Master's Thesis

ON THE RELATION OF SOCIO-TECHNICAL CONGRUENCE AND CODE REVIEW MEASURES

KRISTELA KAJA

May 16, 2023

Advisor:

Christian Hechtl	Chair of Software Engineering

Examiners:

Prof. Dr. Sven Apel	Chair of Software Engineering
Prof. Dr. Ingmar Weber	Societal Computing (Alexander von Humboldt Chair)

Chair of Software Engineering
Saarland Informatics Campus
Saarland University

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken,______________________ ______________________________
(Datum/Date) (Unterschrift/Signature)

ABSTRACT

The software development process contains both social and technical factors. The social factors involve the people who develop the software, while the technical factors involve the artifacts (e.g., code and documentation) that are produced. A socio-technical perspective can demonstrate valuable insights into the characteristics of a software project. This perspective is particularly important for Open-Source Software (OSS) projects due to their peer-review-based development process, globally dispersed teams across different time zones, ad-hoc development procedures, and other factors. Thus, we analyze the Socio-Technical Congruence (STC) to investigate any impact it may have on the code review process across ten OSS projects.

We measure the STC by using the Socio-Technical Motif Congruence (STMC) framework proposed by Mauerer et al. [21]. The STMC utilizes socio-technical networks to measure the alignment between the social and technical aspects of the development process, by quantifying the level of communication among developers contributing to the same file or related files. To evaluate the code review process, we measure the acceptance rate, review rate, and first review interval for each pull request in the chosen OSS projects. We then examine the correlation between the STMC and these code review measures to determine if the STMC has any influence on them.

According to our results, the STMC has no statistically significant influence on the review rate for almost all OSS projects we analyze (8/10). Nevertheless, we cannot conclude whether the STMC has any influence on the acceptance rate and first review interval since the results are mixed; some projects show statistically significant correlations while others do not.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

OSS	Open-Source Software
STC	Socio-Technical Congruence
STMC	Socio-Technical Motif Congruence
IT	Information Technology
dSTMC	degree of Socio-Technical Motif Congruence
OSSD	Open-Source Software Development
STMC_r	refined Socio-Technical Motif Congruence
dSTMC_r	degree of refined Socio-Technical Motif Congruence
PaStA	Patch-stack analysis
TD	technical debt

1 INTRODUCTION

Open-Source Software (OSS) systems, such as web browsers[1], operating systems[2], and many others, are a large part of the Information Technology (IT) field. Nowadays, OSS systems are the spine of technology because they are decentralized, open to everyone, and often distributed freely. However, developing such software has its own challenges. Open-Source Software Development (OSSD) is based on peer-review [19] and the team is distributed worldwide in different time zones. Additionally, the risky development practices in OSSD, such as the ad-hoc development process, the little attention paid to documentation, and poor project management can pose risks to the OSSD process. These factors increase the difficulty of achieving effective communication, coordination, and collaboration among developers working on interdependent modules of the same project [34]. Despite these challenges of developing OSS systems, developers can produce successful software systems [1]. Developing successful software systems requires not only technical expertise but also effective collaboration, communication, and coordination among team members [15]. Thus, communication, coordination, and collaboration play a critical role in software development.

The STC approach measures the alignment between social and technical aspects of the development process in OSSD, highlighting potential issues and areas for improvement. By providing insights into how well the social and technical aspects fit together, STC can help developers work more effectively together, likely resulting in better OSS systems. Therefore, STC can be a valuable tool for enhancing collaboration and communication in OSSD and improving the quality of OSS systems.

In this thesis, we perform a socio-technical analysis on ten OSS projects. For each OSS project we measure the STC to asses whether STC has an influence on code review measures, namely *acceptance rate*, *review rate*, and *first review interval*. These code review measures are commonly used indicators of the effectiveness and efficiency of the code review process in software development [28, 29]. The acceptance rate reflects the percentage of accepted contributions, indicating the contributions' quality and appropriateness. The review rate shows the percentage of contributions that are reviewed, indicating the thoroughness of the review process. Finally, the first review interval measures the time between the initiation of a contribution and its first review, indicating the efficiency and responsiveness of the review process. By examining the influence of STC on these code review measures, we can gain insights into how the alignment between social and technical aspects of the development process affects the effectiveness and efficiency of the code review process. The obtained information can be utilized to identify potential issues that may improve collaboration and communication among developers.

There exist several methodologies how to measure STC [5, 10, 12, 21, 25, 32]; in this thesis, we use the concept presented by Mauerer et al. [21]. They propose the STMC framework, a

1 Mozilla Firefox - `https://www.mozilla.org/en-US/firefox/new/`, last accessed on 24/02/2023
2 Linux - `https://www.linux.org/`, last accessed on 24/02/2023

quantitative and operational notion of STC that is used in the research community [6, 24, 30]. STMC is a comprehensive and flexible framework that measures the alignment between social and technical aspects of the development process.

In concrete terms, STMC measures the degree to which developers contributing to the same file or to two related files, need to communicate. The bases of STMC are heterogeneous socio-technical graphs. A socio-technical graph is a graph that shows the interactions and dependencies between people and technical components in a software development project. The configuration of socio-technical graphs can vary and different methods have been proposed in literature [12, 30, 32].

Mauerer et al. use a heterogeneous socio-technical graph that has nodes representing either developers or files, and edges representing the relationship between two developers, two files, or between a developer and a file. Mauerer et al. extract two types of sub-graphs from this graph, which they call motifs - triangle and square motifs. The triangle motifs capture the direct collaboration, while the square motifs capture the indirect collaboration in an OSS project. Furthermore, Mauerer et al. use triangle and square anti-motifs to capture the direct non-collaboration or indirect non-collaboration, hence the term anti-motif. To quantify the STMC, Mauerer et al. propose the degree of Socio-Technical Motif Congruence (dSTMC) which is composed of the triangle and square congruence.

Additionally, we propose a refined version of the STMC that we call refined Socio-Technical Motif Congruence (STMC_r). STMC_r filters out the false positive motifs that the STMC detects.

We employ the STMC and STMC_r framework and calculate the acceptance rate, review rate, and first review interval for ten OSS projects, including Deno, Vs Code, Tensorflow, Moby, Bootstrap, Atom, TypeScript, React, Electron, and Next.js chosen for their varying sizes, project ages, and domains. We obtain the necessary data for each project, by mining the version control systems of these projects. Additionally, to capture the ever-changing structures of the chosen OSS projects, we split them into six-month windows.

Next, we assess if the STMC or STMC_r affect the acceptance rate, review rate, and first review interval. We accomplish this by correlating the STMC or STMC_r with each of these metrics, using Kendall rank correlation coefficient (Kendall's Tau)[3]. We choose Kendall's Tau because it is suitable for non-normalized data and small data sets, which are both characteristics of the data produced by our STC analysis.

Our findings reveal that we are unable to determine if the STMC or STMC_r impact the acceptance rate since some projects demonstrate statistically significant correlations while others do not. Similarly, we cannot conclude if the STMC or STMC_r impact the first review interval, as the results are inconclusive, with some projects showing statistically significant correlations and some not. On the other hand, we can conclude that the STMC or STMC_r does not affect the review rate, as nearly all projects indicate no statistically significant correlations. This outcome suggests that other factors likely influence the review rate.

3 https://en.wikipedia.org/wiki/Kendall_rank_correlation_coefficient, last accessed on 12/03/2023

2 BACKGROUND

In this chapter, we present the necessary information for comprehending this thesis. We describe OSS development contribution tools and the often-used workflows. Additionally, we explain and formalize socio-technical networks, focusing on those used in this thesis. Lastly, we operationalize STMC and quantify it by providing the formulas for how STMC is calculated.

2.1 OPEN-SOURCE SOFTWARE DEVELOPMENT (OSSD)

Developers use various tools for contributing to OSS projects, including mailing lists, version control systems like Git[1], code review tools such as Gerrit[2], issue trackers like JIRA[3], and communication channels such as Gitter[4] or Discord[5]. While the use of mailing lists to contribute to OSS projects is declining, more and more projects are opting to use GitHub[6] [13], a hosting service for software development and version control systems using Git. In this thesis, we focus on analyzing projects that, to the best of our knowledge, primarily utilize GitHub as a tool for contributing to OSS. Our decision is based not only on the increasing number of projects utilizing GitHub but mainly due to the simplicity of applying our STC analysis methodology to the data extracted from GitHub.

Each OSS project in GitHub has its repository where the developers manage and store their contributions, such as code and documentation, as well as collaborate and communicate with each other. To fulfill these goals, GitHub offers two options for OSS projects, namely issues and pull requests. Issues are used to track bugs, feature requests, and other important information about the project.

On the other hand, developers use pull requests to suggest modifications to the project's code or other associated files, like the `README.md` file. Usually, a developer changes the code in a branch[7] or a fork[8] and then submits a pull request to merge those changes into another branch. Afterward, a different developer reviews the pull request and subsequently decides to either accept it, reject it, or request additional changes. Furthermore, a pull request can be linked to an issue to show that a fix or implementation is in progress.

However, note that the workflow of a pull request and issue may vary between OSS projects, as it depends on the rules and guidelines set by project maintainers. Some OSS projects allow a pull request to be merged after receiving a specific number of approvals

1 `https://git-scm.com/`, last accessed on 30/03/2023
2 `https://gerritcodereview.com/`, last accessed on 12/03/2023
3 `https://atlassian.com/software/jira`, last accessed on 12/03/2023
4 `https://gitter.im/`, last accessed on 12/03/2023
5 `https://discord.com/`, last accessed on 12/03/2023
6 `https://github.com/`, last accessed on 30/03/2023
7 `https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-branches`, last accessed on 30/03/2023
8 `https://docs.github.com/en/get-started/quickstart/fork-a-repo`, last accessed on 30/03/2023

from other developers while in others, a project maintainer must review and approve the pull request before it can be merged.

2.1.1 *Pull Requests*

A developer creates a pull request when they make code changes to a project and want to merge those changes into another branch. The *author* of the pull request is the developer who initiates it and provides the initial commit with their changes. In some cases, several developers contribute to the same pull request, resulting in several co-authors. We refer to both authors and co-authors simply as *'author'*, without distinguishing between them.

The pull request contains a list of all the files that the author changed or added, which can be viewed via the *Files Changed* tab in GitHub. These changed files are important because they demonstrate the specific changes made to the code base by the included commits. For simplicity, we use the term *file* to refer to any changed file in a pull request, regardless of the change type or author. Moreover, the pull request contains all the initial commits that are part of the pull request or commits that are added subsequently to the same pull request.

Once a pull request is opened, a *review* is initiated. A review also referred to as a *code review*, is a quality assurance process used to improve code quality, ensure that the code complies with the project standards, and find potential issues or bugs before they are merged into the project's code base. The developer who conducts the code review is called the *reviewer*.

To conduct a code review, the author may explicitly request a review from another developer by utilizing GitHub's *review functionality*[9]. Nevertheless, the reviewer can conduct a code review using the same review functionality without any explicit request from the author. Alternatively, reviewers can conduct a code review by leaving comments on the pull request's *Conversation* tab[10]. In all these cases, a developer conducts a code review of the changes made by another developer. This process is known as *peer code review*. Once the initial peer code review is complete, the reviewer can approve the pull request, request changes, or reject it.

If changes are requested, authors and reviewers can *communicate* through comments on the conversation tab. The conversation tab serves not only for communication between the author and the reviewer but also enables communication among other developers involved in the project. Developers, regardless of their role in the pull request, can use this tab to leave general comments, ask questions or provide props, that everyone can read and react to.

After the pull request changes are approved, they can be *merged* into the code base. Typically, communication and collaboration between developers regarding this pull request end once it has been merged. The pull request is now considered closed and part of the project's code base. However, there may be cases where the pull request is reopened or referenced in other pull requests (although these cases are not within the scope of this thesis).

9 https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/requesting-a-pull-request-review, last accessed on 15/03/2023

10 https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/reviewing-changes-in-pull-requests/commenting-on-a-pull-request, last accessed on 15/03/2023

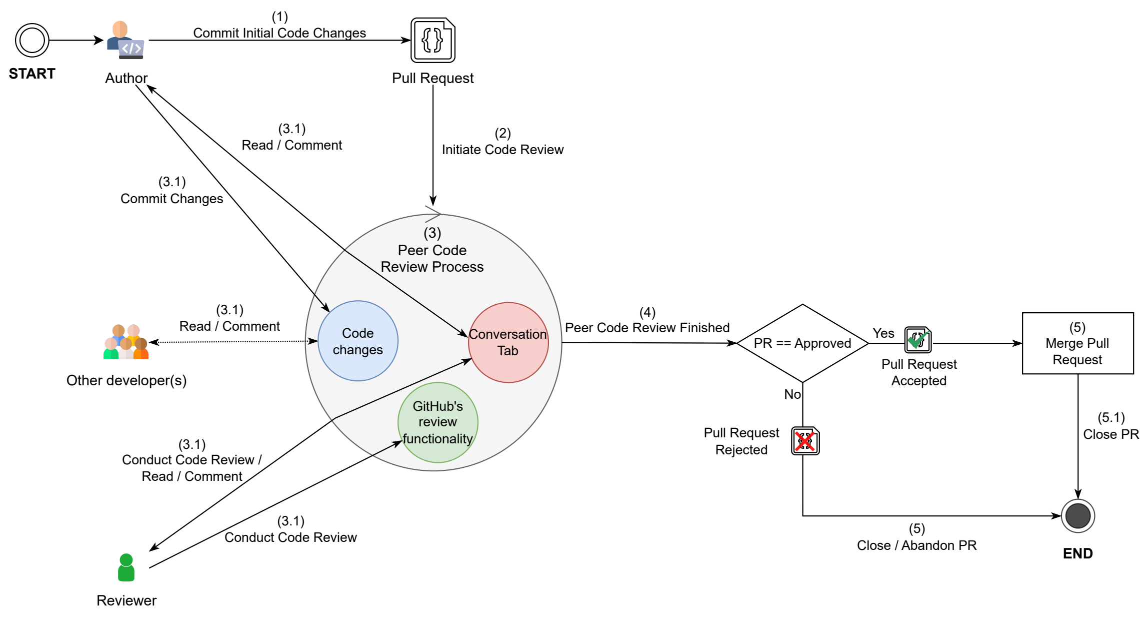


Figure 2.1: In this figure, we illustrate the peer code review workflow that we consider in this thesis. Initially (1) the pull request author creates a pull request by committing the code changes to GitHub. Afterward (2), the code review process starts. The reviewer conducts the code review (3) by using the GitHub functionality or by commenting on the pull request; the author of the pull request commits the code changes and interacts with the reviewer through the communication tab; other developers might join the communication in this PR (3). The reviewer approves or rejects the pull request (4). If the pull request is approved, it is merged into the project's code base (4.1); otherwise, it is abandoned or closed (4.2).

If the reviewer refuses the changes of the pull request, the pull request is not merged into the code base, and no further changes are added to it. We illustrate this workflow in Figure 2.1.

2.2 SOCIO-TECHNICAL CONGRUENCE (STC)

The concept of STC has been formalized many times throughout the years [30]. Melvin Conway first formally defines it in 1968 [8], suggesting that *"Organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations"* - this is known as "Conway's Law". Cataldo et al. [7] propose another STC formalization based on this law, namely *"A technique for measuring task dependencies among people, and the 'fit' between these task dependencies and the coordination activities performed by individuals"*. Kwan, Schröter, and Damian [17] define STC as *"An approach that measures coordination by examining the alignment between the technical dependencies and the social coordination in the project"*. As seen above, researchers often provide a new STC definition when analyzing it. In contrast, we do not formalize the concept of STC but we focus on conducting a quantifiable analysis of STC using the STMC concept presented by Mauerer et al. [21].

Besides the various formalizations of STC, researchers use numerous techniques to measure it. Matrices [6, 7, 16] and socio-technical networks [5, 21, 33] are two widely used techniques to analyze STC. Socio-technical networks are a standard tool used to represent how people communicate and collaborate with each other and have been deployed in many analysis scenarios [4, 14, 20]. Furthermore, socio-technical networks are also used to implement the STMC [21] approach employed in this thesis.

2.2.1 *Socio-Technical Networks*

Socio-technical networks refer to the interdependent relationships between people and technology within a system or organization. They are constructed by the combination of social networks and technical networks. Social networks represent the relationship or communication between people that are part of a system or an organization. Meanwhile, technical networks represent the physical and digital components that enable the activities of the system or organization. Together, social and technical networks create a network that is often used in organizational behavior, human-computer interaction, and IT to better understand the communication and collaboration between people and technology. Nevertheless, in this thesis, we focus only on socio-technical networks in IT specifically in software engineering.

2.2.1.1 *Formalization*

In this chapter, we formalize a socio-technical network mathematically in the context that it is used in this thesis. A socio-technical network can be defined as a tuple $G = (V, E)$, where G is the graph (or the network), V is the set of vertices, and E is the set of edges between the vertices. The vertices represent people with different roles (e.g., author, reviewer, developer) or technical artifacts (e.g., files, issues, pull requests). The edges represent the dependencies or relationships between the vertices (e.g., communication, collaboration). A socio-technical network is a heterogeneous graph because it has two or more types of vertices and/or two or more types of edges.

An edge can also have weights that show the strength of the relationship dependent on the context. For instance, the weight of an edge that shows the communication between two people can refer to the number of communication messages between these two people. Moreover, an edge can be directed or undirected, depending on whether the relationships they represent are one-way or bidirectional. Additionally, multiple edges can exist between the same vertices.

An example of operationalization of a socio-technical network is the one used by Mauerer et al. [21] when defining STMC. See Figure 2.2. Mauerer et al. [21] use a heterogeneous, unweighted, undirected graph to show the communication between the developers, the artifact dependencies, and the contribution of the developers to the artifacts.

2.3 SOCIO-TECHNICAL MOTIF CONGRUENCE (STMC)

STMC is a concept introduced by Mauerer et al. [21] and it is essential in this thesis. Therefore, in this chapter, we explain in detail the operationalization of STMC and the results of correlating STMC with software quality measurements that Mauerer et al. conclude in their study.

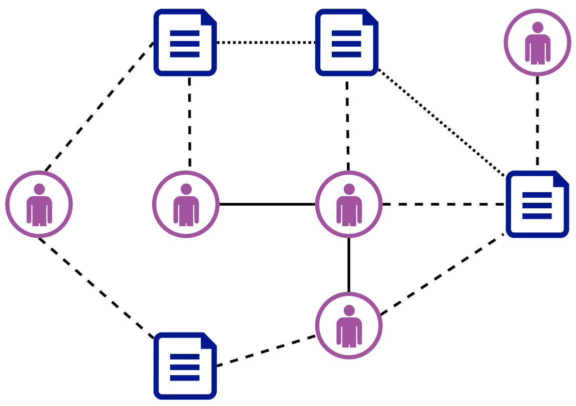


Figure 2.2: The blue square icons are artifacts and the purple circles are developers. The dotted edge between two artifacts shows a dependency between two artifacts. The dashed edge between a developer and an artifact shows a change that the developer did to the artifact. The bold edge between the two developers shows that these developers are communicating with each other.

Mauerer et al. use socio-technical networks to explore whether alignment or misalignment of social communication structures and technical dependencies in large software projects influence software quality. To do that, they focus on motifs in socio-technical networks. *Motifs* are social and technical relationship patterns embedded as subgraphs in a larger socio-technical network. Motifs found in networks are commonly used in other scientific fields [2, 11] but were never used in software engineering organizational research prior to the study of Mauerer et al. In addition, motifs are also useful to understand the evolution of socio-technical networks over time, like open-source software communities and their software architectures [27].

2.3.1 *Operationalizing STMC*

Mauerer et al. focus on two basic types of socio-technical collaboration, such as: direct and indirect collaboration because based on previous research [6], they assume that simple patterns are the "basis" that can capture socio-technical interactions.

Direct collaboration arises when two developers modify the same artifact and they also communicate with each other. To concretize direct collaboration, Mauerer et al. introduce a "*triangle motif*" which is a subgraph that shows two developers modifying the same artifact and communicating with each other. Figure 2.3 (a) illustrates this scenario. Nevertheless, in some cases, developers make changes to the same artifact without any communication, which is the opposite of a motif and referred to as a "*triangle anti-motif*". This is depicted in Figure 2.3 (b).

On the other hand, *indirect collaboration* emerges when two developers communicate with each other and modify *two* artifacts that are interconnected. Similarly to direct collaboration, Mauerer et al. introduce a *square motif* which is a subgraph that captures the indirect collaboration, namely two developers communicating with each other and modifying two artifacts that depend on each other. Refer to Figure 2.3 (c). Analogously to the above, there are cases where developers change the interconnected artifacts but do not communicate with each other. These are called *square anti-motifs*. Figure 2.3 (d) shows a formalization of a square anti-motif.

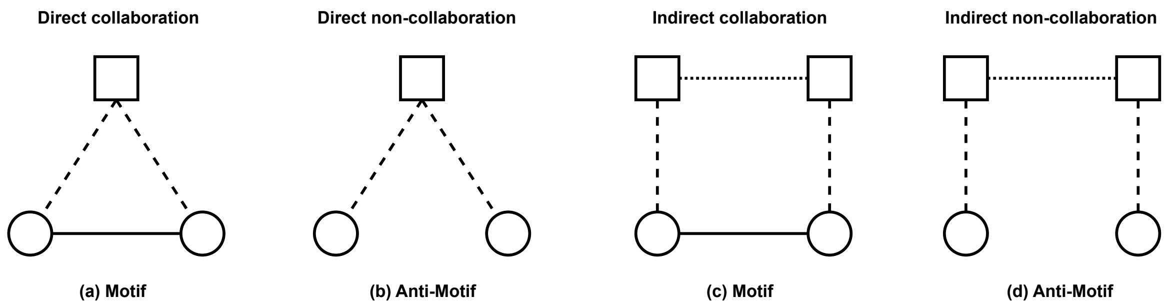


Figure 2.3: We adapt this Figure from [21] Figure 3, which illustrates motifs and anti-motifs. Figures (a) and (c) show a triangle and a square motif respectively, or direct and indirect collaboration. Meanwhile, figures (b) and (d) show a triangle and a square anti-motif respectively or direct and indirect non-collaboration. The circle vertices are developers. The square vertices are artifacts. The solid edge between the two developers indicates there is communication between these developers. The dashed edge between a developer and the artifact indicates that the developer is modifying the artifact. The dotted edge between the two artifacts indicates that these artifacts depend on each other.

To show how motif subgraphs are embedded in a bigger socio-technical network, in Figure 2.4, we highlight examples of motifs and anti-motifs in the socio-technical network illustrated above in Figure 2.2.

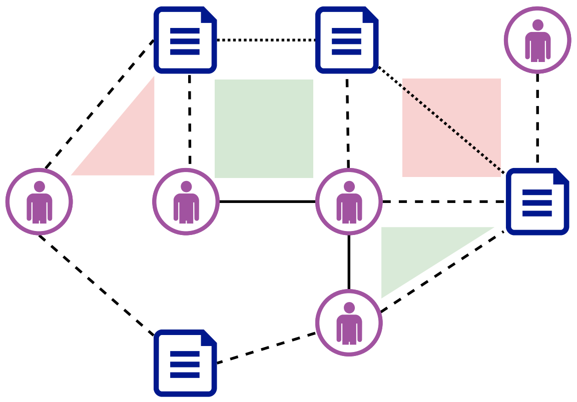


Figure 2.4: We highlight one example for each motif and anti-motif. The green triangle and square show a triangle and square motif respectively. On the other hand, the red triangle and square show a triangle and square anti-motif respectively. They are anti-motifs because the communication edge between the developers is missing in this case.

In summary, the STMC measures the socio-technical collaboration (direct or indirect) and communication between developers and artifacts.

2.3.1.1 *Degree of STMC*

Mauerer et al. propose that it is possible to assess the alignment between social and technical dependencies by measuring the degree of congruence between motifs and anti-motifs. Consequently, they introduce a quantitative measure of STMC, namely the dSTMC. The dSTMC is calculated for both types of motifs - triangle and square motifs. It is calculated as the number of motifs divided by the number of anti-motifs. The result is called triangle congruence (2.1) or square congruence (2.2) based on the motif type.

$$triangle\ congruence = \frac{number\ of\ triangle\ motifs}{number\ of\ triangle\ anti\text{-}motifs} \tag{2.1}$$

$$square\ congruence = \frac{number\ of\ square\ motifs}{number\ of\ square\ anti\text{-}motifs} \tag{2.2}$$

A low triangle or square congruence (generally called congruence) suggests that the developers are not communicating or collaborating effectively. Consequently, it indicates that the patterns of social dependencies (e.g., communication and collaboration) and technical dependencies (e.g., files, issues, and pull requests) do not match which can lead to delays, misunderstandings, and errors. For example, team members may not be aware of the technical dependencies required to complete their tasks, or they may not be able to coordinate effectively.

In contrast, a high congruence suggests that the developers are communicating and collaborating effectively. This means that there is an alignment between social and technical dependencies in an OSS project. A high congruence can lead to better outcomes such as improved productivity [6]. For instance, team members can work more efficiently and avoid unnecessary delays or conflicts.

3 APPROACH

In this chapter, we present the ten OSS projects that we use in this thesis as case studies and explain how we extract their data. Additionally, we explain how we split these OSS projects into time frames to analyze them.

3.1 CASE STUDIES

In this thesis, we analyze ten OSS projects which are: Deno[1], VS Code[2], TensorFlow[3], Moby[4], Bootstrap[5], Atom[6], TypeScript[7], React[8], Electron[9], and Next.js[10]. We choose these OSS projects because they differ in their domain, contribution size (measured by the number of issues), and project age (the time since release). These projects operate in diverse domains, including code editors, front-end frameworks, runtimes, libraries, and others. In addition, we categorize the projects based on the number of issues they have. Large projects, such as VS Code, TensorFlow, and Moby, have a higher number of issues compared to medium-sized projects such as Bootstrap, Electron, Atom, React, and Next.js. The remaining projects are considered small projects. In Table 3.1, we present the number of issues, for each project. In terms of project age, we consider the time passed since the release date, also displayed in Table 3.1.

3.1.1 *Data Extraction*

We extract data from ten OSS projects using codeface[15], a tool developed by Siemens. First, we collect the required data by mining the Git repositories of the ten projects. Next, we feed this data into codeface, which processes the data and later stores it in a database. We also fetch GitHub issues and pull requests using another tool called GitHubWrapper[16]. Then, another tool called codeface-extraction[17] fetches the data from codeface, the database, and the GitHubWrapper and saves them in several .csv files.

The .csv files include *authors.list*, *bots.list*, *commits.list*, *commitMessages.list*, *issues-github.list*, and others. The term "author" in the filename *authors.list* refers to all the developers in the

1 `https://deno.land/`, last accessed on 31/03/2023
2 `https://code.visualstudio.com/`, last accessed on 31/03/2023
3 `https://tensorflow.org/`, last accessed on 31/03/2023
4 `https://mobyproject.org/`, last accessed on 31/03/2023
5 `https://getbootstrap.com/`, last accessed on 31/03/2023
6 `https://github.com/atom`, last accessed on 31/03/2023
7 `https://typescriptlang.org/`, last accessed on 31/03/2023
8 `https://react.dev/`, last accessed on 31/03/2023
9 `https://electronjs.org/`, last accessed on 31/03/2023
10 `https://nextjs.org/`, last accessed on 31/03/2023
15 `https://github.com/siemens/codeface`, last accessed on 31/03/2023
16 `https://github.com/se-sic/GitHubWrapper`, last accessed on 23/04/2023
17 `https://github.com/se-sic/codeface-extraction`, last accessed on 31/03/2023

Table 3.1: In this Table, we illustrate the OSS projects analyzed in this thesis with their number of issues. Additionally, it shows the project release date which we call project age. Finally, we provide a short description of each project.

Project Name	Number of Issues	Release Date	Short Project Description
Deno	8,764	13/05/2018	A runtime for JavaScript[11], TypeScript, and WebAssembly[12].
VS Code	111,282	29/04/2015	A source-code editor made by Microsoft[13].
TensorFlow	45,694	09/09/2015	A software library for machine learning and artificial intelligence.
Moby	41,740	13/03/2013	A framework created by Docker[14] to assemble specialized container systems.
Bootstrap	31,844	19/08/2011	A front-end development framework for the creation of websites and web apps.
Atom	21,190	25/06/2015	A text and source code editor developed by GitHub.
TypeScript	4,1267	01/10/2012	A high-level programming language developed and maintained by Microsoft.
React	20,258	29/05/2013	A front-end JavaScript library for building user interfaces based on components.
Electron	26,739	15/07/2013	A framework for building desktop GUI applications using web technologies.
Next.js	15,628	25/10/2016	A React framework that enables several extra features, including server-side rendering and generating static websites.

project regardless of their role, and not just to the authors of pull requests, as defined in Section 2.1.1. Therefore, the file *authors.list* contains information about all developers in an open-source project.

The other files contain data such as commit hashes, commit messages, event types (e.g., commented, reviewed, merged, closed), timestamps, and more. However, we do not provide an example of the exact data required for our analysis, as the specific data needed depends on the research question.

We extract data from OSS projects starting from the very beginning when the repository was initiated until approximately 2020. The exact dates for each project are listed in Table 3.2.

Table 3.2: In this Table, we show the repositories of each project that we clone to extract the data. Additionally, it shows the covered years of data that we get by mining Git repositories.

Project Name	Project Repository	Time Frame
Deno	`https://github.com/denoland/deno`	2018 - 2020
VS Code	`https://github.com/microsoft/vscode`	2015 - 2020
TensorFlow	`https://github.com/tensorflow/tensorflow`	2015 - 2020
Moby	`https://github.com/moby/moby`	2013 - 2020
Bootstrap	`https://github.com/twbs/bootstrap`	2011 - 2020
Atom	`https://github.com/atom/atom`	2012 - 2020
TypeScript	`https://github.com/microsoft/TypeScript`	2014 - 2020
React	`https://github.com/facebook/react`	2013 - 2020
Electron	`https://github.com/electron/electron`	2013 - 2020
Next.js	`https://github.com/vercel/next.js/`	2016 - 2020

3.2 TIME-SPLIT ANALYSIS

The development of OSS is a dynamic process that changes over time. New team members join, priorities shift, and technical dependencies evolve. To capture these changes, we use an approach called *time-split analysis* to analyze how the project's communication and collaboration patterns change over time. A time-split analysis is an approach that evaluates the performance of a system over time. This approach involves dividing a data set into two or more subsets based on time. Additionally, by employing a time-split analysis approach we can analyze the STMC for each pull request (since we focus on pull requests) and draw conclusions for the entire project timeline. Furthermore, as the data at the beginning of a project may not be directly comparable to data at a later point in the project, a time-split analysis allows for a more localized analysis, mitigating this risk.

We split each project's data set into six-month windows to perform the time-split analysis. We chose this time range because tests have shown that pull requests in OSS projects typically take six months from when they are created until they are merged, closed, or abandoned.

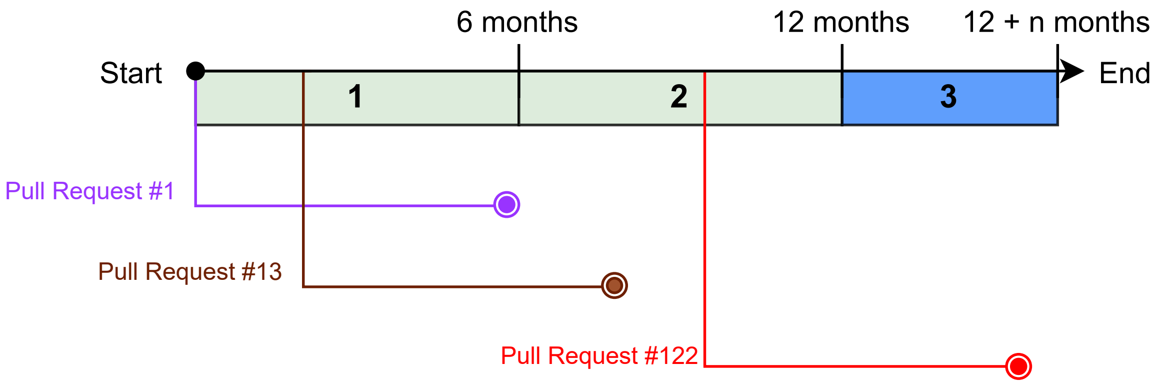


Figure 3.1: In this figure, we present an example of a time-split analysis using a six-month window size. The first two windows (in green) are complete six-month periods, while the last window (in blue) represents the remaining time until the end of our data set. We also illustrate different scenarios where pull requests start at various points in time, such as Pull Request #13, which starts in the first window but is completed (merged, closed, or abandoned) in the second window. To simplify the visualization, we depict all pull requests as having the same duration of six months

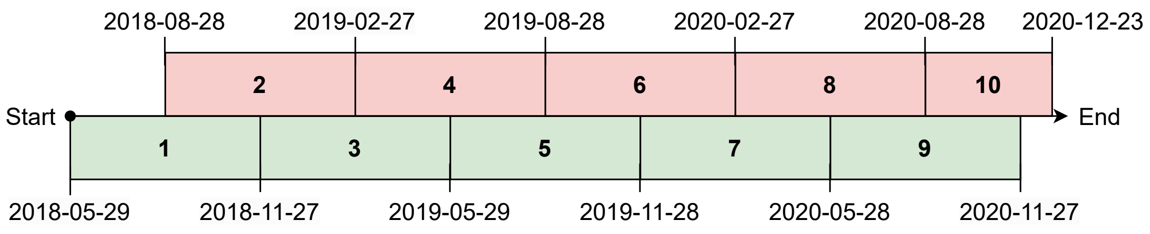


Figure 3.2: By applying a time-split analysis with a sliding window to Deno's data set, we obtain ten windows in total. The green windows with odd numbers are the result of applying a time-split approach without a sliding window, while the even-numbered windows are added by the sliding window approach. Together, they form the complete time series that we analyze for Deno.

We leave it to future work to perform a sanity check to verify that this holds true for the pull requests of the ten OSS projects that we analyze.

As we show in Figure 3.1, pull requests can start or finish every day and not only after exactly six months. Therefore, to capture these scenarios, we apply a *fixed-size sliding window* of three months. This means that the analysis is performed on a series of overlapping six-month periods that overlap with each other by three months.

In Figure 3.2, we present the results of applying a time-split analysis using a sliding window approach for the Deno project. This approach results in a different number of time windows for each project because projects have different ages. Note that while we always utilize a time-split analysis, we do not necessarily use a sliding window approach in all of our research questions. We will explain the reasoning behind this in chapter 4.

We use CORONET[18] to apply the time-split analysis with or without a sliding window. CORONET constructs analyzable networks based on data extracted from CODEFACE-EXTRACTION. This network library reads the written or extracted data from disk and constructs intermediate data structures for convenient data handling, such as data containers and developer networks. We use CORONET to split, access, and filter the data sets. Additionally, we utilize CORONET's ability to build networks, detect triangle or square motifs, and many other functionalities later in the thesis.

To split the data, we use the CORONET function called SPLIT.DATA.TIME.BASED. Below is a code snippet that shows how to split the data into six-month time windows with a sliding window of three months. We do not explicitly specify the size of the sliding window, namely three months, because it is implemented in the function by default.

```r
source("coronet/util-split.R")

split.data.time.based(project.data,             # Project data which is fetched previously.
                      time.period = "6 months",  # The size of the time window.
                      split.basis = c("issues"), # The data name to use as split basis.
                      sliding.window = TRUE      # Using a sliding window approach. We set it
                     )                           # to FALSE if we do not want to use the
                                                 # sliding window approach.
```

18 https://github.com/se-sic/coronet, last accessed 02/04/2023

4 OPERATIONALIZATION

In this chapter, we describe the networks that we can build with CORONET and show what kind of network we use. Furthermore, we configure and refine the STMC by utilizing the available data set and focusing on the socio-technical value that the chosen configuration adds to our analysis. Finally, we operationalize the research questions of this thesis.

4.1 CONFIGURING STMC

As mentioned previously, STMC utilizes socio-technical networks to detect square and triangle motifs and anti-motifs within these networks. However, configuring a triangle or square motif offers several possibilities due to the varying types of nodes, edges, and types of networks. For our STMC analysis, we use CORONET to construct the socio-technical networks and only consider the types of nodes, edges (also called relations in CORONET), and networks that CORONET offers. We also take into account the available data from CODEFACE-EXTRACTION. By combining these constraints, we select one STMC configuration that aligns best with the goal of this thesis, which is to correlate STMC with code review measures such as acceptance rate, review rate, and first review interval.

4.1.1 *Coronet Networks*

We can construct four types of networks using CORONET: *author networks*, *artifact networks*, *bipartite networks*, and *multi-networks*, which combine author, artifact, and bipartite networks. In the following, we provide more details on each type of network and illustrate a simple example for each.

- Author networks

  In an author network, each node represents a unique author that can be identified by their name. The edges in this network are unipartite, meaning they only connect authors within the same group or category. In Figure 4.1, we illustrate an example of an author network with a unipartite edge connecting two authors.

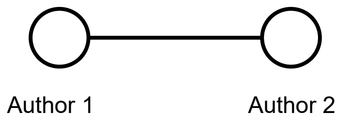


Figure 4.1: The nodes in the network represent authors, depicted as circles. The edges between these nodes are unipartite, connecting authors within the same category. This network is considered a social network as it solely consists of information related to authors.

- Artifact networks

  In an artifact network, nodes represent various types of artifacts, including source-code artifacts such as features, files, or functions, as well as communication artifacts such as mail threads or issues. Each artifact node is uniquely identifiable by its name. The edges in this network are unipartite, connecting only artifacts within the same category. In Figure 4.2, we provide an illustration of this network.

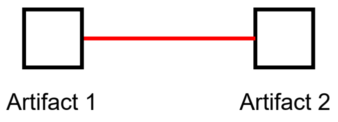


Figure 4.2: The nodes in the network represent artifacts, depicted as squares. The edges between these nodes are unipartite, connecting artifacts within the same category. This network is considered a technical network as it only contains information related to artifacts.

- Bipartite networks

  In a bipartite network, nodes represent both authors and artifacts. The edges in this network are bipartite, connecting only authors to artifacts and vice-versa. A bipartite edge connects an author node to an artifact node. In Figure 4.3, we provide a visualization of this network structure.

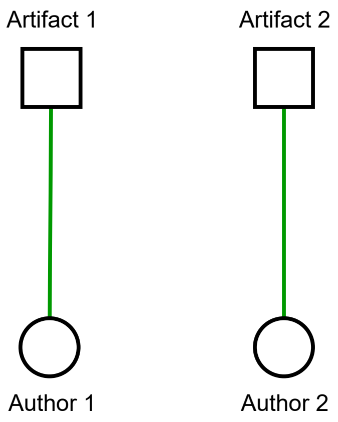


Figure 4.3: Circles are authors. Squares are artifacts. The edges are bipartite because they connect two nodes of different types, namely an author with an artifact. This is a socio-technical network because it contains information about authors and artifacts.

- Multi-networks

  The nodes in a multi-network denote both authors and artifacts. There are both unipartite and bipartite edges among the nodes in this type of network. Essentially, a multi-network is the combination of all other types of networks. We depict an example of a multi-network in Figure 4.4.

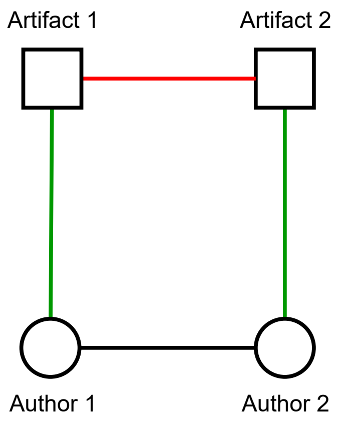


Figure 4.4: This multi-network comprises nodes representing both authors and artifacts. The nodes representing authors are denoted by circles while the nodes representing artifacts are denoted by squares. The edges in this network are unipartite - connecting two authors or two artifacts - and bipartite - connecting an author node to an artifact node. This network is classified as socio-technical as it contains information about both authors and artifacts.

To obtain a comprehensive view of socio-technical aspects in an OSS project, we utilize *multi-networks* in this thesis. With a multi-network, we can obtain information on the relationships between two authors, two artifacts, and an author and an artifact. Consequently, it provides us with a complete view of the socio-technical aspects of an OSS project.

4.1.1.1 *Nodes*

We classify the nodes in CORONET into two main categories: *author nodes* and *artifact nodes*. An author node represents a developer who makes changes to the project files. On the other hand, an artifact node can take different forms, such as features, files, functions, mail threads, and issues. In this thesis, we use *developers* as author nodes and *files* as artifact nodes. We use a file as an artifact in our configuration because it aligns with the relational structure of the overall graph, as depicted in Figure 4.5. More importantly, selecting files supports our STMC goal because they are technical artifacts in OSS projects. Files can represent a variety of artifact types, including source code, documents, configuration files, and data files, among others, and can effectively track the evolution of these artifacts over time.

4.1.1.2 *Edges*

CORONET supports four types of edges, or relations, namely *cochange*, *mail*, *issue*, and *callgraph*. The meaning of each type of relation varies depending on the type of network being analyzed. For instance, in an author network, the edges represent a relationship between two developers. In an artifact network, the edges represent a relation between two artifacts. Conversely, in a bipartite network, the edges indicate the type of contribution made by a developer. In Table A.1, we outline the meaning of *cochange*, *mail*, *issue*, and *callgraph* relations in author, artifact, and bipartite networks. We do not include multi-networks because multi-network is the result of all other networks together.

In this thesis, we use *cochange* as a relation type between two files and *issue* as a relation type between two developers. The *cochange* relation is appropriate for files because it connects two files that are changed in the same commit, which is a common occurrence in OSS projects. Additionally, it detects technical dependencies between the artifacts and it is the most suitable relation to connect two files. Regarding the *issue* relation, it is suitable for two developers because they can communicate using the comment section on an issue. Hence, we can extract the communication between two developers from an issue. Together, the cochange and issue relation provide us with information about the technical dependencies and communication that occur around them. Therefore, this configuration is best aligned with our goal, and we select it for our square motif in the STMC analysis.

4.1.2 *Configuring Triangle & Square Motifs*

Our chosen configuration for the STMC analysis consists of using file as the artifact, cochange as the artifact relation, developer as an author, and issue as the author relation. We depict this configuration in Figure 4.5 for all types of motifs and anti-motifs. This configuration allows us to identify the following patterns: (a) the triangle motif, which detects two developers who have collaborated on the same issue while contributing to the same file; (b) the triangle anti-motif, which identifies two developers who have contributed to the same file but have not collaborated on any issue; (c) the square motif, which reveals two developers who have contributed to two files that were changed in the same commit and have collaborated on an issue; and (d) the square anti-motif, which highlights two developers who have contributed to two files that were changed in the same commit but have not collaborated on any issue. In our analysis, we consider developers to have collaborated if they have *communicated* with each other through commenting, reviewing, reacting to comments, or any other type of communication in an Issue or Pull Request. Therefore, going forward, we refer to collaboration as *communication*.

4.2 REFINING STMC

The STMC approach, as proposed by Mauerer et al. [21] and configured by us, has a limitation. The limitation is that the files involved in a motif or anti-motif *may not be related* to the issue in the same motif or anti-motif. In other words, an issue is used to *extract the communication* between the developers in a motif, but the communication in an issue may also be about other files that are not part of the motif under analysis. To address this matter, we propose a new concept called STMC_r, which is essentially the same as STMC, but with an additional filter.

For each triangle motif, we begin by extracting all the issues that involve a modified file forming a triangle motif in their list of changed files. Then, we compare the extracted issues with the issue involved in the same motif, where the file is included. If we find at least one issue that matches the issue involved in the motif, the motif remains a motif. However, if none of the modified issues match the motif's issue, the motif becomes an anti-motif.

For square motifs, since they involve two files, we first obtain a list of issues that modified each file included in the motif. Using this list, we apply the same filtering approach as we did for the triangle motifs, filtering out any issues that did not change at least one of the

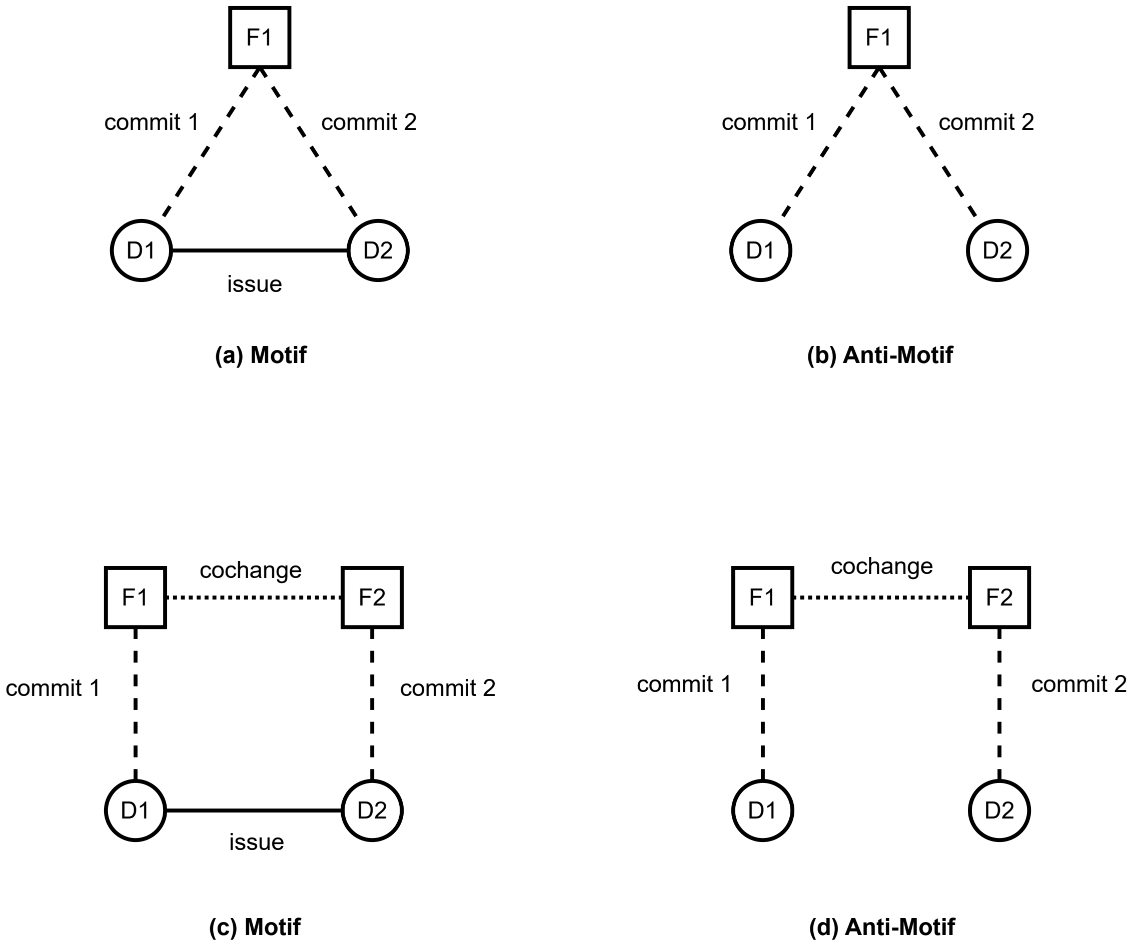


Figure 4.5: D1 and D2 are Developer 1 and Developer 2 respectively. F1 and F2 are File 1 and File 2 respectively. The solid line between the two developers means that these two developers have collaborated on the same issue. The dashed line between a developer and a file means that a developer has changed the file in a commit. The dotted line between two artifacts means that these artifacts are both changed in the same commit.

files in the motif. If there is at least one issue that changed one of the two files that form the motif, then it remains a motif. Otherwise, it becomes an anti-motif.

Furthermore, we use the same formula to calculate the degree of $STMC_r$, which we refer to as degree of refined Socio-Technical Motif Congruence ($dSTMC_r$), as the one used for dSTMC in equations 2.1 and 2.2. We use the same approach for $STMC_r$ as for STMC to get the number of motifs or anti-motifs in any OSS project that we analyze.

4.2.1 *Sanity Check*

To validate the effectiveness of our $STMC_r$ filtering approach, we conduct a qualitative study. We randomly select three triangle motifs and three square motifs from each project that are identified by STMC but not by $STMC_r$. Next, we gather all the pull request IDs that modified the files in these motifs and manually inspect each issue in GitHub to verify if the modified files are included in any of the issues. If none of the issues include the modified files, then our $STMC_r$ approach is validated as it filters out false positive motifs that the STMC approach generates.

4.3 RESEARCH QUESTIONS

In this section, we explain the research questions of this thesis. We use the dSTMC and $dSTMC_r$ concepts to measure the socio-technical congruence in our subject projects and correlate it with the acceptance rate, review rate, and first review interval. In the configuration of the STMC mentioned above, we extract the communication between developers from issues. However, it is not feasible to measure the acceptance rate, review rate, and first review interval for issues because they typically do not undergo the merging and review process. Therefore, we only measure these metrics for pull requests.

We analyze the correlation between the dSTMC or $dSTMC_r$ and the acceptance rate, review rate, and first review interval using the Kendall rank correlation coefficient (Kendall's Tau)[1]. Kendall's Tau is a non-parametric test that examines the similarities in the ordering of data when ranked by quantities. Instead of using the observations as the basis of the correlation, Kendall's Tau uses pairs of observations and determines the strength of association based on the pattern of concordance and discordance between the pairs[2]. We chose Kendall's Tau since our data is not normally distributed and we have a small dataset.

Moreover, we differentiate between statistically significant correlations and those that are not statistically significant, which is determined through the calculation of the p-value. In statistics, the p-values vary from 0 to 1, with lower values indicating higher statistical significance. A p-value of 0 suggests that the probability of obtaining the observed result by chance is extremely low and provides very strong evidence to reject the null hypothesis. The null hypothesis typically assumes that there is no correlation between the variables being analyzed. Therefore, a p-value of 0 would suggest strong evidence that there is indeed a correlation between the variables being analyzed. On the other hand, a p-value of 1

1 https://en.wikipedia.org/wiki/Kendall_rank_correlation_coefficient, last accessed on 24/04/2023
2 https://towardsdatascience.com/kendall-rank-correlation-explained-dee01d99c535, last accessed 04/05/2023

suggests that there is no statistical significance and that the observed result could easily have occurred by chance, assuming no correlation between the variables.

In this thesis, we choose a significance level of 0.05, which is commonly used and represents a 5% chance of obtaining the observed result by chance, assuming a correlation between the variables. It is important to note that the p-value alone does not indicate the strength of the correlation, only the statistical significance of the correlation coefficient. Therefore, it is important to analyze both the correlation coefficient and the p-value together to draw meaningful conclusions.

4.3.1 *Research Question 1*

RESEARCH QUESTION 1: *Is there a correlation between the dSTMC or $dSTMC_r$ with the acceptance rate of pull requests?*

In the first research question, we examine the potential correlation between the dSTMC and $dSTMC_r$ and the acceptance rate of pull requests. Specifically, we aim to investigate whether the degree of socio-technical congruence measured by both, dSTMC and $dSTMC_r$, has an impact on the acceptance rate of pull requests on GitHub. As we have already defined the measurement of dSTMC and $dSTMC_r$, we will now define the acceptance rate.

We begin by defining acceptance of a pull request as its merge into the code base of a project, based on previous research [9, 18, 31]. To analyze the correlation between socio-technical congruence and acceptance rate, we use a time-split analysis with a sliding window approach, using six-month windows with a three-month overlap. This allows us to capture more pull requests within one time frame.

To count the number of merged pull requests for each time frame, we need to address the problem of overlapping time frames. If a merge event occurs in the three-month overlap period, we may end up counting it twice. To avoid this, we count the number of issue events (such as comments, reviews, references, and others) for each time frame that a pull request spans. Since we assume a pull request typically does not last longer than six months, it may span three consecutive time frames at most. As an example of how pull request events can span across different time frames, we provide a visualization in Figure 4.6.

We consider the time frame with the most events as the time frame of the merge, because we want to capture the communication that contributed to the acceptance of the pull request. We illustrate this in Figure 4.6. In case of a tie, where two or three time frames have the same number of events, we choose the time frame closest to the merge event in terms of time, as we believe that communication closer to the merge event has a greater influence on the acceptance of the pull request.

After counting the number of merged pull requests for each time frame, we calculate the total number of pull requests for each time frame. To determine the acceptance rate, we divide the number of merged pull requests by the total number of pull requests for each time frame, as shown in Equation 4.1. We provide an example of how the acceptance rate looks for the DENO project in Figure A.1 in the Appendix.

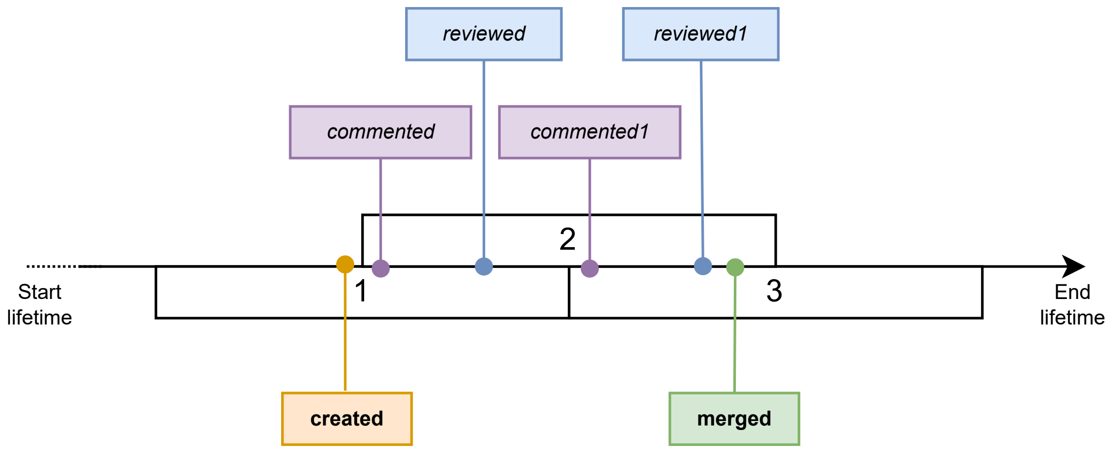


Figure 4.6: In this Figure, we depict the lifetime of a pull request spanning three consecutive time frames labeled as 1, 2, and 3. Each time frame is six months. During time frame 1, the pull request has three events: created, commented, and reviewed. During time frame 2, there are five events: commented, reviewed, commented1 (labeled as 1 to differentiate it from the previous comment), reviewed1 (labeled as 1 to differentiate it from the previous review), and merged. Finally, during time frame 3, there are three events: commented1, reviewed1, and merged. We consider time frame 2 as the time frame where the pull request was merged, as it has the highest number of events.

$$Acceptance\ Rate = \frac{number\ of\ merged\ pull\ requests}{number\ of\ pull\ requests} \tag{4.1}$$

Next, to analyze the correlation between the Acceptance Rate and dSTMC, we use Kendall's Tau to assess the:

- *Acceptance Rate* as one variable and the *triangle congruence* defined in Formula 2.1 as the other variable.
- *Acceptance Rate* as one variable and the *square congruence* defined in Formula 2.2 as the other variable.

We follow the same approach for analyzing the correlation between Acceptance Rate and dSTMC_r.

4.3.2 *Research Question 2*

RESEARCH QUESTION 2: *Is there a correlation between the dSTMC or $dSTMC_r$ with the review rate of pull requests?*

In the second research question, we investigate if there is a correlation between the dSTMC or $dSTMC_r$ and the review rate to explore possible relationships between these variables.

To measure the review rate, we first need to define what we consider a review. By review, we refer to peer code reviews conducted on GitHub. Since every OSS project has a different workflow, this also affects the methodology that the project communities choose to review a pull request. Some projects may use the "review" option on Github, while others may just leave a comment in the conversation tab in a pull request. We refer to the former methodology as a "reviewed" event and the latter as "commented" (based on the event names extracted from GitHub), but both mean that a developer who is not the author of the pull request conducted a peer code review for a pull request[3].

To determine which event(s) should be considered a review, we analyze the events "commented", "reviewed", and "merged", either alone or in combination. We present the various use cases we consider in Table 4.1, along with the implications of a high number of pull requests for each use case in every project.

The analysis of the projects' events showed a mixed result, with some projects using "commented," some using "reviewed," some using both, and some being inconclusive. Therefore, to be inclusive of all possible peer code review methodologies, *we decide to consider both "commented" and "reviewed" events*. This means that if a pull request has at least one "commented" or "reviewed" event, it indicates that the pull request is probably reviewed by someone other than the author of the pull request.

As depicted in Figure 4.7, we opt for a time-split analysis approach, utilizing a six-month window to track the evolution of socio-technical congruence over time. This method allows us to observe the general trend of peer code review activities in each project while avoiding double counting of reviewed pull requests. Additionally, since "reviewed" and "commented" events may take place numerous times throughout the lifespan of a pull request, we consider them only once per time frame for each pull request.

To calculate the rate of reviewed pull requests, we count the number of pull requests that have been reviewed and divide it by the total number of pull requests within each time frame. This approach is represented in Equation 4.2. Finally, to check if there is or is not a correlation between the Review Rate and dSTMC or $dSTMC_r$ we use Kendall's Tau.

$$Review\ Rate = \frac{number\ of\ reviewed\ pull\ requests}{number\ of\ pull\ requests} \tag{4.2}$$

3 Typically, the author of a pull request also comments on their own submission. However, we exclude these comments and focus solely on those made by other developers.

Table 4.1: In the second column of this Table, we depict the GitHub events we consider, which include "reviewed" (referring to pull requests reviewed using the GitHub review option), "commented" (about pull requests that have comments), and "merged" (about pull requests that have been merged into the project's code base). Meanwhile, the third column indicates the significance of a high number of pull requests having any of the events mentioned in the second column.

	Events	Implication
Number of Pull Requests that have the event:	reviewed	A high number of pull requests may indicate that the project utilizes the "reviewed" event for conducting peer code reviews.
	commented	A high number of pull requests may suggest that the project employs the "commented" event for performing peer code reviews.
	reviewed and commented	A high number of pull requests may imply that the project utilizes both the "reviewed" and "commented" events for conducting peer code reviews.
	merged and commented and not reviewed	A high number of pull requests may indicate that the pull requests being merged are being reviewed using "commented" rather than "reviewed". This suggests that the project relies on comments to conduct peer code reviews.
	merged and reviewed	A high number of pull requests may indicate that the pull requests being merged are being reviewed using the "reviewed" options from GitHub. This suggests that the project uses the review option from GitHub to conduct peer code reviews.

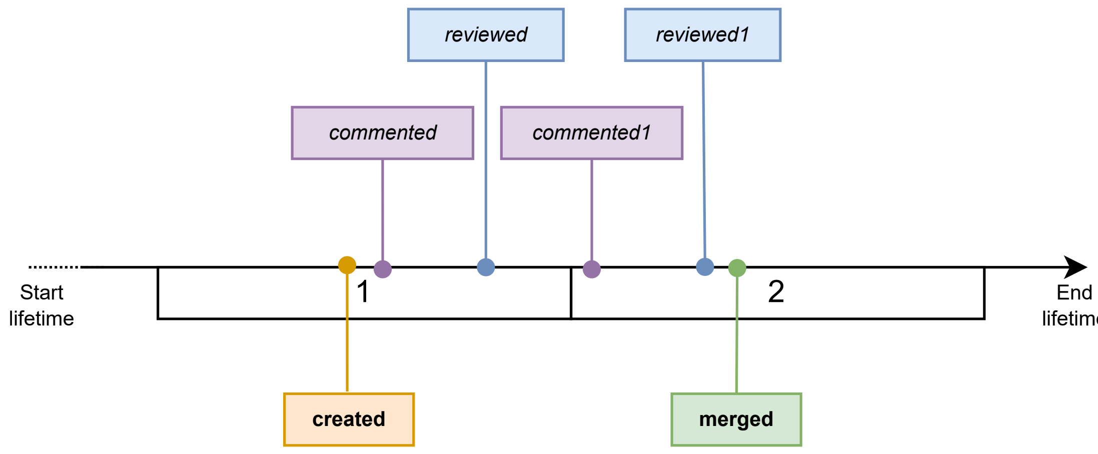


Figure 4.7: In this Figure, we illustrate the lifetime of a pull request spanning 2 consecutive time frames labeled as 1, and 2. Each time frame is six months. The "commented" and "reviewed" events are present multiple times in two different time frames.

To analyze the correlation between the Review Rate and dSTMC, we use Kendall's Tau with the:

- *Review Rate* as one variable and the *triangle congruence* defined in Formula 2.1 as the other variable.
- *Review Rate* as one variable and the *square congruence* defined in Formula 2.2 as the other variable.

We follow the same approach to analyze the correlation between the Review Rate and the dSTMC_r.

4.3.3 *Research Question 3*

RESEARCH QUESTION 3: *Is there a correlation between the dSTMC or dSTMC$_r$ with the first review interval?*

The aim of this research question is to investigate whether there is a correlation between the dSTMC or dSTMC$_r$ and the first review interval. The first review interval is defined as the time elapsed between the creation of a pull request and its initial review. Although a pull request may undergo multiple reviews, we focus only on the first review.

We define the "created" event from our data set as the beginning of a pull request. To determine the time interval from the creation of a pull request until the first review, we use the time stamp of the first review event which can either be the "reviewed" or "commented" event as described in Section (4.3.2). If only one of these events is present, we calculate the time from the creation of the pull request until the occurrence of the event. However, in the case where both events "reviewed" and "commented" are present, we consider the time from the creation of the pull request until the first event occurrence. We illustrate these cases in Figure 4.8.

Similarly to research question 1, we employ a time-split analysis with a sliding window for each OSS project. This approach involves creating six-month windows that overlap with three months to mitigate abrupt changes between two windows and enables capturing more pull requests within one window. The sliding window technique allows us to capture a higher number of event tuples, such as `["created", "commented"]` or `["created", "reviewed"]`, in comparison to a time-split analysis without a sliding window. In Figure 4.9 we provide an illustration of this approach.

To measure the first review interval for each time frame, and as a consequence for the whole OSS project, we calculate the mean of all first review intervals encountered in one window. The formula for each time frame is as shown in equation 4.3. Afterward, we use Kendall's Tau, a non-parametric statistical test, to assess if there is a correlation between the First Review Interval and dSTMC or dSTMC$_r$.

$$First\ Review\ Interval = mean(first\ review\ interval\ for\ each\ pull\ request) \tag{4.3}$$

To investigate the correlation between the First Review Interval and the dSTMC, we use Kendall's Tau with the following pairs of variables:

- *First Review Interval* as one variable and the *triangle congruence* defined in Formula 2.1 as the other variable.

- *First Review Interval* as one variable and the *square congruence* defined in Formula 2.2 as the other variable.

We follow the same approach to analyze the correlation between the First Review Interval and the dSTMC$_r$.

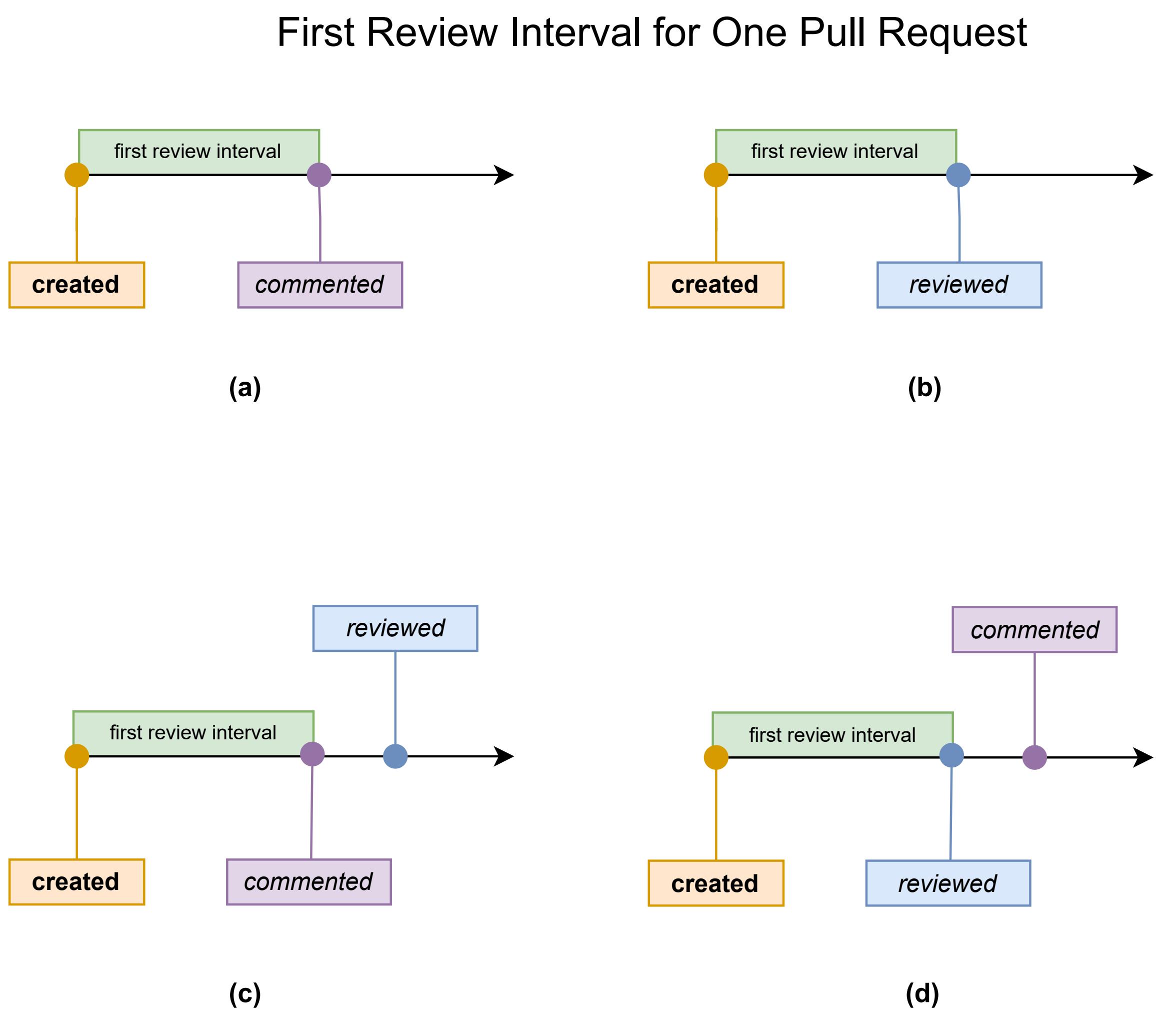


Figure 4.8: Since the pull request is reviewed only by comments in Figure (a), we consider it as the review event and calculate the first review interval for this pull request starting from the "created" event until the "commented" event. In Figure (b), since the pull request is reviewed only by the review functionality in GitHub, we consider it as the review event and calculate the first review interval for this pull request starting from the "created" event until the "reviewed" event. In Figure (c), the pull request is reviewed by comments first and then the review functionality in GitHub, hence we consider the "commented" event as the review since it occurs first. Thereafter, we calculate the first review interval for this pull request starting from the "created" event until the "commented" event. Finally, in Figure (d), the pull request is reviewed using the review functionality in GitHub first and then using comments, hence we consider the "reviewed" event as the review since it occurs first. Subsequently, we calculate the first review interval for this pull request starting from the "created" event until the "reviewed" event.

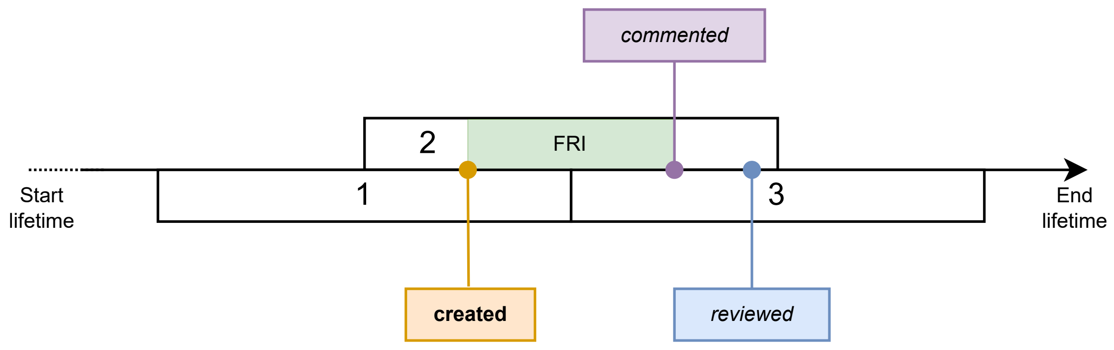


Figure 4.9: The number labels 1, 2, and 3 correspond to the number of windows generated in a time frame analysis with a sliding window. The yellow bar represents the "created" event, which marks the beginning of a pull request. The purple and blue bars represent the "commented" and "reviewed" events, respectively, which are the two events that indicate a review. The first review interval (FRI) is the time between the "created" and "commented" events.

5 EVALUATION

In this chapter, we present the results of this thesis to answer the research questions presented in Section 4.3. Furthermore, we discuss the results and what implications they may have for a project. Lastly, we explain the threats to the validity of this thesis.

5.1 RESULTS

In this section, we present the results of our analyses to answer our research questions. We utilize Kendall's Tau correlation coefficients and the respective p-values to analyze the results. The Kendall's Tau correlation coefficient ranges from -1 to 1, with a value of 0 indicating no correlation. Values closer to -1 or 1 indicate stronger negative or positive correlations, respectively. A negative correlation coefficient means that the two analyzed variables develop in opposite directions, while a positive correlation coefficient means that the two analyzed variables develop in the same direction. We interpret Kendall's Tau correlation coefficient based on the guidelines[1], as illustrated in Figure 5.1.

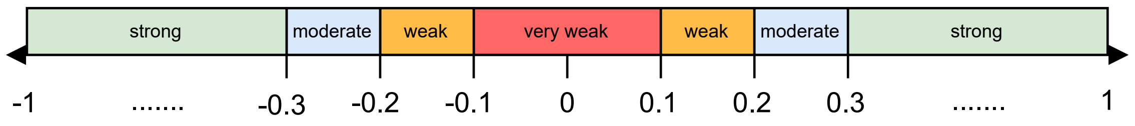


Figure 5.1: A correlation coefficient between -0.1 and 0.1, shown in red on the scale, is considered a *very weak* correlation. When the correlation coefficient falls between -0.1 to -0.19 or 0.1 to 0.19, shown in orange, the correlation is considered *weak*. A correlation coefficient between -0.2 to -0.29 and 0.2 to 0.29, shown in blue, is considered to have a *moderate* correlation. Finally, a correlation coefficient smaller than -0.3 and greater than 0.3, depicted in green on the scale, is considered to have a *strong* correlation.

In the following sections, we present the outcomes of our analyses for each of the OSS projects that we examined in this thesis, addressing all three research questions outlined in Section 4.3. We classify our answers to these research questions as affirmative, negative, or inconclusive.

To simplify the presentation of our results, we use only the names of the congruences when referring to the correlations between the acceptance rate, review rate, and first review interval and the congruences, as follows:

- Triangle = correlating acceptance rate, review rate, or first review interval with *triangle* congruence of dSTMC
- Square = correlating acceptance rate, review rate, or first review interval with *square* congruence of dSTMC

1 https://polisci.usca.edu/apls301/Text/Chapter%2012.%20Significance%20and%20Measures%20of%20Association.htm, last accessed on 04/05/2023

- Refined Triangle = correlating acceptance rate, review rate, or first review interval with *triangle* congruence of dSTMC_r
- Refined Square = correlating acceptance rate, review rate, or first review interval with *square* congruence of dSTMC_r

5.1.1 *Sanity Check Results*

In Section 4.2.1, we presented the results of our sanity check for dSTMC_r. Our results showed that, except for the MOBY project, none of the issues we examined involved communication between authors or modifications to files included in the motifs for all three randomly selected motifs. For MOBY, none of the issues we examined involved communication between authors or modifications to files included in the square motifs, but multiple issues belonging to two different triangle motifs involved communication between authors or modifications to files included in these two triangle motifs. Considering the results of the sanity check, we can imply that the dSTMC_r approach was successful in filtering out false positive motifs. The implications of this finding on our results will be discussed in the next section, namely the Discussion section 5.2.

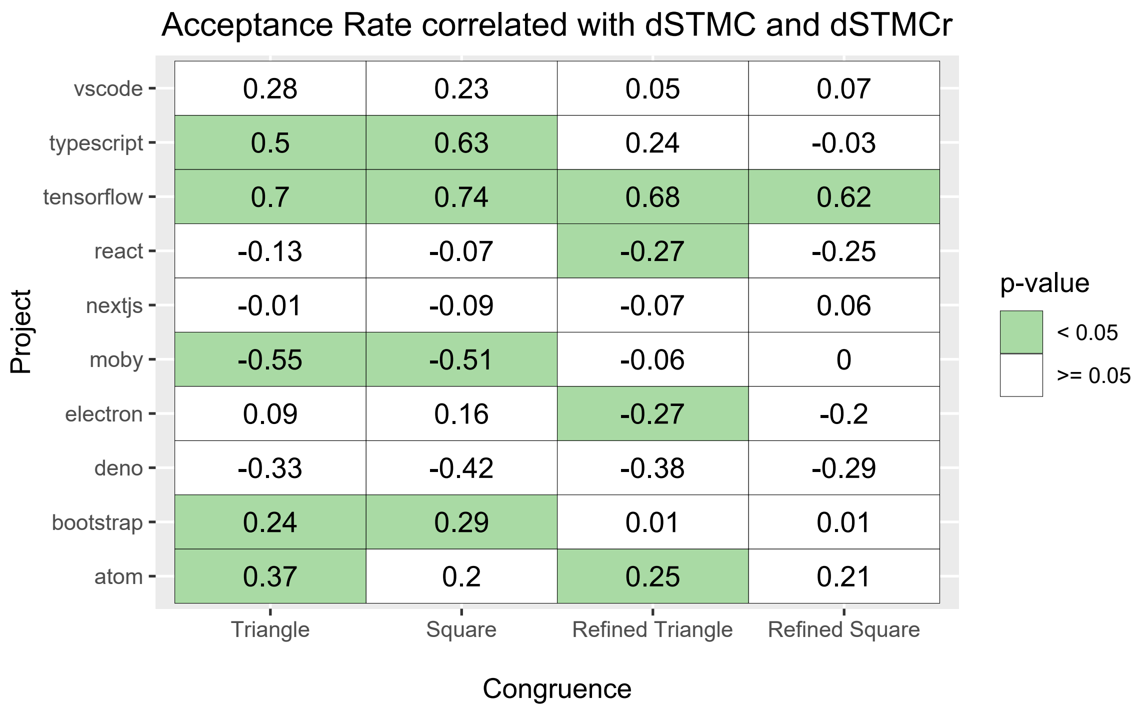
5.1.2 *Research Question 1*

In Figure 5.2, we illustrate in the heatmap the results for Triangle, Square, Refined Triangle, and Refined Square for all of our subject projects. In each cell of the heatmap, we represent the correlation coefficient between two variables, and the color of each cell indicates whether the correlation is statistically significant or not. The values in each cell provide information about the strength and direction of the correlation. This visualization allows us to quickly identify which projects have significant correlations and which ones do not, providing a comprehensive overview of our analysis results.

5.1.2.1 *Acceptance Rate and dSTMC*

In Figure 5.2, we demonstrate in the Triangle column a range of correlation coefficients between -0.55 to 0.7. However, only half of the projects analyzed (i.e., five out of ten) show statistically significant results: TENSORFLOW, MOBY, BOOTSTRAP, ATOM, and TYPESCRIPT. Among them, TENSORFLOW, TYPESCRIPT, and ATOM exhibit strong positive correlation coefficients of 0.7, 0.5, and 0.37, respectively. BOOTSTRAP shows a moderate positive correlation coefficient of 0.24. In contrast, MOBY shows a strong negative correlation coefficient. Hence, for these projects, we can accept the hypothesis that there is a statistically significant correlation between acceptance rate and triangle congruence.

On the other hand, the other five projects do not demonstrate statistically significant correlations for the triangle congruence. For instance, VS CODE exhibits a moderate positive correlation coefficient of 0.28 and a p-value of 0.08, indicating a moderate positive correlation between the acceptance rate and triangle congruence. However, this correlation is not statistically significant at the 5% significance level. Therefore, for these projects where the correlation coefficient is not statistically significant, we cannot accept the hypothesis that there is a correlation between acceptance rate and triangle congruence. Consequently, we



Project	Triangle	Square	Refined Triangle	Refined Square
vscode	0.28	0.23	0.05	0.07
typescript	🟩 0.5	🟩 0.63	0.24	-0.03
tensorflow	🟩 0.7	🟩 0.74	🟩 0.68	🟩 0.62
react	-0.13	-0.07	🟩 -0.27	-0.25
nextjs	-0.01	-0.09	-0.07	0.06
moby	🟩 -0.55	🟩 -0.51	-0.06	0
electron	0.09	0.16	🟩 -0.27	-0.2
deno	-0.33	-0.42	-0.38	-0.29
bootstrap	🟩 0.24	🟩 0.29	0.01	0.01
atom	🟩 0.37	0.2	🟩 0.25	0.21

Figure 5.2: In this heatmap, we represent the correlation coefficients for each project and congruence analyzed, where the y-axis denotes the subject projects, and the x-axis shows the congruences. Each cell in the heatmap corresponds to the correlation coefficient between the review rate and congruence. The color of the cell indicates whether the correlation coefficient is statistically significant or not. Green indicates that the p-value is less than 0.05, while white indicates that the p-value is greater than or equal to 0.05. For instance, considering the first column (Triangle) and the first row (Atom), the first cell represents a statistically significant correlation between the acceptance rate and the triangle congruence (for dSTMC) for the Atom project, with a correlation coefficient of 0.37.

cannot conclude that there is a statistically significant correlation between acceptance rate and triangle congruence.

In the case of the Square column, we observe correlation coefficients ranging from -0.51 to 0.74. Among the ten analyzed projects, four of them exhibit statistically significant correlation coefficients for the acceptance rate and square congruence: TensorFlow, Moby, Bootstrap, and Typescript. Similarly to the Triangle column, TensorFlow, Typescript, and Moby display strong correlation coefficients, while Bootstrap has a moderate correlation coefficient. For these projects, we can conclude that there is strong enough evidence to support the hypothesis that there is a statistically significant correlation between the acceptance rate and the square congruence.

However, for the remaining six projects, we do not have sufficient evidence to establish a statistically significant correlation between the acceptance rate and the square congruence. For instance, the project Deno has a negative correlation coefficient (-0.42), indicating a negative relationship between the acceptance rate and the square congruence measure, but this correlation is not statistically significant at the 5% significance level (p-value: 0.09).

We cannot give a definitive answer to this part of the first research question due to the mixed results obtained for both triangle and square congruence. While some projects exhibit a statistically significant correlation, others do not. Therefore, we *cannot conclude whether there is a correlation between acceptance rate and the dSTMC.*

5.1.2.2 *Acceptance Rate and dSTMC$_r$*

We illustrate the correlation results between acceptance rate and dSTMC$_r$, specifically the triangle and square congruences, on the second pair of columns in the heatmap.

The correlation coefficients for the Refined Triangle range from -0.38 to 0.68 considering all projects. Only four out of ten projects show a statistically significant correlation: TensorFlow, Atom, React, and Electron. TensorFlow shows the highest correlation (0.68), followed by Atom (0.25). Meanwhile, Electron and React show the same negative correlation coefficient, namely -0.27. The other five projects do not show any statistically significant correlation between the acceptance rate and the triangle congruence.

Regarding the Refined Square column, in Figure 5.2, the range of the correlation coefficients is from -0.29 to 0.62. In this case only TensorFlow shows a statistically significant correlation with a strong correlation coefficient of 0.62 and a p-value of 0 which suggests that there is a very low probability that the observed correlation occurred by chance. The React project shows a negative correlation with a correlation coefficient of -0.25 and a p-value of 0.05, indicating a weak negative association between acceptance rate and square congruence. However, this correlation is not statistically significant as the p-value is greater than the threshold of 0.05. Also, all the other projects show no statistically significant correlations in this case.

Our analysis of the correlation between the acceptance rate and the Refined Triangle and Square congruence yields mixed results. While only one out of ten projects show a statistically significant correlation with the refined square metric, four out of ten projects show a statistically significant correlation with the refined triangle metric. As a result, *we cannot draw a conclusion regarding the correlation between the acceptance rate and the $dSTMC_r$ metrics*.

To summarize, our analysis of the first research question did not provide a conclusive answer for either use case - when correlating acceptance rate with dSTMC or $dSTMC_r$.

We answer the first research question *inconclusively* because we cannot determine whether there is a correlation between the acceptance rate and the dSTMC or $dSTMC_r$.

5.1.3 *Research Question 2*

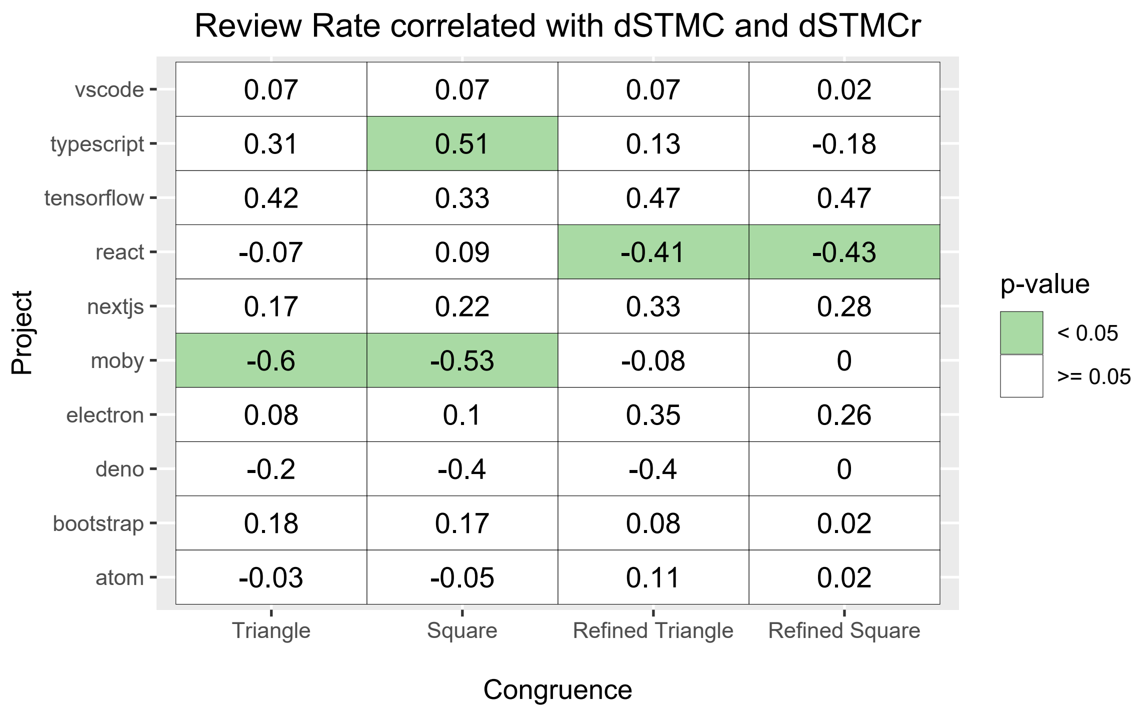
We illustrate the correlation between the review rate and both dSTMC and $dSTMC_r$ for each project in Figure 5.3.

5.1.3.1 *Review Rate and dSTMC*

The correlation coefficients for the Triangle column range from -0.6 to 0.42. Only one project, Moby, shows a statistically significant negative correlation (-0.6) with a p-value of 0, indicating that the review rate and triangle congruence tend to develop in opposite directions for this project. Although TypeScript and TensorFlow have the highest correlation coefficients (0.31 and 0.42, respectively), their p-values are greater than 0.05, indicating no statistically significant correlation. The remaining projects show very weak, weak, or moderate correlation coefficients and none of them are statistically significant (p-values > 0.05). Therefore, we cannot support the hypothesis that there is a correlation between these projects.

Similarly, in the Square column, the correlation coefficients range from -0.53 to 0.51. The only projects with statistically significant correlations are Moby and TypeScript (with p-values of 0 and 0.01, respectively) suggesting a strong and statistically significant correlation between review rate and square congruence. The remaining projects show very weak, weak, moderate, or strong correlation coefficients, but none of them are statistically significant.

To summarize, only Moby shows statistically significant correlations for both the triangle and square congruences, while TypeScript displays a statistically significant result solely for square congruence. For the remaining projects, there are no statistically significant findings in either metric. With 9 out of 10 projects for Triangle and 8 out of 10 for Square not showing statistically significant correlations, we *conclude that there is no significant correlation* between review rate and the dSTMC.



Project	Triangle	Square	Refined Triangle	Refined Square
vscode	0.07	0.07	0.07	0.02
typescript	0.31	🟩 0.51	0.13	-0.18
tensorflow	0.42	0.33	0.47	0.47
react	-0.07	0.09	🟩 -0.41	🟩 -0.43
nextjs	0.17	0.22	0.33	0.28
moby	🟩 -0.6	🟩 -0.53	-0.08	0
electron	0.08	0.1	0.35	0.26
deno	-0.2	-0.4	-0.4	0
bootstrap	0.18	0.17	0.08	0.02
atom	-0.03	-0.05	0.11	0.02

Figure 5.3: We represent the correlation coefficients for each project and congruence analyzed in the heatmap. The y-axis denotes the subject projects, while the x-axis shows the congruences. Each cell in the heatmap shows the correlation coefficient between the review rate and congruence. The color of the cell indicates whether the correlation coefficient is statistically significant or not. Green represents a p-value of less than 0.05, while white represents a p-value of greater than or equal to 0.05. For example, in the first column (Triangle) and the first row (Atom), the first cell represents a not statistically significant correlation between the review rate and the triangle congruence (for dSTMC) for the Atom project, with a correlation coefficient of -0.03.

5.1.3.2 *Review Rate and dSTMC$_r$*

In the Refined Triangle column of the heatmap, we observe that three projects, namely Deno, Moby, and React, exhibit negative correlations with coefficients ranging from -0.41 to -0.08. Conversely, the rest of the projects, including TensorFlow, Atom, TypeScript, Electron, Next.js, Vs Code and Bootstrap show positive correlations with coefficients ranging from 0.7 to 0.47. However, only the correlation for React is statistically significant with a p-value of 0.03, while the other projects show no statistically significant results.

Looking at the Refined Square column, six projects (Vs Code, TensorFlow, Bootstrap, Atom, Electron, Next.js) have a positive correlation (coefficients ranging from 0.02 to 0.47), while two projects (TypeScript and React) have a negative correlation (coefficients ranging from -0.43 to -0.18). Surprisingly, both Deno and Moby have a zero correlation coefficient and a p-value of 1. This suggests that there is no linear relationship between the review rate and square congruence, but the p-value of 1 indicates that this lack of correlation is not statistically significant. In other words, the observed correlation coefficient of zero could have easily happened by chance. Nevertheless, React is the only project that shows a statistically significant correlation, whereas the correlation between the review rate and the square congruence is not statistically significant for the remaining 9 projects.

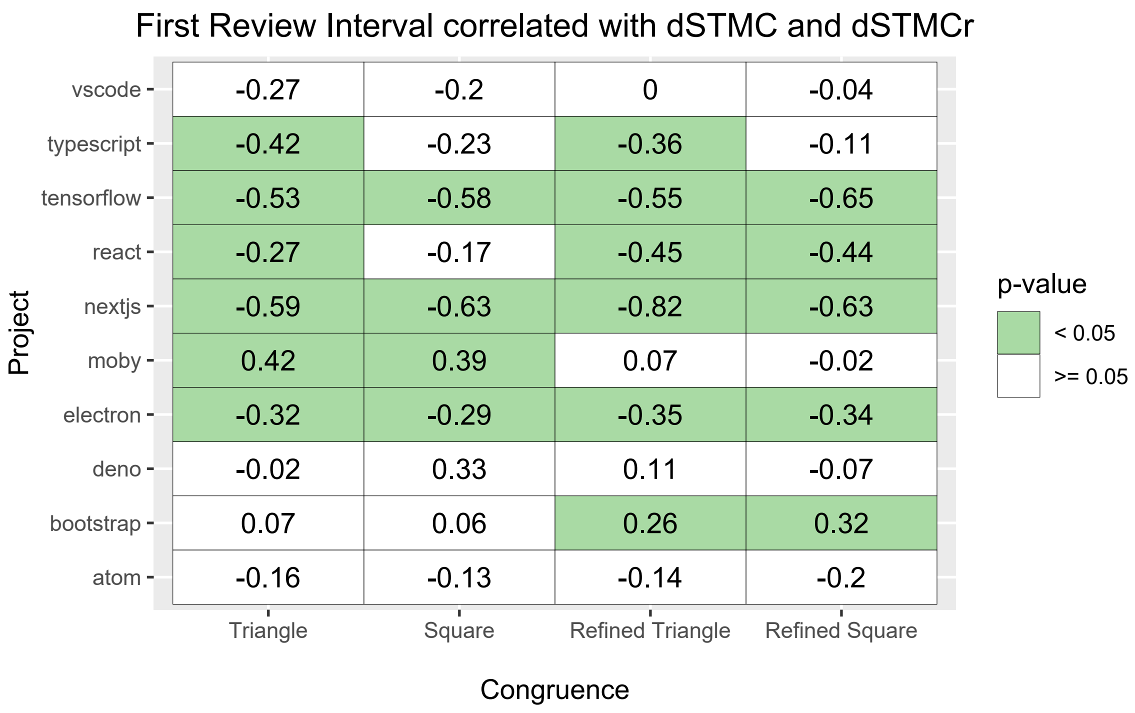
Overall, the results suggest that the correlation between the review rate and the triangle or square congruence varies among the analyzed OSS projects, with only one project demonstrating a statistically significant correlation for triangle and square congruence. Therefore, we *conclude that there is no statistically significant correlation* between the review rate and the dSTMC$_r$.

Based on the analysis of the correlation between the review rate and the four congruences, we find that only a small number of projects exhibit statistically significant correlations. Specifically, the Triangle, Refined Triangle, and Refined Square congruences show that only one project has a statistically significant correlation, while the Square congruence shows that two projects have statistically significant correlations.

Hence, we conclude that there is *no correlation between review rate and either dSTMC or dSTMC$_r$*, as only one or two projects out of 10 exhibit statistically significant correlations for each of the four congruences analyzed.

5.1.4 *Research Question 3*

In Figure 5.4, we depict the correlation between the first review interval and both dSTMC and dSTMC$_r$ for each OSS project that we analyzed.



Project	Triangle	Square	Refined Triangle	Refined Square
vscode	-0.27	-0.2	0	-0.04
typescript	🟩 -0.42	-0.23	🟩 -0.36	-0.11
tensorflow	🟩 -0.53	🟩 -0.58	🟩 -0.55	🟩 -0.65
react	🟩 -0.27	-0.17	🟩 -0.45	🟩 -0.44
nextjs	🟩 -0.59	🟩 -0.63	🟩 -0.82	🟩 -0.63
moby	🟩 0.42	🟩 0.39	0.07	-0.02
electron	🟩 -0.32	🟩 -0.29	🟩 -0.35	🟩 -0.34
deno	-0.02	0.33	0.11	-0.07
bootstrap	0.07	0.06	🟩 0.26	🟩 0.32
atom	-0.16	-0.13	-0.14	-0.2

Figure 5.4: We present the correlation coefficients for each project and congruence in the heatmap. We use the y-axis to denote the subject projects, while the x-axis displays the congruences. Each cell in the heatmap represents the correlation coefficient between the first review interval and a congruence. We use the color of the cell to indicate whether the correlation coefficient is statistically significant or not. Green represents a p-value of less than 0.05 and white represents a p-value of greater than or equal to 0.05. For instance, in the first column (Triangle) and the first row (Atom), we see a non-statistically significant correlation between the first review interval and the triangle congruence (for dSTMC) for the Atom project, with a correlation coefficient of -0.16.

5.1.4.1 *First Review Interval and dSTMC*

We find that the correlation coefficients for the first review interval and the triangle congruence range from -0.59 to 0.42, which are both considered strong correlation coefficients according to the guidelines used in this thesis. Interestingly, six out of ten projects show statistically significant correlations. These projects are: ELECTRON, MOBY, NEXT.JS, TENSORFLOW, REACT, and TYPESCRIPT. All of these projects, except REACT which shows a moderate statistically significant correlation (-0.27), show strong statistically significant correlations. Meanwhile, the other four projects show very weak, weak, or moderate correlations that are not statistically significant.

For the Square column in the heatmap, we find that only four out of ten projects show a statistically significant correlation. These projects are: ELECTRON, MOBY, NEXT.JS, and TENSORFLOW. MOBY, NEXT.JS, and TENSORFLOW show strong correlation coefficients, 0.39, -0.63, and -0.58 respectively. Only ELECTRON shows a moderate correlation coefficient with -0.29. In contrast, all the other projects are not statistically significant.

As we obtain mixed results for both triangle and square congruence, we cannot give a definitive answer to this part of the third research question. While some projects exhibit a statistically significant correlation, others do not. Therefore, we *cannot conclude whether there is a correlation between the first review interval and the dSTMC*.

5.1.4.2 *First Review Interval and dSTMC$_r$*

We observe a range of correlation coefficients varying from -0.82 to 0.26 in the Refined Triangle column. Similarly to the Triangle column, six projects show a statistically significant correlation. These projects include BOOTSTRAP, ELECTRON, NEXT.JS, REACT, TENSORFLOW, and TYPESCRIPT. Among these, only BOOTSTRAP exhibits a moderate statistically significant correlation. The remaining five projects show strong correlation coefficients, with NEXT.JS having the highest correlation coefficient of -0.82, not only in the Refined Triangle column but among all correlation coefficients of our findings. Conversely, the other four projects exhibit very weak or weak correlations that are not statistically significant.

Moving on to the Refined Square column, the correlation coefficients vary from -0.65 to 0.32. Out of the ten projects, half of them show statistically significant correlations. These projects are BOOTSTRAP, ELECTRON, NEXT.JS, REACT, and TENSORFLOW with 0.32, -0.34, -0.63, -0.44, -0.65 correlation coefficients respectively. The remaining half of the projects show no statistically significant correlation and their correlation coefficients are very weak, weak, or moderate.

We *cannot provide a conclusive answer* to whether there exists a correlation between the first review interval and the dSTMC$_r$ due to the mixed results obtained for both triangle and square congruence.

In summary, our analysis of all congruences correlated to the first review interval shows statistically significant correlations for 4 to 6 projects, depending on the type of congruence. However, these results are not conclusive enough to determine whether there is a correlation between the first review interval and the dSTMC or dSTMC_r.

Thus, we *answer our third research question inconclusively* meaning that we cannot show if there is or is not a correlation between the first review interval and the dSTMC or dSTMC_r.

5.2 DISCUSSION

In this section, we discuss our findings from the results section 5.1 by explaining what the correlation coefficients mean and what might have caused them.

We noticed that the correlation coefficients for all our research questions were sometimes positive and sometimes negative among all projects. A positive correlation coefficient indicates that the code review measure (acceptance rate, review rate, or first review interval) and dSTMC or dSTMC_r (we refer to dSTMC and dSTMC_r together as *congruence*) develop in the same direction. Meanwhile, a negative correlation coefficient indicates that the code review measure and the congruence develop in opposite directions. However, it is important to note that correlation does not necessarily imply causation. For example, a high dSTMC does not necessarily and exclusively cause a high acceptance rate as other factors might affect the acceptance rate.

In our analysis, we consider the code review measure as the dependent variable and the congruence as the independent variable. We regard a high congruence (regardless if it's a square or triangle congruence) as suggesting that developers contributing to the same files or related files often communicate with each other (supposedly) about their contributions. On the other hand, we view a low congruence as implying that developers are contributing to the same files or related files but not communicating with each other about their contributions.

5.2.1 *Comparing dSTMC and dSTMCr*

Based on our sanity check, we observed that in most cases, by using dSTMC_r we effectively filtered out the false positive motifs that we detected by the dSTMC method. Consequently, the triangle or square congruence of the dSTMC_r is either lower than or equal to that of the dSTMC. We also noticed that some projects exhibit statistically significant correlations for both dSTMC and dSTMC_r. Moreover, the correlation coefficients for these projects either increase, decrease, or remain unchanged.

Interestingly, certain projects exhibit statistically significant correlations for dSTMC but not for dSTMC_r, and in such cases, the correlation coefficients for dSTMC_r are always lower. On the other hand, some projects do not show a statistically significant correlation for dSTMC, but they do show one for dSTMC_r, and in these instances, the correlation coefficient for dSTMC_r is always higher. Based on these observations we can point that *in these two cases*, the correlation between the code review measure and the dSTMC or dSTMC_r tends to be lower for projects with non-statistically significant correlations.

Furthermore, based on our sanity check results, we demonstrated that almost all motifs we manually checked were false positives hence, can argue that *dSTMC$_r$ is a more reliable measure than dSTMC*. However, we cannot dismiss the results obtained from dSTMC entirely since our sanity check only involved randomly selecting three motifs per project. Further investigations are necessary to confirm this finding.

Lastly, we can argue that the results obtained from projects that exhibited a statistically significant correlation with both dSTMC and dSTMC$_r$ are more reliable than those that did not. These projects include Atom (only triangle congruence) for the first research question, TensorFlow for the first and third research questions, and TypeScript (only triangle congruence), React (only triangle congruence), Next.js, and Electron for the third research question.

5.2.2 *Research Question 1*

The first research question that we investigated is the correlation between the acceptance rate of pull requests and congruence. We found that there is a statistically significant correlation for some projects, but not for others.

Below, we discuss the implications or the causes that may have caused the positive or negative correlation coefficients.

1. Positive Correlation Coefficient: Both acceptance rate and congruence *increase*

   Communication, in the context of STMC, in this scenario may lead to fewer conflicts when changing the code, resulting in the faster merging of pull requests and an increased acceptance rate. Another reason might also be clearer expectations of what is required for a pull request to be accepted. Improved team motivation when developers work closely together may contribute to a high acceptance rate.

2. Positive Correlation Coefficient: Both acceptance rate and congruence *decrease*

   This lack of communication may cause conflicts in the code base, resulting in delays in merging pull requests and a decreased acceptance rate.

3. Negative Correlation Coefficient: acceptance rate *increases* and congruence *decreases*

   This scenario may be due to developers becoming more familiar with the code base and each other's work over time, leading to less communication. Additionally, a lack of code reviews may lead to an increase in the acceptance rate as pull requests are merged without being reviewed.

4. Negative Correlation Coefficient: acceptance rate *decreases* and congruence *increases*

   This scenario may be due to low-quality contributions that are not relevant to the code base. Alternatively, pull requests may be scrutinized very thoroughly, causing a lot of communication for one pull request but fewer pull requests being merged overall.

Most of the projects that showed a statistically significant correlation (Atom, Bootstrap, TensorFlow, and TypeScript) have moderate to strong *positive* correlation coefficients because when developers communicate and collaborate (i.e., high congruence), it is more likely that pull requests are merged quickly, leading to a high acceptance rate. On the other hand, in projects where the developers do not communicate and collaborate, the acceptance rate is likely to be lower, and the congruence is likely to be lower as well.

In contrast, in the rest of the projects that showed statistically significant correlations such as Moby, Electron, and React, there is a moderate to strong *negative* correlation between the acceptance rate and congruence. Since Moby is the project that shows the strongest negative correlation, we investigated the possible reasons for this result. We found that the contributing guidelines for this project strongly encourage getting in touch with the project maintainers before submitting a pull request[2] as they value communication a lot. Their reviewing policy also gives great importance to comments in pull requests. In addition, the acceptance rate of OSS projects usually decreases over time because, usually, at the beginning of a project, only the core developers are involved, and pull requests may be merged without proper review. As the project grows and more contributors join, the number of pull requests increases, which may lead to longer review times and lower acceptance rates.

Taking these factors into account, we believe that the negative correlation coefficient in the case of Moby project may be due to a decrease in the acceptance rate as the project becomes more complex and the number of pull requests increases, despite the efforts to maintain communication and review practices.

To sum up, our analysis suggests that the STMC has an impact on the acceptance rate in a few projects with a statistically significant correlation. For other projects without a significant correlation, several factors, such as the author's past contributions, code base size, and code review frequency, may affect the acceptance rate. Nevertheless, we cannot conclude that the STMC has a statistically significant correlation with the acceptance rate or *causes* the acceptance rate for either group of projects (those that show statistically significant correlations and those that do not).

5.2.3 *Research Question 2*

The second research question that we investigated is the correlation between the Review Rate of pull requests and the congruence. We found that for most of the projects, there is no statistically significant correlation between the review rate and congruence.

For this research question, we found that only three projects demonstrated statistically significant correlations: Moby, React, and TypeScript. Interestingly, Moby showed statistically significant correlations for both Triangle and Square congruence, while TypeScript showed statistically significant correlations for Refined Triangle and Refined Square (see Figure 5.3). Additionally, both projects showed strong *negative* correlations, indicating that direct and indirect collaboration impact the review rate. There could be several reasons for these negative correlations, such as a decrease in communication when developers review a high volume of pull requests or a perception that less communication is necessary when working on related files. However, it's important to note that while a negative correlation between review rate and congruence suggests an inverse relationship, other factors may independently influence these variables.

2 `https://github.com/moby/moby/blob/master/CONTRIBUTING.md#successful-changes`, last accessed 28/04/2023

For most of the projects analyzed, there was no significant correlation observed between review rate and congruence, indicating that the correlation may have occurred by chance. Factors such as project size, complexity, level of engagement within the community, and quality of documentation could influence the review rate. Larger projects or those with a more complex architecture may require more effort to review, possibly resulting in a lower review rate. A highly engaged community can lead to more contributors willing to review pull requests, probably leading to a higher review rate. The high quality and completeness of the project's documentation can also play a role in the time needed for reviewing pull requests, possibly leading to a higher review rate.

Our research indicated that there is no strong evidence of a relationship between the review rate and STMC for the majority of the projects analyzed. This suggests that other factors are likely to be influencing the review rate. Therefore, further investigation is needed to determine which factors are most influential and how they affect the review rate.

5.2.4 *Research Question 3*

Our third research question focused on the correlation between the first review interval and congruence. We observed a statistically significant correlation for certain projects, but not for others, and as a result, we were unable to provide a definitive answer.

Among the projects that exhibited a statistically significant correlation, most of them had negative correlation coefficients, including Electron, Next.js, React, TensorFlow, and TypeScript. In these cases, the negative correlation between the first review interval and congruence indicated that when the first review interval was longer, the congruence was lower, and vice versa. Longer first review intervals and lower congruence could arise from an ineffective review process where pull requests are not prioritized. Additionally, this might also stem as a consequence of poor documentation where the code review process is not clearly stated.

Conversely, a shorter first review interval and higher congruence might be linked to the skill or experience level of the developers. Experienced developers can review more pull requests, reducing the first review interval for a significant number of pull requests. Similarly to the above, documentation can also shorten the first review interval but increase the communication needed afterward.

Interestingly, two projects (Bootstrap and Moby) showed a *positive* statistically significant correlation coefficient, indicating that as the first review intervals increased or decreased, so did the congruence, meaning more or less communication about a file or related files. This could be attributed to a not effective review process, leaving developers waiting for their reviews for an extended period and causing more communication after the review process has begun. A well-established review process, on the other hand, could have the opposite effect, as developers prioritize the pull requests for review, necessitating less communication. One possibility is that longer first review intervals give developers more time to communicate and coordinate their efforts, resulting in higher levels of congruence. This would be consistent with the Moby project's contribution guidelines, which state: *"...make the effort to coordinate with the maintainers of the project before submitting a pull request."*

For the projects that did not exhibit statistically significant correlations, the first review interval can be influenced by other factors, such as reviewer availability (the amount of time reviewers have to conduct pull requests), the complexity of the pull requests (complex changes may necessitate more time and effort to review), and the size of the maintainer team (for those projects where maintainers are responsible for code reviews, the number of maintainers might impact the first review interval).

5.2.5 *Characteristics of OSS Projects*

We conducted an analysis of ten OSS projects that varied in age, as presented in Table 3.2. Our analysis covered a time range of a minimum of two years and a maximum of nine years, containing a diverse set of projects at different stages of development. This approach allowed us to test the effectiveness of our methods across a range of project establishments. It is important to analyze OSS projects of different ages since they undergo not only technical but also social changes throughout their lifespan, as noted by Nakakoji et al. [26].

In Table 3.1, we presented data on the size of our ten selected OSS projects, measured in terms of the number of issues. The number of issues varied significantly across the projects, ranging from approximately 4k for TypeScript to around 111k for VS Code. This variation in size is due to several factors, such as the complexity and activity level of the project, the diversity of contributors, and the level of community engagement.

Studying OSS projects of different sizes is important because it can reveal unique insights into the dynamics of collaboration and communication among contributors. For example, analyzing a large project with a high level of activity may uncover patterns of collaboration and coordination among a large and diverse group of contributors. On the other hand, studying a smaller project with fewer contributors may highlight the critical role of a core group of contributors in maintaining the project. By analyzing projects of different sizes, we can develop more generalizable insights that can be applied to a wide range of OSS projects.

Our analysis of OSS projects is more generalizable by the fact that each project has its own set of contribution guidelines. For example, the Deno project partially relies on communication through its Discord forum, as outlined in their contribution guidelines[3]. Additionally, TensorFlow provides very detailed documentation about their review process, with maintainers responsible for assigning reviewers to pull requests[4]. Meanwhile, Atom places a greater emphasis on the code of conduct rather than specific contribution guidelines, which can confuse new contributors[5]. The presence of distinct contribution guidelines across different projects allows us to consider and compare a wide range of factors that may influence the project's development, communication, and collaboration.

Like Moby, the Bootstrap project also places a strong emphasis on communication through GitHub, as outlined in their contribution guidelines which encourage contributors to get in touch with maintainers before creating pull requests[6]. In contrast, TypeScript encourages contributors to solve issues without contacting the maintainers or informing the other contributors[7], which could suggest a lower level of communication in this project.

3 `https://deno.com/manual@v1.33.1/references/contributing`, last accessed 30/04/2023
4 `https://github.com/tensorflow/tensorflow/blob/master/CONTRIBUTING.md#how-to-become-a-contributor-and-submit-your-own-code`, last accessed 30/04/2023
5 `https://github.com/atom/atom/blob/master/CODE_OF_CONDUCT.md`, last accessed 30/04/2023
6 `https://github.com/twbs/bootstrap/blob/main/.github/CONTRIBUTING.md`, last accessed 30/04/2023
7 `https://github.com/microsoft/TypeScript/blob/main/CONTRIBUTING.md#issue-claiming`, last accessed 30/04/2023

Our selected OSS projects have different communication, contribution, and collaboration methodologies, which can significantly affect our results. By analyzing these different methodologies, we can gain insights into the factors that influence OSS project. of different types and sizes.

5.3 THREATS TO VALIDITY

In this section, we discuss the threats to the validity of our study.

While conducting our analysis, we identified several potential threats to its validity. One such threat is that our study considers only the communication that occurs in GitHub issues or pull requests. Considering that some GitHub OSS projects may utilize multiple communication tools such as Discord, Slack, or external forums, which we have not included in our dataset, it may lead us to miss out on communication between developers. However, we consider this threat to be minimal because the important and most relevant communication about issues or pull requests usually occurs in the issue or pull request itself.

Additionally, our analysis relies on specific tools like CORONET for network building. This limits the scope of our analysis to only the networks that CORONET offers. Nevertheless, CORONET offers three important types of networks in the STC context, such as social networks, technical networks, and socio-technical networks, which cover a high amount of patterns that can occur in STC. Moreover, we use socio-technical networks constructed from different data sources, one threat is that the socio-technical networks do not reflect the real world. This risk is very low because previous research proves that socio-technical networks are an adequate representation of relationships in OSS projects [23].

It is important to note that our analysis relies on the STMC methodology to measure STC in the analyzed OSS projects. It is possible that the STMC formalization may not reflect other socio-technical patterns that exist. Nonetheless, given that STMC captures the two fundamental ways of collaboration, Mauerer et al. [21] believe that any other patterns would likely appear as sub-(anti-) motifs, which means that the correlation should not differ significantly from our results.

Moreover, the terms "commented" or "reviewed" may not accurately reflect when a pull request is reviewed and bias the results for the review rate and first review interval of a pull request. This is because any GitHub user can leave a comment on a pull request, even if their comment is not related to the review process. We classify these comments as reviews since, although they may not include a review from the project maintainers, they still provide feedback and assessment for the pull request.

Lastly, our study uses a quantitative analysis as we did not conduct an extensive qualitative analysis. As a result, we base our discussion section on hypotheses of what might have caused these obtained results, hence, we can not make definitive statements about the implications of the obtained results. Regardless, we try to mitigate this limitation by reviewing contribution guidelines for each analyzed OSS project and presenting implications in the discussion section.

6 RELATED WORK

In this chapter, we present research papers that are related to the main concepts of this thesis, namely, socio-technical congruence implemented using socio-technical networks and socio-technical congruence correlated with code review measures.

The most related work for this thesis is of course the work from Mauerer et al. [21] because we use their operationalization of STMC to measure the STC in this thesis. Mauerer et al. correlate the STMC with software quality measures, namely, code bugs and code churn. They use a mixed-methods statistical analysis to investigate the relationship between STMC and code bugs and code churn. The authors collect data from 25 large OSS projects and measure STMC based on the occurrence of triangle anti-/motifs and square anti-/motifs in the projects.

In their STMC configuration, the authors consider files as artifacts and capture the file dependencies (artifact relationship) from function calls, co-changes in the same commit, and comments and key terms associated with source code. Moreover, they capture the developer's communication (author relationship) from issues and emails.

Mauerer et al. collect data on code bugs and code churn from the same projects and correlate them with STMC. They count the number of bugs associated with an artifact (bug density) and compute the magnitude of churn (changed lines per artifact). An increasing number of bugs is equivalent to decreasing software quality, and a high change rate (churn) over extended periods of time on a given artifact indicates low software quality.

Their study's main finding is that there is no correlation between STMC and bugs or churn, regardless of the time frame considered. In other words, the authors do not identify any statistically significant association between the coordination of developer tasks and communication and the outcomes of a project. Based on their findings, they conclude that if there exists a quantifiable connection between STMC and software quality measures, it would have to appear in a higher level of abstraction than connections between individual software files.

Another study that shares similarities with Mauerer et al.'s work [21] and is relevant to our own research is the research conducted by Baldwin and Colfer [3]. This study can be compared to Mauerer et al.'s research because, similar to their hypothesis, Baldwin and Colfer put forth the idea that the organizational structure (specifically communication within the context of the STMC framework) and software architecture (particularly file dependencies within the STMC) should mirror each other.

In their study, Baldwin and Colfer propose the mirroring hypothesis, which suggests that organizational relationships (social aspect) within a project or firm correspond to the technical dependencies present (technical aspect). Baldwin and Colfer analyze a total of 142 empirical studies, which are classified into three categories: industry, firm, and open collaborative projects. These studies are further classified into two types: descriptive and normative. Descriptive studies aim to establish relationships between technical dependencies and organizational ties, while normative studies investigate the performance outcomes of mirrored versus unmirrored systems. Based on their analysis of industry and firm studies, Baldwin and Colfer find that over 70% of the descriptive studies provide strong evidence in support of the mirroring hypothesis. However, the normative studies suggest the existence of a "mirroring trap", where firms may become too focused on their current technical architecture and miss out on architectural innovations that arise outside their boundaries. Regarding the evidence from open collaborative projects, Baldwin and Colfer report that the majority of descriptive studies (56%) do not support the mirroring hypothesis. However, there are not enough normative studies available to provide a conclusive decision.

In this thesis, we focus on socio-technical networks as a critical component, and in this regard, we present three studies that also utilize socio-technical networks in their research. First, we present the study by Joblin and Apel [12]. They analyze socio-technical congruence using socio-technical networks, in the context of project success. They define successful projects as those with a high level of popularity, tens of thousands of commits, and an active developer community with hundreds or thousands of contributors. Conversely, failed projects are those that have high levels of popularity but are eventually abandoned. Joblin and Apel examine 32 open-source projects using version control systems and classify them as successful or failed via a keyword search and qualitative evaluation with project maintainers. Among their findings, they discover that even a simple socio-technical network model, based purely on version control system data, can provide accurate and generalizable indicators of future project success.

The second study is the one by Syeed and Hammouda [32]. In their study, Syeed and Hammouda explore the notion of Conway's Law and utilize socio-technical networks to evaluate socio-technical congruence in the FreeBSD[1] project. They examine both social and technical aspects of congruence, studying how communication patterns of the OSS developer community align with the software architecture and vice versa. Syeed and Hammouda also analyze the evolution of socio-technical congruence over time. Their findings indicate that the communication needs established by the software architecture and the interdependency among software modules are effectively embedded in the communication patterns of the developer community in the FreeBSD project. Moreover, Syeed and Hammouda discover that the level of socio-technical congruence in the project remains stable as the project matures.

The third study we present regarding socio-technical networks is the one by Bird et al. [5]. They investigate the influence of socio-technical networks on the fault-proneness of individual software components for Windows Vista and Eclipse releases. Their first hypothesis proposes that the role of a software component in the technical network and its contribution to the social network influence its proneness to defects. The second hypothesis suggests that software components that have key roles in the socio-technical network are more likely to have defects. These hypotheses hold true for both projects, with the exception of one Eclipse release. The researchers also find that incorporating socio-technical networks in predictive models enhances the predictive power of the models in comparison to only social or technical networks.

As we correlate STMC with code review measures, we refer to a study conducted by Zabardast, Gonzalez-Huerta, and Tanveer [35]. Similar to this thesis, the authors employ code reviews as a source of data and investigate a measurement similar to socio-technical

1 https://www.freebsd.org/, last accessed on 19/04/2023

congruence in relation to code review time. Specifically, Zabardast, Gonzalez-Huerta, and Tanveer investigate the effect of team ownership and team contribution alignment on communication overhead such as lead time and code review time, as well as the pace of technical debt (TD) accumulation in services. The authors define team ownership as the official team responsible for a component. Meanwhile, they define team contribution as the contribution coming from code production, issuing tickets, and code reviews. They analyze data from 267 components of a company. Their early findings suggest that early identification of such misalignment can help organizations prevent the faster accumulation of TD. Currently, they conclude that team ownership and team contribution alignment have a significant impact on software development processes, and identifying and addressing such misalignments can help mitigate communication overhead and minimize TD accumulation.

7 CONCLUDING REMARKS

In this chapter, we summarize the approach that we followed to perform an STC analysis and our findings. Additionally, we explain our suggestions of what can be done in the future to perform several studies based on this thesis.

7.1 CONCLUSION

In this thesis, we conducted an STC analysis for OSS projects to determine if there is a correlation between STC and code review measures. We used the STMC formalization, as presented by Mauerer et al. [21], to measure STC and considered the Acceptance Rate, Review Rate, and First Review Interval for code review measures.

Our case studies were ten OSS projects including Vs Code, TypeScript, TensorFlow, React, Next.js, Moby, Electron, Deno, Bootstrap, and Atom. We chose these projects because they vary in size, age, and domain. To get their data, we mined their git repositories until 2020 and used codeface and GitHubWrapper to process the data.

For each project, we built multi-networks using coronet to capture socio-technical relationships. We used files and developers as nodes, and pull requests and commits as edges. In these multi-networks, we applied the STMC framework which consists of extracting triangle or square motifs or anti-motifs. By using the triangle motifs we captured the communication (or lack thereof for anti-motifs) between two developers contributing to the same file. Meanwhile, by using the square motifs, we captured the communication (or lack thereof for anti-motifs) between two developers contributing to two related files.

We quantified the STMC using the dSTMC which includes triangle (see 2.1) and square congruence (see 2.2). Additionally, we proposed a new version of the STMC, called the STMC_r. By using STMC_r we checked whether the file(s) in the motif is modified in any of the pull requests that form the same motif. Similarly to dSTMC, we proposed dSTMC_r that quantifies the STMC_r.

Next, we formalized the code review measures, including the Acceptance Rate, Review Rate, and First Review Interval. We defined Acceptance Rate as the number of merged pull requests divided by the total number of pull requests, Review Rate as the number of reviewed pull requests divided by the total number of pull requests, and First Review Interval as the time from the creation of a pull request until it is merged, closed, or abandoned.

Additionally, we applied a time-split analysis for each project, splitting them into six-month time windows and using a sliding window of three months for the Acceptance Rate and First Review Interval. We analyzed each time window individually to gain a comprehensive result for the complete project. Moreover, we used Kendall's Tau to correlate the code review measures with the dSTMC or dSTMC_r and a significance level of 5% to assess statistical significance.

In our first research question, we correlated the Acceptance Rate with the dSTMC or $dSTMC_r$ and found statistically significant correlations for some projects but not for others thus, our results were inconclusive. For those projects that showed statistically significant correlations, this suggests that the STMC or $STMC_r$ affect the Acceptance Rate presumably due to factors such as expectations when reviewing a pull request, developers' familiarity with the code base, and the quality of the contributions. Meanwhile, for the other projects, the Acceptance Rate is probably affected by factors such as authors' past contributions, code base size, or code review frequency.

Our second research question investigated the correlation between the Review Rate and dSTMC or $dSTMC_r$. We found no statistically significant correlation for most of the projects, suggesting that there is no relationship between the Review Rate and STMC. This suggests that other factors such as project size, complexity, level of engagement within the community, and quality of documentation could influence the Review Rate.

In our third research question, we examined the correlation between the First Review Interval and the dSTMC or $dSTMC_r$. However, the results of this study were inconclusive since some projects demonstrated statistically significant correlations while others did not. For projects that demonstrated a statistically significant correlation, it implies that the STMC or $STMC_r$ has an impact on the First Review Interval. This observation could be caused by various factors, such as the quality of documentation, the pull request process, and the importance placed by the project community on communication and collaboration.

In conclusion, this thesis aimed to investigate the correlation between the STC and code review measures. Our study used STMC and $STMC_r$ to measure the STC and Acceptance Rate, Review Rate, and First Review Interval as code review measures. However, we were not able to draw a conclusion for the Acceptance Rate and First Review Interval, but our findings indicate that there is no statistically significant correlation between the Review Rate and the STMC or $STMC_r$.

7.2 FUTURE WORK

In this section, we offer proposals for those interested in further exploring this area of research in the future.

Our first suggestion is to expand the range of OSS projects analyzed in this study by including more OSS projects that use not only GitHub but also other contribution tools, such as mailing lists, Jira issues, forums, and others. Doing so would improve the robustness of the study and increase its generalizability across different types of OSS projects. However, this would also introduce new challenges, such as configuring the STMC with different types of nodes and edges.

Additionally, we propose that in cases where data or STC constraints limit the use of a square motif, which is more complex than the triangle motif, researchers could analyze solely the triangle congruence to measure direct collaboration. As a result, researchers will capture only a subset of the STC within a project, specifically, direct communication among the projects' community. This trade-off may limit the researchers to fully understanding the alignment of the project's social and technical structures, as indirect communication and coordination among team members are not included in the analysis. Nonetheless, this approach can still provide valuable insights into the direct collaboration patterns within a project.

We recommend conducting a qualitative study before analyzing the STC in OSS projects. The qualitative study would involve examining how the code review process is managed in the selected OSS projects by checking the project contribution guidelines or consulting with project maintainers to obtain a better understanding of how code reviews are conducted. This information would be necessary to ensure that the measures that we used to assess the code review process, such as Acceptance Rate, Review Rate, and First Review Interval, are accurate and suitable for the chosen OSS projects. For instance, in our second research question, we discovered that different projects have different review processes. Some projects used the Review option in GitHub, while others relied on comments. In some cases, the review process was unclear. The results could become robust by conducting a qualitative analysis.

Furthermore, we recommended grouping the analyzed OSS projects into categories based on their review methodologies, rather than using a general measurement for all projects. For instance, one group could be for projects that use GitHub's Review option for code review, while another group could be for projects that rely on comments for code review. This approach would help to ensure that the analysis is more similar to the specific review methodologies employed by the OSS projects, leading to more accurate results.

Another recommendation is to refine and test the STMC_r further. As demonstrated by our preliminary sanity check, the STMC_r successfully filters out false positive motifs that the original STMC framework may detect. To improve the accuracy of the STMC_r, additional manual checks could be conducted.

Lastly, it is worth considering other code review measures when investigating the correlation between STC and code review measures. For example, in their study on socio-technical code review measures and security vulnerabilities, Meneely et al. [22] explored various code review measures such as the number of people invited to provide feedback on a file, the number of different people who provided feedback on a file across all its code reviews, the percentage of reviews that exceeded 200 lines per hour, and others. While Meneely et al. applied these measures in the context of security vulnerabilities, we believe that they could also be utilized in the context of STC and correlated with STMC. For instance, we could investigate the correlation between the number of different people who provided feedback on a file in a pull request and STMC, and investigate whether communication or lack thereof impacts the number of different people who provided feedback on a file.

A
APPENDIX

In the appendix, we provide plots, tables, and additional information for this thesis.

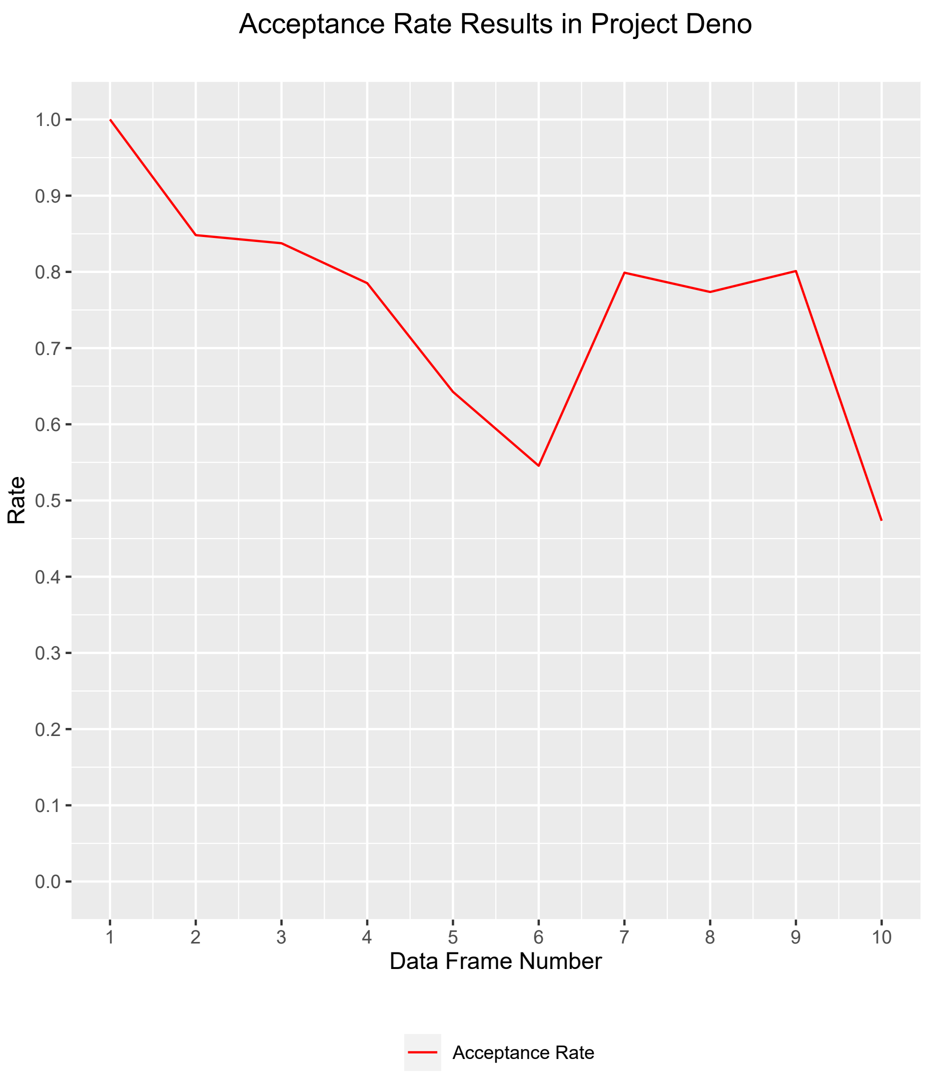


Figure A.1: In this Figure, we present the acceptance rate for the lifetime of the Deno project, with the y-axis representing a range from 0 to 1. The x-axis displays the number of data frames when using a data split analysis with a sliding window. We observe that the acceptance rate at the start of the project is high, likely due to the small number of contributors and the maintainers merging pull requests without a thorough review. Over the years, the acceptance rate fluctuates.

Table A.1: In this table, we summarize all the types of edges that can be used in author, artifact, or bipartite networks, and we leave out the multi-network as it is a result of these networks. In the second row, we specify the type of network, and in the second column, we list the types of relations that exist. For example, the cell highlighted in yellow (third row and fourth column) describes what the cochange relation means in an artifact network. Assuming that the artifact is a file, the cochange relation means that the two files are changed in the same commit.

		Network types		
		Author	**Artifact**	**Bipartite**
Relation (Edges) types	**cochange**	Authors who change the same source-code artifact are connected with an edge.	Source-code artifacts that are concurrently changed in the same commit are connected with an edge.	Authors get linked to all source-code artifacts they have changed in their respective commits.
	mail	Authors who contribute to the same mail thread are connected with an edge.	Mail threads are connected when they reference each other.	Authors get linked to all mail threads they have contributed to.
	issue	Authors who contribute to the same issue are connected with an edge.	Issues are connected when they reference each other.	Authors get linked to all issues they have contributed to.
	callgraph	Does not apply	Source-code artifacts are connected when they reference each other (i.e., one artifact calls a function contained in the other artifact).	Authors get linked to all source-code artifacts they have changed in their respective commits (same as for the relation cochange).

A.1 CONFIGURING TRIANGLE & SQUARE MOTIFS

In this section, we explain the possible configurations that CORONET offers. Additionally, we explain why we do not choose the other possible configurations but only one and provide an example of what would be an unsuitable configuration for our analysis.

In Figure A.2, we present a feature model that shows the possible configurations for a multi-network using CORONET. Feature models are commonly used to generate specific configurations of a software product line by selecting and combining appropriate features based on stakeholders' needs and requirements. In this case, we use it to show the possible configurations for the multi-network and not for a product line. However, it is important to note that not all configurations may be valid, as we explain later.

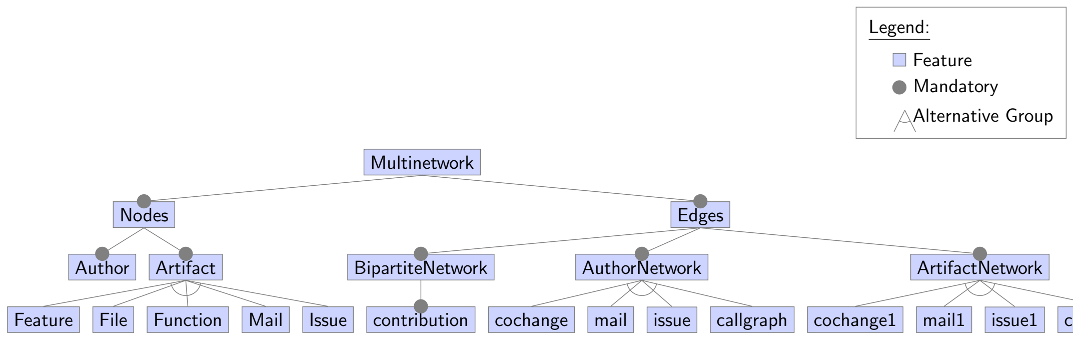


Figure A.2: The root of the feature model is a multi-network that represents a triangle or square motif. Regarding the legend: Feature - each blue rectangle in the feature model. Mandatory – child feature must be selected. Alternative Group (XOR) – exactly one of the sub-features must be selected. The features beginning with a lowercase letter signify a relation such as "cochange," "cochange1", "mail," "mail1," "issue," and others. There is no difference between "cochange" and "cochange1" or the other sibling features; we add the 1 to differ that it has another meaning in another type of network.

In the feature model, we can only configure the "Artifact" feature[1], the type of relationship between two authors (the children of the AuthorNetwork feature), and the type of relationship between two artifacts (the children of the ArtifactNetwork feature). In the following, we briefly explain why we either consider or do not consider the children of the "Artifact" feature from the feature model. We do not mention the "Author" feature because it is mandatory.

- Artifact

  Feature →In software systems with multiple modules or components, the use of features can lead to two common issues: feature scattering and feature tangling. Feature scattering arises when the implementation of a single feature is distributed across multiple modules, resulting in difficulties in maintaining and modifying that feature independently from others. Contrarily, feature tangling occurs when code related to multiple features is entangled within a single module, making it challenging to modify or maintain a single feature without impacting others. Due to these challenges, we

1 Note that with "feature" we refer to the features in the feature model and not Feature as an artifact.

opt not to select a Feature as an artifact because identifying Features in our data set is very problematic.

File →We decide to consider a File as a potential artifact in our configuration because it aligns well with the relational structure of the overall graph, as depicted in Figure 4.5. More importantly, selecting Files supports our STMC goal because they are a technical artifact in OSS projects. Files can represent a variety of artifact types, including source code, documents, configuration files, and data files, among others, and can effectively track the evolution of these artifacts over time.

Function →We do not choose a Function as an artifact because we lack the necessary data. Furthermore, Functions are too granular for the type of analysis we intend to conduct.

Mail →A Mail artifact is typically employed for projects that rely on mailing lists as a means of contribution, collaboration, and communication. However, in the present study, we analyze ten OSS projects that use GitHub for these purposes, and thus, we decide not to consider Mail as an artifact. Additionally, if we were to include projects that utilize mailing lists, we would need to gather additional information to facilitate the mapping of emails to commits, such as Patch-stack analysis (PaStA)[2] data, which links patches sent to mailing lists with upstream commits.

Based on the above reasoning, we are left with two artifact types to use, namely File or Issue. We must also consider four relation types, namely cochange, issue, mail, and callgraph.

We exclude mail as a relation because it requires matching the artifact (File or Issue) with the corresponding mail thread, which necessitates the use of PaStA data. Incorporating PaStA data would decrease the soundness of our STMC analysis as the mapping is not entirely accurate. Moreover, we are focusing solely on GitHub projects, hence, we do not have mail data for the ten OSS projects that we are analyzing.

There are a few projects that use both, namely mail threads and GitHub issues to communicate. Nevertheless, incorporating mail as a relation would still require the use of PaStA data. Figure A.3 illustrates an example of a square motif configuration containing Issue as an artifact and mail as a relation that would not work. We use a square motif in this illustration as it is more complicated than a triangle motif because the square motif has one edge more. We use the same configuration for the triangle motif as for the square motif but simplify it by removing the edge between the two artifacts.

Furthermore, we exclude callgraph because it is too finely grained as it connects artifacts if they reference each other in the code. Callgraph would be a suitable solution if we were using Function or Feature as an artifact.

Therefore, we have two artifacts to choose from, File and Issue, and two relation types, cochange, and issue. These possibilities result in eight possible configurations, but only one of them suits our goal the best. In the following, we explain the rationale for why we choose or do not a configuration.

2 https://github.com/se-sic/coronet#data-sources, last accessed on 11/04/2023

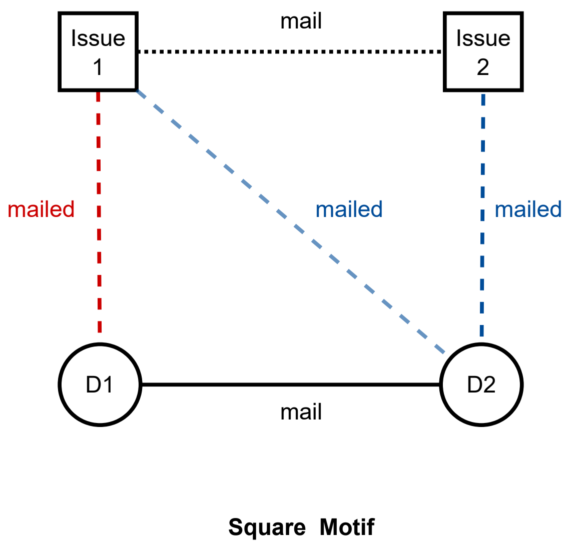


Figure A.3: The dashed edges show that the developers (or the author; we call them developers not to confuse the term with the author of the pull request) send an email to a mail thread. The blue diagonal edge shows that developers D1 and D2 receive the solid edge "mail" from this mail because they have both contributed to the same mail thread. Meanwhile, the two artifacts Issue1 and Issue2 have an edge "mail" because the mail threads that Issue1 and Issue2 are part of, reference each other. However, the data that we have does not allow us to check if two Issues reference each other (in this exact configuration), only if two mail threads reference each other. Additionally, we would need to add another condition to verify that D1 and D2, who are communicating via a mail thread, have also contributed to Issue1 and Issue2. As a result, it does not align with the socio-technical value and a Mail artifact would be more appropriate (for projects that use mailing lists as a primary communication tool).

The format of the following points is [Artifact], [Artifact Relation], [Author Relation] followed by the reasoning.

1. File, cochange, cochange

   The cochange relation for the Author Relation is intended for Files, which are typically changed in the same commit. However, this relation does not provide us with any information about communication since it only focuses on commits. As we need to consider both artifact dependencies and communication between developers for an STMC analysis, the cochange relation does not add any communication value to our analysis.

2. File, issue, issue

   When the Artifact is a File, the issue relation is not suitable for the Artifact Relation because it connects two Issues that reference each other. In this case, we would need to check whether the File is part of the Issue when the developer is contributing to the Issue. Also, this type of relationship is mostly suitable to detect collaboration and not artifact dependencies.

3. File, issue, cochange

   We apply the same reasoning as configurations 1 and 2.

4. File, cochange, issue

   The cochange relation is appropriate for files because it connects two files that are changed in the same commit, which is a common occurrence in OSS projects. Additionally, it detects technical dependencies between the artifacts. Moreover, the issue relation is suitable for two developers because they can communicate using the comment section on an Issue. Together, the cochange and issue relation provide us with information about the technical dependencies and communication that occur around them. Therefore, this configuration is best aligned with our goal, and we select it for our square motif in the STMC analysis.

5. Issue, cochange, cochange

   Same reasoning as configuration 1.

6. Issue, issue, issue

   This configuration is feasible and valuable. However, we do not choose it because using the Issue as an artifact and the issue as a relation for two artifacts would result in a higher level of granularity than necessary. Issues are typically a high-level representation of a problem or task, and may not provide enough detail to fully capture the complexity of relationships between artifacts in a project. Likewise, an issue is not the most suitable relationship to capture technical dependencies, since a lot of communication is involved there. Therefore, this configuration may not provide us with the necessary level of granularity required for our analysis.

7. Issue, issue, cochange

   We apply the same reasoning as configuration 1.

8. Issue, cochange, issue

   The cochange relation requires two issues to be changed in the same commit, which is highly unlikely. Typically, a commit addresses changes relevant to only one issue rather than two or more.

BIBLIOGRAPHY

[1] Mark Aberdour. “Achieving Quality in Open Source Software.” In: *IEEE Software* 24.1 (2007), pp. 58–64.

[2] Uri Alon. “Network motifs: theory and experimental approaches.” In: *Nature Reviews Genetics* 8 (2007), pp. 450–61.

[3] Carliss Baldwin and Lyra Colfer. “The Mirroring Hypothesis: Theory, Evidence and Exceptions.” In: *Industrial and Corporate Change - Oxford Academic* 25.5 (2016), pp. 709–739.

[4] Andrew Begel, Jan Bosch, and Margaret-Anne Storey. “Social Networking Meets Software Development: Perspectives from GitHub, MSDN, Stack Exchange, and TopCoder.” In: *IEEE Software* 30.1 (2013), pp. 52–66.

[5] Christian Bird, Nachiappan Nagappan, Harald Gall, Brendan Murphy, and Premkumar Devanbu. “Putting It All Together: Using Socio-Technical Networks to Predict Failures.” In: *International Symposium on Software Reliability Engineering (ISSRE)*. IEEE Computer Society, 2009, pp. 109–119.

[6] Marcelo Cataldo, James Herbsleb, and Kathleen Carley. “Socio-Technical Congruence: A Framework for Assessing the Impact of Technical and Work Dependencies on Software Development Productivity.” In: *International Symposium on Empirical Software Engineering and Measurement (ESEM)*. ACM, 2008, pp. 2–11.

[7] Marcelo Cataldo, Patrick Wagstrom, James Herbsleb, and Kathleen Carley. “Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools.” In: *ACM Conference On Computer-Supported Cooperative Work And Social Computing*. ACM, 2006, pp. 353–362.

[8] Adrian M. Colyer. “How Do Committees Invent? and Ironies of Automation.” In: *ACM Queue* 18.1 (2020), pp. 37–38.

[9] Tapajit Dey and Audris Mockus. “Which Pull Requests Get Accepted and Why? A study of popular NPM Packages.” In: *Computing Research Repository (CoRR)* abs/2003.01153 (2020).

[10] James Herbsleb and Rebecca Grinter. “Splitting the Organization and Integrating the Code: Conway’s Law Revisited.” In: *International Conference on Software Engineering (ICSE)*. ACM, 1999, pp. 85–95.

[11] Claus Hunsen, Janet Siegmund, and Sven Apel. “On the Fulfillment of Coordination Requirements in Open-Source Software Projects: An Exploratory Study.” In: *Empirical Software Engineering (EMSE)* 25.6 (2020), pp. 4379–4426.

[12] Mitchell Joblin and Sven Apel. “How Do Successful and Failed Projects Differ? A Socio-Technical Analysis.” In: *ACM Transactions on Software Engineering and Methodology* 31.4 (2022), 67:1–67:24.

[13] Verena Käfer, Daniel Graziotin, Ivan Bogicevic, Stefan Wagner, and Jasmin Ramadani. "Communication in Open-Source Projects - End of the E-Mail Era?" In: *International Conference on Software Engineering (ICSE)*. ACM, 2018, pp. 242–243.

[14] Rick Kazman, Dennis Goldenson, Ira Monarch, William Nichols, and Giuseppe Valetto. "Evaluating the Effects of Architectural Documentation: A Case Study of a Large Scale Open Source Project." In: *IEEE Transactions on Software Engineering* 42.3 (2016), pp. 222–247.

[15] Robert Kraut and Lynn Streeter. "Coordination in Software Development." In: *Communications of the ACM* 38.3 (1995), pp. 69–81.

[16] Irwin Kwan, Adrian Schröter, and Daniela Damian. "A weighted congruence measure." In: *Workshop on SocioTechnical Congruence*. 2009, pp. 1–4.

[17] Irwin Kwan, Adrian Schröter, and Daniela Damian. "Does Socio-Technical Congruence Have an Effect on Software Build Success? A Study of Coordination in a Software Project." In: *IEEE Transactions on Software Engineering* 37.3 (2011), pp. 307–324.

[18] Valentina Lenarduzzi, Vili Nikkola, Nyyti Saarimäki, and Davide Taibi. "Does Code Quality Affect Pull Request Acceptance? An Empirical Study." In: *Journal of Systems and Software* 171 (2021), p. 110806.

[19] Juho Lindman, Matti Rossi, and Pentti Marttiin. "Applying Open Source Development Practices Inside a Company." In: *The International Federation for Information Processing (IFIP)*. Vol. 275. Springer, 2008, pp. 381–387.

[20] Yuan Long and Keng Siau. "Social Network Structures in Open Source Software Development Teams." In: *Journal of Database Management (JDM)* 18.2 (2007), pp. 25–40.

[21] Wolfgang Mauerer, Mitchell Joblin, Damian Tamburri, Carlos Paradis, Rick Kazman, and Sven Apel. "In Search of Socio-Technical Congruence: A Large-Scale Longitudinal Study." In: *IEEE Transactions on Software Engineering* 48.8 (2022), pp. 3159–3184.

[22] Andrew Meneely, Alberto Rodriguez Tejeda, Brian Spates, Shannon Trudeau, Danielle Neuberger, Katherine Whitlock, Christopher Ketant, and Kayla Davis. "An Empirical Investigation of Socio-Technical Code Review Metrics and Security Vulnerabilities." In: *Foundations of Software Engineering (SIGSOFT/FSE)*. ACM, 2014, pp. 37–44.

[23] Andrew Meneely and Laurie Williams. "Socio-Technical Developer Networks: Should We Trust Our Measurements?" In: *International Conference on Software Engineering (ICSE)*. ACM, 2011, pp. 281–290.

[24] Andrew Meneely, Laurie Williams, Will Snipes, and Jason Osborne. "Predicting Failures with Developer Networks and Social Network Analysis." In: *Foundations of Software Engineering (SIGSOFT/FSE)*. ACM, 2008, pp. 13–23.

[25] Nachiappan Nagappan, Brendan Murphy, and Victor Basili. "The Influence of Organizational Structure on Software Quality: an Empirical Case Study." In: *International Conference on Software Engineering (ICSE)*. ACM, 2008, pp. 521–530.

[26] Kumiyo Nakakoji, Yasuhiro Yamamoto, Yoshiyuki Nishinaka, Kouichi Kishida, and Yunwen Ye. "Evolution Patterns of Open-Source Software Systems and Communities." In: *International Workshop on Principles of Software Evolution (IWP SE)*. ACM, 2002, pp. 76–85.

[27] Ashwin Paranjape, Austin Benson, and Jure Leskovec. “Motifs in Temporal Networks.” In: *Web Search and Data Mining (WSDM)*. ACM, 2017, pp. 601–610.

[28] Peter Rigby and Christian Bird. “Convergent Contemporary Software Peer Review Practices.” In: *European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 2013, pp. 202–212.

[29] Peter Rigby, Daniel Germán, Laura Cowen, and Margaret-Anne Storey. “Peer Review on Open-Source Software Projects: Parameters, Statistical Models, and Theory.” In: *ACM Transactions on Software Engineering and Methodology* 23.4 (2014), 35:1–35:33.

[30] José Sierra, Aurora Vizcaíno, Marcela Genero, and Mario Piattini. “A Systematic Mapping Study about Socio-Technical Congruence.” In: *Information and Software Technology* 94 (2018), pp. 111–129.

[31] Daricélio Soares, Manoel Limeira de Lima, Leonardo Murta, and Alexandre Plastino. “Rejection Factors of Pull Requests Filed by Core Team Developers in Software Projects with High Acceptance Rates.” In: *International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2015, pp. 960–965.

[32] Mahbubul Syeed and Imed Hammouda. “Socio-technical Congruence in OSS Projects: Exploring Conway’s Law in FreeBSD.” In: *IFIP Advances in Information and Communication Technology* 404 (2013), pp. 109–126.

[33] Giuseppe Valetto, Mary Helander, Kate Ehrlich, Sunita Chulani, Mark Wegman, and Clay Williams. “Using Software Repositories to Investigate Socio-Technical Congruence in Development Projects.” In: *Mining Software Repositories (MSR)*. IEEE, 2007, p. 25.

[34] Dindin Wahyudin, Alexander Schatten, Dietmar Winkler, and Stefan Biffl. “Aspects of Software Quality Assurance in Open Source Software Projects: Two Case Studies from Apache Project.” In: *Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE Computer Society, 2007, pp. 229–236.

[35] Ehsan Zabardast, Javier Gonzalez-Huerta, and Binish Tanveer. “Ownership vs Contribution: Investigating the Alignment Between Ownership and Contribution.” In: *International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 2022, pp. 30–34.