire product. Pull

[1]https://smallbusinessprogramming.com/optimal-pull-request-size/

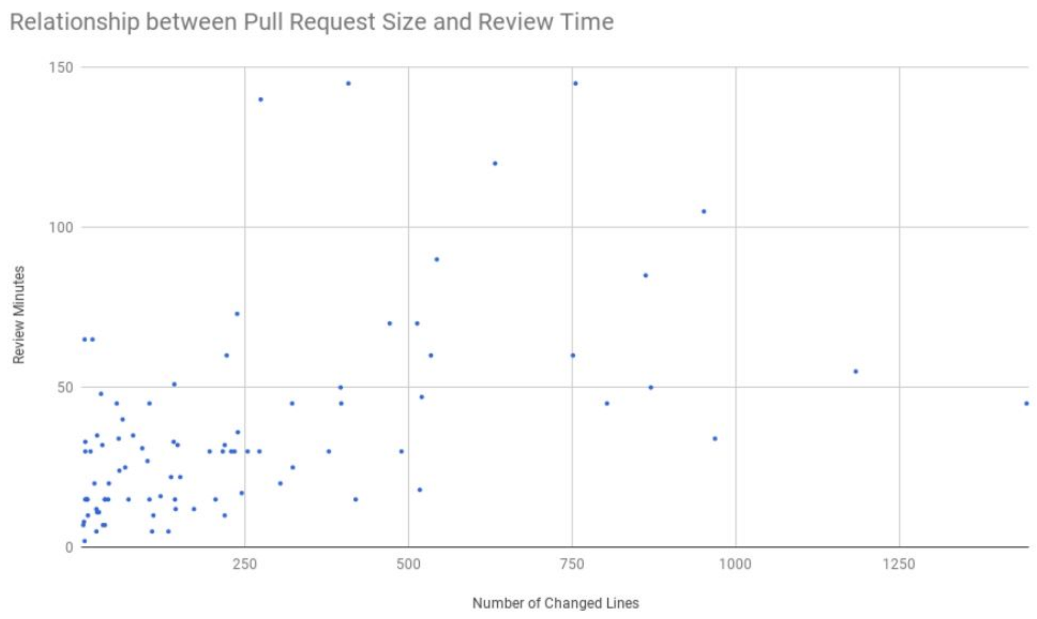


Figure 2: Relation Between Pull Request and Review Time

requests go via a standard five-step procedure:

- Create a local clone by forking the central repository. The developer first forks the primary repository, which they then clone onto their local system.
- Make the necessary local changes. Whether they are trying to fix a problem or add a new feature, the developer can add or update the code as necessary.
- Push Local Modifications into the Forked Repository. The developer pushes the updated code to the forked repository they established in step one after finishing and testing it.
- Publish a pull request. This is where the pull request happens! The main repository maintainer is notified following a pull request for review. Following a review of the

developer's forked repository's work, the maintainer will submit any comments or modification requests that require approval

- The developer is then notified of any revisions and asked to make any necessary extra commits (code changes).
- The maintainer accepts the pull request if no changes are required.
- Join the main project. New modifications made by the developer in the forked repository are merged with the primary project repository when the repository maintainer has accepted a pull request. After the new feature or bug remedy is added, the product is updated and made available for end-users to examine.

Table 1 lists the characteristics of a pull request [11]. It is established from our research on pull requests that these factors alone might not be sufficient when evaluating a pull request. Additionally, the review process will move more quickly if we give visual representations of software structural modifications and software metrics.

2.2.4 Test Coverage

To improve the dependability and quality of their products, many modern software development teams test their code. We can demonstrate the existence of bugs in the program by testing, but never their absence. Bezier[2] asserts that testing is straightforward and only requires that developers "identify a graph and cover it." To do this, a test criterion that will be used to evaluate the program must be specified. But what exactly are the test

[2]https://dl.acm.org/doi/10.5555/79060

Table 1: Pull Request Characteristics

Metric	**Description**
lifetime minutes	Minutes between opening and closing
merge time minutes	Minutes between opening and merging (only for merged pull requests)
num commits	Number of commits
src churn	Number of lines changed (added plus deleted)
test churn	Number of test lines changed
files added	Number of files added
files deleted	Number of files deleted
files modified	Number of files modified
files changed	Number of files touched (sum of the above)
num commit comments	The total number of comments included in a commit
num issue comments	The total number of comments included in an issue (of a commit)
num comments	The total number of comments included in a commit (sum of the above)
num participants	Number of participants in the discussion

criteria? What qualifies as an adequate test is determined by a test criterion. Or, to put it another way, a test adequacy criterion specifies the characteristics of a program that must be put to the test for it to be considered thorough, which means that the tested program has no errors.

A formal definition of test adequacy is the function C:

$$Formula : C : P \times S \times T \to [0, 1] \tag{2.1}$$

If C(p,s,t) = r, then the testing program p's suitability as measured by the test set's adequacy concerning the specification s is of the degree r as determined by criterion

C[3]. To put it another way, when a program is evaluated using a specific test set, a result between 0 and 1 is generated, indicating how successfully the program was tested about a specific criterion. This is typically reported as a percentage.

Different test sufficiency standards have been developed throughout many years of research. The criteria employed in this thesis are program-based structural criteria that depend on a program's control flow. A unit test typically uses a program-based criterion to assess relatively tiny program components. The test cases for structural testing, also known as white-box testing, are derived from the program's internal structure, including its branches and specific circumstances. The most significant structural criteria are described in the sections that follow.

2.2.4.1 Statement Coverage

Statement coverage is the most fundamental control-flow requirement to consider. It is the most basic and ineffective type of testing. It states that at least one test case must cover each statement in the program [12]. Full statement coverage is unfortunately not always achievable owing to unreachable code, which is undetected because it is not always feasible to determine whether a piece of code is inaccessible.

2.2.4.2 Branch or Decision Coverage

Branch coverage, also called decision coverage, is a more stringent condition for an appropriate test. Every control transfer, according to this, needs to be verified. In other words, every option should be tested at least once for each decision in the program.

[3]https://onlinelibrary.wiley.com/doi/abs/10.1002/stvr.4370020204

Statement coverage is less than branch coverage because all statements will be tested if all branches are tested. The subsumes relation is the name given to this relationship. However, branch coverage is still somewhat shaky in the end, mainly when decisions involve multiple conditions. There are exponentially more possible input variable combinations in this situation, but only the decision's outcome is considered [13].

2.2.4.3 Condition Coverage

Condition coverage is a factor that occasionally has more weight. For this requirement, a test case must allow each condition in every choice to experience every potential result at least once. We claim that it is sometimes more substantial because even if we get 100% condition coverage, that does not guarantee 100% decision coverage. This is so because condition 5 coverage does not account for the result of choice.

2.2.5 Code Review Process

Many large projects, both open-source and commercial, employ pull requests because they make it simple for developers to contribute to projects, and they do so to evaluate the efforts of other developers. Contributors fork and clone projects to their private repositories, edit the code and tests, and then submit their edits as pull requests. Reviewers are in charge of assessing pull requests after submission and deciding whether or not to accept or reject the modifications, see Fig. 3.

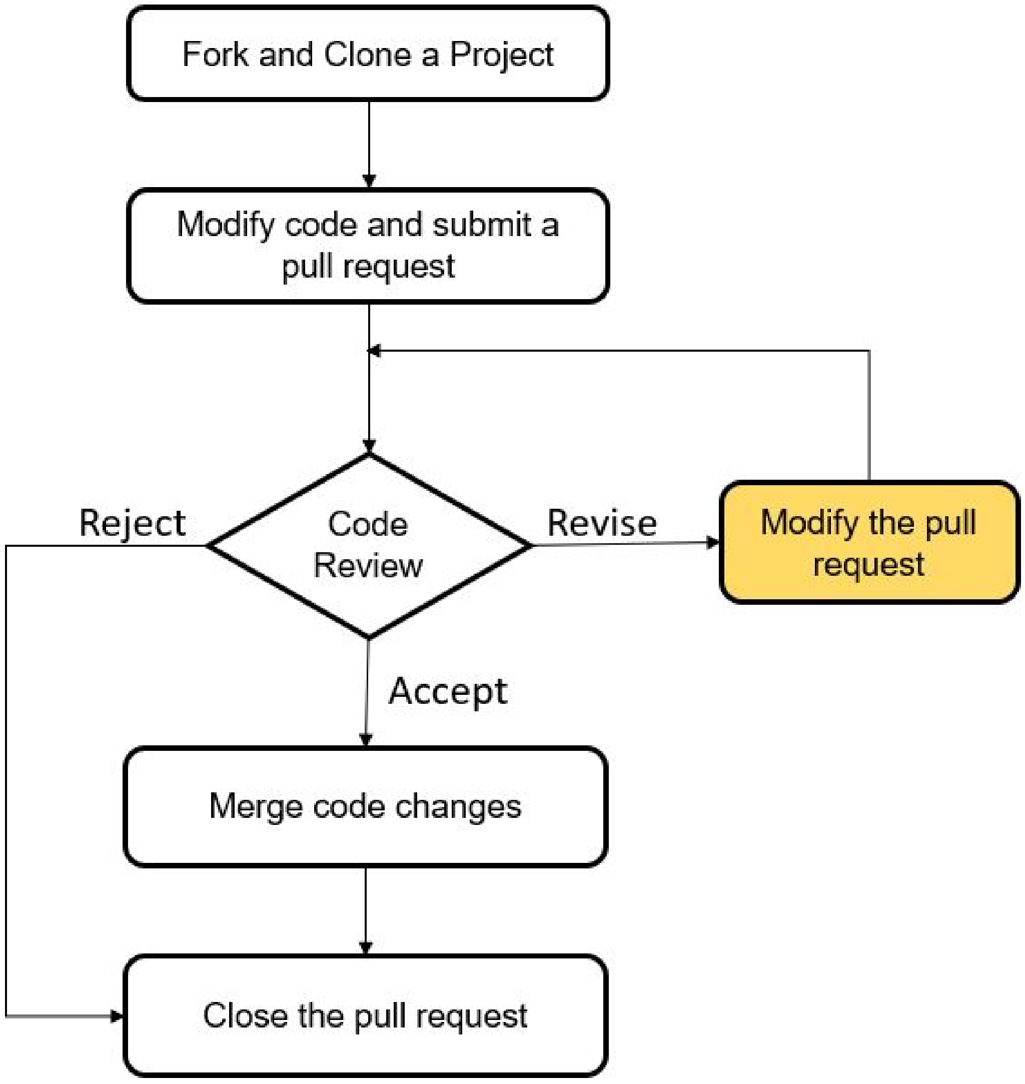


Figure 3: Code Review Process

2.3 Pull Request Review

A well-liked method of contemporary collaborative software development that is housed on GitHub is known as pull-based development. A distributed development paradigm can review a pull request before being merged into the main branch or repository. A pull request takes into account new functionality, bug fixes, and maintenance requests from both the integrator and the contributor.

It appears that numerous empirical research is carried out to ascertain the process of pull request evaluation. Furthermore, little research has been done to our knowledge to evaluate the quality and preparedness of pull requests. Numerous pull requests made at once, according to a study, can cause software systems to include problems. Thus, a rigorous examination of the pull requests is necessary [6]. So, the topic of whether the code review process improves the quality of the code is consequently one that merits further investigation.

2.3.1 Good Pull Request

One of the many things to consider when making a pull request is writing clean code. Large pull requests significantly slow down the code review process and make it easier for defects to enter the source. The developer should therefore give the pull request itself some consideration. It should only accomplish one task, be succinct, and have a strong title and description.

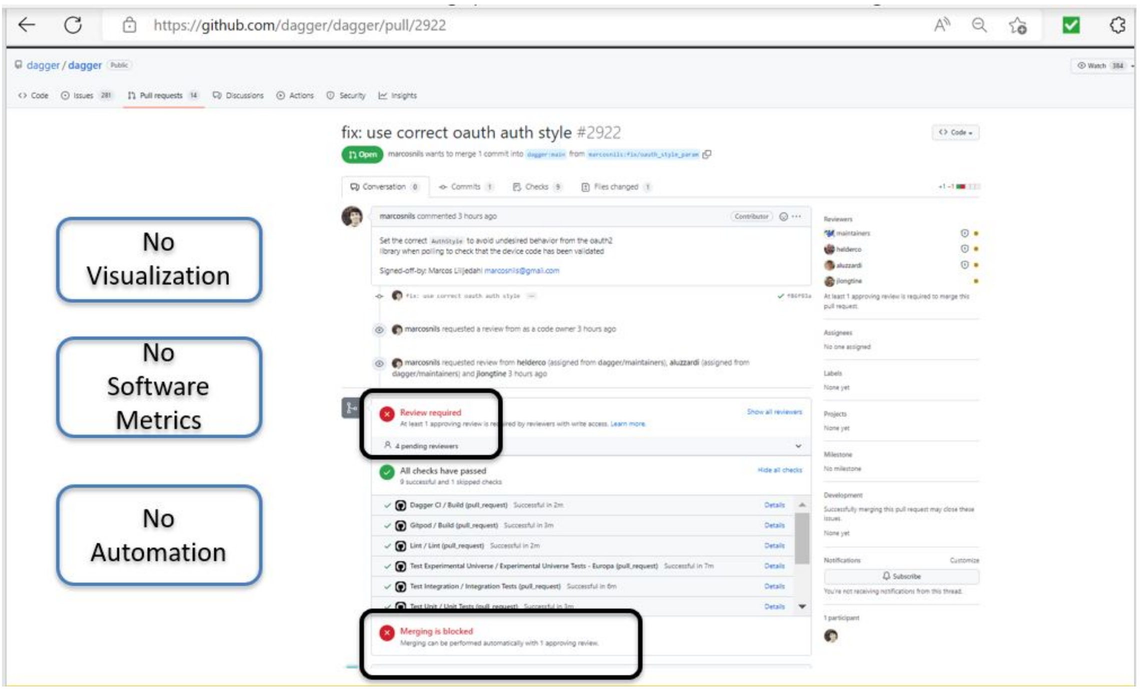
- A good pull request is one which can be reviewed quickly. It means that it should contain minimum changes as much as possible.

- A newly, added member in a team should be able to contribute in a pull request review process. It should ease an ob-boarding of new member.
- It is important to note that, a pull request should not introduce new bugs in an existing code base.
- Multiple developers or team members should not spend time on a single pull request for longer.
- It should expedites evaluation and, ultimately, product development.

2.3.2 Pull Request Structure Review

Pull Request (PR) reviews can be challenging. If someone claims that going through a pull request is simple, likely, they haven't done it. The size of the pull request is important while evaluating them. The size of the pull request should be kept to a minimum. No more than 250 lines of alteration should be used. The features should be divided into smaller pull requests wherever possible. A single item of work should be the focus of each pull request. The pull request's title should make clear what it does and should be self-explanatory. What, why, and how it was changed should be mentioned in the pull request's description. According to our research, very few reviewers concentrate on the pull request's software structure review.

PR shows the changes the author made to the files. A reviewer comments on a particular line of code when they find a potential issue or wish to suggest a better approach. Typically while reviewing the PR developers or peers looks for - What does a particular



fix: use correct oauth auth style #2922

Open marcosnils wants to merge 1 commit into dagger:main from marcosnils:fix/oauth_style_param

marcosnils commented 3 hours ago

Set the correct `AuthStyle` to avoid undesired behavior from the oauth2 library when polling to check that the device code has been validated

Signed-off-by: Marcos Lilljedahl marcosnils@gmail.com

`fix: use correct oauth auth style` f86f93a

marcosnils requested a review from as a code owner 3 hours ago

marcosnils requested review from helderco (assigned from dagger/maintainers), aluzzardi (assigned from dagger/maintainers) and jlongtine 3 hours ago

Review required
At least 1 approving review is required by reviewers with write access. Learn more.
4 pending reviewers

All checks have passed
9 successful and 1 skipped checks

- Dagger CI / Build (pull_request) Successful in 2m
- Gitpod / Build (pull_request) Successful in 3m
- Lint / Lint (pull_request) Successful in 2m
- Test Experimental Universe / Experimental Universe Tests - Europa (pull_request) Successful in 7m
- Test Integration / Integration Tests (pull_request) Successful in 6m
- Test Unit / Unit Tests (pull_request) Successful in 3m

Merging is blocked
Merging can be performed automatically with 1 approving review.

At least 1 approving review is required to merge this pull request.

No one assigned

None yet

None yet

No milestone

Successfully merging this pull request may close these issues.

None yet

You're not receiving notifications from this thread.

Figure 4: Pull Request Example

PR do, does PR do what it suppose to do, does PR do what it suppose to do it correctly, are there any Logical errors, is the code readable, is the function is too long, are there to many nested conditions, can code be optimised, what are the return condition, are the changes covered by tests, are the changes covered by enough tests, is code clean, etc.

2.3.3 Pull Request Examples

Fig. 4 shows few things about a pull request. It is in blocked status because it is pending with review. Also, there is no visual representation or software metrics of the changes being submitted. This pull request may remain open because of unavailability of reviewer or no one wants to review it because it's time consuming, or lack of skills etc.

2.4 Visualization in Software Engineering

For software development tasks like reuse, testing, maintenance, and evolution, understanding the program structure and its organization is a crucial and essential prerequisite [14]. A software system's capabilities and internal interconnections grow larger and more sophisticated as it develops over time. The processes of adding or modifying a feature raise the dangers of making errors or altering the behavior pattern of the software program without a thorough understanding of the software program and the connections between its components [15]. Investigating the system's evolution, or contrasting the changes between several software system versions over time, makes this challenge considerably more difficult. It is common knowledge that the most expensive activities in software creation are maintenance duties, particularly those brought on by changing software. The cost of software growth and maintenance might range from 40% to 80% of the total cost of development [16, 17].

In order to comprehend and illustrate the feature changes over the releases, software developers and testers frequently need to look into numerous software system versions. In order to create new test scenarios, for instance, software testers must concentrate their testing efforts on the new features and execution routes that were added in a new version of the software. The analysis and modeling of complex systems in many areas has been transformed by recent developments in information theory, such as call graph representation of software systems and network structure [18]. A call graph [19] is a visual representation that shows the structure of a software system with each node standing for a function and each directed edge also representing a function relationship

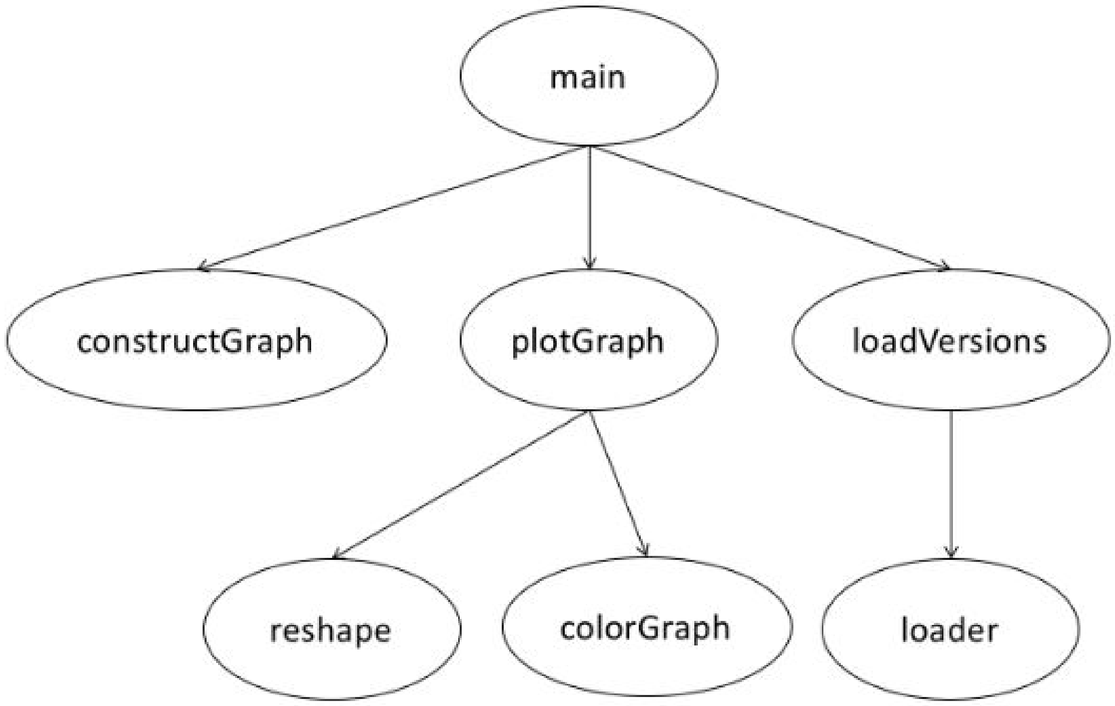


Figure 5: Code to Graph

Fig.5 illustrates the static call graph of GraphEvo Java library. The node $main$ represents an entry point to the system, which is the head of the execution path and is not called by any other function in the system. It is represented as a node without incoming edges in the call graph. The leaf nodes represents an exit points, and it does not call any other function in the system. It is represented as a node without outgoing edges in the call graph. An execution path is an ordered sequence of function calls from an entry point to an exit point; it does not visit the same function or function call twice. Note that the cycles within the call graphs can be eliminated or restricted to a specific number of iterations to reduce the length of the execution paths.

Static call graphs have been proven to assist software developers in understanding the overall system interactions and structure. For example, the tool in [20] can detect and visualize the static interactions between the classes of software systems written in Java. Code2graph [21] is a tool that automates the construction and visualization of static call graphs for programs written in Python. The work in [22] utilized machine learning algorithms to cluster the execution paths in large call graphs to facilitate system comprehension. The authors [23] of studied the call graphs of 223 releases of the Linux kernel to identify similar graph structures. Bhattacharya et al. [18] analyzed the releases of well-known projects to evaluate the use of call graphs in understanding software evolution. Similar to the existing methods and tools, our tool aims at constructing and visualizing call graphs. However, unlike existing tools, our tool uses *static call graphs to visualize software evolution in a monolithic color-coded graph and graph metrics to quantify software change*.

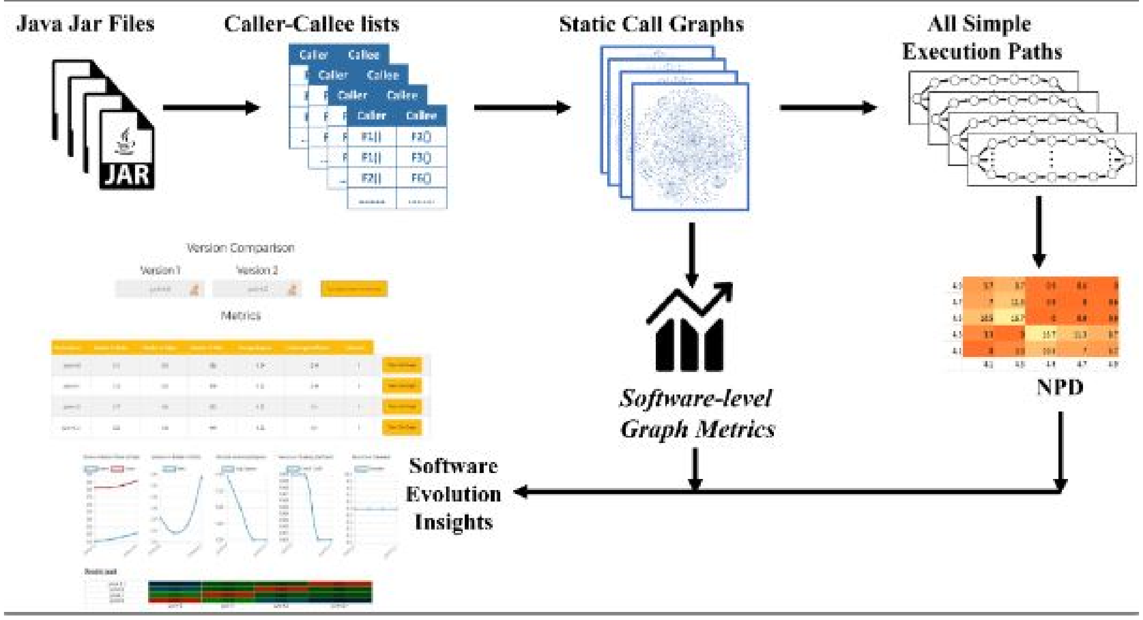

Java Jar Files
Caller-Callee lists
Static Call Graphs
All Simple Execution Paths
Software-level Graph Metrics
NPD
Software Evolution Insights
Version Comparison
Metrics

Figure 6: GraphEVO System Overview

2.4.1 GraphEVO Overview

The approach behind our tool is to use the graph theory of call graphs to facilitate and improve the characterization and visualization of software evolution. We introduce an open-source tool that can automatically visualize and characterize the software evolution for a given number of software releases with features to highlight the differences in execution paths to assist stakeholders in making better decisions. For example, assisting software testers in developing new test scenarios and answering a wide range of questions, such as *"What execution paths have been added that were not tested before?"*

The underlying approach of GraphEvo consists of two main steps. First, it constructs a call graph for each release of the software system under investigation. While the call graph can be used to understand the functionality and behavior of a single software system at a time, the goal here is to utilize the call graphs in characterizing the software evolution over time. Second, it calculates a set of graph metrics across several system releases to quantify evolution. In addition, the tool provides comparison and visual analysis features to help software testers better understand the code and execution-level changes, see Fig. 6.

2.5 Pull Request Quality and Characteristics

2.5.1 Software Quality of Pull Request

Research on pull requests has been done from several angles, including acceptance, reviewer assignment, and pull-based development. Latency is another issue with pull requests that have been looked into. Pull requests are accepted based on the size of the change, comments, perceived quality, and context. Integrators evaluate contributions according to quality, style, documentation, granularity, and compliance with project standards. Additionally, testing is vital to the decision to accept a pull request, favorably affecting the majority of authorized contributions (85%) [1]. Pull request rejection may rise if technical problems are not sufficiently fixed and the number of forks rises. Other grounds for rejection include lack of experience with pull requests, the complexity of contributions, the location of altered artifacts, and policy contribution. Investigated free static analysis software like PMD (Static Code Analyzer). Defect density was used as a proxy to assess how PMD rules affected the caliber of software. The defect density varied statistically significantly between the data with and without PMD. However, they did not remark on the quality of the tested code in accepted pull requests [24, 25].

Evaluated the pull request acceptance procedure for a commercial project [1] with an emphasis on developers' perceptions of review quality. They carefully examined the pull requests after data mining on the project's GitHub repository to comprehend the merge. They investigated many issues, such as the size of the pull request and the 108 people interested in the conversation. These studies looked at the software quality of pull requests according to the affiliation and experience of the developers. Our study advances this by quantifying code quality in pull requests submitted to well-known open-source projects, accounting for structural changes to the code and their impact on tests as quality criteria. These modifications are also apparent in the tool and other class-level metrics.

2.5.2 Call graph and Network portrait divergence for comparisons

The use of call graphs to represent the internal operations of a software system through function calls [26, 27] has been extensively studied. While other studies focus on collaborative software graphs, several existing studies concentrate on creating and analyzing call graphs for a single system. Code2Vec [28] studied the software components as software networks. It used Abstract Syntax Tree to extract all software pathways as vectors, aggregating them to create the Code2Vec neural model. It was evaluated using design quality criteria.

An innovative method for comparing networks based on their portraits is called portrait divergence. Network Portrait Divergence is based on information theory, in contrast to earlier ad hoc comparison measures, allowing for consistent interpretation of the divergence measure [29, 30]. As a result, rather than assuming that networks are defined on the same nodes, it may compare them based on their different topology structures. GraphEvo employed the metric to compare software versions and pinpoint pivotal moments in the development of programs.

2.5.3 Factors Affecting the Decision to Reject a Pull Request

We explaining here the main characteristics which are affecting the decision of a pull request. And they are mainly Project, Developer and Pull Request characteristics.

2.5.3.1 Project Characteristics

The studies on project characteristics typically discuss the fundamental details of target projects that are sent through pull requests, which may be distilled into the following elements: programming language, project adequacy as determined by the number of forks, stars, and viewers, and the age of the project the quantity of open pull requests is a measure of workload. The difference in time in seconds between the two most recent pull requests opening times is used to assess activity, while the number of open problems is used to indicate openness [31, 32]

2.5.3.2 Developer Characteristics

The contributor and integrator traits are connected to developer characteristics. This group of factors includes things that have to do with people and how two contributors or a contributor and a project interact. Basic information about developers, such as their gender, affiliation, and country of residence, is included in this category. While some studies examined the relationship between the developer and the target project, including the developer's experience, others focused on personal characteristics, including the personality and emotion of developers, which is defined as the number of prior pull requests, accepted commits, days since the creation of the account, whether this is the contributor's first pull request, the integrators prior reviews, the contributors coreness, the contributor's social distance and social strength from the integrator, and the integrators response time to the pull request.

2.5.3.3 Pull Request Characteristics

The related works concentrate on the essential details of pull requests, such as the size of the change measured at the file, commit, and code levels; the complexity of a pull request measured by the length of the description; the nature of pull requests measured by bug fixes; the inclusion of tests in pull requests; and the hotness or relevance of a PR. A few studies also concentrate on the process details of pull requests created during the code review process, such as the citation of a contributor, issue, or pull request; the conflict of a pull request; the complexity of discussion; the emotion in discussion; and the use of CI (Continuous Integration) tools during the review process.

CHAPTER 3

METHODOLOGY

3.1 Motivation

Many tools are available to display differences in source code and many more to compute code coverage, but very few solutions attempt to support the pull request review process. GitHub, SonerQube and Coveralls.io are few available online resources.

3.1.1 GitHub

GitHub is used by 83+ million developers, 4+ million organizations and has over 200+ million repositories.However, when developers submit a pull request, the review process typically takes a long time and demands a lot of work. Though GitHub offers a number of features to facilitate software development, it lacks a visual depiction of the program structure and analytics that might speed up the review process for pull requests[1].

3.1.2 SonarQube

A project's manual inspection can be supported by SonarQube, an open-source software. Multiple languages are supported by SonarQube, which can also examine code duplication, coverage, complexity, and other factors. Only the coverage feature and its interaction with SonarQube's source changes are worth exploring in this study. You can

[1]https://github.com/

see a demo site where you can see the many reports SonarQube generates using the web-based application. SonarQube does not view the files that have changed between versions as Coveralls.io does; instead, after selecting a file, it just displays the class's current coverage statistics. The modifications can be filtered in a variety of ways. It is possible to display the uncovered lines, uncovered branches, and lines that need to be covered. Although a timeline can be chosen, the default display is the file's present state. We can choose a time period of 90 days, 30 days, or since the previous analysis[2].

3.1.3 Coveralls.io

Coverall.io displays an overview with detailed coverage data. It displays the variance in coverage percentage and a view that displays the number of lines in each class, the number covered or missed, and the average number of hits on each line. Which classes were modified or had different coverage may be seen. The overview is cluttered because classes that only slightly alter the number of hits per line are also labeled as altered.We can see the current coverage status in a class's code. By manually comparing the source files and the changes in coverage, the previous version button must be clicked from the top of the page to see what truly changed.

Many developers would prefer not to utilize coveralls because manually verifying these changes is frequently problematic. The overview table on which the altered files are presented, along with occasionally shown files unrelated to the pull request or commit, is a feature of coveralls that can be utilized in a review process[3].

[2]https://www.sonarqube.org/
[3]https://coveralls.io/

3.2 Approach

The suggested effort in this thesis entails creating a tool called GraphEVOPR that will compile changes to test coverage and code changes into visualization and software metrics that will support the pull request review process. The software review process will be aided by this tool, which will shorten the review cycle duration and improve accuracy, see Fig. 7.

GraphEvoPR's objective is to use recent advancements in network comparison to discover structural changes that occurred to code within pull requests and to support the pull request review process. To assist decision-makers, we offer an open-source tool that computes metrics at the class level, quantifies structural changes, and identifies variations in execution paths. A software project's two versions can be compared using the tool. Both the code modifications and the changes in test coverage will be computed. Based on these adjustments, a report is produced that developers and reviewers can use to look through the changes in the new version.

Along with building such a tool, the tool will also be assessed by looking at reports based on actual alterations in open-source projects. To conduct the evaluation, GraphEVOPR has been enhanced to interact with Github and gather data on pull requests, which stand for project changes that need to be examined before approval. Following several intriguing report instances, a brief analysis of the effect of pull requests on test coverage is given. With this research, we hope to ascertain whether GraphEVOPR is a beneficial complement to the software review process.

Fig. 7 shows a high level process overview of pull request review process with

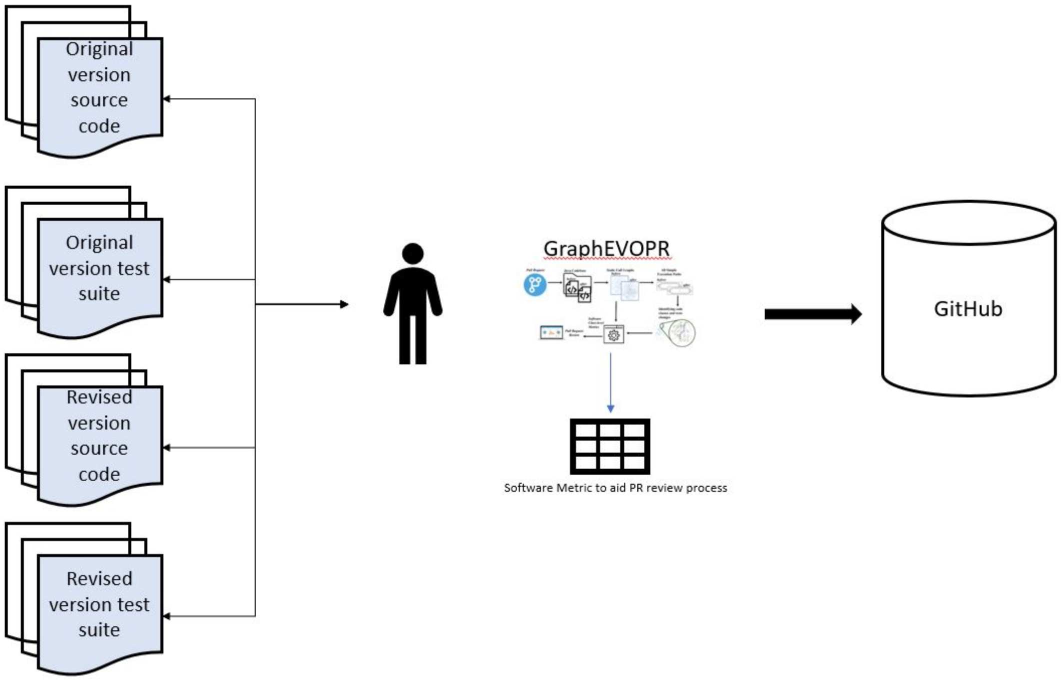

Original version source code
Original version test suite
Revised version source code
Revised version test suite
GraphEVOPR
Software Metric to aid PR review process
GitHub

Figure 7: Visualization of Process Flow with GraphEVOPR

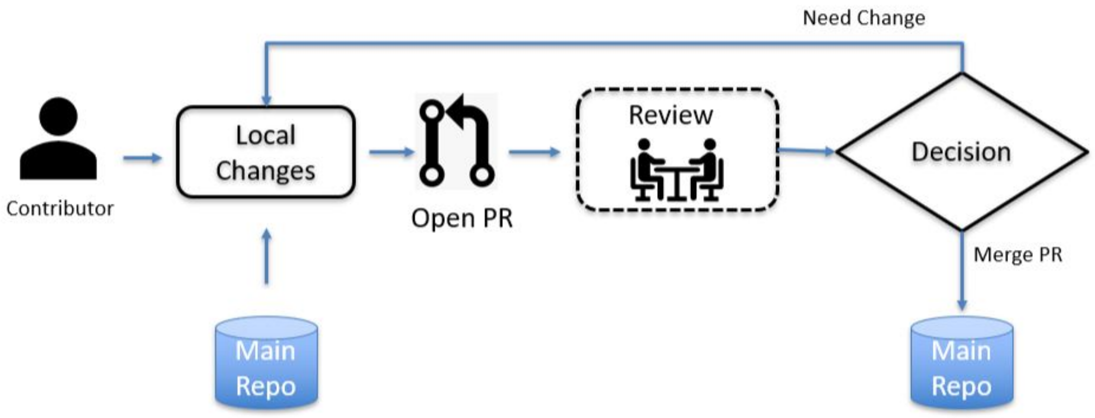


Figure 8: Typical Pull Request Review Process

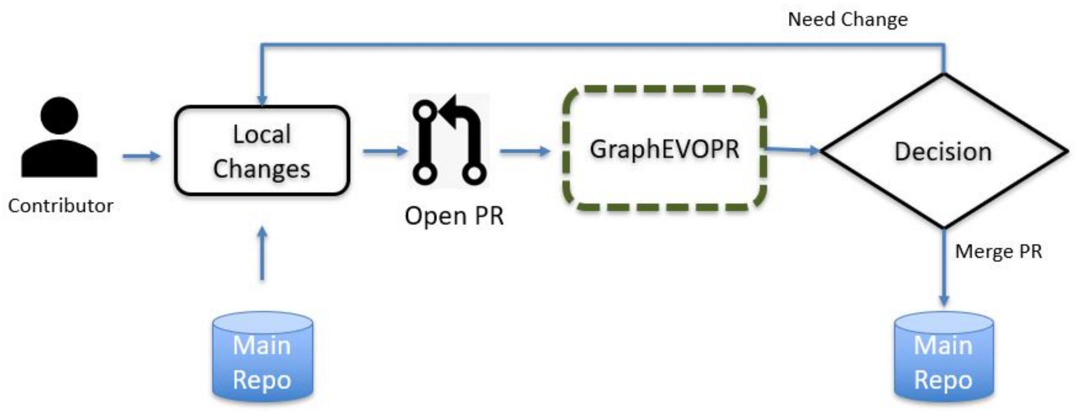


Figure 9: Pull Request Review Process with GraphEVOPR

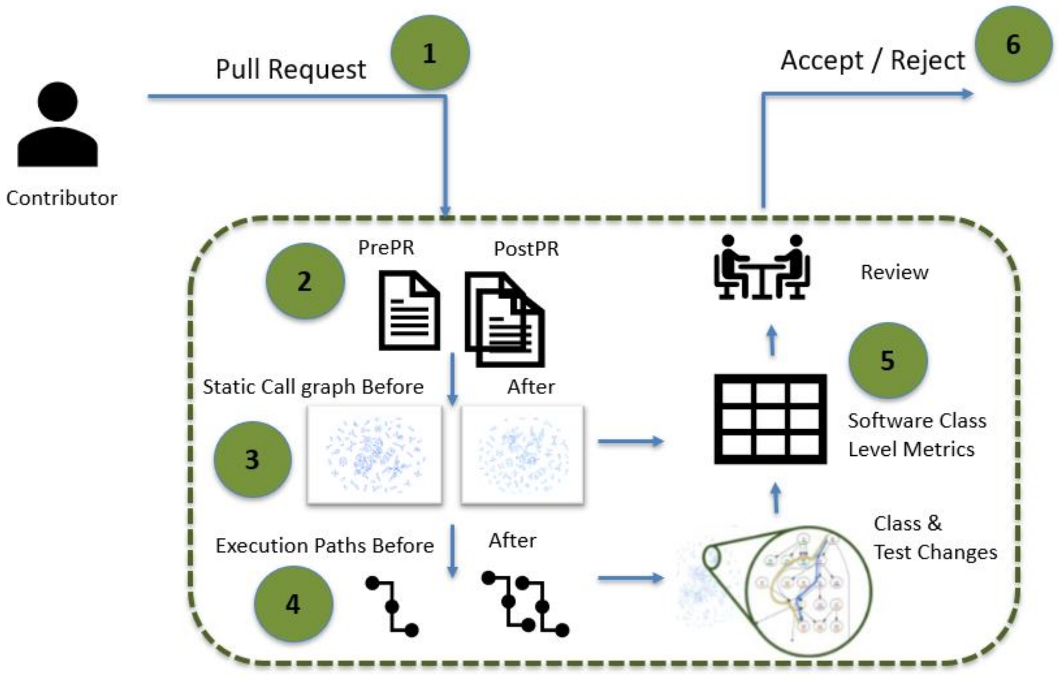


Figure 10: GraphEVOPR High Level Design

GraphEVOPR. It includes the details of code and test changes before and after, which are treating as prePR and postPR in our case. This changes will be pushed through GraphEVOPR for comparison and produce software metrics along with graph visualization. The output of this tool is used in review process of pull request and take a decision to accept or reject it.

Fig. 8 shows the typical pull request review process which involves manual review of the code and test changes in a pull request. In which developer creates a copy of central repository and add the changes to local copy. After that a pull request is raised and it will go through the review process. This process is time consuming and requires effort. Where as in Fig. 9 shows pictorial representation of GraphEVOPRs introduction in pull request review process. We mainly adding a mechanism to generate software metircs and visual represenation of the code. It is then used as part of pull request review process. The details steps invloved in GraphEVOPR are shown in Fig. 10. It has 6 different steps - pull request mining, code extraction, generation of call graphs, extraction of execution paths, generation of software metrics and last decision making.

Our approach starts with mining PRs from a list of open source Java-based repositories. We extract the source code before and after the PRs are merged. We pair prePR and postPR files, where each pair serves as an example of a meaningful change. Classes are then extracted, filtered, and organized in the dataset for each prePR and postPR version. We compute software metrics for each class, including test classes, and create a dataset. Finally, we analyze the dataset from a path changes perspective and perform a user story about the usability of this tooling.

3.3 Step 1 - Pull Request Mining

We built a graphEvoPR utility to collect the PR data needed to perform the path-based analysis. Given a list of Github projects, the utility extracts the list of PRs in respective repositories having “merged” as the final status project-wise. Then, for each project, the crawler retrieves the list of all PRs and then processes each merged PR P using the following steps. First, let us define the set of Java files submitted in P as

$$FS = F1, F2, ..., Fn \tag{3.1}$$

We ignore the PRs having only non Java files since our utility only supports Java. For each PR, we use the Github API to retrieve their version before the changes are implemented in the PR. The two (before/after) file sets might not be exactly the same due to files created/deleted during the review process. The output of the utility is, for each PR, the version of the code-base files before (prePR) and after (postPR, merged) the PR. At the end of the mining process, we obtain the datasets of code-base for all PRs.

3.4 Step 2 - Code Extraction

Each mined PR is represented as

$$PR = (f1, ..., fn), (ff1, ..., ffm) \tag{3.2}$$

Where f1,...,fn are the source code files before the PR, and ff1,...,ffm are code files after the PR.

As previously explained, the two sets may or may not be the same size since files could have been added or removed during the PR process. Next, the dataset is organized in pairs of each subsequent PRs {(pr1,pr2),(pr2,pr3)...,(prn-1,prn)}. Each pair of PRs (pri-1, pri) is again processed using the utility, which identifies AST operations performed between methods and then establishes class-level mapping for post-PR code-base within the pr. Then, software metrics are . for each class {(c1,....,ck),(c1,....,cl)} of pair (pri-1,pri) where post-PR code-base of pri-1 contains k classes while pri contains l classes.

3.5 Step 3 - Generation of Call Graphs

Extraction of the caller-callee relationships from the codebase is the first step in creating the call graph. The function calls for a certain Java codebase or Jar files were extracted using an open-source program that was used. The utility creates a list of caller and callee pairs for each function call made by the software system. We utilize this list to create a generic call graph by adding each caller-callee relationship as a directed edge, (u, v), to the graph, where the caller, u, and the callee, v, represent nodes in the graph and a call from u to v. At the level of function calls, this produces a valid call graph for the entire system. Every pull request goes through the same procedure. The next of our study uses the call graphs as an input to extract the execution paths. For the feature branch, GraphEVOPR creates a call graph that displays the changes both before and after. Even while the call graph may also be used to comprehend the functionality and behavior of a particular software system at a time, the objective in this instance is to utilize it to specify the pull request.

3.6 Step 4 - Extraction of Execution Paths

The same module or set of functions is not visited more than once in an execution path, which is an orderly arrangement of functional calls from an entry point to an exit point. GraphEVOPR (1) collects all basic execution pathways from each pull request; (2) builds a set of network portraits for each pull request based on their execution paths in order to determine the portrait divergence between the portraits of various releases of a software system.

3.7 Step 5 - Generation of Software Metrics

Information theory, on which Network Portrait Divergence is based, offers a trustworthy interpretation of the divergence measure. So, rather than assuming that they are defined on the same nodes, it may compare the networks based on their different topology structures. Moreover, Network Portrait Divergence is a graph invariant, making it more computationally efficient than the current expensive node matching optimization techniques. It should be listed that this method can handle both directed and undirected networks equally. In order to determine the Network Portrait Divergence between the PrePR and PostPR pictures -

- Using the call graphs for each pull request, extract all potential execution pathways.
- Create a set of network pictures depending on the execution pathways of each pull request.

- Utilizing the portraits of the investigated pull requests, calculate the portrait divergences.

3.8 Step 6 - Pull Request Review Decision

We made a few assumptions for this study and came up with the decision criteria. The pull requests are primarily divided into two classes: Accept and Reject. If the software structure modifications and the software code changes are in sync, which is shown by the software metric network portrait divergence together with LOC and AMC. We studied variety of with statement, expression, parameter, conditions, class definition and multiple level changes to decide either accept or reject the pull request.

Type of code changes -

- Statement change(s) - Such as variable declaration, method modifications etc.
- Condition change(s) - Such as if else, while , try catch blocks etc.
- Class files add or remove - This is addition or removal of entire class file
- Expression change(s) - This is change in an expression
- Class definition change(s) - This is entirely changing the definition of the class, like interface to abstract
- Multiple changes - Includes 2 or more changes of above type

Complexity - Using the Weka tool, we grouped the NPD values from the GraphEvo dataset. Simple EM was the clustering method employed (expectation maximization).

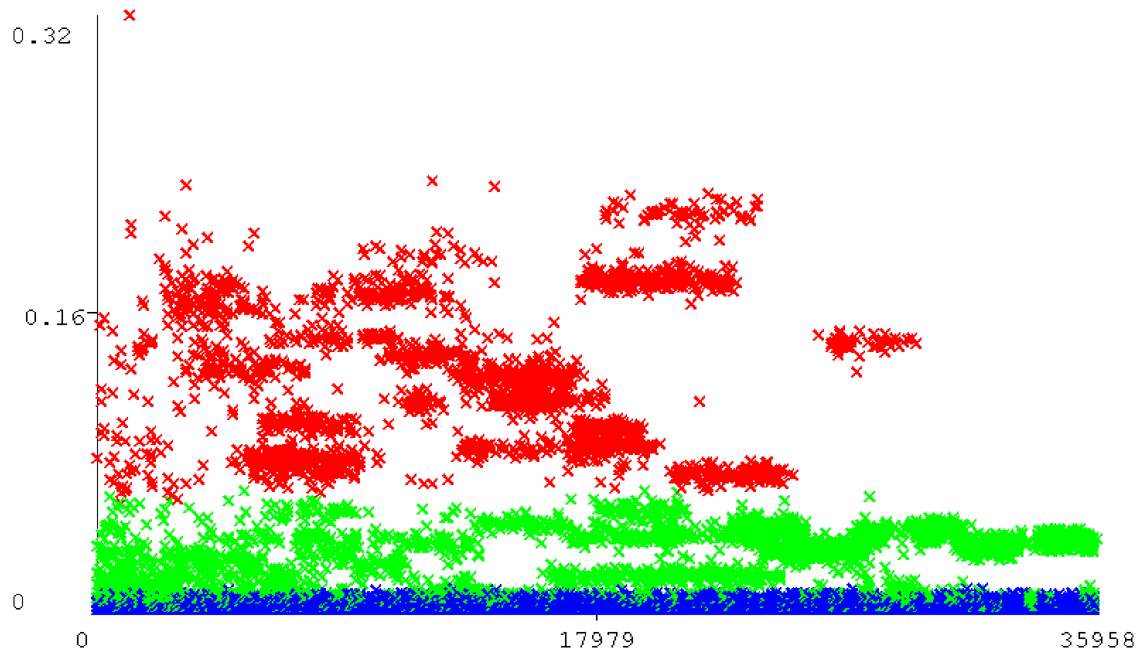


Figure 11: NPD Vs Complexity Clustering using Weka Tool

Each instance is given a probability distribution by EM, reflecting the likelihood that it belongs to each cluster. We can tell EM how many clusters to generate a priori, or it can select how many clusters to generate based on cross-validation. In the range of 0 to 0.02, 0.03 to 0.1, and more than 0.1, we found 3 clusters, as shown in Fig. 11.

Using such data, we reasoned that we could categorize the clusters as small, medium, and large. We also employ this clustering for individual Java projects and discovered that the NPD range varies depending on the software's size and domain, Fig. 12. We discovered that certain minor program displayed a variety of alterations. It is obvious that any modification will have a greater significance if the software has fewer classes and functions. The Java program under examination, which must change depending on project size, will start with this kind of range.

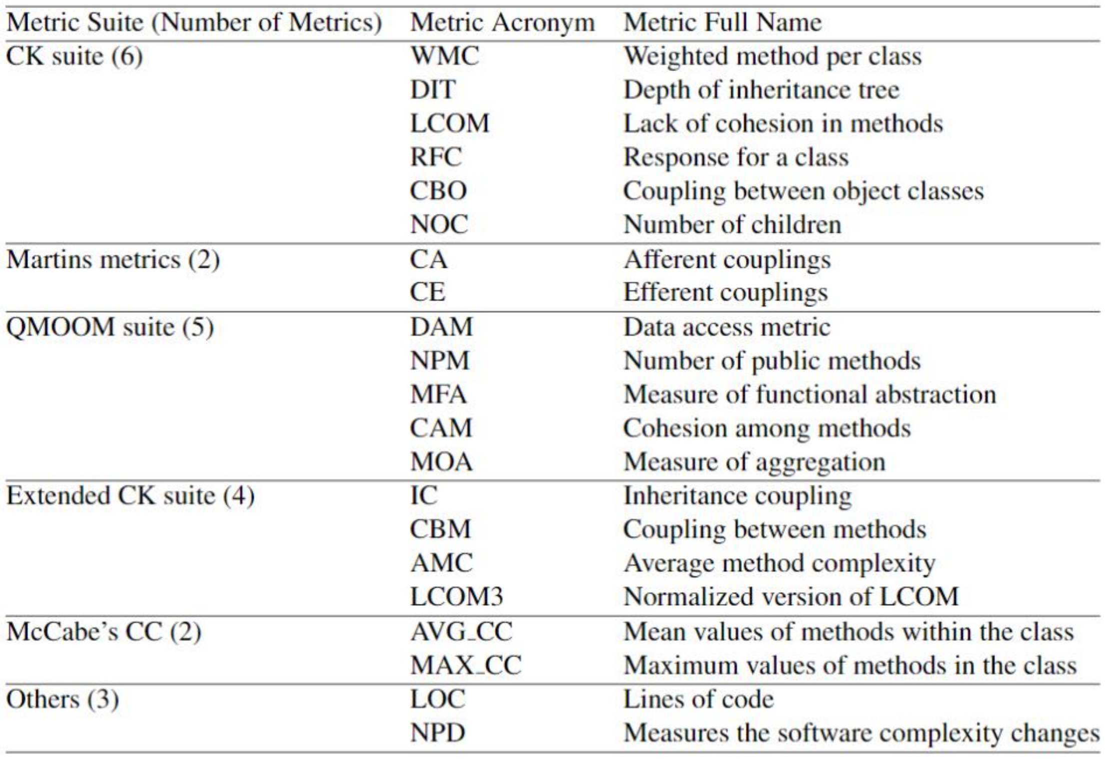
Table 2 below shows the level of complexity and the range of values for the respective software metrics network portrait divergence. To determine the pull request criteria, this was categorized.

Table 2: Code Complexity and Software Metric Classification

Complexity	**Network Portrait Divergence Range**	**Assumption**
Low	0.0000 - 0.0200	Small degree of change
Medium	0.0200 - 0.1000	Considerable degree of change
High	0.1000 - 0.3000	Significant degree of change
Very High	0.3000 - 1.0000	Severe change

Decisions -

- Accept - If software structure change depicts the degree of change and its in an acceptable range with respect to code and test
- Reject - If software structure change is not in an acceptable range with respect to code and test change



Metric Suite (Number of Metrics)	Metric Acronym	Metric Full Name
CK suite (6)	WMC	Weighted method per class
	DIT	Depth of inheritance tree
	LCOM	Lack of cohesion in methods
	RFC	Response for a class
	CBO	Coupling between object classes
	NOC	Number of children
Martins metrics (2)	CA	Afferent couplings
	CE	Efferent couplings
QMOOM suite (5)	DAM	Data access metric
	NPM	Number of public methods
	MFA	Measure of functional abstraction
	CAM	Cohesion among methods
	MOA	Measure of aggregation
Extended CK suite (4)	IC	Inheritance coupling
	CBM	Coupling between methods
	AMC	Average method complexity
	LCOM3	Normalized version of LCOM
McCabe's CC (2)	AVG_CC	Mean values of methods within the class
	MAX_CC	Maximum values of methods in the class
Others (3)	LOC	Lines of code
	NPD	Measures the software complexity changes

Figure 12: Static Code Metric

CHAPTER 4

RESULTS AND EVALUATIONS

4.1 Overview

To determine whether or not the GraphEVOPR tool is valuable, we examine pull requests on GitHub. First, some intriguing pull requests are chosen, and the GraphEVOPR tool's visual capabilities are examined to see whether they can provide relevant data. The next step is to gather analytics on pull requests using GraphEVOPR to ascertain the potential utility of the tool. How often pull requests, for instance, do we see freshly added code and tests that aren't covered? In this chapter we will discuss on dataset, configuration/setup, software and hardware details , experimental design and user study survey.

4.2 Dataset

Table 3 shows the total number of pull request from each project that we mined. We collected data from variety of software project hosted on GitHub, see Tables 4 and Table 5 shows the description of the open source projects that we shortlisted.

4.2.1 Selected Projects

There are 14 projects chosen for this review. The choices of projects are limited by GraphEVOPR's capabilities. GraphEVOPR currently only supports Java-based applications with Maven-based test suites. The chosen projects are Defect4j-repair (Math, Chart, Closure, Lang, Mockito, and Time), any23, dagger, grafika, incubator-shenyu, MOE, and rejoiner, and they are all accessible on Github. For these projects, the GraphEVOPR receives the pull requests for analysis. Since not every pull request was approved and some branches or forked repositories no longer exist, not every pull request could be investigated. It is impossible to decide which versions should be compared because of deleted branches.

There has been a big code change, to be exact. fix the test because it directly affects the code structure, demonstrating that the link for modified classes are rewired with related courses. Divergence in Network Portraits has resonance with the modifications made to the fixed broken test. Additionally, RFC and LOC denote suitable metrics changes.

4.2.2 Defect4j-repair

This repository contains the unprocessed experimental data from tests on the automatic correction of Defects4J dataset issues conducted at INRIA Lille. In 2016, Matthias Martinez, Thomas Durieux, Romain Sommerard, Jifeng Xuan, and Martin Monperrus published Automatic Repair of Real Bugs in Java: A Large-Scale Experiment on the Defects4J Dataset. Defects4J is a huge, systematic dataset of actual Java defects that have undergone peer review. Each flaw in Defects4J [1] has a test suite and at least one flaw-triggering test case that fails, see Table 4.

[1]http://program-repair.org/defects4j-dissection/

4.2.3 Projects and Pull requests

Here are the total number of pull request which we have planned to evaluate from 14 open source java projects, see Table 3.

Table 3: Projects and Pull Request

Project	**Number of Pull Requests**
android-classyshark	81
any23	120
caliper	99
curator	4
dagger	62
grafika	3
incubator-shenyu	28
lang	65
math	10
moe	9
open-location-code	32
rejoiner	42
xquery	42
walt	49
Total	**627**

4.2.4 Selected Pull Requests

Our initial strategy entails gathering pull requests that demonstrate intriguing scenarios. The objective of this evaluation is to assess GraphEVOPR's utility from the perspective of the code reviewer. Pull requests from the projects are carefully examined, and interesting situations are chosen to ascertain their usefulness. To demonstrate how GraphEVOPR is helpful in the review process of these pull requests, we looked for pull requests that demonstrate good examples of testing and a handful that demonstrate less

Table 4: Selected Projects: Project and Description Part I

Project	**Description**
Defect4j-repair:Chart	This module provides all project-specific configurations and subroutine for the JFreeChart project
Defect4j-repair:Closure	This module provides all project-specific configurations and subroutines for the Closure compiler project
Defect4j-repair:Lang	This module provides all project-specific configurations and subroutines for the Commons-lang project
Defect4j-repair:Math	This module provides all project-specific configurations and subroutines for the Commons-math project
Defect4j-repair:Mackito	This module provides all project-specific configurations and methods for the mockito project
Defect4j-repair:Time	Time - This module provides all project-specific configurations and subroutines for the Joda-Time project
any23	Anything To Triples (any23) is a library, a web service, and a command-line tool that extracts structured data in RDF format from various Web documents
dagger	A fast dependency injector for Java and Android. Dagger is a compile-time framework for dependency injection. It uses no reflection or runtime bytecode generation, does all its analysis at compile time, and generates plain Java source code
grafika	Grafika is: A collection of hacks exercising graphics features. An SDK app developed for API 18 (Android 4.3). While some of the code may work with older versions of Android, some sporadic work is done to support them. Open-source (Apache 2 license), copyright by Google. So you can use the code according to the terms of the license (see ”LICENSE”). A perpetual work-in-progress. It’s updated whenever the need arises
incubator-shenyu	Apache ShenYu is a native Java API Gateway for service proxy, protocol conversion, and API governance. Scalable, High Performance, Responsive API Gateway Solution for all MicroServices
MOE	MOE (Metric Optimization Engine) is an efficient way to optimize a system’s parameters when evaluating parameters is time-consuming or expensive. A global, black-box optimization engine for real-world metric optimization
rejoiner	Generates a unified GraphQL schema from gRPC microservices and other Protobuf sources
xquery	Extract data or evaluate value from HTML/XML documents using XPath
walt	A device for measuring the latency of physical sensors and outputs on phones and computers

Table 5: Selected Projects: Project and Description Part II

Project	Description
google-java-format	Reformats Java source code to comply with Google Java Style
caliper	Caliper is an instrumentation and performance profiling library
OLC	OLC(Open Location Code) is a library to generate shortcodes, called ”plus codes”, that can be used as digital addresses where street addresses don’t exist
android-classyshark	Android and Java bytecode viewer
curator	Apache Curator includes a high-level API framework and utilities to make using Apache ZooKeeper much more reliable. It also includes recipes for common use cases and extensions such as service discovery and a Java 8 asynchronous DSL

tested components.

4.3 Configuration/Setup

To explore a range of pull requests, we chose 14 open-source Java projects of various sizes. In the beginning, we used a method that involved creating the jar files for prePR and postPR in order to compare and produce call graphs and software metrics. Later, when we realized how time-consuming this method was, we switched to using GitAPI to pull the source code directly from GitHub.

- **Method 1 -** Use of jar files to compare the version of pull request before and after fix. This worked well with smaller projects.
- **Method 2 -** Use of GitAPI to pull the source code directly from the GitHub to compare the code changes and generate software metrics. This method worked

well for projects with bigger size.

4.4 Software and Hardware

- **Editor:Intellij -** We used Intellij[2] to analyze the open-source java projects and pull request.
- **Editor:PyCharm -** We used pycharm[3] to enhance the GraphEVO and converted it to GraphEVOPR for this study
- **GraphEVOPR -** Python and Java de-compiler libraries
- **Data retrival from server -** Using JavaScript
- **Graph Visualization -** Using HTML and CSS

4.5 Implementation

Three steps made up the implementation tasks. Initially, we enhanced and integrated GraphEvo, an already-existing tool that extracts a list of the caller-callee functions for a particular Java project. Second, the GraphEvo tool uses the caller-callee relationships in two graph data structures to compute class-level metrics and net-work portrait divergence. Third, we may view class-level metrics, spot structural software changes, and display the pull request as a color-coded call graph.

A web-based user interface is used to operate GraphEvoPR locally in its entirety.

[2]https://www.jetbrains.com/idea/
[3]https://www.jetbrains.com/pycharm/

We used HTML and CSS for the graphics, and Javascript was used for client-side requests to get data from the server. We used Flask, a well-known open-source micro web framework, to construct the backend server. Using various open-source tools, the tool also computes 21 class-level metrics, such as network portrait divergence. Currently, the number of failed tests is manually added to the system; however, in future development, we intend to automate this process.

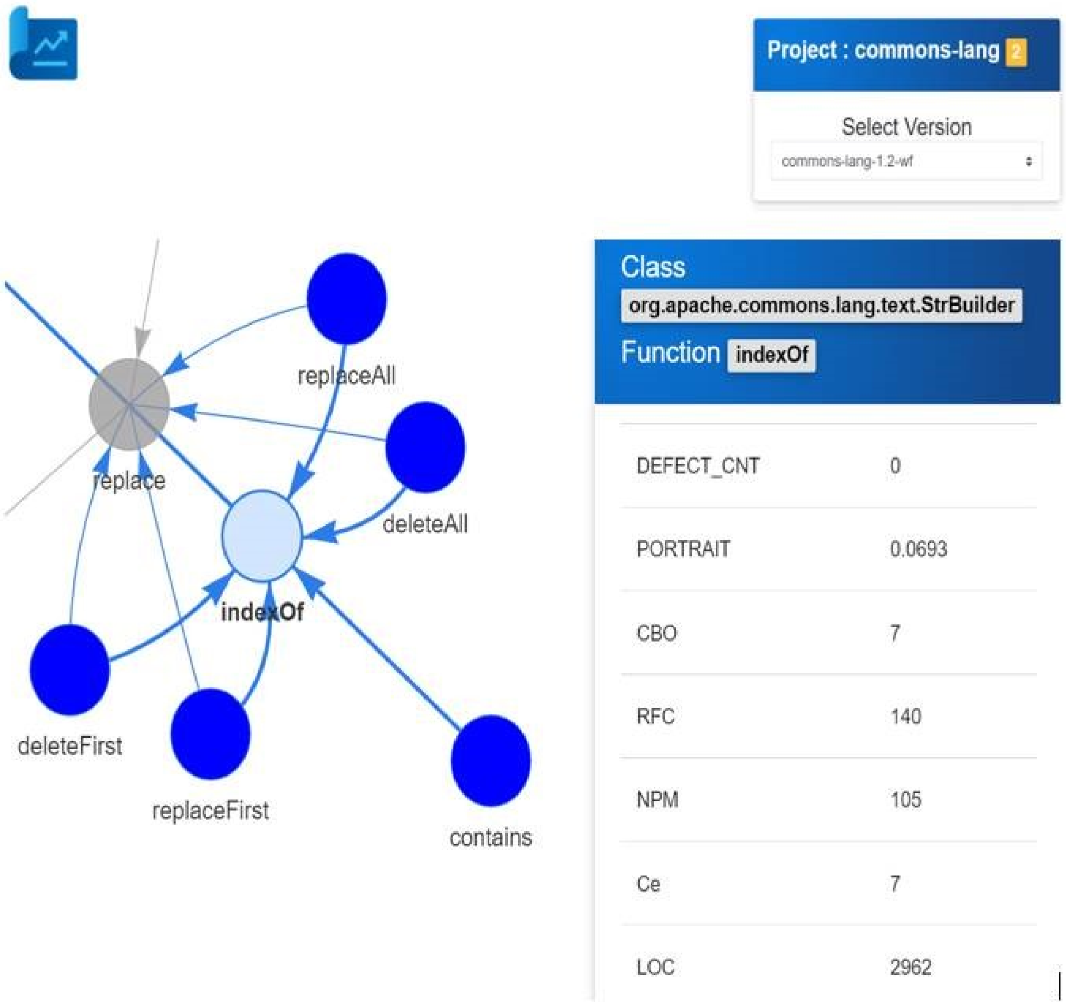
Fig. 13 shows the landing page for GraphEvoPR. Users can initially set up the project by entering the project name, Github URL, default master branch, and authorization key. The user can then choose the project and input the URL for the pull request.

4.6 Experimental Design

Out of 627 pull request that we mined, we are discussing 5 different variety of pull request in details in this section. Table 6 below shows the experiment with type of change and degree of change we are going to discuss further.

Table 6: Pull Request Experiment Details

Experiment	**Type of change**	**Degree of change**
1	Statement(assignment addition)	Low
2	Statement (assignment expression modification)	Low
3	Conditional and method call addition	Medium
4	Statement(assignment addition)	Low
5	Statement(assignment addition)	Medium



Project : commons-lang 2

Select Version

commons-lang-1.2-wf

Class org.apache.commons.lang.text.StrBuilder

Function indexOf

DEFECT_CNT	0
PORTRAIT	0.0693
CBO	7
RFC	140
NPM	105
Ce	7
LOC	2962

Figure 13: GraphEVOPR Tool

4.6.1 **Experiment 1 -** Defect4j-repair:Branch-Math 10, Class-DSCompiler - **Accept**

One method update in the class DSCompiler adds two repair actions and patterns to correct the flawed test, see Fig. 14. The modifications show that they affect 43 execution routes, both directly and indirectly, see Fig. 15. The methods of execution that utilize or call this class. The right modifications are reflected in every metric. Specifically, the shift in values for Network Portrait Divergence from 0 to 0.0203 illustrates the change in the code's structure and how it captures the impacted execution routes, see Table 7. The one test also fails, which is usual for such a slight structural change.

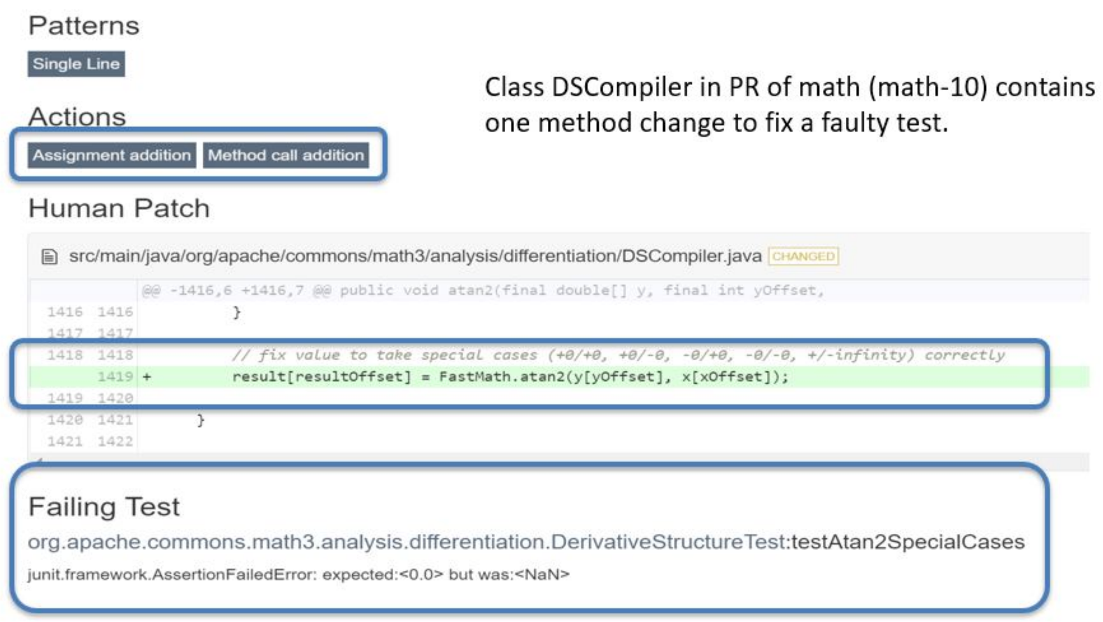
Table 7: Software Metric Change for DSCompiler

Description	**Before Fix**	**After Fix**
Version	1.1	1.2
Total Test	4625	4625
Failed Test	1	0
AMC	74.0265	74.2708
PORTRAIT	0.0000	0.0203

If we open the class and check, their is the new addition of code to fix the failing test. With the GraphEVOPR tool, we can easily identify that before fixing the test, the software metric values, and after the fix. Specifically, the network portrait divergence value is changed, which signifies a code change.

4.6.2 **Experiment 2 -** Defect4j-repair:Branch-Lang 61, Class-StrBuilder **Accept**

Compared to all execution pathways, class StrBuilder Fig. 16, had a 14.66 percent influence on the execution path (absolute execution paths for before the change after the change are 709, respectively, and StrBuilder class appeared in 104 execution paths for



Patterns

Single Line

Actions

Assignment addition | Method call addition

Class DSCompiler in PR of math (math-10) contains one method change to fix a faulty test.

Human Patch

src/main/java/org/apache/commons/math3/analysis/differentiation/DSCompiler.java CHANGED

```
@@ -1416,6 +1416,7 @@ public void atan2(final double[] y, final int yOffset,
            }

            // fix value to take special cases (+0/+0, +0/-0, -0/+0, -0/-0, +/-infinity) correctly
+           result[resultOffset] = FastMath.atan2(y[yOffset], x[xOffset]);

        }

```

Failing Test

org.apache.commons.math3.analysis.differentiation.DerivativeStructureTest:testAtan2SpecialCases

```
junit.framework.AssertionFailedError: expected:<0.0> but was:<NaN>
```

Figure 14: Defect4j-repair Math10 DCCompiler Class

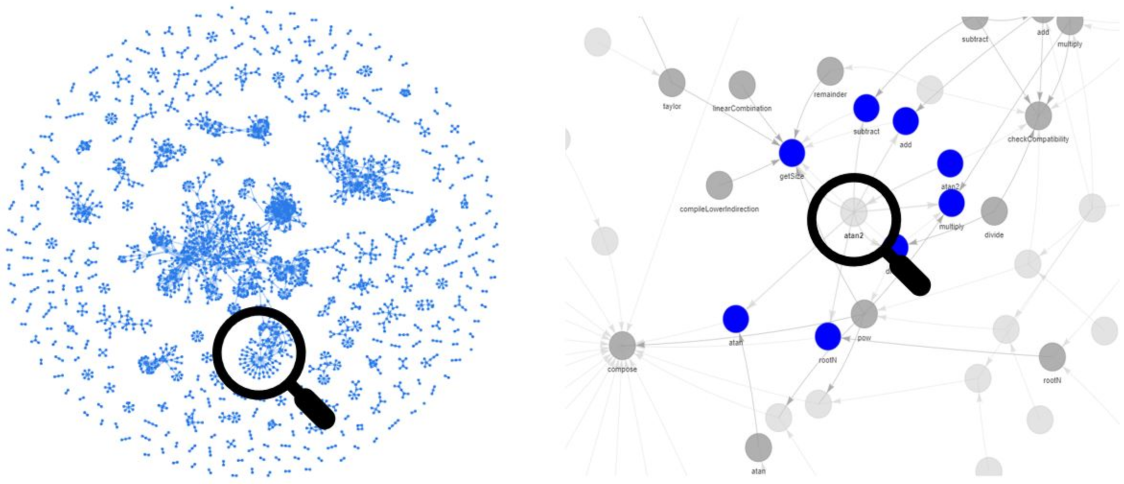
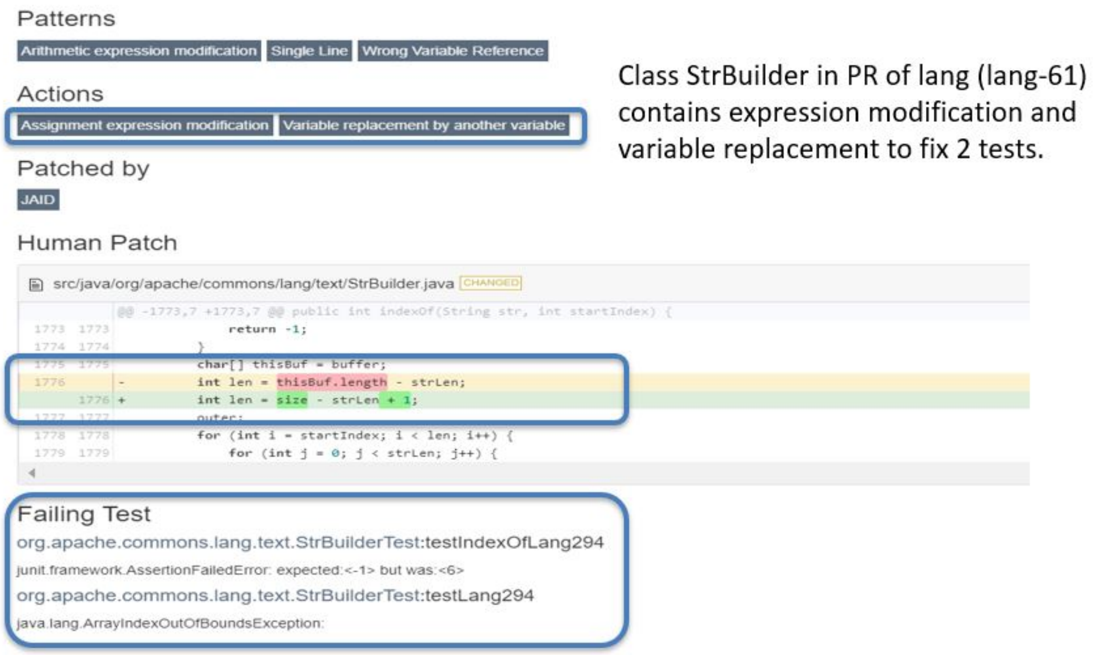


Figure 15: Defect4j-repair Math10 DCCompiler Class after Changes



Patterns

Arithmetic expression modification | Single Line | Wrong Variable Reference

Actions

Assignment expression modification | Variable replacement by another variable

Patched by

JAID

Human Patch

src/java/org/apache/commons/lang/text/StrBuilder.java CHANGED

```
@@ -1773,7 +1773,7 @@ public int indexOf(String str, int startIndex) {
             return -1;
         }
         char[] thisBuf = buffer;
-        int len = thisBuf.length - strLen;
+        int len = size - strLen + 1;
         outer:
         for (int i = startIndex; i < len; i++) {
             for (int j = 0; j < strLen; j++) {
```

Failing Test

org.apache.commons.lang.text.StrBuilderTest:testIndexOfLang294

```
junit.framework.AssertionFailedError: expected:<-1> but was:<6>
```

org.apache.commons.lang.text.StrBuilderTest:testLang294

```
java.lang.ArrayIndexOutOfBoundsException:
```

Class StrBuilder in PR of lang (lang-61) contains expression modification and variable replacement to fix 2 tests.

Figure 16: Defect4j-repair Lang61 StrBuilder Class

each version). This suggests that it significantly affects the quality and coverage of the code. The Network Portrait Divergence accurately detected these changes along with RFC and LOC. Additionally, we observe that both tests fail, indicating that DSCompiler has undergone a more significant modification, see Fig. 17 and Table 8.

Table 8: Software Metric Change for StrBuilder

Description	Before Fix	After Fix
Version	1.1	1.2
Total Test	4985	4985
Failed Test	2	0
AMC	25.8727	25.8909
PORTRAIT	0.0000	0.0693

The impact of the code modification on related classes and associated tests is shown in the Fig. 18. The second test was impacted and failed before installing the

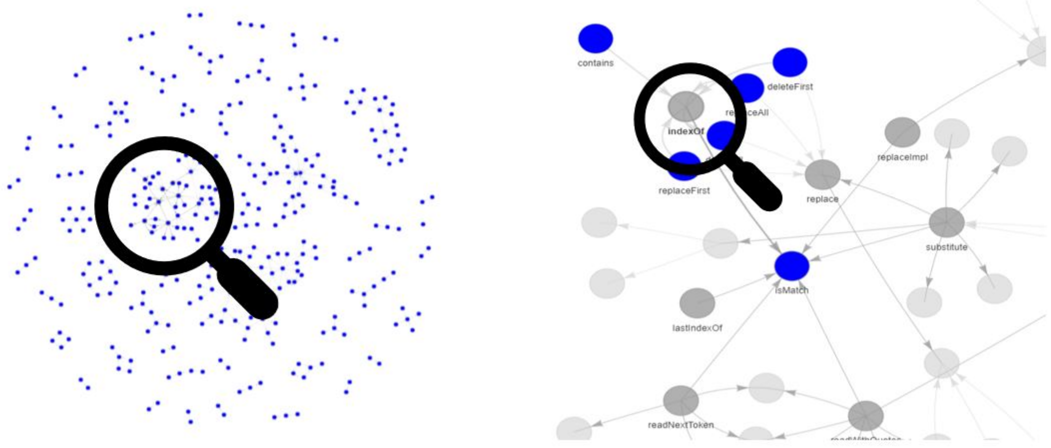
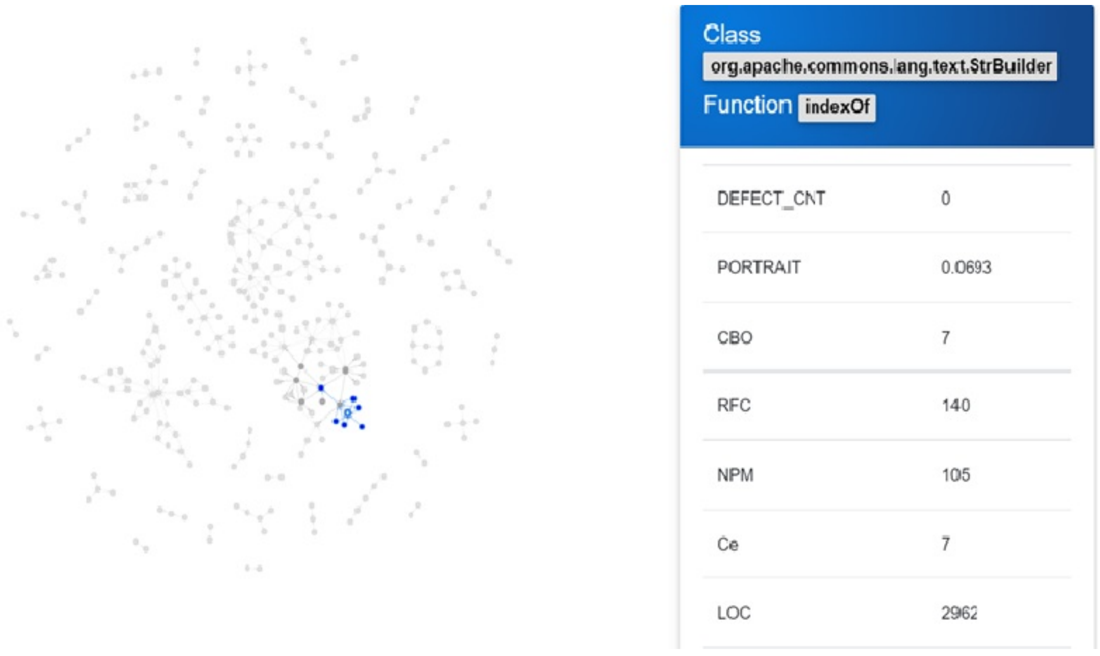


Figure 17: Defect4j-repair Lang61 StrBuilder Class after Change

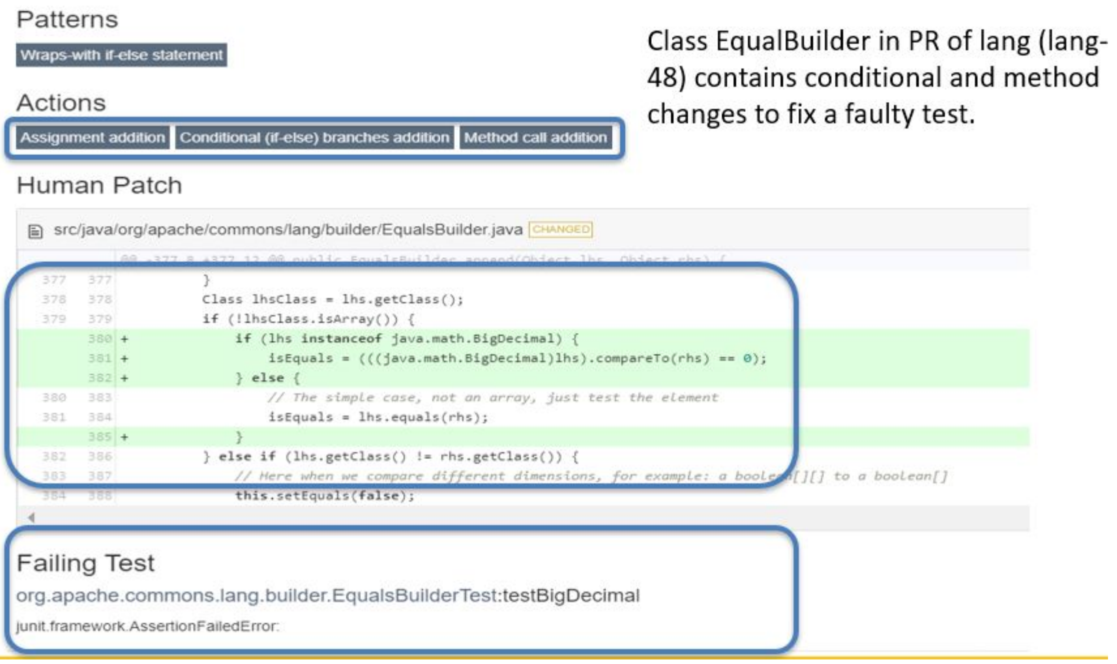


Class org.apache.commons.lang.text.StrBuilder

Function indexOf

DEFECT_CNT	0
PORTRAIT	0.0693
CBO	7
RFC	140
NPM	105
Ce	7
LOC	2962

Figure 18: Defect4j-repair Lang61 StrBuilder Class GraphEVOPR



Class EqualBuilder in PR of lang (lang-48) contains conditional and method changes to fix a faulty test.

src/java/org/apache/commons/lang/builder/EqualsBuilder.java

```
@@ -377,8 +377,12 @@ public EqualsBuilder append(Object lhs, Object rhs) {
            }
            Class lhsClass = lhs.getClass();
            if (!lhsClass.isArray()) {
+               if (lhs instanceof java.math.BigDecimal) {
+                   isEquals = (((java.math.BigDecimal)lhs).compareTo(rhs) == 0);
+               } else {
                    // The simple case, not an array, just test the element
                    isEquals = lhs.equals(rhs);
+               }
            } else if (lhs.getClass() != rhs.getClass()) {
                // Here when we compare different dimensions, for example: a boolean[][] to a boolean[]
                this.setEquals(false);
```

org.apache.commons.lang.builder.EqualsBuilderTest:testBigDecimal

junit.framework.AssertionFailedError:

Figure 19: Defect4j-repair Lang48 EqualBuilder Class

update. Both of these tests pass following the fix. The amount of change is shown by the portrait value of 0.0693. It was accessing the incorrect variable and needed the arithmetic expression to be modified.

4.6.3 **Experiment -3** Defect4j-repair:Branch-Lang 48, Class-EqualBuilder **Accept**

Entities, FastDateFormat, and Class EqualsBuilder Fig. 19,are notably large classes with various execution paths. There were a large number of affected execution pathways. These classes' Network Portrait Divergence values are within a narrow range, indicating that the code changes in these classes had a similar effect on the execution paths. The one test that failed also shows a slight structural alteration, see Fig. 20 and Fig. 21.

The impact of the code modification on related classes and associated tests is

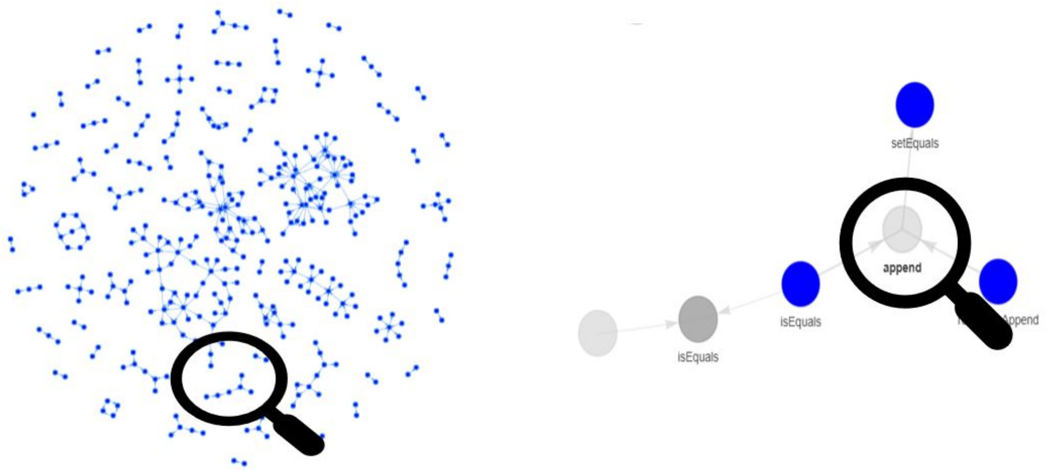
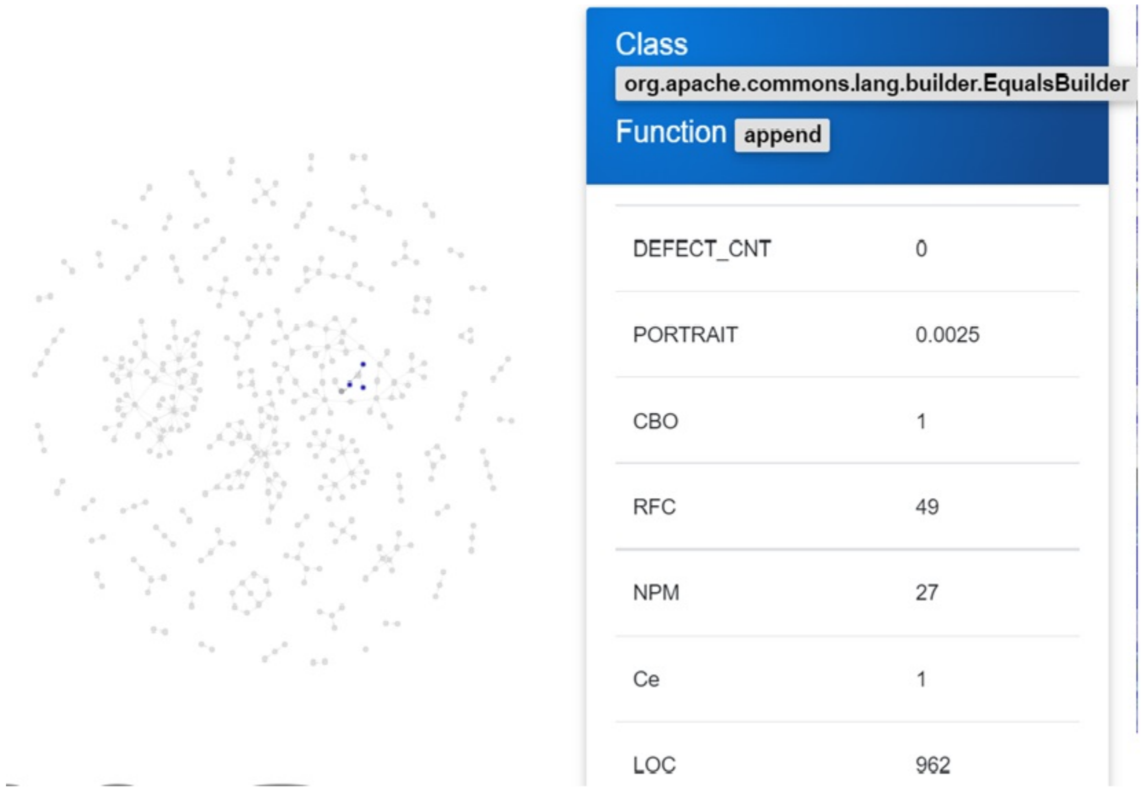


Figure 20: Defect4j-repair Lang48 EqualBuilder Class after Change



Class org.apache.commons.lang.builder.EqualsBuilder

Function append

DEFECT_CNT	0
PORTRAIT	0.0025
CBO	1
RFC	49
NPM	27
Ce	1
LOC	962

Figure 21: Defect4j-repair Lang48 EqualBuilder Class GraphEVOPR

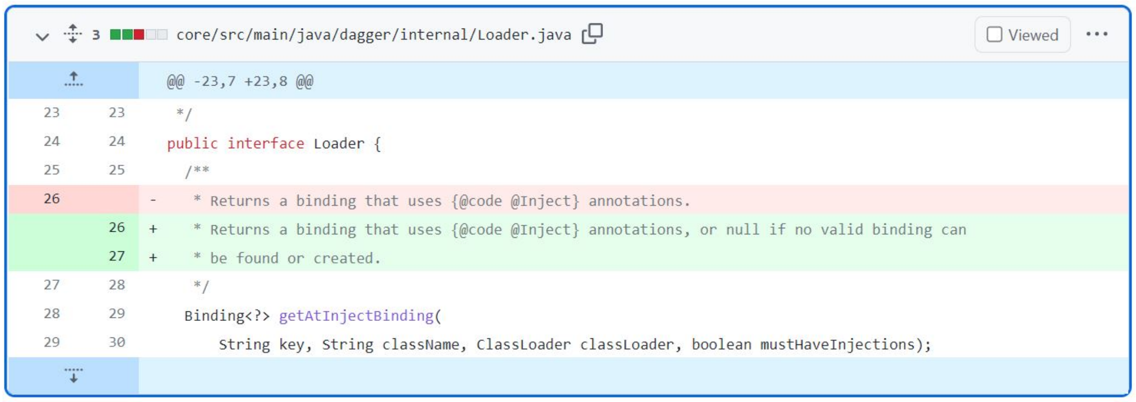
shown in the figure. The test “testBigDecimal” was impacted because of this change. The amount of change is shown by the portrait value of 0.0025, which is very minimal in this case. The fix of conditional branch addition has been added here. The minimal value of portrait divergence can be considered safe to accept the pull request.

4.6.4 **Experiment 4** - Pull Request number 7 and 8 from dagger **Reject**

The class is declared as an interface in pull request number 7, as shown in Fig. 22, and no related test cases are provided to verify the coverage. In contrast, the class declaration in pull request number 8 is changed from an interface to an abstract class, see Fig. 23. With this modification, three new test classes have been introduced for test coverage.

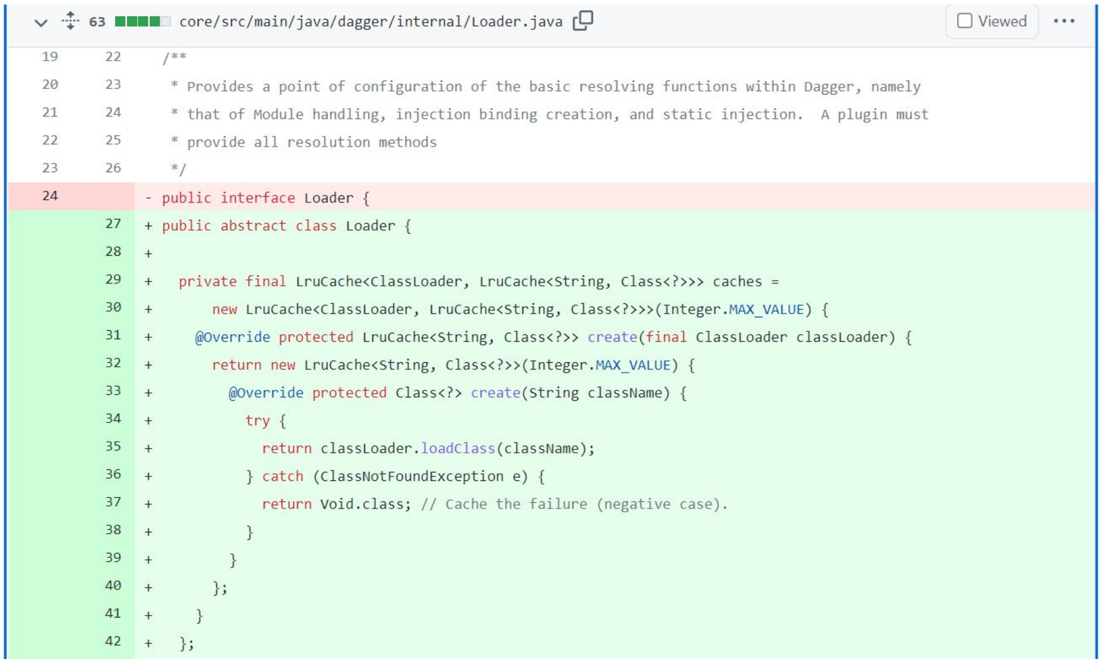
Here, the code change, its influence, and the test coverage can be identified using the GraphEVOPR tool. When pull request 7 is run via the tool with this being the first version, the network portrait divergence result is 0.0005 compared to the code change in pull request 8. For pull request 8, the network portrait value has been slightly adjusted to 0.0004. This shows that the influence of a code change on the software architecture is relatively minimal, only one class path from the call graph flow. However there is significant code changes in this pull request. The degree of change is not in alignment with code changes. So, we are inclined toward rejecting this change.

We also observed that other software metrics are changing along with network portrait, like LOC, modifier, cobModified, fanin, fanout, totalMethodQty, abstractMethodsQty, etc, see Table 9.



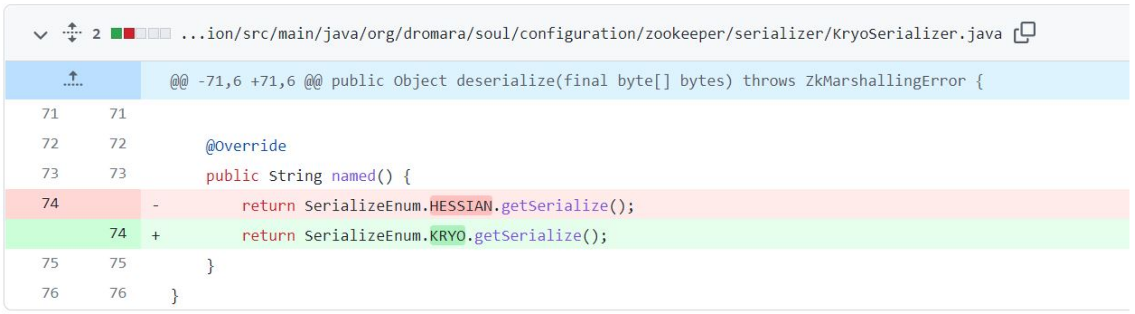
```
core/src/main/java/dagger/internal/Loader.java
@@ -23,7 +23,8 @@
   */
  public interface Loader {
    /**
-    * Returns a binding that uses {@code @Inject} annotations.
+    * Returns a binding that uses {@code @Inject} annotations, or null if no valid binding can
+    * be found or created.
     */
    Binding<?> getAtInjectBinding(
        String key, String className, ClassLoader classLoader, boolean mustHaveInjections);
```

Figure 22: Dagger Pull Request Number 7



```
core/src/main/java/dagger/internal/Loader.java
  /**
   * Provides a point of configuration of the basic resolving functions within Dagger, namely
   * that of Module handling, injection binding creation, and static injection.  A plugin must
   * provide all resolution methods
   */
- public interface Loader {
+ public abstract class Loader {
+
+   private final LruCache<ClassLoader, LruCache<String, Class<?>>> caches =
+       new LruCache<ClassLoader, LruCache<String, Class<?>>>(Integer.MAX_VALUE) {
+     @Override protected LruCache<String, Class<?>> create(final ClassLoader classLoader) {
+       return new LruCache<String, Class<?>>(Integer.MAX_VALUE) {
+         @Override protected Class<?> create(String className) {
+           try {
+             return classLoader.loadClass(className);
+           } catch (ClassNotFoundException e) {
+             return Void.class; // Cache the failure (negative case).
+           }
+         }
+       };
+     }
+   };
```

Figure 23: Dagger Pull Request Number 8



```
...ion/src/main/java/org/dromara/soul/configuration/zookeeper/serializer/KryoSerializer.java
@@ -71,6 +71,6 @@ public Object deserialize(final byte[] bytes) throws ZkMarshallingError {

    @Override
    public String named() {
-       return SerializeEnum.HESSIAN.getSerialize();
+       return SerializeEnum.KRYO.getSerialize();
    }
}
```

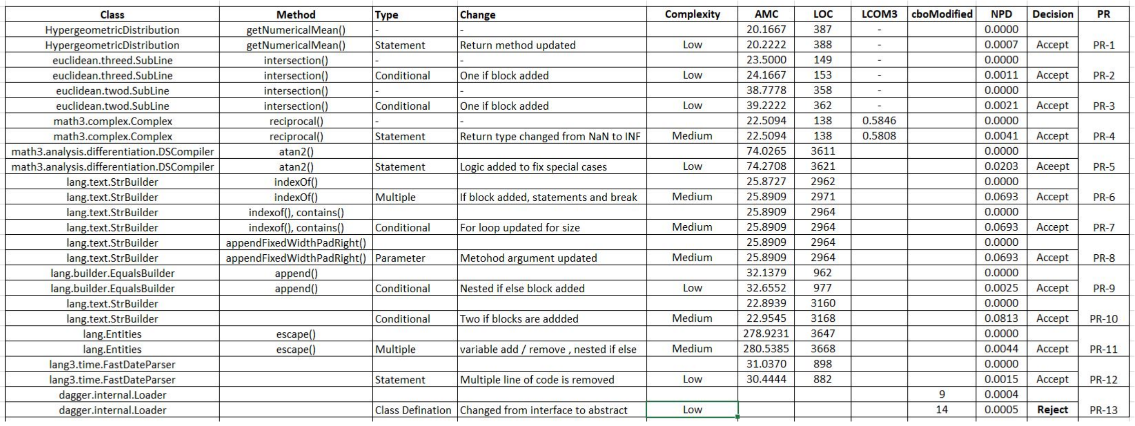
Figure 24: Incubator-shenyu Pull Request Number 17

4.6.5 **Experiment-5** Pull Requests number 16 and 17 from incubator-shenyu **Accept**

There is one modified line of code in pull request number 17, see Fig. 24. In essence, references to methods are switched from one to the other. In this instance, the portrait value changed from 0 to 0.0913, see Table 10. However, other software metric values remained unchanged. Additionally, no relevant tests have been modified, added, or withdrawn. Furthermore, this adjustment does not influence any pathways. We can conclude from this data that it is safe to accept this pull request.

Table 9: Software Metric for Dagger Pull Requests 7 and 8

Description	**PR7**	**PR8**
Portrait	0.0005	0.0004
Modifiers	1	1025
LOC	5	48
cobModified	9	14
fanin	5	8
fanout	4	6
abstractMethodsQty	4	6



Class	Method	Type	Change	Complexity	AMC	LOC	LCOM3	cboModified	NPD	Decision	PR
HypergeometricDistribution	getNumericalMean()	-	-		20.1667	387	-		0.0000		
HypergeometricDistribution	getNumericalMean()	Statement	Return method updated	Low	20.2222	388	-		0.0007	Accept	PR-1
euclidean.threed.SubLine	intersection()	-	-		23.5000	149	-		0.0000		
euclidean.threed.SubLine	intersection()	Conditional	One if block added	Low	24.1667	153	-		0.0011	Accept	PR-2
euclidean.twod.SubLine	intersection()	-	-		38.7778	358	-		0.0000		
euclidean.twod.SubLine	intersection()	Conditional	One if block added	Low	39.2222	362	-		0.0021	Accept	PR-3
math3.complex.Complex	reciprocal()	-	-		22.5094	138	0.5846		0.0000		
math3.complex.Complex	reciprocal()	Statement	Return type changed from NaN to INF	Medium	22.5094	138	0.5808		0.0041	Accept	PR-4
math3.analysis.differentiation.DSCompiler	atan2()				74.0265	3611			0.0000		
math3.analysis.differentiation.DSCompiler	atan2()	Statement	Logic added to fix special cases	Low	74.2708	3621			0.0203	Aceept	PR-5
lang.text.StrBuilder	indexOf()				25.8727	2962			0.0000		
lang.text.StrBuilder	indexOf()	Multiple	If block added, statements and break	Medium	25.8909	2971			0.0693	Accept	PR-6
lang.text.StrBuilder	indexof(), contains()				25.8909	2964			0.0000		
lang.text.StrBuilder	indexof(), contains()	Conditional	For loop updated for size	Medium	25.8909	2964			0.0693	Accept	PR-7
lang.text.StrBuilder	appendFixedWidthPadRight()				25.8909	2964			0.0000		
lang.text.StrBuilder	appendFixedWidthPadRight()	Parameter	Metohod argument updated	Medium	25.8909	2964			0.0693	Accept	PR-8
lang.builder.EqualsBuilder	append()				32.1379	962			0.0000		
lang.builder.EqualsBuilder	append()	Conditional	Nested if else block added	Low	32.6552	977			0.0025	Accept	PR-9
lang.text.StrBuilder					22.8939	3160			0.0000		
lang.text.StrBuilder		Conditional	Two if blocks are addded	Medium	22.9545	3168			0.0813	Accept	PR-10
lang.Entities	escape()				278.9231	3647			0.0000		
lang.Entities	escape()	Multiple	variable add / remove , nested if else	Medium	280.5385	3668			0.0044	Accept	PR-11
lang3.time.FastDateParser					31.0370	898			0.0000		
lang3.time.FastDateParser		Statement	Multiple line of code is removed	Low	30.4444	882			0.0015	Accept	PR-12
dagger.internal.Loader								9	0.0004		
dagger.internal.Loader		Class Defination	Changed from interface to abstract	Low				14	0.0005	**Reject**	PR-13

Figure 25: Sample of Pull Request Decisions

4.6.6 **Pull Request Samples**

We reviewed the data of various pull requests and decided whether to accept or reject the pull request based on the outcome of our tooling GraphEVOPR. We have a variety of types of change and degrees of change in the software structure. We observed that more than 70% of the pull request we reviewed fall under the low or medium category. Fig.25 shows some sample pull request from total 627 pull that we studied.

Table 10: Software Metric for Incubator-shenyu Pull Requests 16 and 17

Description	PR7	PR8
Portrait	0	0.0913
Modifiers	1	1
LOC	32	32
cobModified	9	9
fanin	1	1
fanout	8	8

4.7 User Study

The primary goal of user studies is to gain understanding of the practical applications of a given technology. Similar to this, our user study seeks to understand the value of our strategy and tool based on user experience input. Additionally, it can aid in determining the advantages and disadvantages of our visualization tool and direct future efforts to enhance current methods [33].

In this part, we use a user research to assess the applicability and usability of our methodology and tool. **24** people took part in the user study out of 100 plus invitees and completed a series of tasks before answering questions on a questionnaire. The questionnaire consisted of **10** questions using to rate the tool's usefulness [34].

4.7.1 Participants

Real software development managers, engineers, and testers with experience levels ranging from 0 to 10 were asked to test out our solution and then complete a survey as part of the user study. We reached out to total more than 100 software development managers, software engineers, and testers. Only 24 opted to participate in the study. 23 out of 24 participants mentioned their industry experience. The participants belong to more than 10 international industries worldwide, including Google, Amazon, Cummins, Capgemini, Deloitte, Tata Consultancy Services (TCS), HARMAN International and others. However, because our participants did not provide enough information about their age and gender, we did not wish to use partial statistics in the study. Our participants were experienced (69% more than 10 plus years, 25% between 3 to 10 years)

4.7.2 Procedure

The participants had to finish a number of quick activities before the study even began. They first had to provide their email address as a means of consent. Second, they were prompted to respond to a series of questions designed to elicit demographic data, including name, experience level, and opinions regarding the study's subject. Third, a brief tool demo film was shown, and the attendees were instructed to observe sample instances with software metrics and visualization.

Participants in this study were given a demo film, a set of activities, and a questionnaire to evaluate the tool. The tasks involved gathering information about the GrapshEVOPR tool's usability and usefulness. The assignments are outlined in Table 11. The questionnaire was then filled out by the participants utilizing the tool's sample examples and demo video. Using Google Forms, the participants' replies were gathered.

4.7.3 Questionnaire

Our questionnaire include 2 types of questions including (1) Likert scale questions to evaluate usability and use-fullness of the tool, (2) open ended questions to gain the feedback on design and improvements of the tool.The list of questions and their types are listed in Table 12.

4.7.4 Results and Discussion

We first report the results of usability & use-fullness questions, Q1 to Q8, respectively, and then discuss the strengths and weaknesses of the tool from user feedback in

Q9 and Q10.

To evaluate the usability and use-fullness of the tool, the participant were asked to perform the set of small tasks, watch the video clip and review the sample examples with visualization and software metrics of different pull request from this study.

Q1: Participants are questioned about their thoughts on the tool's user interface in this question. 22 out of 24 participants, or more than 90%, agreed that the tool's UI is simple and easy to use.

Q2: We asked participants if they could speed up the time and effort spent on the review of pull requests. More than 85% of the participants agreed with the majority of the positive comments. And 8% of participants said that it would be helpful.

Q3: We asked participants if they would suggest this tool to other developers and engineers for pull request review, and we received a very positive response. Participant support for this was 91%.

Q4: The purpose of this inquiry was to determine whether seasoned developers the majority of whom had more than three years of experience in the field would prefer manual review of a pull request to GraphEVOPR, which shows the software metrics and call graphs along with the code changes. The results indicate that 75% of participants voted in favor of it, while 25% thought it may aid in the review of pull requests. After speaking with a small number of the participants from the 25% who may have voted, we learned that they would still prefer to directly review the code for upkeep and bug-fixing duties. The purpose of this inquiry was to learn more about software architecture, not debugging. The responses to Q2 confirm this.

Q5: To find out if it can help with the issue of detecting the test coverage while reviewing a pull request, we asked the participant this question. More than 95% of participants strongly agreed on this, according to the results of the poll.

Q6: In this question, we asked the participant whether or not a visual depiction of classes, methods, and their related pathways will be helpful in the pull request review process. One individual disagreed with this, but the other 23 participants strongly agreed.

Q7: We wanted to know if the Network Portrait Divergence value may aid pull request reviewers in determining how a code change will affect the software's structure. Will this help in removing the reliance on knowledgeable resources in a group setting while reviewing pull requests? More than 95% of participants responded positively to this and agreed with it.

Q8: Participants were asked to vote on the GraphEVOPR tool's general usability and usefulness in this question. The final score demonstrates that every participant agreed to it.

4.7.5 Survey Response

Fig. 26 shows the actual response details from the 24 participants. It indicated that more than 95% of participants strongly agreed that GraphEVOPR is helpful in the pull request review process and expedites the process. We got an opportunity to discuss with a few of the participants on the call, and they shared that visual representation helps understand the impact and can help in scoping the testing tasks.

Fig. 27 shows the coverage of industry experience participants has. 69% of the participants have more than ten years of experience in the software industry. We aimed to cover users from 0 to 10+ years of experience and understand how GraphEVOPR can support them in the pull request review process. 50% of the participant out of 24 were software developers; the other 50% were software development managers, data engineers, and software development engineers in test and quality engineers.

4.7.6 Participant's Feedback

Two open-ended questions are included in the questionnaire. The participants were asked to rate the tool overall, provide us input that could help us improve the instrument, and contribute insightful information about the challenges faced during the analysis. We summarize and synthesize the participants' comments as follows:

Java version and other languages: One participant specifically asked, *"Will this tool help with different versions of java?"*. Yes, GraphEVOPR can support different vesion of java. Hoever it needs further enhancements to support other languages like python, C sharp etc.

Lack of Information: Along with Network Portrait Divergence we used basic software meerics such as AMC, LOC, fanin, fanout etc. to evaluate the impact on software structure and how it can help in a pull request review process. However, 2 participants suggested to include more data points to support this. One participant commented *"More data to back up this analysis"* and another participant asked *"Can also include statement coverage, branch coverage etc like what Testing framework like jest, jasmine,Junit etc.. do."*

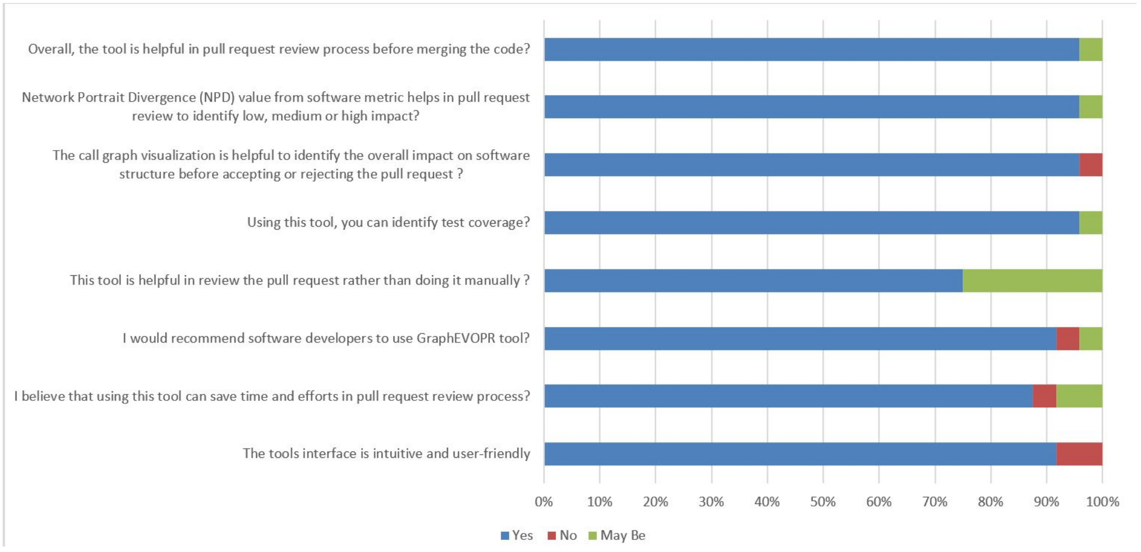


Figure 26: Likert Scale Representation of Survey Responses

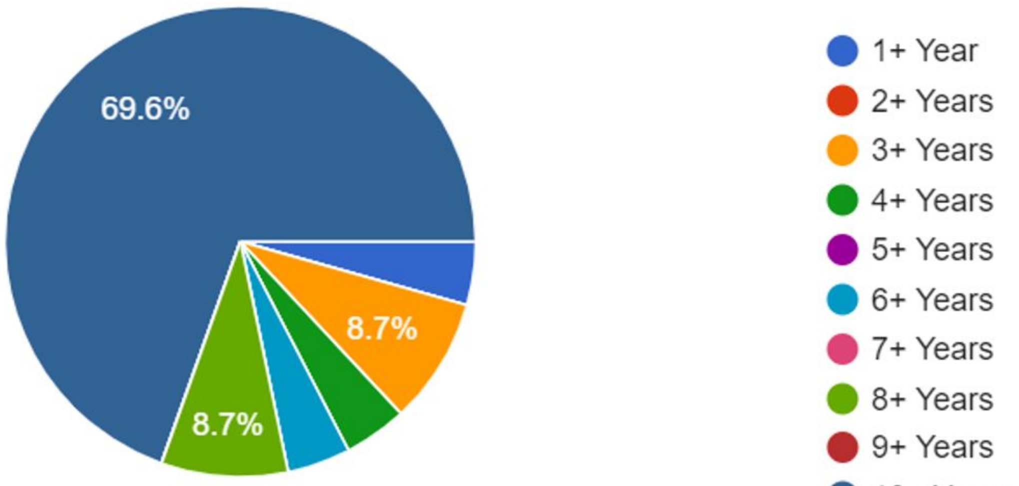


Figure 27: Survey Participants Industry Experience

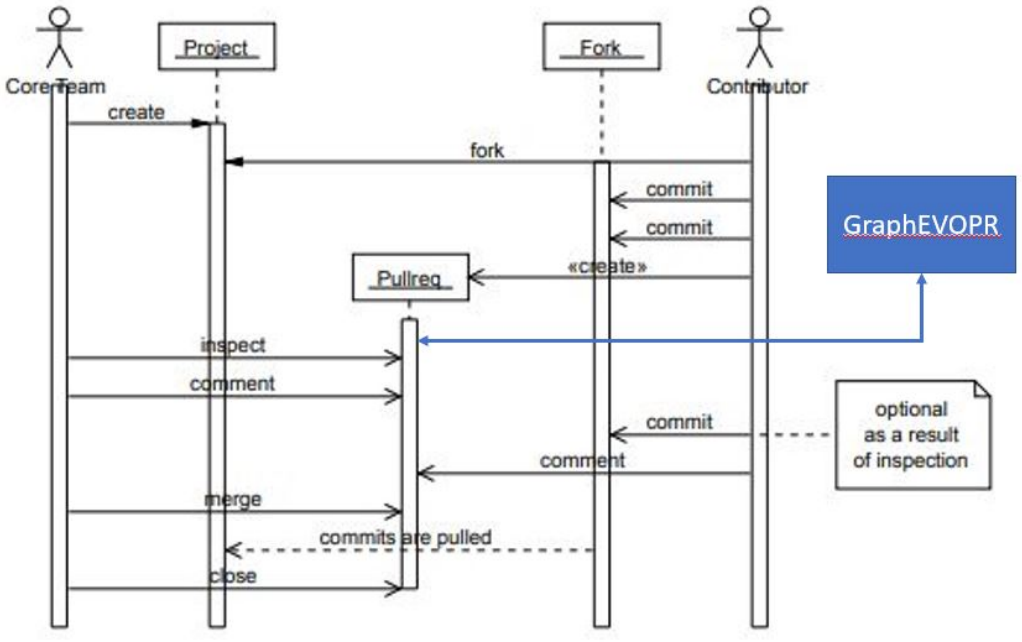


Figure 28: GraphEVOPR in Pull Request Review Process

4.8 Application Demo

Using PyCharm, we can run the GraphEVOPR program locally. We can upload the jar files of the relevant project versions once the program has been launched in order to compare them and create call graphs.

Fig. 28 shows where exactly GraphEVOPR fits in the pull-based development and review process. The software developers can be plugged in quickly in the process. When developers submit the pull request, GraphEVOPR can run the prePR and postPR code comparison and generate software metrics along with software structure visualization. The reviewers can use these details and decide to accept or reject them.

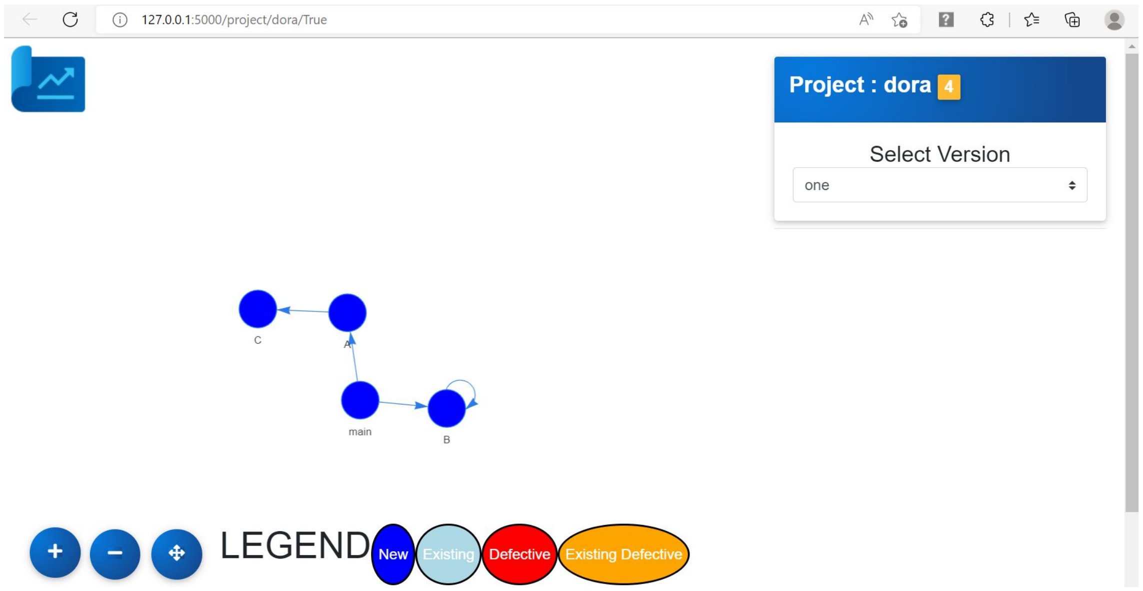


Figure 29: GraphEVOPR Landing Page

Fig. 29 shows the landing page of the GraphEVOPR application. We ran this tool locally on the machine. You can navigate to the upload file page from the left top icon. The top right corner is where you can select the software version under test. We also have legends for new, existing, defective, and existing defective changes identification and color coding. Based on a user study, it's an easy interface to use.

Fig. 30 shows the load of a project page. A user can navigate to this page from the application landing page. It has an option to choose the jar files; a user can select the prePR and postPR files to run. It also provides an opportunity to enter a project name. It has a load button that will pull the jar files submitted from the front end to the backend in the GraphEVOPR process.

Fig. 31 shows the zoomed-in view of a StrBuilder class from Lang61 pull request. As you can see in the details, the function indexOf is modified as part of the pull request.

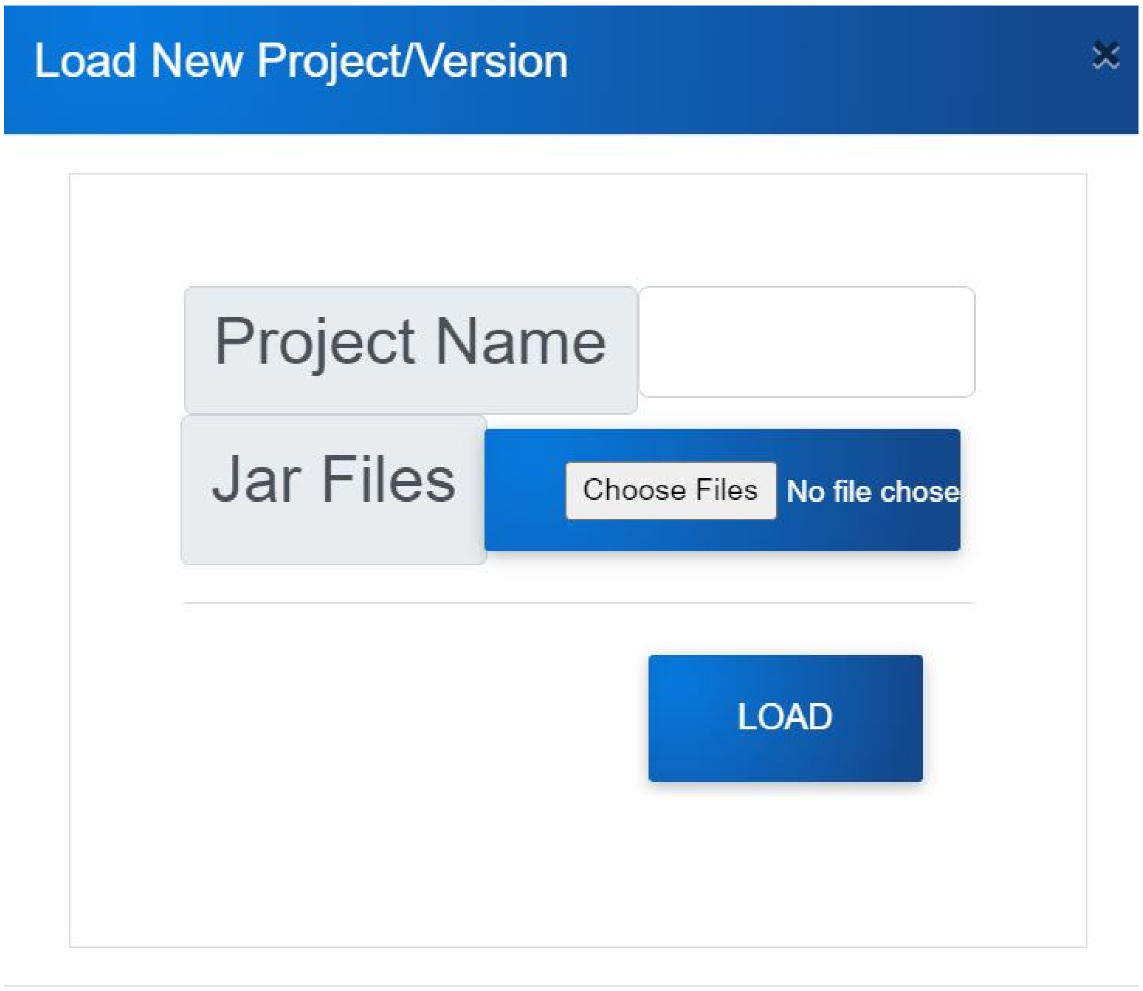
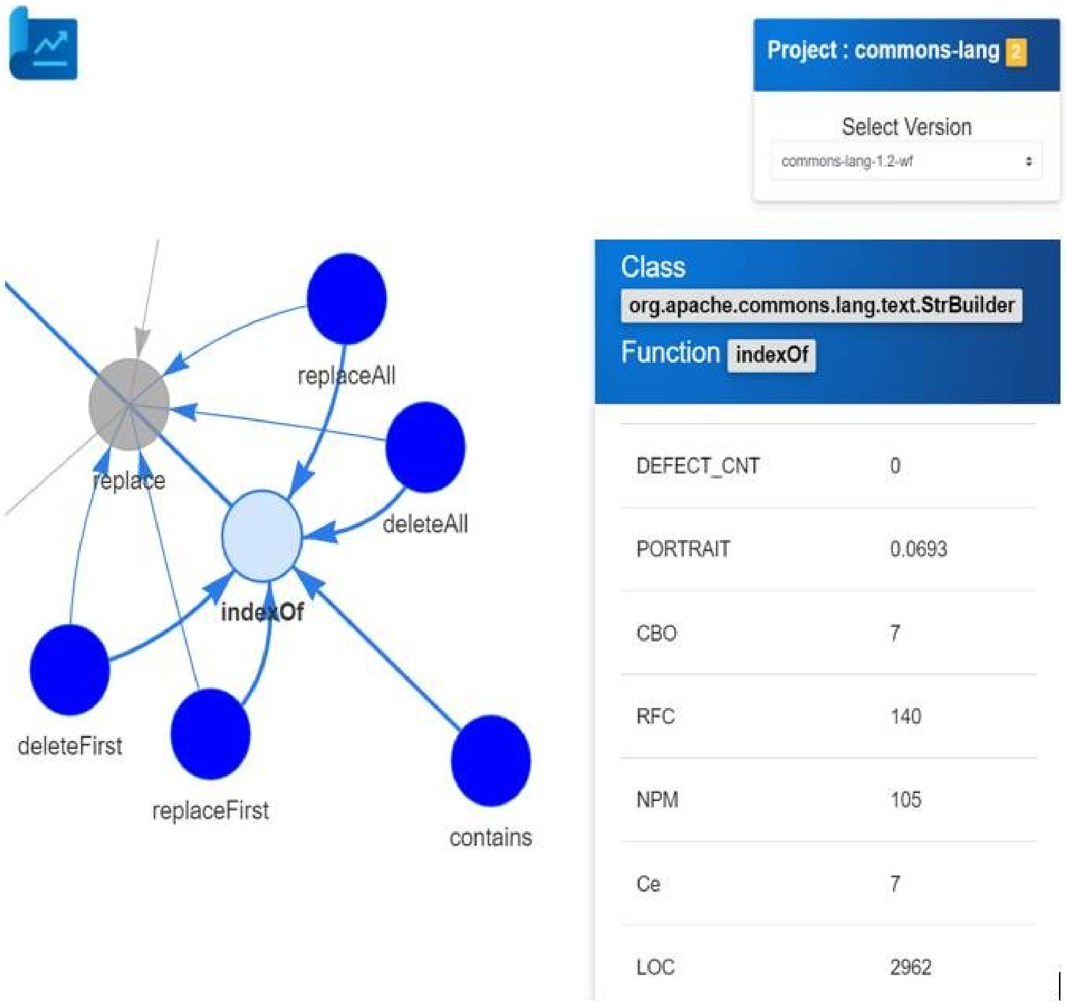


Figure 30: GraphEVOPR Load Project Page



Class org.apache.commons.lang.text.StrBuilder

Function indexOf

DEFECT_CNT	0
PORTRAIT	0.0693
CBO	7
RFC	140
NPM	105
Ce	7
LOC	2962

Figure 31: GraphEVOPR Result View

It shows the paths are impacted due to this change. The dark blues are the functions that interact with indexOf function; with this change, they are also potentially impacted. We can also see the software metric data on the right-hand side. This is a very helpful visualization in the pull request review process.

Table 11: Description of Tasks for User Study

Task ID	Description
T1	Name two classes that has high value of network portrait divergence
T2	Name two classes where you see significant change in respective test cases
T3	Name two classes where you see no code changes however impact on the path or network portrait value
T4	Impact of code change on test coverage
T5	Code and test change analysis using path based visualization

Table 12: User Study Questions

Question	Description	Type
Q1	The tools interface is intuitive and user-friendly	Likert Scale
Q2	I would recommend software developers to use this tool	Likert Scale
Q3	This tool is helpful in review the pull request rather than doing it manually	Likert Scale
Q4	I believe that using this tool can save time and efforts in pull request review process	Likert Scale
Q5	Using this tool you can identify test coverage	Likert Scale
Q6	The call graph visualization is helpful to identify the overall impact on software structure before accepting or rejecting the pull request	Likert Scale
Q7	Network portrait value from software metric helps in pull request review to identify low, medium or high impact	Likert Scale
Q8	Overall, the tool is helpful in pull request review process before merging the code	Likert Scale
Q9	Please list any reasons for your answer to previous questions	Open Ended
Q10	Would you like to add any other suggestion? feedback? or limitations?	Open Ended

CHAPTER 5

CONCLUSION

5.1 Overview

In this chapter, we are discussing on conclusion and future work. We have studied and analyzed 14 open-source java projects. From these 14 projects, we deep-dived into 627 plus pull requests and compared different software metric data. Primarily, we compared the network portrait divergence data from previous and current pull requests to understand the impact of code change. We further analyzed the behavior of tests related to the code change. We also studied how visual representation of the software structure supports the pull request review process, reduces the cycle time of a pull request review process, and expedite the overall process.

5.2 Conclusion

We studied the use of changes in software class-level metrics, such as Network portrait divergence, for evaluating and visualizing pull requests in this study. Specifically, We collected class-level metrics for code in pull requests before and after modifications.We reviewed a set of metrics and identified associated changes to test failures.We provided tool support for our technique, GraphEvoPR.We studied the usefulness of visual representation of software structure and how it supports the pull request review process.

We evaluated the performance of class-level metrics using 627 Pull requests from

14 popular Java open-source systems. The study demonstrated how network portrait divergence measures identify the structural changes and how visual representation of software supports the pull review process.

5.3 Future Work

We will concentrate on growing the amount of software and pull requests in the near future. Furthermore, we intend to use machine learning models (Decision Trees, Random Forest, Extremely Randomized Trees, AdaBoost, Gradient Boosting, and XGBoost) to estimate code quality in terms of a set of tests that may be required to pass a pull request.

REFERENCE LIST

[1] G. Gousios, M. Pinzger, and A. v. Deursen, “An exploratory study of the pull-based software development model,” in *Proceedings of The 36th International Conference on Software Engineering*, 2014, pp. 345–355.

[2] O. Kononenko, T. Rose, O. Baysal, M. Godfrey, D. Theisen, and B. De Water, “Studying pull request merges: a case study of shopify’s active merchant,” in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, 2018, pp. 124–133.

[3] F. Calefato, F. Lanubile, and N. Novielli, “A preliminary analysis on the effects of propensity to trust in distributed software development,” in *2017 IEEE 12th International Conference on Global Software Engineering (ICGSE)*. IEEE, 2017, pp. 56–60.

[4] V. Walunj, G. Gharibi, D. H. Ho, and Y. Lee, “Graphevo: Characterizing and understanding software evolution using call graphs,” in *2019 IEEE International Conference on Big Data (Big Data)*, 2019, pp. 4799–4807.

[5] V. Sobreira, T. Durieux, F. Madeiral, M. Monperrus, and M. de Almeida Maia, “Dissection of a bug dataset: Anatomy of 395 patches from defects4j,” in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2018, pp. 130–140.

[6] D. Spadini, M. Aniche, M.-A. Storey, M. Bruntink, and A. Bacchelli, “When testing meets code review: Why and how developers review tests,” in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE).* IEEE, 2018, pp. 677–687.

[7] Z. Li, Y. Yu, T. Wang, G. Yin, S. Li, and H. Wang, “Are you still working on this? an empirical study on pull request abandonment,” *IEEE Transactions on Software Engineering*, vol. 48, no. 6, pp. 2173–2188, 2022.

[8] G. Gousios, M.-A. Storey, and A. Bacchelli, “Work practices and challenges in pull-based development: The contributor’s perspective,” in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, 2016, pp. 285–296.

[9] V. Stray, N. B. Moe, M. Mikalsen, and E. Hagen, “An empirical investigation of pull requests in partially distributed bizdevops teams,” in *2021 IEEE/ACM Joint 15th International Conference on Software and System Processes (ICSSP) and 16th ACM/IEEE International Conference on Global Software Engineering (ICGSE)*, 2021, pp. 110–119.

[10] M. Ortu, G. Destefanis, D. Graziotin, M. Marchesi, and R. Tonelli, “How do you propose your code changes? empirical analysis of affect metrics of pull requests on github,” *IEEE Access*, vol. 8, pp. 110 897–110 907, 2020.

[11] S. M. Syed-Mohamad and N. S. Md. Akhir, “Soready: An extension of the test and defect coverage-based analytics model for pull-based software development,” in *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*, 2019, pp. 9–14.

[12] E. Sherif, A. Liu, B. Nguyen, S. Lerner, and W. G. Griswold, “Gamification to aid the learning of test coverage concepts,” in *2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEEandT)*, 2020, pp. 1–5.

[13] N. Williams, “Towards exhaustive branch coverage with pathcrawler,” in *2021 IEEE/ACM International Conference on Automation of Software Test (AST)*, 2021, pp. 117–120.

[14] K. H. Bennett and V. T. Rajlich, “Software maintenance and evolution: a roadmap,” in *Proceedings of the Conference on the Future of Software Engineering*. ACM, 2000, pp. 73–87.

[15] J. Krüger, G. Calikli, T. Berger, T. Leich, and G. Saake, “Effects of explicit feature traceability on program comprehension,” in *Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2019.

[16] R. S. Pressman, *Software Engineering: A Practitioner’s Approach*. Palgrave Macmillan, 2005.

[17] C. Ghezzi, M. Jazayeri, and D. Mandrioli, *Fundamentals of Software Engineering*. Prentice Hall PTR, 2002.

[18] P. Bhattacharya, M. Iliofotou, I. Neamtiu, and M. Faloutsos, “Graph-based analysis and prediction for software evolution,” in *Proceedings of the 34th International*

Conference on Software Engineering, ser. ICSE '12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 419–429.

[19] D. Grove, G. DeFouw, J. Dean, and C. Chambers, "Call graph construction in object-oriented languages," *ACM SIGPLAN Notices*, vol. 32, no. 10, pp. 108–124, 1997.

[20] R. Vasa, J.-G. Schneider, and O. Nierstrasz, "The inevitable stability of software change," in *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*. IEEE, 2007, pp. 4–13.

[21] G. Gharibi, R. Tripathi, and Y. Lee, "Code2graph: automatic generation of static call graphs for python source code," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 880–883.

[22] G. Gharibi, R. Alanazi, and Y. Lee, "Automatic hierarchical clustering of static call graphs for program comprehension," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 4016–4025.

[23] L. Wang, Z. Wang, C. Yang, L. Zhang, and Q. Ye, "Linux kernels as complex networks: A novel method to study evolution," in *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*. IEEE, 2009, pp. 41–50.

[24] D. M. Soares, M. L. D. L. Junior, L. Murta, and A. Plastino, "Rejection factors of pull requests filed by core team developers in software projects with high acceptance rates," in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2015, pp. 960–965.

[25] A. Trautsch, S. Herbold, and J. Grabowski, "A longitudinal study of static analysis warning evolution and the effects of pmd on software quality in apache open source projects," *Empirical Software Engineering*, vol. 25, no. 6, pp. 5137–5192, 2020.

[26] P. Bhattacharya, M. Iliofotou, I. Neamtiu, and M. Faloutsos, "Graph-based analysis and prediction for software evolution," in *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 2012, pp. 419–429.

[27] M. Savić, M. Ivanović, and L. C. Jain, "Analysis of software networks," in *Complex Networks in Software, Knowledge, and Social Systems*. Springer, 2019, pp. 59–141.

[28] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "code2vec: Learning distributed representations of code," *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, pp. 1–29, 2019.

[29] J. P. Bagrow and E. M. Bollt, "An information-theoretic, all-scales approach to comparing networks," *Applied Network Science*, vol. 4, no. 1, pp. 1–15, 2019.

[30] M. Tantardini, F. Ieva, L. Tajoli, and C. Piccardi, "Comparing methods for comparing networks," *Scientific Reports*, vol. 9, no. 1, pp. 1–19, 2019.

[31] M. M. Rahman and C. K. Roy, "An insight into the pull requests of github," in *Proceedings of the 11th working conference on mining software repositories*, 2014, pp. 364–367.

[32] D. Chen, K. T. Stolee, and T. Menzies, “Replication can improve prior results: A github study of pull request acceptance,” in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 2019, pp. 179–190.

[33] R. Kosara, C. Healey, V. Interrante, D. Laidlaw, and C. Ware, “Visualization viewpoints,” *IEEE Computer Graphics and Applications*, vol. 23, no. 4, pp. 20–25, 2003.

[34] M. A. Kabir, M. U. Rehman, and S. I. Majumdar, “An analytical and comparative study of software usability quality factors,” in *2016 7th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, 2016, pp. 800–803.

VITA

Vasim Shaikh is a computer science graduate student. Under the direction of Dr. Yugyung Lee, he is conducting research and writing a thesis. His main areas of research interest are artificial intelligence, machine learning in software testing, defect forecasting, and the software review process. Vasim has approximately 12 years of management and quality engineering expertise in the software business. He is currently employed by AMAZON as a Quality Engineering Leader, contributing to the development and testing of a top-notch digital reading experience for millions of customers worldwide. He formerly served as CAPGEMINI AMERICA INC's Quality Engineering Manager. Vasim is happy to be the proud father of a lovely daughter and likes to play cricket, listen to music, and cook.