Doctoral Dissertation

Knowledge Sharing in Software Development: Uncommunicated Update, Communication Channels, and Human Aspects

Yusuf Sulistyo Nugroho

August 31, 2020

Graduate School of Information Science
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Yusuf Sulistyo Nugroho

Thesis Committee:

Professor Kenichi Matsumoto	(Supervisor)
Professor Hajimu Iida	(Co-supervisor)
Associate Professor Takashi Ishio	(Co-supervisor)
Assistant Professor Hideaki Hata	(Co-supervisor)
Assistant Professor Raula Gaikovina Kula	(Co-supervisor)

Knowledge Sharing in Software Development: Uncommunicated Update, Communication Channels, and Human Aspects*

Yusuf Sulistyo Nugroho

Abstract

Knowledge, a fundamental resource that allows people to function intelligently, has become a critical success factor for organizations. To acquire knowledge, knowledge sharing through communication is necessary. However, several challenges may complicate knowledge sharing in software development, such as managing coordination between distributed teams, encouraging people to share embedded knowledge, and handling various social identities of individuals in the teams. Although some solutions have been offered in prior studies, it is important to know where and how knowledge is captured and shared to support the process of knowledge sharing. To achieve the goal, this thesis attempts to (a) empirically study one case of uncommunicated knowledge which is not shared explicitly, (b) analyze multiple communication channels to share knowledge over open source projects, and (c) investigate the human factors of users' interaction in forums while knowledge sharing. This thesis reveals how knowledge sharing in software ecosystems unfolds and that when there are problems with the communication of information, such as uncommunicated update, this determines the extent of its implication on the relevant domains. The results finally highlight the need for better understanding of knowledge sources and the nature of knowledge sharing at the software ecosystem level, thus leading to a better knowledge sharing.

Keywords:

knowledge sharing, uncommunicated update, communication channels, human aspects

*Doctoral Dissertation, Graduate School of Information Science, Nara Institute of Science and Technology, August 31, 2020.

List of Publications

1. Nugroho, Y.S., Hata, H. & Matsumoto, K. How different are different `diff` algorithms in Git? Use --histogram for Code Changes. *Empirical Software Engineering* 25, 790–823, 2020.

2. Tantisuwankul, J., Nugroho, Y. S., Kula, R. G., Hata, H., Rungsawang, A., Leelaprute, P., & Matsumoto, K. A topological analysis of communication channels for knowledge sharing in contemporary GitHub projects. *Journal of Systems and Software*, 158, 2019.

Acknowledgements

Firstly, I would like to thank **Almighty Allah** for giving me opportunity, determination and strength during my study. His continuous grace and mercy was with me throughout my life and ever more during my PhD life.

Secondly, I would like to express my sincere gratitude to my advisor Professor Kenichi Matsumoto for the continuous support of my doctorate and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my PhD study.

To my thesis review committee: Professor Hajimu Iida, Associate Professor Takashi Ishio, Assistant Professor Hideaki Hata and Assistant Professor Raula Gaikovina Kula, thank you for your insightful advices, comments and encouragement, but also for the hard question which incented me to widen my research from various perspectives. In particular, I am grateful to Assistant Professor Hideaki Hata for enlightening me the first glance of research. Thank you for the support, idea, and insights during my PhD life.

My sincere thanks also goes to Associate Professor Hoa Khanh Dam and the University of Wollongong, Australia, who provided me an opportunity to join their team as a visiting research student, and who gave an access to the laboratory and research facilities. Also to Abdulaziz Alhefdhi, thank you for everything he had done for me during my visiting period in Wollongong, Australia. Without their precious support it would not be possible to conduct the research.

I thank to all lab members of Software Engineering for the stimulating discussions and for the time we were working together during research and writing papers before deadlines. Also special thank my Indonesian friends in NAIST for their warm and great friendship and for all the fun we have had in the last three years.

Last but not the least, I would like to thank my family: my parents and parents in law, my beloved wife ***Mulyta Erwien Lestari*** and lovely children (**Keisha**, **Kimi**, **Kenzo**, and **Mecca**), and to all brothers and sisters for supporting me spiritually throughout writing this dissertation and my life in general.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

Knowledge can be presented as combination of values, information, and proficient understanding obtained from experiential [6] or educational activity [109], and the expertise on theory or practice for assessing and integrating new experiences and facts [25]. For an organization, knowledge has become a resource and critical success element that offers competitive values that can be maintained [25, 34, 38, 102]. According to Nonaka and Takeuchi [79], knowledge is categorized into two types, tacit and explicit. Tacit knowledge embeds in the human mind which is usually gained through experiences or jobs, but difficult to codify and document. In contrast, explicit knowledge is easy to identify, articulate, document, and share to other people.

Knowledge that is considered crucial is documented into other formats, such as books, technology, practices or other forms of documents to facilitate the knowledge sharing over time. Knowledge sharing is a process through which knowledge, either tacit or explicit, is communicated to other individuals [8]. In an organization, knowledge sharing between individuals or groups is a crucial part since it is the fundamental means through which team members can contribute to knowledge application, renewal, and the advantage of the team. Knowledge sharing between people within and across teams enables organizations to capitalize the knowledge-based resources [17, 25] and discuss critical factors of projects and work coordination between distributed sites [21].

However, maintaining knowledge sharing in organizations is challenging, especially in software development teams [19, 95]. For example, organizing different elements of activity across distributed teams [21], motivating developers to share their embedded knowledge [23], and managing cross-functionality and different

social identity of individuals in a team [20, 35]. The understanding of the sources where knowledge can be extracted may also challenge the process of knowledge sharing in software development teams. This is because such specific knowledge is not always explicitly documented and communicated between people within or across the teams [92, 94] even though the knowledge already exists. This may influence the results of some related works.

Prior study attempts to observe missing documentation from software artifacts in an agile software development project [94], which might lead the knowledge uncommunicated. However, not much is known about the impact of this undocumented knowledge on the other relevant works. Other studies also offer some solutions relates to knowledge sharing, such as, the design of component-based tools development to reduce the need for communication [60], and the implementation of social cognitive theory to form the conceptual model on knowledge sharing behaviour [5]. Despite the valuable findings of previous studies in facilitating knowledge sharing, questions about where and how knowledge is captured and shared within software ecosystems are still emerging.

To better understanding of knowledge sources and the nature of knowledge sharing, this thesis therefore aims to (i) identify the source of knowledge that may be uncommunicated, (ii) analyze the implementation of communication channels within software ecosystems at topological level, and (iii) characterize the social interaction between individuals while knowledge sharing in online forums. To the best of my knowledge, an understanding of the uncommunicated knowledge and its implication on the relevant works, a topological analysis of communication channels to share knowledge, and an investigation of human aspects in online discussion forums while knowledge sharing have never been undertaken.

1.2 Scope of Dissertation

This section describes the scope of this dissertation in terms of knowledge sharing as illustrated in Figure 1.

Knowledge-sharing platforms, such as GitHub,[1] have changed how developers share knowledge and seek information on the web. As the world's largest developers community to develop software, GitHub hosts millions of repositories, from

[1] `https://github.com`

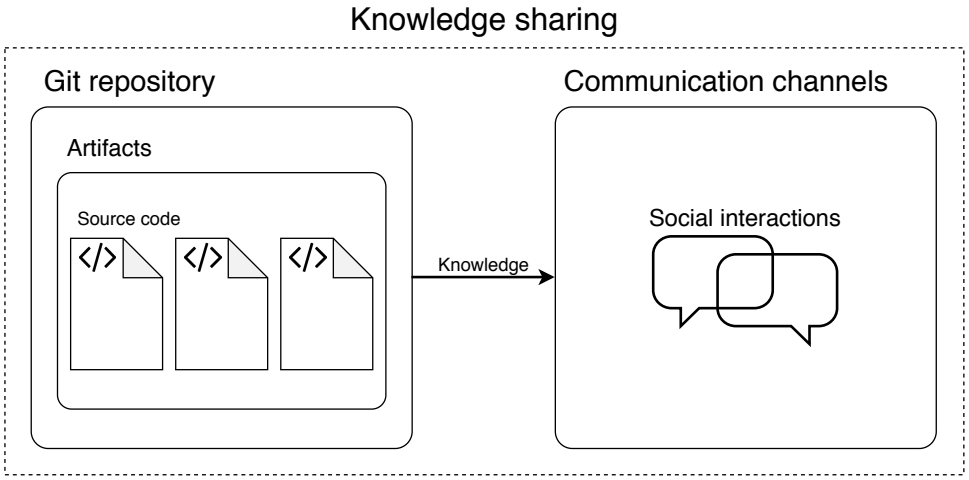


Figure 1: Area of thesis

private team repositories to open source projects. In a repository, developers collectively contribute various types of knowledge sources, such as source code. Source code is one of many artifacts which is produced when doing continuous builds and stored in a software repository. However, knowledge that might be sourced from code could be missed to document [94], leading to the knowledge uncommunicated.

For example, `diff` algorithm in Git has update. The definition and the procedure of each version of `diff` algorithm in producing the lists of the changed lines of code might have been known by the software engineering community. However, the knowledge regarding the differences between the `diff` outputs generated by the old algorithms and the latest version of `diff` algorithm and their implications on the relevant domains have never been studied comprehensively, documented, and communicated between individuals in the community. Practically, people tend to neglect the update of `diff` algorithm and this is possible to have an impact on the result of relevant works.

In addition to GitHub, developers commonly share their knowledge on social media, such as Twitter [39, 72], news aggregator sites [3], and project communication channels such as project fork [13], software license [113, 119], wiki pages [16] for open source software projects, or other textual electronic media [14]. The usage of communication channels critically impacts on how knowledge is formed,

shared, and captured [106]. In the environment of software development, different software projects might adopt different channels in communicating and sharing knowledge. The use of various media and channels to share knowledge might have different implications for software development [111]. However, how knowledge is shared between communication channels of contemporary projects at the ecosystem level is not yet well understood.

As one type of communication channels, Eclipse community forums support mass communication and social interactions between developers in the Eclipse community. Social interaction which information is directly communicated [54] between software developers, is found to have the strongest impact on knowledge sharing behavior [5]. The occurrence of social interactions in software projects is typically associated with a component selection that begins with creating issues or requesting features [81]. However, not much is known about how human aspects play a role in an online discussion such as Eclipse forums and affect the nature of knowledge sharing in the ecosystem.

Limited by the discussion topics based on Figure 1, this thesis investigates comprehensively one case of uncommunicated knowledge and the impact on its relevant study, analyzing the role of multiple communication channels with knowledge sharing over software projects, and understanding the human factors while knowledge sharing in the Eclipse community forums.

1.3 Contributions of Dissertation

This dissertation has three main contributions. First, it provides the analyses of such uncommunicated knowledge and its implication on study results. Second, it discusses the implementation of communication channels to support the knowledge sharing over open source software projects at the topological level, and third, it investigates the human aspects in the Eclipse forums while knowledge sharing.

The main results of this dissertation are described as follows:

1.3.1 Uncommunicated Update: The Differences of Different `diff` Algorithms in Git

To understanding how previous studies used `git diff` command, a systematic mapping study was carried out. 52 papers were selected from eight top journals and three top international conferences published between 2013 and 2017. The findings from the systematic mapping revealed the three most common purposes for using the `git diff` command. This encourage this thesis to undertake comparison analyses between *Myers* and *Histogram* algorithms in three applications: metrics collection, bug-introducing identification using SZZ algorithm, and patches. The intention of this work is to investigate the level of differences between the two `diff` algorithms used in these three applications and their possibility of affecting the result of studies.

The findings from the metrics comparison provide clear evidence that the use of various of `diff` algorithms could differentiate the `diff` lists. This thesis shows that different metric values were obtained from 0.8% to 6.2% in the file-level and 1.7% to 8.2% in the commit-level. These differences can have impacts on studies using diff-related metrics.

The results from the application of SZZ algorithm confirm that different `diff` algorithms possibly generate different results, from 6.0% to 13.3% in the total of the identified bug-fix commits. The *Myers* and *Histogram* sometimes produced a different number and location of the deleted lines (bug-related lines) in several files. Therefore, the comparison result indicates that several prior studies that had used the SZZ algorithm to locate bugs have the possibility of producing inaccurate analyses.

The manual comparison between *Myers* and *Histogram* reveals that the differences are the number of the changes, the order of the changed lines, and even the identified added and deleted code. They certainly affect the readability of the `diff` outputs. Importantly, the result of this thesis provides evidence that *Histogram* frequently produced better `diff` results compared to *Myers* in extracting the differences in source code.

1.3.2 A Topological Analysis of Communication Channels for Knowledge Sharing in Contemporary GitHub Projects

This study discusses the use of topological data analysis to generate the topology of multiple communication channels and their evolution in three years from 2015 to 2017.

Using the Nonaka and Takeuchi's SECI model which describes the transformation of tacit and explicit knowledge into organizational knowledge [79], this thesis is able to map how knowledge is shared through the communication channels. From seven library package projects, thirteen channels are classified. Four channels (i.e. GitHub page, ReadMe file, security audit, and wiki page) are considered as externalization (tacit to explicit), and nine channels (i.e. changelog, code of conduct, licence, contributing guideline, fork, issue tracker, security threat model, number of forks, and number of open issues) are categorized as combination (explicit to explicit). Then, in a large-scale topological analysis of seven library package projects, the results of this thesis show that contemporary GitHub Projects tend to adopt multiple communication channels, especially younger projects that adopt different channels compared to the older projects. The analysis also reveals that communication channels change over time, and they are used to capture new knowledge (i.e., externalization) and update the existing knowledge (i.e. combination).

1.3.3 Discussions in Eclipse Community Forums: Participation, Content, and Sentiment

In this study, an investigation of human aspects when sharing knowledge within the Eclipse community forums with such connective information was conducted. In a large-scale study, forums contribute to help knowledge sharing by referencing various development resources. The findings indicate that Eclipse community forums are essential platform for linking various resources in Eclipse ecosystem. This is shown by various types of links included in the forum threads. Furthermore, active contributors in the ecosystem are likely to be also active in forums, making it a source of expert knowledge for other development systems. The results of this study also show that forum membership-based participation and social interactions plays a significant role in shaping the knowledge sharing within

the Eclipse community forums. Different statuses of users likely to share different types of discussion and different kinds of interaction for both posting and responding to the threads.

1.4 Organization of Dissertation

The rest of this dissertation is structured as follows. Section 2 presents how such uncommunicated knowledge can impact the other studies. In this section, an empirical study of different `diff` algorithms in Git is conducted. The first part of this section discusses a systematic mapping study on 11 top journals and international conference proceedings published between 2013 and 2017 to investigate the application of `git diff` command in prior works. In the second part, it describes the comparison analyses between two popular `diff` algorithms (*Myers* and *Histogram*) in three major applications based on the findings in the systematic mapping survey, that are, metrics collection, SZZ algorithm application to identify bug-introducing changes, and manual comparison of patches.

Section 3 presents a topological analysis of communication channels to share knowledge among projects within software ecosystems. Different knowledge forms of channels from seven library ecosystems are identified. This section also explores the evolution of different channels and distinguishes the differences between these seven ecosystems.

Section 4 is divided into two studies. First, it presents the preliminary study of statistical description. Second, it discusses the investigation of human aspects in Eclipse forums relates to the participation, discussion and social interactions between forum users.

Finally, Section 5 concludes this dissertation.

2 Uncommunicated Update: The Differences of Different *diff* Algorithms in Git

2.1 Background

The `diff` utility calculates and displays the differences between two files, and is typically used to investigate the changes between two versions of the same file. Since understanding and measuring changes in software artifact is essential in empirical software engineering research, `diff` is commonly used in various topics, such as defect prediction where code churn [76, 99] and process metrics [42, 69, 52] are used, code authorship [88, 70], clone genealogy [59, 31], and empirical studies of changes [7, 90].

Along with the growth of GitHub, recent studies analyze software changes from Git repositories by using the `git` command. Git, a version control system, offers `diff` utility for users to select the algorithms of `diff`. Git offers four `diff` algorithms, namely, *Myers*, *Minimal*, *Patience*, and *Histogram*. Without an identifying algorithm, *Myers* is used as the default algorithm.

In textual differencing, all `diff` algorithms are computationally correct in generating the `diff` outputs. However, the `diff` outputs are sometimes different due to different `diff` algorithms. Different `diff` algorithms might identify different change hunks, that is, a list of program statements deleted or added

contiguously, separated by at least one line of unchanged context [90]. We expect that a set of changing operations done by developers can be represented by change hunks. However, there can be inappropriate identifications of change hunks. Although *Histogram* that was introduced in `git 1.7.7`[2] in 2011 might give better performance to `git diff`, it is not popular among software engineer communities. Thus, we focus on the *Myers* and *Histogram* algorithms to empirically investigate the impact on software engineering research. The motivation of this study is try to clarify the impact of adopting different `diff` algorithms on empirical studies and investigate which `diff` algorithm can provide better `diff` results that can be expected to recover the changing operations. Furthermore, our study provides a comprehensive procedure of *Myers* and *Histogram* in generating the `diff`s and shows the differences between their outputs. To the best of our knowledge, empirical comparisons of different `diff` algorithms in `git diff` command have never been undertaken. In this chapter, we carry out two sequential analyses: systematic mapping and empirical comparisons.

For the systematic mapping, we collect papers from three high ranking journals and eight top international conference proceedings published from 2013 to 2017. We then map 52 identified papers in the following four aspects: frequency of `diff` algorithms, analyzed software artifact, purpose of mining Git repositories, and data origins. The results of the systematic mapping revealed that the advanced `diff` algorithms had not been considered in the previous studies. In terms of the focus of the `git` command, 51 out of 52 papers centralized on mining the code changes. We also found that the purposes of using the `git` command were to get patches (46.2%), followed by metrics collection (25%), and bug-introducing change identification (SZZ algorithm) (23.1%). Regarding the dataset, most papers investigated OSS projects (98%), even though the remaining work analyzed industrial data.

In our empirical analyses, we conduct three comparisons based on the most popular usages of `git diff` found in our mapping study: collecting metrics, identifying bug introduction, and getting patches. We investigate the disagreement between two `diff` algorithms: *Myers* and *Histogram*, and take a manual mea-

[2]`https://github.com/git/git/blob/77bd3ea9f54f1584147b594abc04c26ca516d987/Documentation/RelNotes/1.7.7.txt#L68-L70`

surement of their quality in generating the `diff` lists. Based on previous related studies, we investigate the code changes from the files in 14 OSS projects that employ Continuous Integration for metrics collection and 10 Apache projects for the bug introduction identification to quantify the differences of the `diff` outputs that resulted from both `diff` algorithms. We analyze the quality of patches derived from *Myers* and *Histogram* by manually comparing their two `diff` from 377 changes, a statistically representative sample of the 21,590 changes identified in the above two comparisons. Our findings show that using various `diff` algorithms in the `git diff` command produced unequal `diff` lists.

This influences the different number of files that have dissimilar added and deleted lines of code in each CI-Java project. The differences of these added and deleted lines that are distinguished by their different number and position range from 0.8% to 6.2% and from 1.4% to 7.6%, respectively. The divergent `diff` outputs also affected the different number of identified files in bug introduction identification. The percentage of files that have different deleted lines of code range from 2.4% to 6.6%. Regarding the result of the patches analysis, we found that, in-code changes, *Histogram* is better in 62.6% files, while *Myers* is better in 16.9% files. However, both `diff` algorithms evenly have a good quality in generating the list of non-code changes.

In sum, the contributions of this work are:

- A systematic survey of studies that use `diff`;
- An analysis of metrics collected from `diff` outputs produced by *Myers* and *Histogram*;
- An analysis of *Myers* and *Histogram* outputs in identifying potential bug-introducing changes;
- A manual comparison between *Myers* and *Histogram* to investigate their output quality.

The remaining parts of this chapter are structured as follows. Section 2.2 presents the application of various category of `diff` algorithms in the literature. Section 2.3 presents a brief explanation of `diff` algorithms used in the `git` command. We explain the differences between two `diff` algorithms in generating

the list of changes. Section 2.4 describes how we conduct a systematic mapping study and present the result of the survey. The overview of the three comparisons and the research questions are presented in Section 2.5. Sections 2.6, 2.7 and 2.8 report our procedures and discuss their results in performing three comparison studies; namely, collecting metrics, identifying bug introduction, and getting patches respectively. In Section 2.9, we discuss the implication of different `diff` algorithms and provide the example, and discuss their threats to validity, and finally we conclude in Section 2.10.

We have provided the data sets used in this chapter publicly on the Web.[3]

2.2 Source Code Differencing

Existing differencing techniques use similarities in names and structure to match code elements at a particular granularity, such as text-based and abstract-syntax-tree-based (AST).

Tree-based differencing techniques are widely used nowadays (e.g., diff in Unix), since they are expected to have better understandability than the text-based. Such AST differencing tools were used in several studies. For example, Change Distilling (CD) that extracts the code changes by finding both a match between the nodes of the compared two abstract syntax trees and a minimum edit script that can transform one tree into the other given the computed matching [33]. In this study, the text-based differencing is used to extract the changes at the beginning of the process as the input before further processed using the proposed AST algorithm. In comparison with textual `diff`, the Change Distiller is able to assign the type of the changes such as declaration or body part of a method, rather than just to a line number. Diff/TS [41] and MTDIFF [30] use moving code to compute the changes. Diff/TS is used to analyze fine-grained structural change between versions of programs but only capable of processing *Python, Java, C*, and *C++* projects, while MTDIFF improves the accuracy of the previous tree-based approaches in detecting moved code. Falleri et al. [32] introduced an algorithm to compute edit scripts at the abstract syntax tree granularity including move actions. In this study, the authors conducted a performance

[3] `https://github.com/yusufsn/DifferentDiffAlgorithms`

study to measure the running time and memory consumption between their proposed algorithm and the other tools, such as `GumTree` and `RTED` algorithm. The classical text `diff` was used to present the reference values when comparing the running time between the involved algorithms. Tree-based differencing approach was also used by Higo et al. [47] to consider copy-and-paste as a type of editing action forming tree-based edit script, and Huang et al. [48] to propose CLDIFF for generating concise linked code differences whose granularity is in between the existing code differencing and code change summarization methods.

Despite many advantages in tree-based differencing techniques, text-based `diff` is widely used for several applications in software engineering research because of its simplicity and lightweight runtime. Therefore, in this chapter we only focus on studying the impact of changing `diff` algorithms, instead of comparing wider categories of differencing techniques.

2.3 Diff Algorithms in Git

`Diff` is an automatic comparison program used to find the disagreements between the older and the newer version of the same file in a storage (including insertions, deletions, document renaming, document movements etc.). The `diff` utility extracts code changes line by line in one file compared to the other file and reports them in a list. The operation of the `diff` program has been fundamentally solved by using the longest common subsequence (LCS) problem initiated by Hunt and MacIlroy [49]. Since its first run on the Unix operating system in 1970, the `diff` command has been widely used in many studies.

The `git diff` command has numerous options in the application of code changes extraction,[4] including extracting changes related to the index and commit, paths on a filesystem, the original contents of objects, or even quantifying the number of changes for each object relatively from the sources. Researchers and practitioners are able to use the variation of these available options depending on their needs in extracting the data, not to mention, the `diff` algorithms. The essence of `diff` algorithms is in contrasting the two sequences and to receive insight of the transformation from the first into the second by a series of operations using the ordered deletion and insertion. The subsequence can be flagged as

[4] `https://git-scm.com/docs/git-diff` (accessed in October 2018)

a change if a delete and an insert concur on the same scope. The `diff` algorithm can be selected with this option `--diff-algorithm=<algorithm>`.

In Git, there are four `diff` algorithms, namely *Myers*, *Minimal*, *Patience*, and *Histogram*, which are utilized to obtain the differences of the two same files located in two different commits. The *Minimal* and the *Histogram* algorithms are the improved versions of the *Myers* and the *Patience* respectively. Each algorithm has its own procedures for finding the items presented in the original document, but absent in the second one and vice versa; as a consequence, different outputs may be produced. Due to the similarity of the basic idea of *Minimal* and *Histogram* algorithms with their precursors, in this chapter we only contrasted the two `diff` algorithms: *Myers* and *Histogram*.

2.3.1 Myers

Myers algorithm was developed by Myers [75]. In the `git diff` command, this algorithm is used as the default. The operation of this algorithm traces the two primary identical sequences recursively with the least edited script. Since the *Myers* only notices the sequences which are actually equal in both, the comparison between the other prior and posterior subsequences is executed repetitively for the entire remaining sequences.

Figure 2 indicates several code changes from the first into the second version of the same file (`GuiCommonElements.java`) taken from Openmicroscopy project.[5] As can be seen in the figure, the code between line 673 and 689 in the first version transformed to the newer version between line 673 and 693. Figure 3 shows how *Myers* algorithm generates the `diff` output from the code changes in Figure 2. First, the *Myers* scans the lines of code sequentially from the first line in both versions of the same file to find a line pair that match up each other. Once the exact same lines between the two versions of the file are found by the algorithm, the lines will be considered as the unmodified lines (e.g. pair of lines 673-675 in both versions in Figure 3a). The algorithm then do the same scanning to extract the other pairs of matched lines for the remaining lines of code repetitively, as depicted in Figure 3b and Figure 3c. In Figure 3c, we can see all unmodified lines

[5] `https://github.com/openmicroscopy/openmicroscopy/commit/844e0fde447d2d069fb17c480e95acf4d372afc4#diff-07322c93ef4fb3f0dd245932b74b10e1`

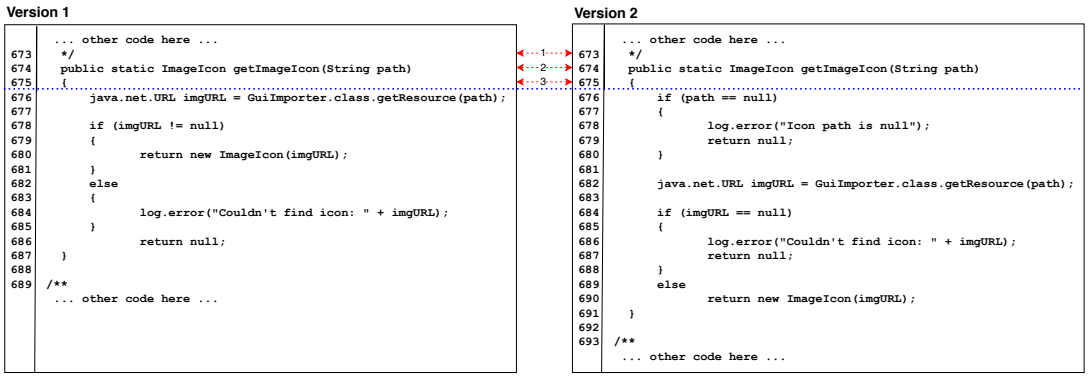
Version 1

```
     ... other code here ...
673  */
674  public static ImageIcon getImageIcon(String path)
675  {
676      java.net.URL imgURL = GuiImporter.class.getResource(path);
677
678      if (imgURL != null)
679      {
680             return new ImageIcon(imgURL);
681      }
682      else
683      {
684             log.error("Couldn't find icon: " + imgURL);
685      }
686             return null;
687  }
688
689  /**
     ... other code here ...
```

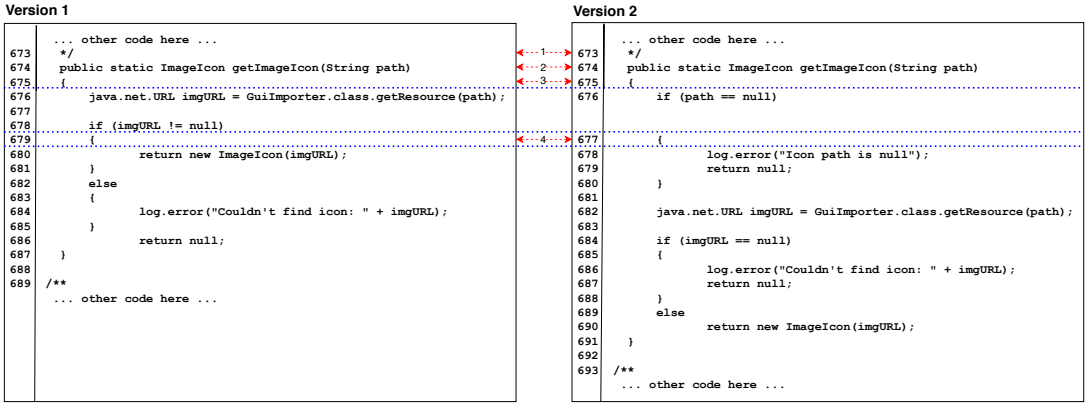
Version 2

```
     ... other code here ...
673  */
674  public static ImageIcon getImageIcon(String path)
675  {
676      if (path == null)
677      {
678             log.error("Icon path is null");
679             return null;
680      }
681
682      java.net.URL imgURL = GuiImporter.class.getResource(path);
683
684      if (imgURL == null)
685      {
686             log.error("Couldn't find icon: " + imgURL);
687             return null;
688      }
689      else
690             return new ImageIcon(imgURL);
691  }
692
693  /**
     ... other code here ...
```

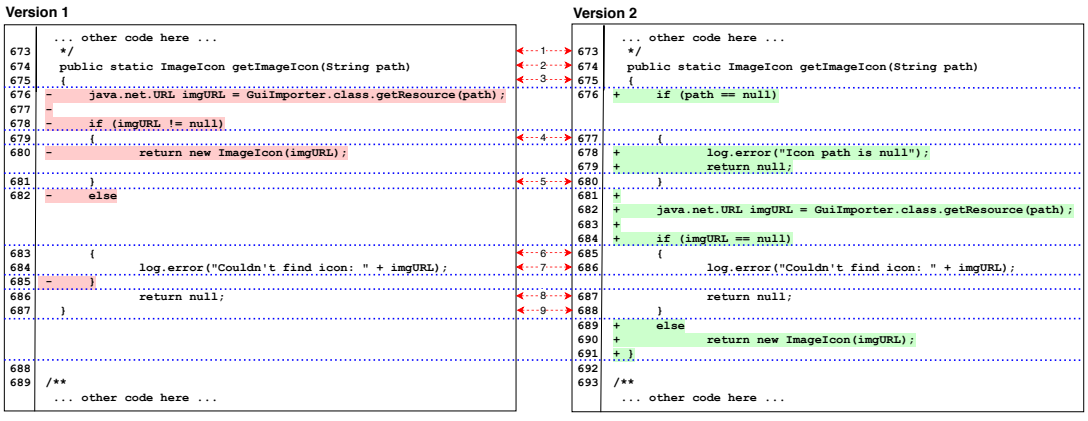
Figure 2: A set of changes from an older file into a newer file



(a) Step 1: pair up the first three matching lines (line 673-675 in both versions)



(b) Step 2: pair up the fourth matching lines (line 679 in version 1 and line 677 in version 2)



(c) Step n: final step after pairing up all matching lines

Figure 3: How *Myers* identifies the `diff`

found by the *Myers* algorithm: pair of line 673-675 in both versions, pair of line 679 in Version 1 and 677 in Version 2, 681 and 680, 683 and 685, 684 and 686, 686 and 687, and 687 and 688). The unpaired lines in Version 1 are subsequently considered as the deleted lines, while the unpaired lines in Version 2 are counted as the added lines. As a result, the *Myers* algorithm produces the paired and unpaired lines from the first and second version of the same file in sequence, as illustrated in Figure 5a.

The *Minimal* algorithm is the extended version of *Myers*. The operation of this algorithm in finding the changes resulted from a comparison of two objects resembling the *Myers*, but an extra attempt was made to keep the patch size as minimal as possible.[6] As a result, the `diff` lists created using this algorithm are often identical with the *Myers*. If we apply the *Minimal* algorithm to the code in Figure 2, the `diff` output is shown in Figure 5a as well.

A major limitation of the *Myers* algorithm is it frequently catches the blank lines or parentheses and conforms the lines to match instead of catching the line that is "unique" (i.e. lines that occur exactly once or the least occurrence in both versions), such as code of function declaration, or a line of assignment. Consequently, the *Myers* sometimes produces unclear `diff` lists that do not describe the actual code changes. The position between changed code and code that replace them is often written distantly in inappropriate lines, or located separately in a line that does not represent the modification. Additionally, there is occasionally a conflict of identification of the changed code; for example, the code in lines 4 and 15 in Figure 5a. In fact, these lines of code were derived from the same unique line that was unmodified. Using the *Myers* algorithm, this unique line is detected as a changed code even though it does not show the alteration. This makes it possible to cause misidentification of a code change.

2.3.2 Histogram

The *Histogram* algorithm is the enhanced version of *Patience*, which was built by Bram Cohen who is renowned as the BitTorrent developer.[7] It supports low-occurrence common elements which are applied to improve efficiency. The *His-*

[6] `http://fabiensanglard.net/git_code_review/diff.php` (accessed in December 2018)

[7] `https://alfedenzo.livejournal.com/170301.html` (accessed in December 2018)

togram was initially built in jgit[8] and was introduced in `git 1.7.7`.

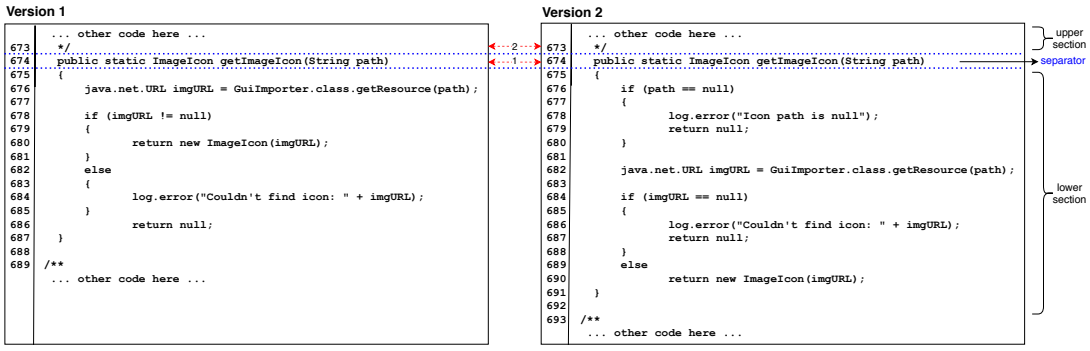
The *Patience* marks the important lines within the text by focusing on the lines that have the smallest number of occurrences, but are essential. This `diff` automated procedure is an LCS-based problem as well, but it uses a different technique. The *Patience* only notices the longest common subsequence of the marked lines attained from the lines which emerge uniquely in a specific range and the lines that are also written precisely similar in both files. This implies that the lines having a single bracket or a new line are usually disregarded; otherwise, the *Patience* retains the distinctive line such as a function definition.

The *Histogram* strategy works similarly to the *Patience* by developing a histogram of the appearances for every line in the first version of a file. Every element in the second version is subsequently shown to match with the first sequence in an orderly way to find the existences of the elements and to count the occurrences. If the elements exist and their presences are less than in the first sequence, they are expected to be a potential LCS. Once the screening is finished for the second sequence, the lowest occurrence of LCS is marked as the separator. Two sections resulting from the partition (i.e. section 1 represents the area before the LCS, while section 2 represents the region after the LCS), are then executed repetitively using the same process as the beginning of the algorithm. This means that the *Histogram* performs similarly to the *Patience* if a unique common element exists in both files; otherwise, it selects the element that has the least occurrences. In comparison with the other two `diff` algorithms, (i.e. the *Myers* and the *Patience*), the *Histogram* nevertheless, has been declared much quicker.[9]

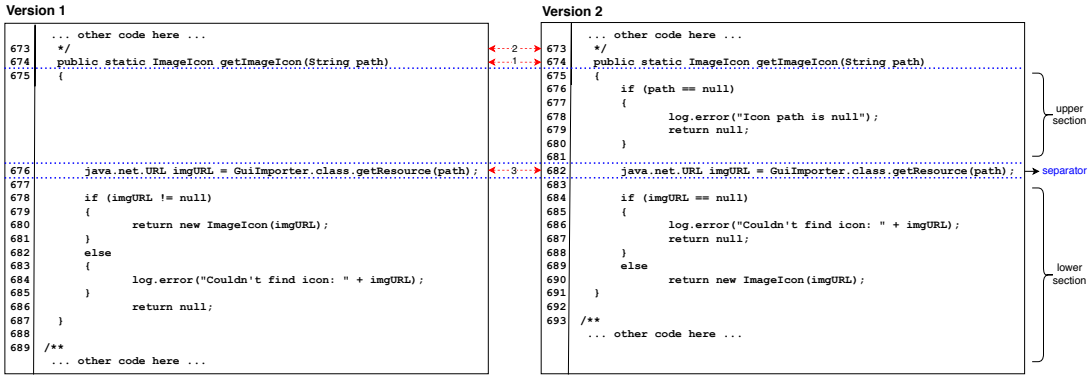
To easily understanding the *Histogram* generates the `diff` output from Figure 2, we describe the procedure in Figure 4. First, the *Histogram* scans all elements in the first version of the file to count the appearances of each line. Every line in the second version is extracted to match with the element in the first version sequentially to find the exact same line and count the occurrences. If the algorithm found the lines in both versions are match and their presences are unique (i.e. occurs exactly once or have the lowest occurrences in both), they are

[8] `http://eclipse.org/jgit/`

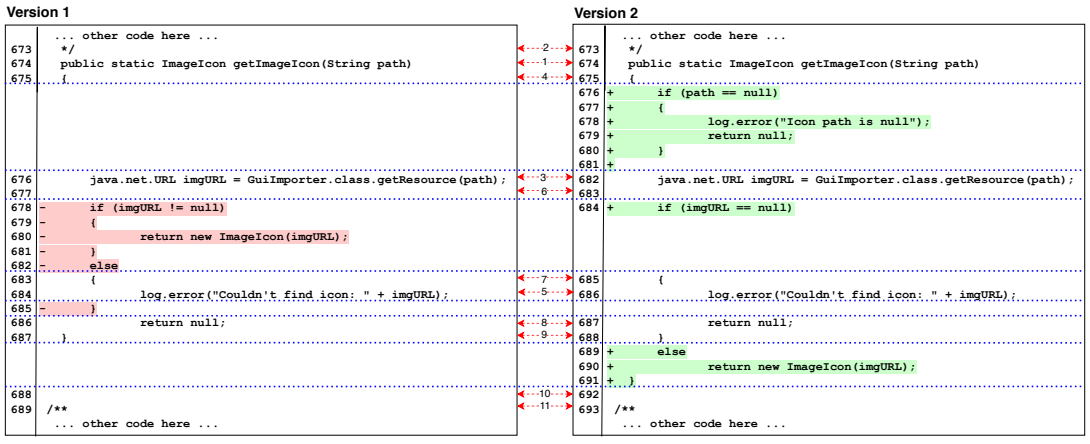
[9] `https://marc.info/?l=git&m=133103975225142&w=2` (accessed in December 2018)



(a) Step 1: pair up the first and second matching unique lines (line 674 in both versions and line 673 at the upper section of the partition)



(b) Step 2: pair up the third matching unique line at the lower section of the partition (line 676 in version 1 and line 682 in version 2)



(c) Step n: final step after pairing up all matching unique lines

Figure 4: How *Histogram* identifies the `diff`

considered as the potential LCS which is then marked as the separator. As shown in Figure 4a, line 674 in both versions are marked as the first separator. Two sub-sections are created after this slicing, that is, the area before and after the separator. Within those sub-sections, the algorithm find more unique pairings; lines that are not unique when scanning the entire document can be unique when the algorithm consider a sub-section. The same process is then applied to both sub-sections. The *Histogram* compares line 673 in the upper section in both versions, and lines 675-689 in Version 1 with lines 675-693 in Version 2 in the lower sections. Due to the least appearances of line 673 only in the upper section in both versions, thus, this line is expected to be the second separator. In the lower section, the scanning process is re-executed from the beginning. As illustrated in Figure 4b, the process yields a new separator (i.e. line 676 in Version 1 and line 682 in Version 2) and two new sub-sections (i.e. line 675 in Version 1 and line 675-681 in Version 2 as the upper section, and line 677-689 in Version 1 and line 683-693 in Version 2 as the lower section). The same process is subsequently executed repetitively for the two new sub-sections resulting from the partition. Figure 4c shows the final step after comparing all elements in both versions. All potential LCS that are marked as the separator are expected to be the unmodified lines, while the other lines are considered as the deleted lines in Version 1 and the added lines in Version 2. As a result, the `diff` output is generated as described in Figure 5b.

In contrast with the *Myers*, the *Histogram* algorithm provides `diff` results that are easier for software archives miners to understand, as the *Histogram* more clearly separates the changed code lines. This algorithm splits the changed lines of code by trying to match up unique lines between two versions of the same file. Thus, it will reduce the occurrences of conflict (i.e. a line of an unchanged code identified as a changed code, so that in the `diff` list, this code is written in duplicate as both a deleted and inserted code). For example, if we extract the differences between the two versions of the same file in Figure 2 using the *Histogram* in the `git diff` command, we obtain the output as depicted in Figure 5b. A unique line of code in line 10 of Figure 5b is not detected as a changed code due to its role as the benchmark to match the line, where this line is identified as a changed code in case of *Myers*. This influences the sequences of the other

```
 ... other code here ...
      */
     public static ImageIcon getImageIcon(String path)
     {
-        java.net.URL imgURL = GuiImporter.class.getResource(path);
-
-        if (imgURL != null)
+       if (path == null)
         {
-                return new ImageIcon(imgURL);
+                log.error("Icon path is null");
+                return null;
         }
-        else
+
+        java.net.URL imgURL = GuiImporter.class.getResource(path);
+
+        if (imgURL == null)
         {
                 log.error("Couldn't find icon: " + imgURL);
-        }
                 return null;
         }
+        else
+                return new ImageIcon(imgURL);
+    }

     /**
 ... other code here ...
```

(a) *Myers'* diff

```
  ... other code here ...
      */
     public static ImageIcon getImageIcon(String path)
     {
+       if (path == null)
+       {
+                log.error("Icon path is null");
+                return null;
+       }
+
         java.net.URL imgURL = GuiImporter.class.getResource(path);

-        if (imgURL != null)
-        {
-                return new ImageIcon(imgURL);
-        }
-        else
+        if (imgURL == null)
         {
                 log.error("Couldn't find icon: " + imgURL);
-        }
                 return null;
         }
+        else
+                return new ImageIcon(imgURL);
+    }

     /**
  ... other code here ...
```

(b) *Histogram*'s diff

Figure 5: `Diff` outputs produced by *Myers* and *Histogram*

changed code. An additional block of *if* condition is written between lines 4 and 9 where it should be placed. This block of code is clearly understood as the new code inserted before the statement of the assignment code (code in line 10 which is used as one of some unique lines to match). It is also obvious that the code between lines 12 and 16 were replaced by one line of code in line 17, while the closing curly brace in line 20 was omitted from the files, and three new lines of code (line 23, 24 and 25) were added at the end of the code in Figure 5b.

2.4 Systematic Mapping: How Previous Studies Used Git Diff?

To understand the ways in which the previous studies use `diff`, we conducted a systematic mapping of papers that used the `git diff` command for their studies. As described by Petersen et al. [83], a systematic mapping study can provide and visualize a statistical insight of a study domain by classifying and quantifying the number of publications related to the research interest within the same study domain. The main activity of the method was searching the relevant literature from a wide range of publications including journal articles, books, documented archives and scripts.

We performed a systematic mapping as we intend to: (i) draw an overview of the research area through quantification in a structured way [61], (ii) confirm the knowledge in the currently published studies [84]. A systematic mapping is reliable because the findings are repeatable and consistent across the time [118], and they are beneficial for better reporting of some empirical findings of the primary studies [15].

To understand how recent studies used `git diff`, we prepared the following research questions for this systematic mapping.

- Which *diff* algorithm is used?
- What kind of software artifact is analyzed, code or other documents?
- What are purposes of using *diff*?
- Where does the data source come from, OSS or industry?

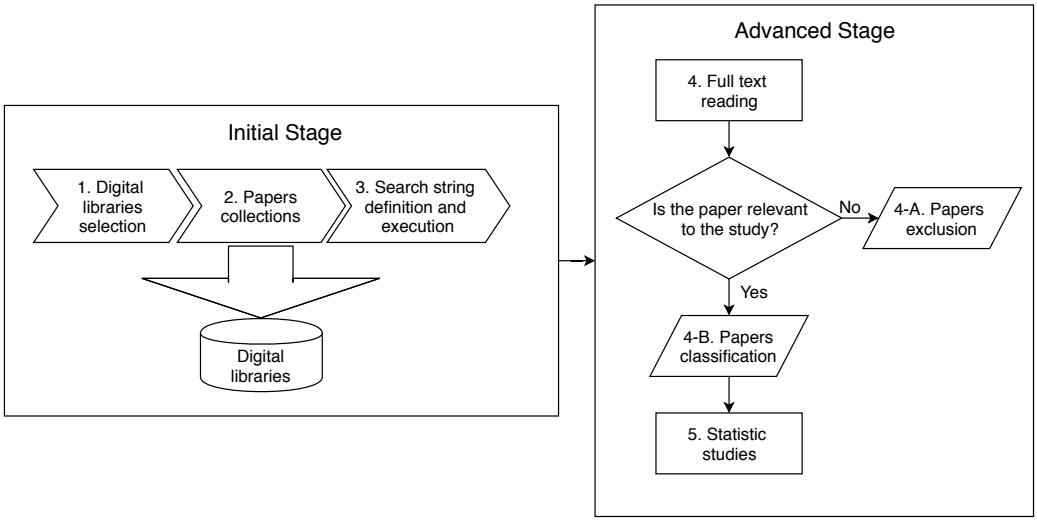


Figure 6: Design of the Survey Procedure

2.4.1 Procedure

Figure 6 illustrates an overview of our systematic mapping procedure, which is divided into an initial stage and an advanced stage. The first stage has three steps including a digital libraries selection, papers collection, search string definition and initial search execution. The second stage begins with repetitive manual exclusion by narrowing the search terms and the reading of full papers, followed by paper classification, and statistical analyses.

Step 1: Digital Libraries Selection. The selection of appropriate literature is essential to guarantee high-quality papers and to grasp the state-of-the-art issues in the software engineering field [55]. We specifically targeted papers which were published in high ranking journals and conference proceedings of the software engineering area. To maximize the probability of finding highly relevant good quality articles, we used three specific digital resources: ACM Digital Library,[10] IEEE Xplore,[11] and SpringerLink.[12] Table 1 shows the list of the

[10] `https://dl.acm.org/`

[11] `https://ieeexplore.ieee.org/`

[12] `https://link.springer.com/`

Table 1: List of Surveyed SE Journals and Conferences

Category	Name of Journal or Conference	IF or Rank
Journal	IEEE Transactions on Software Engineering (TSE)	IF = 3.331
	Empirical Software Engineering (EMSE)	IF = 2.933
	ACM Transactions on Software Engineering and Methodology (TOSEM)	IF = 1.946
Conference	ACM Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA)	Rank = A*
	ACM-SIGPLAN Conference on Programming Language Design and Implementation (PLDI)	Rank = A*
	International Conference on Software Engineering (ICSE)	Rank = A*
	Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)	Rank = A*
	Automated Software Engineering (ASE)	Rank = A
	International Conference on Software Maintenance and Evolution (ICSME)	Rank = A
	International Conference on Mining Software Repositories (MSR)	Rank = A
	International Symposium on Software Testing and Analysis (ISSTA)	Rank = A

publication sources used in our survey including their impact factors (IF)[13] and rankings published in 2018 CORE Rankings.[14] We gathered published papers from these three digital sources between the years of 2013 and 2017.

Step 2: Papers Collection. To reduce bias in the context of the study, we only collected technical papers. Papers which did not meet our criteria (i.e shorter-than-10-page papers, editorials, panels, poster sessions, and opinions) were excluded. As depicted in Figure 7, by applying our criteria, we sourced 3,057 papers in total from the three digital sources in a 5-year time span.

Step 3: Search String Definition and Execution. In this step, we formulated search keywords to filter the targeted papers into more specific works that use the `git diff` command. We defined three specific search terms related to the command, namely *git*, *log* and *diff*. Papers that contained one of three words with an exact match without affixes or suffixes (e.g. *github, blog, logarithm, logging, different, difficult* etc.) were collected. Since we only focus on the study

[13] `https://www.scimagojr.com/`

[14] `http://www.core.edu.au/conference-portal/2018-conference-rankings-1` (accessed in November 2018)

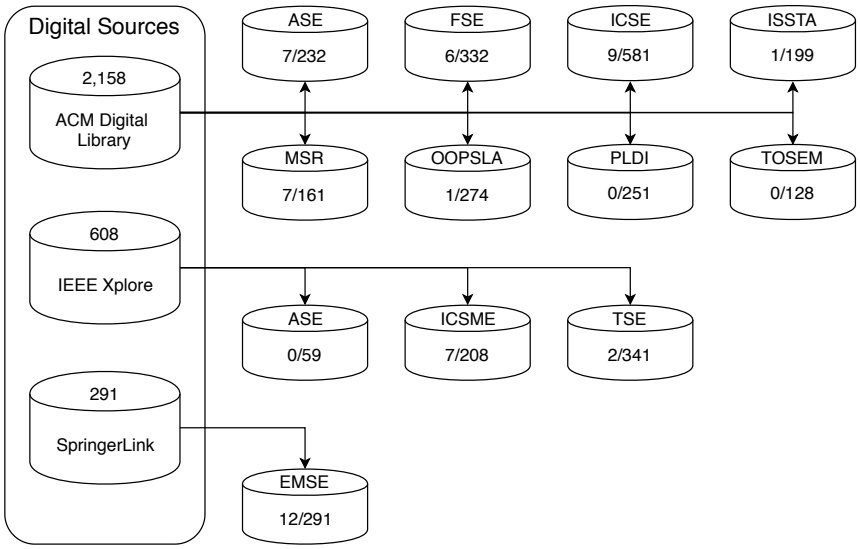


Figure 7: Number of collected papers from each source

that used `diff` command in `git` repositories, papers that do not exactly mention at least one of the three keywords are excluded despite they use other terms such as *differencing* which might indicates the implementation of the other `diff` tools. The command `git log` was also targeted because this command can produce *diff* with specific options. By using these three search terms, all papers extracted from the databases were then manually scanned in full text. Consequently, only published works containing these three search strings were included. As a result of Step 3, we were able to identify 137 papers.

Step 4: Full Text Reading. To ensure the collected previous studies are relevant to our objectives, we then performed a full text reading of the papers. This process was undertaken by the first and the second authors to avoid obscurity and to separate the primary studies more exhaustively based on their contents. We applied the inclusive and exclusive criteria to the full paper which is described in Table 2. Papers that fit the inclusive criteria were kept for further processing while other papers that met the exclusive criteria were excluded from the study. After this step, we had 52 papers.

Table 2: Inclusive and Exclusive Criteria

Group	Criteria
Inclusive	Paper mentions the `git` command. Paper applies the `git` command to extract the data from `Git` repositories for further analyses in support of their works.
Exclusive	Paper does not use the `git` command as part of their studies. For example, `git diff` is used only for motivating examples.

2.4.2 Results of the Mapping

Figure 8 indicates the distribution of the number of papers in each journal and conference in the last 5 years. As can be seen in the heat map, all journals and conference proceedings published the works related to the `git diff` command application in at least one paper in 5 years except for the PLDI and TOSEM. Most papers that applied the `git diff` command are published on EMSE especially in 2017, accounting for 6 papers.

2.4.2.1 Which `Diff` Algorithm Is Used?

Out of the 52 primary studies,we identified the application of different `diff` algorithms in the command in extracting the changes. Of particular note is that even though most instructions applied different options in the use of the `git` command to extract the required data, none of the previous selected works considered different `diff` algorithms. This shows that all of the collected studies used *Myers* as the default algorithm.

2.4.2.2 What Kind of Software Artifact Is Analyzed?

To understand the components that were extracted using the `git` command in the previous studies, two main focuses emerged as our parameters to classify the documents; namely, code changes and license changes as depicted in Figure 9. As can be seen in the figure, code changes were prominently the focus for researchers in extracting software repositories using the `git` command over five years. Thus, in our comparisons we analyze code changes extracted from the data source.

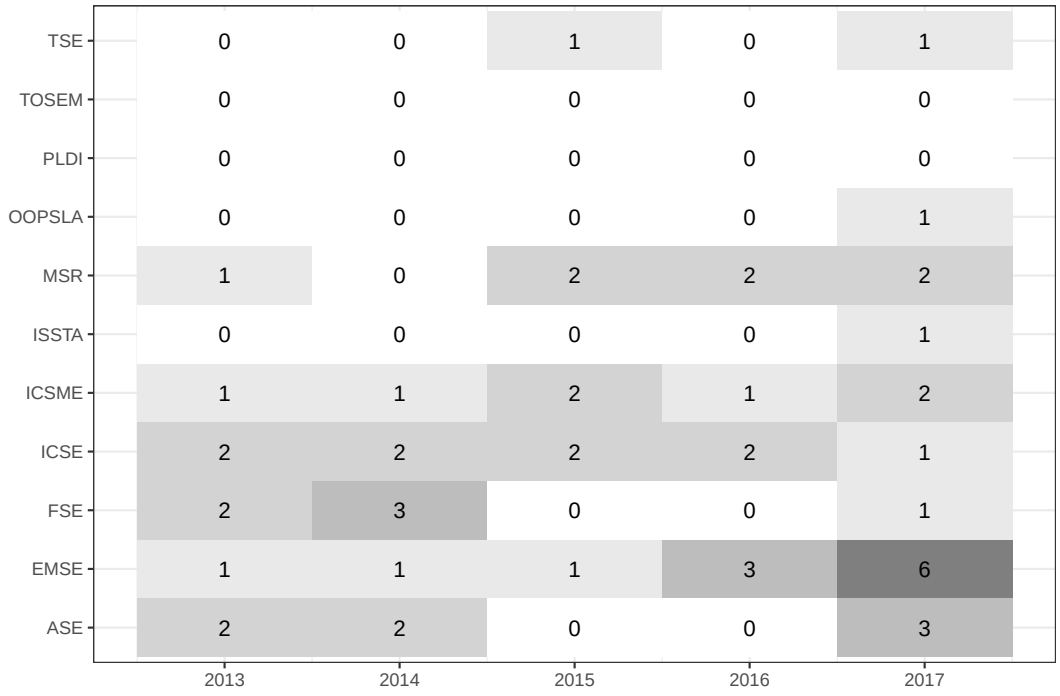


Figure 8: Number of papers per journals and conferences between 2013 and 2017

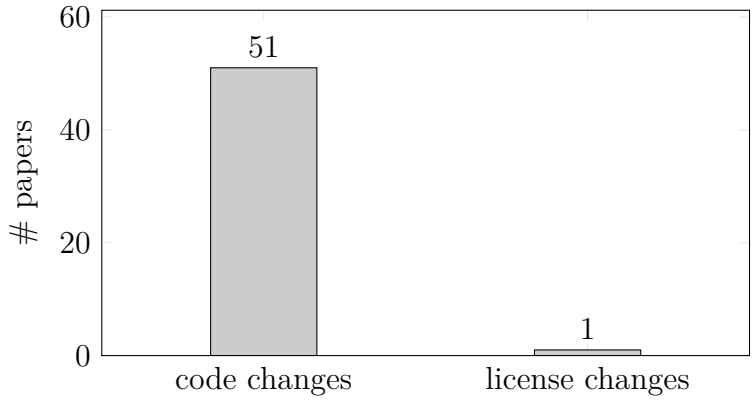


Figure 9: Number of papers based on parameter searched using `git` command

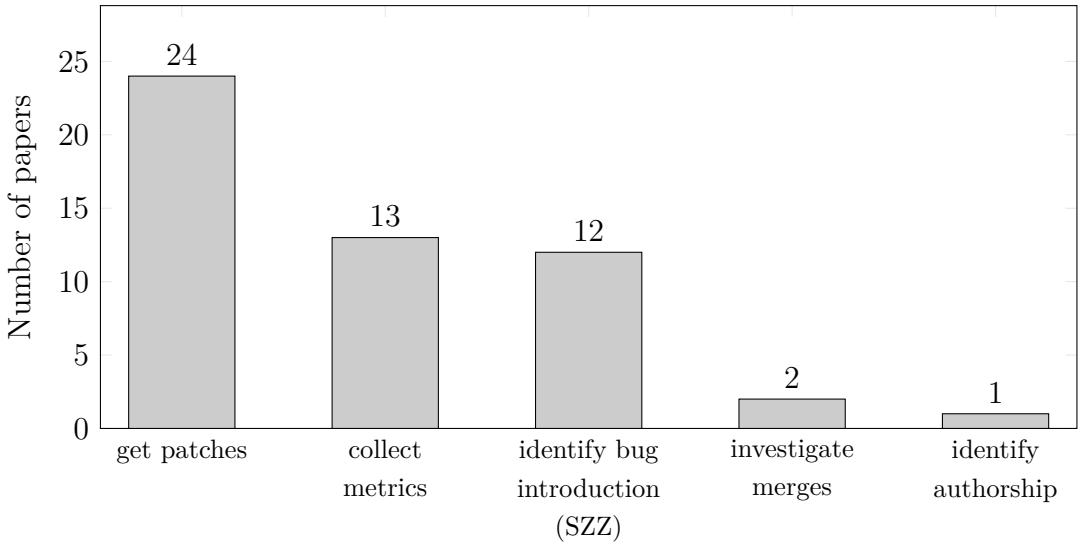


Figure 10: Number of papers classified with the purpose of using the `git` command

2.4.2.3 What Are Purposes of Using `Diff`?

By reading the papers manually, we summarized the purposes from the extraction of software development records and grouped them into five categories, as can be seen in Figure 10.

From the figure, we see that the most common purposes is to *get patches*, amounting to as many as 24 studies, followed by *collecting metrics* and *identifying bug-introductions*, which covered 13 and 12 studies, respectively. A few studies addressed *merges investigation* and *authorship identification*. This finding motivated us to carry out a further investigation of the impact of different `diff` algorithms in the extraction of the added and deleted lines for metrics collection, bug-introducing change identification, and getting the patches.

2.4.2.4 Where Does the Data Source Come From?

Our intention is to provide a comprehensive understanding of the different outcomes generated by different `diff` algorithms; thus, we need to run a set of tests of the algorithms' implementations in the `git diff` command. From the result of our dataset classification, open source software (OSS) is found to be dominated

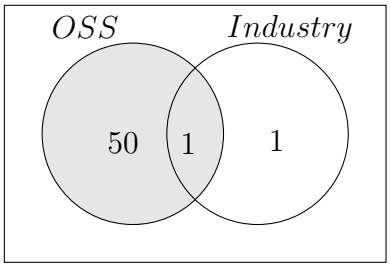


Figure 11: Distribution of the type of data sources used in prior studies

as the data source over the industrial type as illustrated in Figure 11. Therefore, we mine the data from OSS projects to support our comparisons.

2.4.3 Summary

The survey results of the usage of the `git diff` command confirm that the previous studies conducted between 2013 and 2017 did not use various `diff` algorithms to extract the differences between the first and the second versions of the same file. In mining the `diff` lists, they applied the standard commands using a default `diff` algorithm with some additional options, but without considering various `diff` algorithms. We also found that the information most sought after in prior studies was code changes in open source projects. The code changes were mostly utilized to thoroughly investigate counting the number of line changes and to record them in the form of metrics, locating the origin of a bug using a specific method (i.e. SZZ algorithm), and analyzing the patches. The results of these types of analyses obviously rely on the `diff` records produced by an applied `diff` algorithm in the `git diff` commands. Thus, different `diff` algorithms in extracting the line of code changes might differentiate the final result of a study and the conclusion of the description as well.

2.5 Overview of Comparisons and Research Questions

The findings from our systematic mapping revealed the three most common purposes for using the `git diff` command. This encouraged us to undertake comparison analyses between the *Myers* and *Histogram* algorithms in three applications: metrics, the SZZ algorithm, and patches. Our intention is to investigate the level

of differences between the two `diff` algorithms used in these three applications and their possibility of affecting the result of studies. To achieve these goals, we address the following research questions:

RQ$_1$: *Can the values of `diff`-related metrics become different because of different `diff` algorithms?*
For metrics (Section 2.6), equal and unequal changed lines in the files identified by the two `diff` algorithms were calculated based on two factors: the quantity and the position of the line of code. We then compared the quantity of the files that have the same and different added and deleted lines of code to understand the significance of the differences of both algorithms in providing the `diff` records.

RQ$_2$: *Are the results of bug-introducing change identification different because of different `diff` algorithms?*
The result of locating bug-introducing changes using the SZZ algorithms relies on the `diff` results. In Section 2.7, we applied the *Myers* and *Histogram* algorithms in the `git diff` command to know whether the `diff` lists affect the result of bug-introducing change identification.

RQ$_3$: *Which `diff` algorithm is better in generating a good `diff`?*
Lastly, we compared the quality of the identified patches manually. In Section 2.8, we investigate 377 changes, a statistically representative sample of the 21,590 changes identified in the above two comparisons.

In our three comparisons, to extract the changes, we apply the `git` command: `git diff -w --ignore-blank-lines --diff-algorithm=<algorithm name> <parent commit ID> <commit ID> -- <filename>`. We use the same options `-w` and `--ignore-blank-lines` to ignore whitespace and the changes whose lines are all blank. The use of various options is common according to the purposes to what extent the `diff` command generates the code changes. However, since our focus is comparing *Myers* and *Histogram* as the `diff` algorithm that can be used at the same circumstances, we do not consider to investigate the impact of other options.

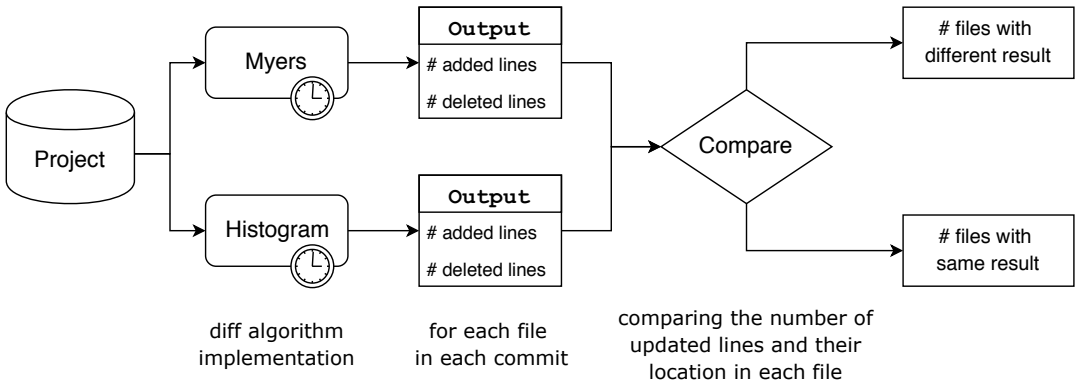


Figure 12: Overview of the metrics collection procedure

2.6 Comparison: Metrics (RQ_1)

RQ_1: *Can the values of* `diff`*-related metrics become different because of different* `diff` *algorithms?*

2.6.1 Analysis Design

As illustrated in Figure 12, we investigate the following two basic `diff`-related metrics with two `diff` algorithms: *Myers* and *Histogram*.

NLA The number of added lines in a file.

NLD The number of deleted lines in a file.

For our empirical analysis, we collected the Git repositories of 14 projects used in the previous study [89], which are identified in our systematic mapping as a study utilizing `git` for collecting metrics. The targeted 14 projects are OSS that employ Continuous Integration (CI) and are written in Java. The descriptions of the projects and the number of commits in the *master* branches are shown in Table 3.

We investigated all modified files in all commits in the master branches. To extract the NLA and NLD from the file, we implement the `git` command: `git diff -w --ignore-blank-lines --diff-algorithm=<algorithm> <parent commit ID> <commit ID> -- <filename>`. We considered the results the same if the values of both NLA and NLD were the same with the two algorithms; otherwise,

Table 3: Targeted 14 open-source Java projects following the previous study [89]

Project Name	Description	#Commits
Apache Storm	Distributed realtime computation system	9,317
Butterknife	View binding library for Android	836
Crate	Distributed SQL database	8,646
Hystrix	Interactions controller library between distributed systems	2,106
JabRef	Reference manager application that uses BibTex	11,940
jcabi-github	Object oriented wrapper of GitHub API	2,521
Openmicroscopy	Application to store biological microscopy light data in a standard format	46,543
Presto	Distributed SQL query engine for big data	13,561
RxAndroid	RxJava bindings for Android	461
SpongeAPI	A minecraft plugin API	2,479
Spring Boot	A framework to create Java applications	17,087
Square OkHttp	An HTTP+HTTP/2 client for Android and Java applications	3,171
Square Retrofit	An open source library to make HTTP communication simpler	1,576
WordPress-Android	WordPress for Android OS	30,295

the results were considered different. However, several software engineering tasks that rely on such metrics do not consider the position of the added and deleted lines, where different position of the changed lines can be occurred by chance despite the same metrics value. We conjecture that different number and position of changed lines can have different impact on empirical studies. Thus, we investigated the disagreement of the identified change locations separately. If the positions of each changed line of code were the same, we considered the results the same; otherwise, the results were considered different. File-level and commit-level results are discussed to see how the different results can appear in a different granularity.

2.6.2 Results

Table 4 summarizes the result from the comparison between two `diff` algorithms in 14 projects. From the total number of modified files identified by both al-

Table 4: Total number of files that have the same and different values in metrics (NLA and NLD) and the position of changes.

Project	#Files	Metrics (NLA and NLD)				Locations of Changes			
		#Same	%	#Different	%	#Same	%	#Different	%
Apache Storm	22,011	21,278	96.7%	733	3.3%	20,979	95.3%	1,032	4.7%
Butterknife	1,873	1,804	96.3%	69	3.7%	1,774	94.7%	99	5.3%
Crate	44,463	43,522	97.9%	941	2.1%	42,723	96.1%	1,740	3.9%
Hystrix	3,310	3,192	96.4%	118	3.6%	3,097	93.6%	213	6.4%
JabRef	55,988	54,375	97.1%	1,613	2.9%	53,609	95.7%	2,379	4.3%
jcabi-github	6,218	6,170	99.2%	48	0.8%	6,131	98.6%	87	1.4%
Openmicroscopy	118,349	115,126	97.3%	3,223	2.7%	112,548	95.1%	5,801	4.9%
Presto	73,572	72,471	98.5%	1,101	1.5%	71,455	97.1%	2,117	2.9%
RxAndroid	627	613	97.8%	14	2.2%	603	96.2%	24	3.8%
SpongeAPI	10,757	10,584	98.4%	173	1.6%	10,395	96.6%	362	3.4%
Spring Boot	62,137	60,805	97.9%	1,332	2.1%	60,214	96.9%	1,923	3.1%
Square OkHttp	7,345	7,206	98.1%	139	1.9%	7,108	96.8%	237	3.2%
Square Retrofit	3,473	3,394	97.7%	79	2.3%	3,351	96.5%	122	3.5%
WordPress Android	58,188	54,555	93.8%	3,633	6.2%	53,789	92.4%	4,399	7.6%

gorithms, we counted the quantity of files in each commit that have same or different number values of NLA and NLD metrics. Similarity, the number of same and different results in changed locations are shown in the table.

We see that the percentages of different metric values are between 0.8% and 6.2%. Considering the different results in locations of changes, ranging from 1.4% to 7.6%, we found that quite a few portions of the metric values are same even though the identified locations are different.

To further explore of the disagreements between *Myers* and *Histogram*, we calculated the number of commits influenced by the different number of code changes and the locations in the `diff` output of files. In each project, we counted the sum of files that have the same and different quantity and the position of lines inserted and removed from each commit across the project. A single commit may contain more than one modified file. If a commit recorded at least one file having unequal changed lines of code either in their number or their location, we classified this commit as 'different'. On the other hand, if all files in a commit had identical

Table 5: The number of commits that contain a different number and the position of added and deleted lines of code in a file

Project	#Commits	Metrics (NLA and NLD)		Locations of Changes	
		#Different	%	#Different	%
Apache Storm	9,317	395	4.2%	587	6.3%
Butterknife	836	34	4.1%	48	5.7%
Crate	8,646	707	8.2%	1,202	13.9%
Hystrix	2,106	87	4.1%	160	7.6%
JabRef	11,940	881	7.4%	1,317	11.0%
jcabi-github	2,521	42	1.7%	70	2.8%
Openmicroscopy	46,543	2,340	5.0%	3,674	7.9%
Presto	13,561	816	6.0%	1,423	10.5%
RxAndroid	461	11	2.4%	18	3.9%
SpongeAPI	2,479	128	5.2%	235	9.5%
Spring Boot	17,087	954	5.6%	1,447	8.5%
Square OkHttp	3,171	106	3.3%	181	5.7%
Square Retrofit	1,576	61	3.9%	90	5.7%
WordPress Android	30,295	2,108	7.0%	3,050	10.1%

changed lines, we categorized the commit in the 'same' class. In this process, we only notify the files that have an unequal number and location of the lines of code.

Our results show that several changed files impacted by the changed lines have similar commits. We grouped the same commits from these several files that contain different changed lines of code into a single commit. We then summarized the percentage of commits that have a different number and position of the changed lines of code resulting from the usage of the *Myers* and *Histogram* algorithms in the `git diff` command as described in Table 5.

In general, our comparisons revealed that the data extraction using two `diff` algorithms in the command produced identical `diff` lists for most files in all commits. However, even though the output has been dominated by the same results for each file in a commit, the `diff` output from the *Myers* and *Histogram* recorded several files that have different added and deleted lines. These disagreements im-

pacted the dissimilar number of commits that have files containing changed lines of code. The level of differences in the number of commits influenced by the amount of lines of code are adequately high, ranging from 1.7% to 8.2%, while the unequal location of lines affects the level of differences in the quantity of commits from 2.8% to 13.9%.

2.6.3 Summary

The finding from the metrics comparison provides clear evidence that the use of multiforms of `diff` algorithms might differentiate the `diff` lists. Since the metrics are insensitive to differences in change locations, the same values can be obtained even if identified change locations are different. However, we see that different metric values were obtained from 0.8% to 6.2% in the file-level and 1.7% to 8.2% in the commit-level. These differences can have impacts on studies using `diff`-related metrics.

2.7 Comparison: SZZ Algorithm (RQ_2)

RQ_2: *Are the results of bug-introducing change identification different because of different* ***`diff`*** *algorithms?*

2.7.1 SZZ Algorithm

The SZZ algorithm proposed by Śliwerski et al. [101] is an approach to identify bug-introducing changes. The SZZ uses a bug-tracking system (e.g. Bugzilla) as the reference to link archived versions of a software (e.g. CVS). Figure 13 depicts the basic idea of the SZZ algorithm.

The SZZ algorithm first identifies bug-fixing commits by searching *bug report identity numbers (bug ID)* in log messages, which have been written by developers when they fix bugs. The commit ID of this bug-fixing commit is subsequently used to track the previous commit (parent commit). The code changes are extracted by applying `diff` to find the differences between the older version of a file in the parent-commit and the newer version of the same file in the bug-fix commit. The identified deleted lines are considered to be candidates of bug-related lines. To identify bug-introducing commits, `cvs annotate` command is used to investigate

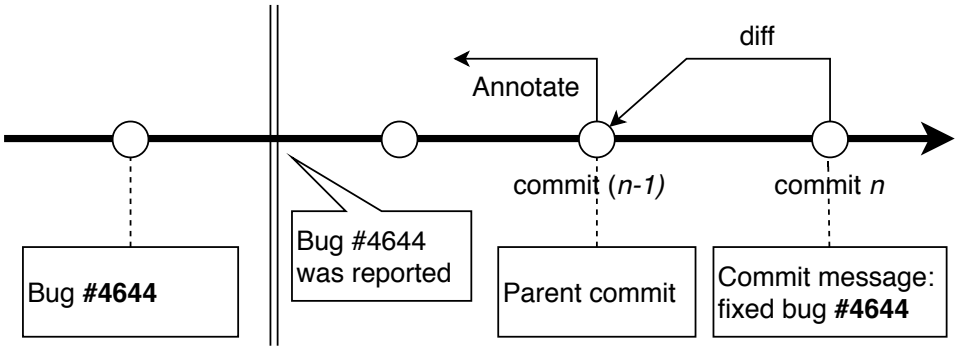


Figure 13: SZZ: Locating bug-introducing changes

when lines are added. Among the candidates of bug-related lines, lines that have been created before the bug reporting time are considered to be *validated* bug-related lines. The commits that introduced those validated bug-related lines are identified as bug-introducing commits.

2.7.2 Analysis Design

Figure 14 describes the validation process of our analysis. For our empirical analysis, we studied 10 open source Apache projects used in the previous study [24], which is identified in our systematic mapping as a study utilizing Git for identifying bug introduction using the SZZ algorithm. The descriptions of projects and the number of commits in the *master* branches are shown in Table 6. We analyzed the impact of using different `diff` algorithms on the original SZZ algorithm. We studied the disagreement between the *Myers* and *Histogram* in the results of the SZZ algorithm based on `diff`.

First, *bug report IDs* in the commit messages are searched with specific keywords (i.e. "bug", "fix", "defect", and "patch" [101]), then the identified commits are marked as candidates of bug-fixing commits. In each candidate bug-fixing commit, we focus on the modified files. The two `diff` algorithms are used to identify deleted lines using the command: `git diff -w --ignore-blank-lines --diff-algorithm=<algorithm> <parent commit ID> <bug-fix candidate commit ID> -- <filename>`. By fetching files in the parent commit ID, we subsequently applied the `git blame` command (similar to `cvs annotate`) to locate

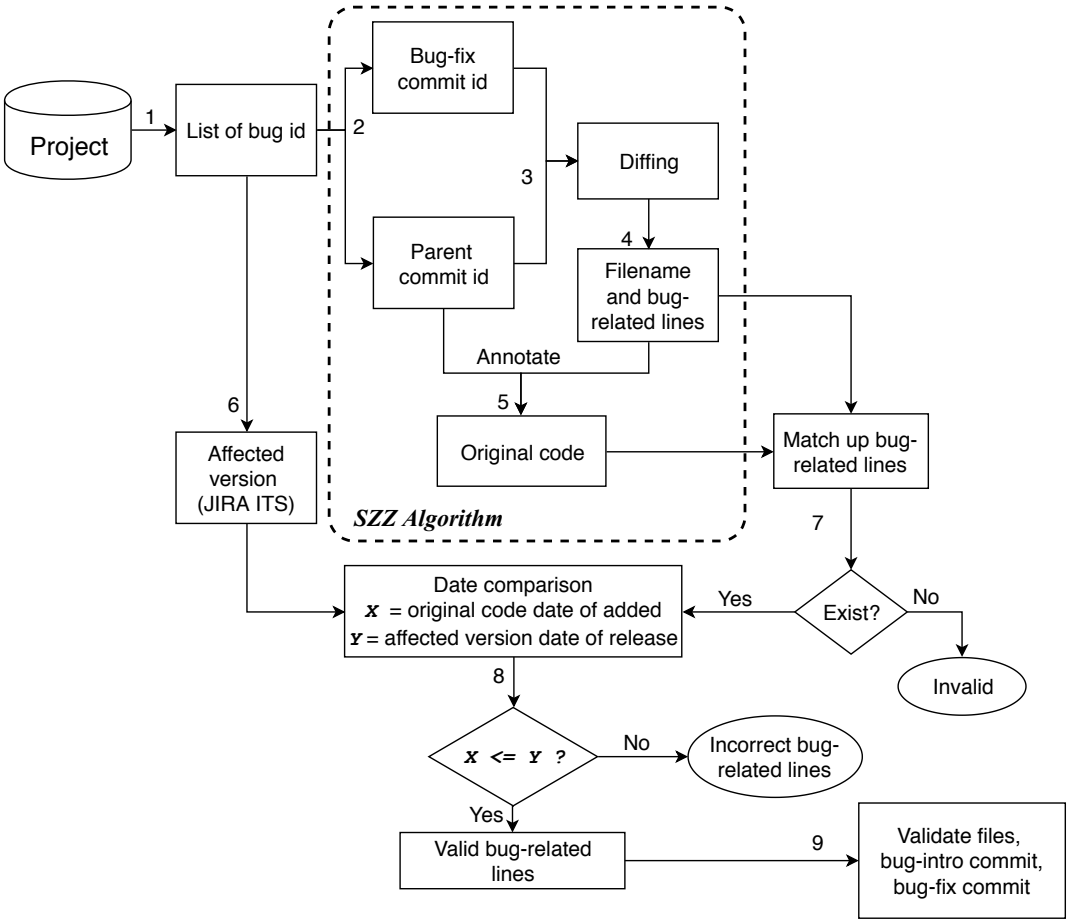


Figure 14: Overview of the validation process of bug-introducing commits

the origin of the deleted lines. Those deleted lines are considered to be candidates of bug-related lines.

Similar to the procedure of da Costa et al. [24], the next step is to find the *affected software versions* of a bug. We extract bug reports and their *affected versions* from the JIRA issue tracking system.[15] If a single bug ID affects more than one version, the earliest version is chosen since the SZZ algorithm targets the initial appearance of a bug. From the collection of *affected-versions*, we compare the dates of the introduction of the candidates of bug-related lines with the release dates of the versions. If the release dates of the affected versions are later than

[15] `https://www.atlassian.com/software/jira` (accessed in April 2018)

Table 6: Overview of the 10 studied Apache projects

Project Name	Description	# Commits
ActiveMQ	Message broker and Java Message Service client	9,962
Camel	A framework to create routing and mediation rules in various domain specific languages	32,124
Derby	Relational database implemented in Java	8,184
Geronimo	Open source application server compatible with Java EE	13,137
Hadoop Common	Collection of common utilities and libraries that support other Hadoop modules	10,509
HBase	A distributed big data store for the Hadoop database	15,091
Mahout	A library to generate the implementation of distributed or scalable machine learning algorithms	3,959
OpenJPA	The implementation of the Java Persistence API	4,864
Pig	High-level mechanism for the parallel programming of MapReduce on Hadoop	3,154
Tuscany	An open source to develop applications based on SCA standard	16,253

the dates of the introduction of the candidates of bug-related lines, we classified them as *valid bug-related lines*; otherwise, we classified them as *invalid*.

With these sets of valid bug-related lines, we validate bug-introducing commits, bug-related files and bug-fixing commits. The validation processes are performed in the opposite direction with the above procedure. A valid bug-introducing commit is a commit that initially adds valid bug-related lines. Files containing bug-related lines are considered to be valid bug-related files. From the candidates of bug-fixing commits, if there is at least one valid associated bug-introducing commit, we consider the candidate bug-fixing commit to be valid, otherwise invalid.

Table 7: Summary of valid bug-related lines, valid files, valid bug-introducing commits, and valid bug-fix commits resulting from *Myers* and *Histogram*

Project	#valid bug-related lines		#valid files		#valid bug-intro commits		#valid bug-fix commits	
	M	H	M	H	M	H	M	H
ActiveMQ	10,671	10,846	1,566	1,565	1,015	1,016	614	613
Camel	12,525	12,626	2,377	2,374	1,514	1,516	716	712
Derby	130,861	131,031	4,372	4,373	1,178	1,180	1,038	1,039
Geronimo	29,543	29,743	2,448	2,448	1,282	1,277	462	462
Hadoop Common	15,053	15,285	805	805	546	550	318	318
HBase	37,558	37,291	2,083	2,079	1,480	1,481	669	668
Mahout	1,542	1,548	182	182	145	144	44	44
OpenJPA	5,160	5,204	794	794	370	370	365	366
Pig	1,789	1,787	205	206	187	187	80	80
Tuscany	750	781	46	46	34	36	16	16

$M = Myers$
$H = Histogram$

2.7.3 Results

Table 7 presents the outputs of the *Myers* and *Histogram* algorithms in the number of valid bug-related lines, files, bug-introducing commits, and bug-fix commits. Two algorithms produced a different number of valid bug-related lines in all 10 projects, which then led to the different number of files, bug-introducing commits, and bug-fix commits.

Similar to the analysis of metrics in Section 2.6, differences in the quantities of changes are relatively small or the same for some projects, because of the insensitivity of change locations.

Since investigating the locations of bug introduction is also important, we perform a comparison of files that have the same and different locations of bug-related lines. Table 8 shows this result. It can be seen that the total number of files that have a different location of the changed code is high in each project, ranging from 2.4% to 6.6%. This means that some files can contain suspicious bug-related lines, only because of different algorithms.

Bringing these data into further analysis, we then summarized the number of valid bug-fixing commits. As shown in Figure 15, all studied projects have a

Table 8: Total number of files that have the same and different positions of valid bug-related lines in all valid bug-fix commits

Project	#Same	#Different	%	Total
ActiveMQ	1,464	103	6.6%	1,567
Camel	2,315	63	2.7%	2,378
Derby	4,198	175	4.0%	4,373
Geronimo	2,318	130	5.3%	2,448
Hadoop Common	777	28	3.5%	805
HBase	1,973	110	5.3%	2,083
Mahout	171	11	6.0%	182
OpenJPA	764	31	3.9%	795
Pig	201	5	2.4%	206
Tuscany	43	3	6.5%	46

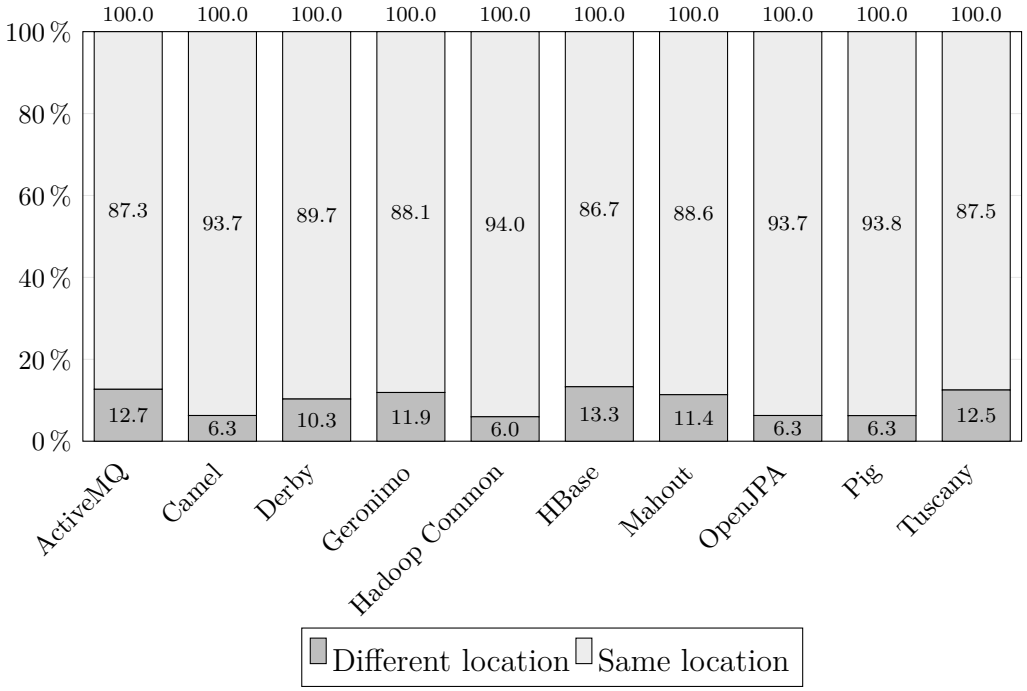


Figure 15: The percentage of valid bug-fixing commits that have the same and different positions of valid bug-related lines

different number of valid bug-fixing commits caused by the different positions of valid bug-related lines resulting from the *Myers* and *Histogram*. The percentage of the different results are between 6.0% and 13.3%, or 9.7% on average. This analysis found evidence that nearly 10% of bug-fixing commits do not guarantee success in locating bug-introducing changes since some deleted lines that were suspected as the candidate bug-introducing changes are different if we applied different `diff` algorithms in the `git diff` command. This is because a valid bug-related line in a file has the possibility of being identified by a particular `diff` algorithm, but it remains undetected while using the other `diff` algorithms.

2.7.4 Summary

The results from the SZZ algorithm confirm that different `diff` algorithms possibly generate different results, from 6.0% and 13.3% in the total of the identified bug-fix commits. The *Myers* and *Histogram* sometimes produced a different number and location of the deleted lines (bug-related lines) in several files. These differences certainly affect the number of disagreement files that have the bug-related lines, the amount of bug-introducing commits, and the bug-fixing commits that actually have the bug-contained files. Therefore, the comparison result indicates that several prior studies that had used the SZZ algorithm to locate bugs have the possibility of producing inaccurate analyses.

2.8 Comparison: Patches (RQ_3)

RQ_3: *Which `diff` algorithm is better in generating a good `diff`?*

2.8.1 Analysis Design

From the previous two comparisons, we showed that different `diff` algorithms can have different results of metrics collection and bug-introduction identification (SZZ algorithm). Computationally, both `diff` algorithms are correct in textual differencing. However, the `diff` outputs are sometimes different due to different `diff` algorithms. The `diff` results might show different change region with a contiguous list of deleted and added lines that is called as a change hunk (Ray et al, 2015). We expect that a set of changing operations done by developers

Table 9: Targeted files that have different locations in identified lines with two `diff` algorithms

Project Group	Type of identified line	#Files
CI-Java Projects (Section 2.6)	Added and deleted lines	20,535
Apache Projects (Section 2.7)	Deleted lines	1,055
Total		21,590

can be represented by change hunks. However, the identification of the change hunks can be inappropriate. In our investigation, this issue could not be identified automatedly. Thus, we analyze the quality of `diff` manually.

To judge the quality of the `diff` algorithms, we define "better" if the algorithms meet our two criteria: (i) it detects the unmodified lines appropriately that should not be identified as changed lines, and (ii) it shows the changed lines more systematically [58]. The sequences of the added and deleted lines of code are expected to be closer to what developers did to the code. If the code elements change together, they are shown explicitly as group systematic changes or report their common structural characteristics.

For this analysis, we used the same dataset that had been used in Section 2.6 and Section 2.7, shown in Table 9. From the CI-Java projects, we considered all modified files in all commit IDs to be targeted, while of the Apache projects, files changed in all bug-fix commit candidates are targeted. We applied the same command as the other two comparisons: `git diff -w --ignore-blank-lines --diff-algorithm=<algorithm name> <parent commit ID> <commit ID> -- <filename>` to generate the `diff` output from *Myers* and *Histogram*. In each project of the first group, we analyzed the files that have different locations of the inserted and removed lines from the execution of the two `diff` strategies. While in the second group, only the files that have a different location of the deleted lines were analyzed.

We divided the comparison into two categories: (i) *in-code diff* and (ii) *in-non-code diff*. The first category of *diff* means the different `diff` lists generated by both algorithms are lines of code or a block of code in a source code file. Otherwise, the second *diff* implies the disagreement between these two algorithms

Table 10: Description of the `diff` assessment

Result	Condition
Histogram	The output of *Histogram* is better.
Myers	The output of *Myers* is better.
Same	Both outputs are same level. One algorithm is not better than the other.

are other than a line of code, for example a change of comments, or a change in a non-code file, such as a modification in a text file.

Qualitative analysis between the two `diff` algorithms was performed manually by the first two authors in multiple steps. Initially, the first author made a list of all files from the two project groups. From this list, the sample size of files was counted using the tool provided in a survey system[16] to statistically represent sample from files in each project, so that the conclusions about the quality of the `diff` algorithm would generalize to all files in all projects with a confidence level of 95% and a confidence interval of 5. As can be seen in Table 9, the total number of files summarized from all project groups is 21,590. From this population, we selected random samples of 377 files.

In the second step, we conducted a manual comparison between two `diff` outputs produced by *Myers* and *Histogram* algorithms from all files in the sample. The first two authors of this chapter were involved to independently annotate the `diff` outputs that makes the result is expected to be more reliable. To specify the comparison result between two `diff` algorithms, we generated three categories as described in Table 10. We assign **Histogram** to the comparison results if the `diff` outputs produced by *Histogram* algorithm show the unmodified lines more appropriately and provide better group systematic changes to show the lines were changed together compared with the *Myers*. If the results produced by *Myers* provide more appropriate unchanged contexts and show the group changes more systematically compared with the *Histogram*'s `diff`, we labeled them as **Myers**. While if the `diff` outputs produced by one algorithm are not better than the other, then we mark them the **Same**. The comparison results between two

[16] `https://www.surveysystem.com/sscalc.htm`

Table 11: Frequency of comparison result in the sample data

Result	*in-Code diff*		*in-Non-Code diff*	
Histogram	152	(62.6%)	18	(13.4%)
Myers	41	(16.9%)	20	(14.9%)
Same	50	(20.6%)	96	(71.6%)
Sum	243	(100%)	134	(100%)

authors from 377 files were subsequently computed to find the kappa agreement.[17] We obtained 70.82%, which is categorized into 'substantial agreement' [114]. This means, the statistic result of our manual study is acceptable.

2.8.2 Results

Table 11 shows how well both `diff` algorithms work in presenting the changes of code. It can be seen that *Histogram* outnumbered the other results in the *in-Code diff* category, which emphasizes that this algorithm is substantially better to differentiate the changes of code specifically.

Figure 16 shows how the *Histogram* algorithm provides better output of code changes compared with the *Myers*. We extracted the `diff` from the file `AmqpMessage.java`[18] in commit `f56ea45e5` from the project of *ActiveMQ*. It is true that none of the algorithms are incorrect in describing changes. However, the *Histogram* algorithm provides a reasonable diff output better describing human change intention, as the *if*-statement is moved to a new method and a new method call is added. While from the result of *Myers*, it is not clear how developer changed the code. Lines that have not modified were identified as removed from the original positions (line 18 and 19) and added to the new positions (line 6 and 7).

This manual investigation also highlighted that the *Myers* and *Histogram* algorithms have almost the same ability to extract the `diff`s from non-code changes.

[17] `http://justusrandolph.net/kappa/`

[18] activemq-amqp/src/test/java/org/apache/activemq/transport/amqp/client/AmqpMessage.java – `https://github.com/apache/activemq/commit/f56ea45e58a17fa3aad46cbe8fc605ef4ffdbc81#diff-5296b90814217d75e272e14834a09dca`

```
@@ -169,11 +284,36 @@ public class AmqpMessage {
      * @throws IllegalStateException if the message is read only.
      */
     public void setText(String value) throws IllegalStateException {
+        checkReadOnly();
+        AmqpValue body = new AmqpValue(value);
+        getWrappedMessage().setBody(body);
+    }
+
+    //----- Internal implementation ------------------------------------------//
+
+    private void checkReadOnly() throws IllegalStateException {
         if (delivery != null) {
             throw new IllegalStateException("Message is read only.");
         }
+    }

-        AmqpValue body = new AmqpValue(value);
-        getWrappedMessage().setBody(body);
```

(a) `diff` output using *Myers*

```
@@ -169,11 +284,36 @@ public class AmqpMessage {
      * @throws IllegalStateException if the message is read only.
      */
     public void setText(String value) throws IllegalStateException {
-        if (delivery != null) {
-            throw new IllegalStateException("Message is read only.");
-        }
-
+        checkReadOnly();
         AmqpValue body = new AmqpValue(value);
         getWrappedMessage().setBody(body);
     }
+
+    //----- Internal implementation ------------------------------------------//
+
+    private void checkReadOnly() throws IllegalStateException {
+        if (delivery != null) {
+            throw new IllegalStateException("Message is read only.");
+        }
+    }
```

(b) `diff` output using *Histogram*

Figure 16: Example of `diff` outputs generated by *Myers* and *Histogram* in extracting the code changes

```
@@ -2,27 +2,21 @@
  * org.openmicroscopy.shoola.agents.hiviewer.ChannelMetadataLoader
  *
  *------------------------------------------------------------------------------
+ *  Copyright (C) 2006 University of Dundee. All rights reserved.
  *
- *  Copyright (C) 2004 Open Microscopy Environment
- *      Massachusetts Institute of Technology,
- *      National Institutes of Health,
- *      University of Dundee
  *
- *
- *    This library is free software; you can redistribute it and/or
- *    modify it under the terms of the GNU Lesser General Public
- *    License as published by the Free Software Foundation; either
- *    version 2.1 of the License, or (at your option) any later version.
- *
- *    This library is distributed in the hope that it will be useful,
+ *     This program is free software; you can redistribute it and/or modify
+ *  it under the terms of the GNU General Public License as published by
+ *  the Free Software Foundation; either version 2 of the License, or
+ *  (at your option) any later version.
+ *  This program is distributed in the hope that it will be useful,
  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
- *    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
- *    Lesser General Public License for more details.
+ *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
+ *  GNU General Public License for more details.
  *
```

(a) `diff` output of comment using *Myers*

```
@@ -2,27 +2,21 @@
  * org.openmicroscopy.shoola.agents.hiviewer.ChannelMetadataLoader
  *
  *------------------------------------------------------------------------------
- *
  *  Copyright (C) 2004 Open Microscopy Environment
  *      Massachusetts Institute of Technology,
  *      National Institutes of Health,
  *      University of Dundee
+ *  Copyright (C) 2006 University of Dundee. All rights reserved.
  *
  *
- *    This library is free software; you can redistribute it and/or
- *    modify it under the terms of the GNU Lesser General Public
- *    License as published by the Free Software Foundation; either
- *    version 2.1 of the License, or (at your option) any later version.
- *
- *    This library is distributed in the hope that it will be useful,
+ *     This program is free software; you can redistribute it and/or modify
+ *  it under the terms of the GNU General Public License as published by
+ *  the Free Software Foundation; either version 2 of the License, or
+ *  (at your option) any later version.
+ *  This program is distributed in the hope that it will be useful,
  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
- *    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
- *    Lesser General Public License for more details.
+ *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
+ *  GNU General Public License for more details.
  *
```

(b) `diff` output of comment using *Histogram*

Figure 17: Example of `diff` lists generated by *Myers* and *Histogram* in extracting the non-code changes

As shown in Table 11, their percentages are nearly equals in the *in-Non-Code diff* (13.4% files are better using the *Histogram* and 14.9% files are preferable using the *Myers*). This is even strengthened by the high percentage of both `diff` algorithms' application that resulted in the same quality for the same files (see the example in Figure 17), which reached 71.6%. This quantification reveals that we can use any of these algorithms to produce the `diff` from non-code changes. As shown in Figure 17, both `diff` algorithms worked well to reveal the comment changes from file `ChannelMetadataLoader.java`[19] in commit `e5924527fa` of *Openmicroscopy* project since both lists are readable and understandable. The only differences between the two lists are the position of the initial added line and the matched line after the first inserted one. However, these disagreements did not change our interpretation about the modifications that occurred.

2.8.3 Summary

Due to the different procedures between *Myers* and *Histogram* in identifying the changed lines of code, they possibly generated different `diff` results. Our manual comparison found that their differences were the number of the changes, the order of the changed lines, or even the detected added and deleted code. They certainly affect the readability of the `diff` outputs, in other words, the quality of the `diff` results produced by the two `diff` algorithms were different. Importantly, our results provide evidence that *Histogram* frequently produced better `diff` results compared to *Myers* in extracting the differences in source code.

2.9 Discussions

2.9.1 Implication and Recommendation

In this section, we present a description of the impact of different `diff`s on the results of a study. In the example shown in Figure 16, we can see both algorithms identify the changed lines of code from line #169. Nevertheless, there are several differences in the identified changed lines shown in both `diff` outputs.

[19] SRC/org/openmicroscopy/shoola/agents/hiviewer/ChannelMetadataLoader.java – `https://github.com/openmicroscopy/openmicroscopy/commit/e5924527fa467e117337b53a769503b6cc48e43f#diff-94a26d7e256533ade03740d86aca1afe`

The first difference is the number of the changed lines. From Figure 16, we can see that the quantity of the detected changed lines are unequal. There are 11 changed lines discovered by the *Myers*, while the *Histogram* found 13 lines. In a study that aims to collect metrics from the code changes, considering different `diff` algorithms is important since it has an impact on the number of changes.

In software quality analysis, one key factor of process metrics used to measure the changes is the number of modified lines (NLA and NLD). For example, a work undertaken by Gousios et al. [37] which proposed an approach to measure a software developer's contribution using `diff` records to compute the number of changed lines in a file. This quantity of the changed lines was then used to calculate the commit size of all affected files. Based on our metrics comparison, we found that 1.7% to 8.2% commits have different NLA and NLD due to different `diff` algorithms application. While our manual investigation shows that more than 60% `diff` outputs are better to extract using *Histogram*. Thus, if this study attempts to apply *Histogram*, it might affect around 1% to 4% different commit size. As a result, this will impact the measurement of software developer's contribution as well. Another study related to metrics analysis was conducted by Rausch et al. [89]. The authors investigated the complexity of changes that can impact software quality. The findings support that higher median values of NLA and NLD lead to an increase in build failures. The study also found that the high mean values of the number of modified files correlates to the failed builds. Based on the result from our metrics analysis, we found 0.8% to 6.2% files have different NLA and NLD. Therefore, if *Histogram* is applied in this study, this will influence around 0.5% to 3.5% of the modified files that correlates to the failed builds.

The second difference is the position of the changed lines. Figure 16 shows that the two `diff` algorithms detect the deleted lines differently. The *Myers* identifies one line of 'Assignment' and one line of a 'Method' call, whereas the *Histogram* specifies a block of '*if* condition'. Related to SZZ application, both `diff` algorithms produce different deleted lines that are considered as the candidate of bug-introducing changes. Thus, the identified bug-related lines might be invalid due to different `diff` algorithms application that can lead to the failure of bug-introducing changes identification.

A study undertaken by da Costa et al. [24] investigates the output of five SZZ procedures in discovering the bug-introducing changes. The study on 10 Apache projects analyzed the validity of bug-introducing changes. The validation process of bug-related lines used by the authors is similar to our study. It compares the release dates of the earliest affected software versions of a bug with the dates of the introduction of the candidates of bug-related lines. However, in our study, we enhanced the process to validate the other three parameters, that is, the bug-introducing commits that initially adds the valid bug-related lines, files containing valid bug-related lines, and bug-fixing commits that relates to valid bug-introducing commits. Our SZZ analysis shows that different `diff` algorithms application can have impact on the results of SZZ algorithm. We found 2.4% to 6.6% valid files have different location of valid bug-related lines. Since the *Histogram* is better in more than 60% `diff` outputs based on our manual analysis, therefore, if the study by da Costa et al. [24] applies *Histogram* in the `diff` command, around 1.5% to 4% files might have different valid bug-related lines in their study results. The SZZ algorithm has also been studied by Rodriguez-Perez et al. [91]. The authors conducted a literature review of published articles that focus on the SZZ algorithm's functionality and its ability to be imitated. The similarity of this study to ours is investigating the changing impact due to the modification of SZZ algorithm. However, the study focus on the usability of the changing SZZ in the academic paper over time while our study analyze the impact of different `diff` algorithms application in the SZZ to study results. Without considering the version of SZZ used in 187 previous studies collected by Rodriguez-Perez et al. [91], we understand that SZZ is a widespread and well-known algorithm over a 10-year period. This bug identification algorithm was commonly used to investigate commit size (26% of the papers), line of code (15% of the papers), number of changes (12% of the papers), number of affected files (8% of the papers), etc. As described in our SZZ analysis, `diff` algorithms also have an impact on SZZ. Thus, if the *Histogram* is applied in those 187 prior studies, it might affect the results of studies.

Our investigations on metrics and SZZ application provide evidences that different `diff` algorithms application in `git` command can have an impact on a study result. It is also acknowledged that the *Histogram* algorithm is substantially

better than the *Myers* to produce the changed lines of code. Thus, we recommend to use the *Histogram* in `git diff` command to extract the changes from source code.

2.9.2 Threats to Validity

Threats to the *construct validity* appear in the mapping study and the SZZ application. In our mapping study, we selected only the papers that specifically mention the `git` commands. As a result, papers that had used `git` commands but do not mention it in the full text had been ignored, which can cause selection bias. Since different `diff` algorithms produce different results, we consider that papers should mention algorithm names of `diff` if the authors intentionally chose them. In the SZZ application, we used a small number of keywords to detect commit messages that describe fixing bugs. This limited our ability to extract all potential candidate bug-fixing commits. Even so, the commits that should not be identified as bug-fixing commits were also possible to be collected as long as they included the keywords in their log messages. However, since our focus is to investigate the level of differences of the `diff` lists produced by *Myers* and *Histogram*, the impact of the incorrect commits to the study result is small. Another threat to the *construct validity* is the definition of *better* for the `diff` algorithm. We consider good quality of the algorithm based on our two criteria, while many could have been considered. Different software engineering tasks may have different requirements for `diff` analysis. However, since our focus is expecting to recover the changing operations from the `diff` outputs, the impact of this issue is not significant. In addition to the *construct validity*, this chapter did not perform a statistical test of its implication to previous studies. Thus, the significance of *Histogram* diff algorithm implementation is still unknown. However, due to different identification results in SZZ algorithm application, using better algorithm is important for location of the identified changes.

Threats to the *external validity* emerge regarding the repository used in our experiments. Although we analyzed 24 OSS Java projects mined from Git repositories, we cannot generalize our study results to other open source projects nor industry.

To reduce the threats to *reliability*, we make our dataset publicly available.

We provided lists of our collected files identified by the *Myers* and *Histogram* algorithms which were used in the three empirical analyses (see on GitHub[20]).

2.10 Section Summary

In software development environment, some specific knowledge might be undocumented and uncommunicated between individuals although the knowledge already exists. In this chapter, for example, we explore the differences between the `diff` outputs of two `diff` algorithms, *Myers* and *Histogram*. Since the differences of these two `diff` algorithms have never been documented and shared in software engineering communities, people do not know that a simple git option in git command may affect the results of relevant works.

Our metrics analysis has shown that the application of different `diff` algorithms in metrics collection could impact around 4% files and 7% commits have different results. The study on the application of SZZ algorithm indicates that different `diff` algorithms can affect around 5% files and 9% commits have different location of potential bug-introducing changes. Since `diff` is the fundamental tool for various software engineering tasks, considering limitations and advantages of algorithms are important. Thus, documenting and communicating various types of knowledge, either tacit or explicit, are necessary since this might determine the implication of relevant studies.

[20] `https://github.com/yusufsn/DifferentDiffAlgorithms`

3 A Topological Analysis of Communication Channels for Knowledge Sharing in Contemporary GitHub Projects

3.1 Background

A key ingredient to the emergence and success of software projects on collaborative platforms such as GitHub[21] has been its distributed information sharing nature, which is the ability to interact and share information between software developers. With over 28 million developers and 67 million repositories reported in 2018, GitHub hosts a multitude of diverse developer ecosystem (also referred to as communities).[22] As well as hosting traditional software projects, GitHub is also the home to sometimes trivial library projects [1] and have been the focus of recent studies [71, 62, 74, 36, 46, 57, 27]. For instance, package managers like npm[23] host around 700 thousand packages on GitHub. Interestingly, we find that a single npm developer could be the maintainer for hundreds of these packages.

Investing in knowledge creates value during software development, especially in the context of human capital [80]. This knowledge can then be represented

[21] `https://github.com/`

[22] The survey result is available at `https://octoverse.github.com/` (accessed in May 2018)

[23] `https://www.npmjs.com/`

and shared in contemporary software through social and technical *'communication channels'*, mostly used to improve and maintain a project's presence in an ecosystem. Examples of such channels include forking, pull requests, the readme file documentation and so on. According to the business management perspective, communication channels can be distinguished into either tacit and explicit knowledge, with its transfer being described as externalization and combination (using the SECI model [79]). In fact, open source projects heavily rely on social markers and its popularity (i.e., star counts and forks) for measuring their abilities to attract and maintain their contributors. Although much work has covered the different communication channels, a mapping of all these communication channels and the knowledge shared has not yet been studied.

The research gap that this chapter fills is understanding at the topological level of how knowledge is shared between communication channels of contemporary projects. There has been work that has studied the social collaborations between projects, but not an analysis of multi-channels over large-scale ecosystem of libraries. A study by Storey et al. [106] showed that communities of FLOSS (Free Libre Open Source Software) projects are shaped through social and communication channels (also referred to as social coding). Recently, Aniche et al. [3] confirmed that news channels also play an important role in shaping and sharing knowledge among developers.

In this chapter, we investigate communication channels to understand how projects share knowledge at the software ecosystem level. Inspired by the knowledge-based theory of the firm [38], our study is to validate the underlying theory behind the transferable of knowledge within these library ecosystems, and to investigate how ecosystems influence social practices within and outside their ecosystems. To achieve our goal that is to analyze how communication channels share knowledge over projects, we first identify different knowledge forms of channels in over 210 thousand library projects from seven different library ecosystems. We then explore the evolution of these channels and distinguish differences between these seven ecosystems. Similar to a study by Lertwittayatrai et al. [64], we use topological data analysis to generate topologies that cover three years (i.e., 2015 to 2017). Using topology data analysis, the results of the study show that (i) contemporary GitHub Projects tend to adopt multiple communication channels, (ii) communi-

cation channels change over time, and (iii) communication channels are used to capture new knowledge (i.e., externalization) and updating existing knowledge. The contributions of the study are two-fold. First, we present a manual categorization of channels forms in software projects. The second contribution is a large-scale analysis of channels for software projects over seven ecosystems using the topological analysis of software library projects for seven different software ecosystems.

The rest of the chapter is organized as follows. Section 3.2 describes our initial work to classify the communication channels into tacit or explicit. Section 3.3 details our experiments using the topological data analysis of the seven GitHub library ecosystems. Section 3.5 is the evaluation of our topological data analysis technique. Section 3.6 discuss the implication of the experimental results, with Section 3.7 defining the threats of validity. We present the related works in Section 3.8, and finally conclude the chapter and summarize potential avenues for future work in Section 3.9. The replication package that contains all the dataset and experiment details are accessible from `https://github.com/NAIST-SE/TDA_Communication_Channels`.

3.2 Preliminary Study: Communication Channels and Knowledge Sharing in Software Projects

Before we proceed with the study, we first carried out a preliminary study to first understand what knowledge exists and is transferred through the communication channels.

3.2.1 Motivation

The study of knowledge sharing has had an impact in fields like Sharing Architectural Knowledge [123], where architectural decision-making and has been shown to increase project consistency, coordination, and communication coherence over time. To understand knowledge sharing, we apply and distinguish different knowledge forms to communication channels. We use existing models of knowledge and how they are transferred. We carry out an empirical study to analyze and answer the formulated research question:

Table 12: Distinctions between Tacit and Explicit Knowledge[24]

Tacit Knowledge	**Explicit Knowledge**
T1: Subjective, cognitive, experiential learning	E1: Objective, rational, technical
T2: Personal	E2: Structured
T3: Context sensitive/specific	E3: Fixed content
T4: Dynamically created	E4: Context independent
T5: Internalized	E5: Externalized
T6: Difficult to capture and codify	E6: Easily documented
T7: Difficult to share	E7: Easy to codify
T8: Has high value	E8: Easy to share
T9: Hard to document	E9: Easily to transferred/taught/learned
T10: Hard to transfer/teach/learn	E10: Exists in high volumes
T11: Involves a lot of human interpretation	

PS_1: Are we able to distinguish knowledge within the communication channels of GitHub projects?

3.2.2 Approach

Our approach to answer the preliminary study question is through analysis of historical information. We first carried-out an investigation of possible communication channels, which we then methodologically classify into the different knowledge forms. We then use the SECI model to understand the knowledge transfer within these channels.

As shown in Table 12, there are two knowledge forms [86, 87, 79]. The first is tacit knowledge (know-how) where the knowledge is embedded in the human mind through experience and jobs. Personal wisdom and experience, context-specific are more difficult to extract and codify. In addition, tacit knowledge includes insights and intuitions. The second is explicit knowledge (know-that) which is codified and digitized in books, documents, reports, memos, etc. This type of knowledge is easily identified, articulated, shared and employed that can facilitate action. To classify the transfer (through sharing) of knowledge within each communication channel, we used the SECI knowledge model. Nonaka and

[24] `https://www.tlu.ee/~sirvir/Information\%20and\%20Knowledge\%20Management/Key_Concepts_of_IKM/tacit_and_explicit_knowledge.html` (accessed in July 2019)

Table 13: Taken from Nonaka and Takeuchi [79], four dimensions of knowledge transfer

Dimension	**Knowledge Transfer**	**Description**
Socialization	Tacit to Tacit	Social interaction as tacit to tacit knowledge transfer
Externalization	Tacit to Explicit	Articulating tacit knowledge through dialogue and reflection. When tacit knowledge is made explicit, knowledge is crystallized, thus allowing it to be shared by others, and it becomes the basis of new knowledge
Combination	Explicit to Explicit	Systemizing and applying explicit knowledge and information
Internalization	Explicit to Tacit	Learning and acquiring new tacit knowledge in practice

Takeuchi's SECI model is a model of knowledge dimensions that describes the transformation of tacit and explicit knowledge into organizational knowledge [79]. Since it was first introduced by Nonaka [78], SECI model has been used in many area of studies. Dávideková et al. [26] used SECI model to analyze various information and communication technology (ICT) tools in bridging virtual collaboration between team members without their physical presence. In comparison with traditional teams that requires the presence of individuals, virtual collaboration demands the motivation of team members, support from team leader, and appropriate technology. Therefore, the preference of such suitable ICT tools for each activity in organizations is necessary. As shown in Table 13, SECI model contains four dimensions of knowledge which together form the acronym "SECI". In this chapter, we focus specifically on the *externalization* and *combination* in our classifications.

The identification of knowledge within communication channels was performed by a group consensus among three of the authors, with rationale clearly aligned with the formal definitions.

Table 14: Seven Library Package Platform Ecosystems

Library Package Manager	Programming Language	Typical Usage Domain	# Stars			
			Min	Max	Median	Mean
Go	GoLang	Developed by Google Applications	592	92,227	6,866.24	2,559
npm	nodeJS JavaScript	Web Services	569	122,630	8,479.49	3,372
Packagist	PHP	Server-side web development	8	122,630	194.13	23
RubyGems	Ruby	Web Applications	11	90,383	433.96	48
PyPI	Python	General scripting	10	122,630	439.77	39
Bower	JavaScript	Web Services	5	122,630	866.11	43
Maven	Java-based languages	Languages that use Java Virtual Machine	107	122,630	1,755.45	454

3.2.3 Data Collection

For the preliminary study, the authors used the `libraries.io`[25] collection of GitHub software projects. This dataset includes various communication channels and covers the largest range of ecosystems. According to its website, libraries.io indexes data from over 3 million library packages from 36 package managers. Package managers represent different ecosystems of libraries. For example, libraries belonging to the nodeJS package manager (npm) are part of the bigger JavaScript ecosystem of projects. Furthermore, libraries.io also monitors and stores package releases, analyzes each project's code, ecosystem, distribution and documentation, and map the relationships between packages. Our dataset has also been used in recent empirical studies [57, 27].

As shown in Table 14, our collected raw dataset is a subset of the seven largest library ecosystems from the `libraries.io` dataset. Furthermore, we used the star count to as to get the more popular repositories within each ecosystem [13]. The higher star ensures that the package has value to the ecosystem. Thus, the top 10,000 ranked projects from each ecosystem was collected. Two authors then identified and mapped 13 communication channels from the raw dataset features. Details of the mapping are discussed in the replication package and presented in Table 15.

[25] `https://libraries.io/data` (accessed in May 2018)

Table 15: Summary of 13 channels classified with rationale.

Dimensions	**Channels**	**Coding (Table 12)** **Rationale** **Source**
Externalization	GitHub Pages	T2, T3 Personal webpage of a project, the content is specific, and it has no standard template to create. https://help.github.com/en/articles/what-is-github-pages
	Readme	T3, T4 The content is specific and is created dynamically without a template. https://help.github.com/en/articles/about-readmes
	Security Audit	T2, E3 Although the audit is personal, the contents are fixed. https://help.github.com/en/articles/reviewing-the-audit-log-for-your-organization
	Wiki	T2, T3 Similar to GitHub Pages, the contents of wiki are personal and specific. It has no specific template to create. https://help.github.com/en/articles/about-wikis
Combination	Changelog	E2, E3 The changes are documented in a structured manner, the contents are fix and cannot be customized. https://github.blog/2018-05-03-introducing-the-github-changelog/
	Code of Conduct	E2, E3 There is a standard template to make the contents of code of conduct. https://help.github.com/en/articles/adding-a-code-of-conduct-to-your-project
	Contributing Guidelines	E2, E3, E4 The contents are structured, fixed and independent. It is created by following a template. https://help.github.com/en/articles/setting-guidelines-for-repository-contributors
	Fork	E2, E3, E4 Fork has structured and fixed content. The context is independent. https://help.github.com/en/articles/about-forks
	Issue Tracker	E2, E4 The contents are independent and adopted from a system in a structured way. https://en.wikipedia.org/wiki/Issue_tracking_system
	License	E2, E3 The contents of license are structured and fixed. https://help.github.com/en/articles/licensing-a-repository
	Security Threat Model	E2, E3, E4 The security regulations that are structured, fixed and independent. http://www.agilemodeling.com/artifacts/securityThreatModel.htm
	# of Forks	E2, E4 The content is structured and independent. https://help.github.com/en/articles/fork-a-repo
	# of Open Issues	E2, E4 Structured and independent content. https://help.github.com/en/articles/opening-an-issue-from-code

Each library ecosystem is described below. Go[26] is a package manager in GoLang programming language which is developed by Google. The npm[27] and Bower[28] which are renowned for the JavaScript are mostly used in the website development. Similar to the npm and Bower, Packagist[29] is very common for the website development but in server-side. The language used for this package is PHP. Meanwhile, RubyGems[30] is a framework of library management contains functions that can be called by a Ruby program. Finally, the python-based library manager, PyPI[31] works for writing script in general, while the Java-based language that use Java Virtual Machine (JVM) is Maven.[32]

3.2.4 Analysis

Answering PS_1: Using the collected dataset, we labeled each of communication channel to a knowledge form (i.e., tacit or explicit). The manual labeling was performed by one author and later validated by other co-authors. Based on Table 12, we found that labeling T2, T3, T4, E2, E3 and E4 were the most identifiable distinctions. To reduce bias, the first author and second author did independent labeling. Then, in a round table, other authors were consulted for any conflicts. In Table 15, we provide a full rationale for each feature.

3.2.5 Results

We now present the results of classifying knowledge of each channel.

3.2.5.1 PS_1: Are we able to distinguish knowledge within the communication channels of GitHub projects?

Yes, we are able to distinguish knowledge forms channels in software projects.

[26] `https://golang.org/`
[27] `https://www.npmjs.com/`
[28] `https://bower.io/`
[29] `https://packagist.org/`
[30] `https://rubygems.org/`
[31] `https://pypi.org/`
[32] `https://maven.apache.org/`

Table 15 shows that channels with tacit forms of knowledge being externalized (i.e., externalized dimension of SECI). Since the classification of tacit and explicit knowledge is not trivial, we applied the most distinguishable features taken from Table 12 (i.e. T2, T3, T4, E2, E3, and E4). In general we used the following rationale as guidance:

- *T2 - Personal*: The knowledge possessed by any individual. Usually accumulated through observation or experiences. For the study, we characterize individual additions with no structure.
- *T3 - Context sensitive/specific:* The content is specific to its original context. It depends on particular time and space. Similar to T2, here the project customizes the channel specific to the project requirements or nature (i.e., library or framework, programming language)
- *T4 - Dynamically created:* The content is capable to change or customize. Since GitHub has templates, we regard these features are not in the templates.
- *E2 - Structured:* The information is organized in a predictable way and usually classified with metadata. For instance, a workflow tool usually has structure to is, when compared to a wiki.
- *E3 - Fixed content:* The content that is not, under normal circumstances, subject to change. This feature is more common with workflow and tools that serve as channels.
- *E4 - Context independent:* The content is unaffected by contextual relevance. For instance, the channel can serve different purpose for different projects.

Interestingly, we find that the security audit is a mix of tacit and explicit forms. Although the audit tends to personal, the contents that describe the strategy, policy and the process related to the management are fixed. Thus, we conclude that the developers can provide the guidelines with regards to reducing the risk of misrepresentation of knowledge when developing software [67].

3.3 A Topological Analysis of Communication Channels Across GitHub Ecosystems

Taking the results from the preliminary study, we are now able to study the knowledge topology of these channels. This topology mapping analysis presents a visual representation of channels within and across projects in the GitHub ecosystems.

3.3.1 Topological Data Analysis

Due to the vast amount of data and the different communication channels, we apply the Topological Data Analysis (TDA) technique. TDA is an approach to extract meaningful information from such data that is insensitive to the chosen metric, high-dimensional, noisy and incomplete without initiating a query or hypothesis [68]. TDA has been employed in many research fields for data exploration and mapping. In comparison with other analysis methods such as the principal component analysis (PCA), multidimensional scaling (MDS), and cluster analysis, TDA is sensitive to both small and large scale patterns that often other techniques fail to detect. Lum et al. [68] showed the significance of understanding the "shape" of data by implemented topology to analyze three different types of data: data of breast tumors to show gene expression, data of voting behavior from members of the United States House of Representatives and performance data of the NBA players. In software engineering, TDA was also applied in a study of software testing by Costa et al. [85]. Similar to Lertwittayatrai et al. [64], a topology of the dataset is generated to provide a visual interpretation of multi-dimensions data analysis.

Figure 18 provides an example of how a TDA is constructed. TDA assumes a choice of a filter or its combination that can be viewed as a map to a space of metric to provides insights based on clustering the various subsets of the dataset related the choices of filter values. As shown, each node is represented as a set of data points, and the connection between nodes occurs if and only if their corresponding collections of data points have a point in common. The topology is constructed by clusters of nodes (i.e., nodes connected together). Then within the cluster, we can find groups of nodes that form a shape of the dataset. The density

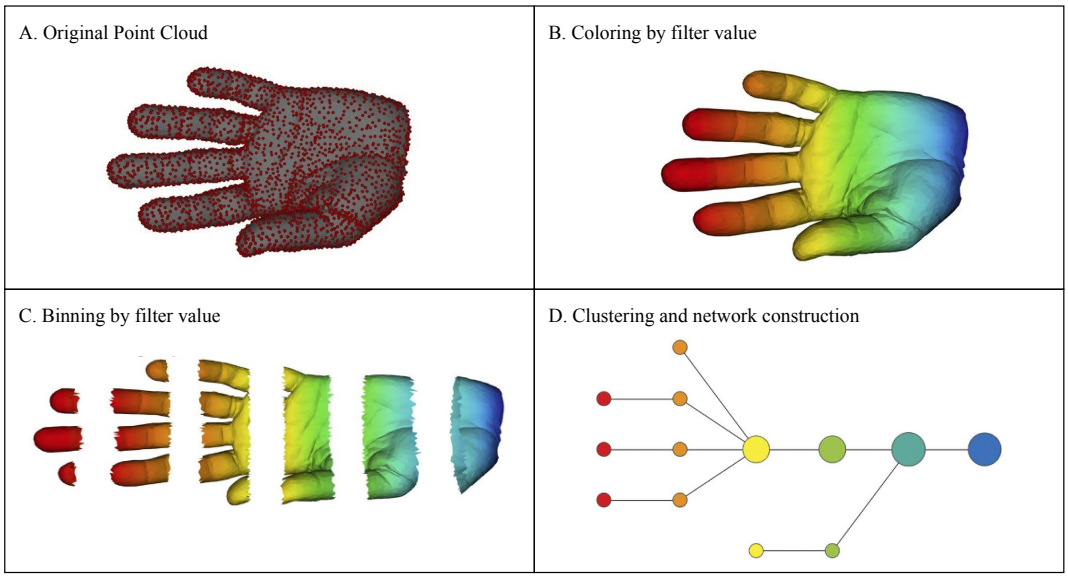


Figure 18: Taken from Lum et al. [68], A) 3D object (hand) represented as a point cloud, B) A filter value is applied to the point cloud and the object is then colored by the values of the filter function, C) The dataset is binned by filter value, D) Each bin is clustered and a network is built. Within each cluster, groups of nodes determine the shape.

of the nodes and their shape gives an indication of the dominant of the features. Tailored to our study, each point is a project that are clustered according to the different features extracted in the preliminary study. The use of color highlights the dominance of a feature, which indicates high occurrence of that channel.

For TDA, the clustering is performed using the t-Distributed Stochastic Neighbor Embedding (t-SNE) [112]. In detail, the algorithm starts by calculating the probability of similarity of points in high-dimensional space, computing in proportion to their probability density under a Gaussian (normal distribution) algorithm. Multi-dimensional data are then mapped by the t-SNE to a lower dimensional space and attempts to find patterns in the data by identifying observed clusters based on similarity of data points with multiple features.

3.3.2 Motivation

Our motivation for the topological mapping study is to present a representation and overview of channels that exists within large-scale ecosystems. As such we formulated two research questions as follows:

- ***RQ_1: Do communication channels change over time?***
  In this research question, our motivation is to investigate how channels evolve and change over time.

- ***RQ_2: Do communication channels differ within ecosystems?***
  For this, we take a closer look at the ecosystem. By studying the seven ecosystems, we are able to understand whether there are differences in knowledge.

3.3.3 Approach

Our approach to answer the two research questions is through the TDA mapping technique. The TDA mapper algorithm [100] uses combinatorial representations of geometric information about high-dimensional point cloud data, which is implemented with the Knotter tool [100]. The tool provides a common framework which includes the notions of density clustering trees, disconnectivity graphs, and Reeb graphs, but which substantially generalizes all three. We use the t-Distributed Stochastic Neighbor Embedding (t-SNE) [112], a technique for dimensionality reduction and clustering, and our defined features as the filters for the visualization construction. For RQ_1, we analyze the map by identifying the most dense clusters (i.e., majority of projects) and then find the dominant features for those clusters. For RQ_2, to find dominant features, we identify groups of nodes within the cluster and label.

3.3.4 Data Collection

As with the preliminary study, the same dataset from `libraries.io` was used in our experiments. The results of the preliminary study in Table 15 were used as feature inputs in the topological mapper construction. For RQ_1, we selected only projects between 2015 and 2017 because (i) they contained the youngest

Table 16: Statistics of Generated Topologies including the Topology Build-time

Library Ecosystem	Pop. Size	RQ_1 #proj. created 2015	RQ_1 #proj. created 2016	RQ_1 #proj. created 2017	RQ_2 #proj.
Go	743,841	10,000	10,000	509	20,000
npm	447,306	10,000	10,000	10,000	20,000
Packagist	176,608	10,000	10,000	10,000	20,000
RubyGems	93,377	10,000	10,000	8,611	20,000
PyPI	69,895	10,000	10,000	10,000	20,000
Bower	64,271	10,000	10,000	6,472	20,000
Maven	62,654	10,000	7,526	428	20,000
build-time per topology (mins.)		35.03	29.60	20.22	70
Totals	1,657,952	70,000	67,526	46,020	

projects and (ii) all seven ecosystems had sufficient sample projects within this time period. Part of the data preparation involved normalizing each of the 13 features into a value that ranges from 0 to 1. Based on the type column in Table 15, we normalize the int, string and boolean values. For integers, we calculate the ratio of the $X_{i,j,k}$ and $X_{i,j,max}$ which $X_{i,j,k}$ is the value of feature i in project k which is in platform j and $X_{i,j,max}$ is the maximum value of feature i in platform j. For boolean and string types, we represent 1 to them if the value is TRUE that indicates the channel exists. On the other hand, represent as 0 if the project does not use that channel. The tool limited the maximum number of projects selected to 10,000, which resulted in selecting the top 10,000 most popular projects (based on the star count).

3.3.5 Analysis

Table 16 shows that on average we used up to 30,000 projects for each of the seven ecosystems from the libraries.io.[33] Note that for RQ_1, we prepared an evolutionary set of topologies, dividing the dataset into three time periods (i.e., 2015, 2016 and 2017). Generation is approximated at up to 20–70 minutes for each of the 28 topologies.

Answering RQ_1: To answer RQ_1, we split the projects to separate the older

[33]dataset available at `https://libraries.io/` (accessed in April 2018)

projects from the younger ones using the date that they were created (2015, 2016 or 2017). Shown in Table 16, we generate a topology that highlight the influencing features to explore differences between the older and younger projects. First, we identify main clusters of points. Then we compare these clusters over the three years. Note that the color filter helps to identify dominant features.

Answering RQ_2: To answer RQ_2, we construct seven library specific topologies to find whether projects that are popular (i.e., has the most stars in that ecosystem) share similar channels across ecosystems. First, we identify and analyze the dominant features of nodes. The nodes of the network represent sets of projects and are coloured according to the value of the features existence. If two nodes have one or more data points in common, they will be connected with an edge. Then, using the median score of stars per project within those nodes, we identify the group that contains more popular projects (i.e., labeled as Popular Group) when compared to the other groups (i.e., labeled as Non-Popular Group).

3.4 Results

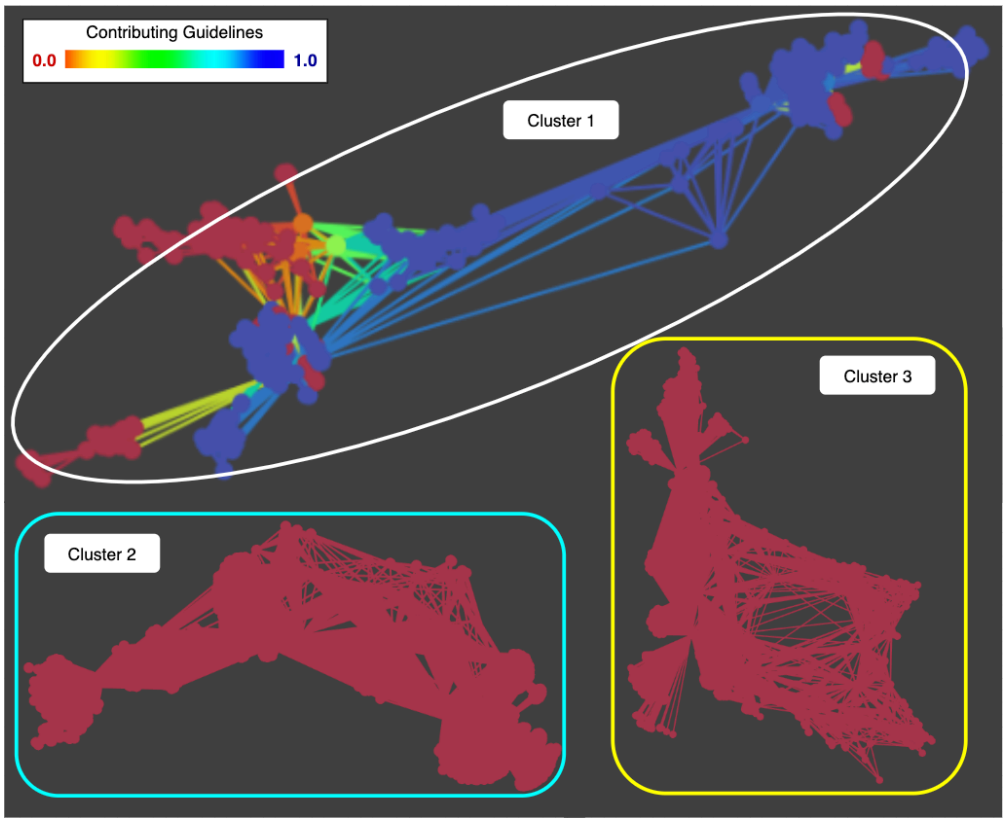
We now introduce our answer to the research questions and then describe the results.

RQ_1: Do communication channels change over time?

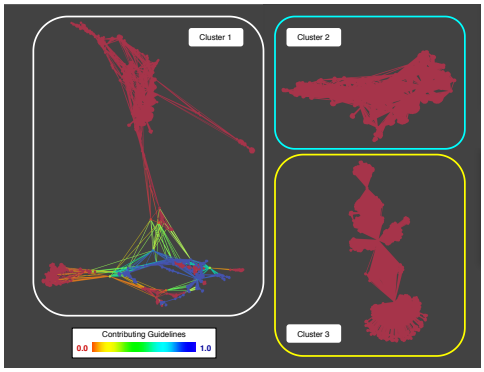
'Younger projects adopt different channels compared to older projects'

We observed two main findings. First, from Table 17, the topology reveals that younger projects are adopting different channels mechanisms when compared to the older projects. To make the topology easier to read, we assigned the color (blue indicates existence while red indicates no existence) to the `Contributing Guidelines` feature. Note that Cluster 1 always indicates the highest number of points (refer to Table 17). Therefore, we can see that the blue nodes first are dominant in Cluster 1 in (i.e., Figure 19(a)), but tend to become less dominant in 2016 and 2017 (i.e., Figure 19(c)).

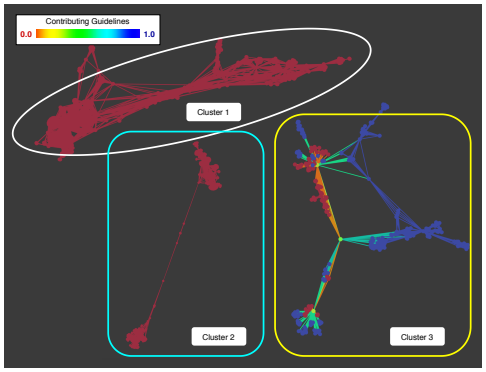
Second, as shown in the Table 17, we see that although some communication channels have changed over time (i.e. GitHub Pages, Security Audit, Changelog, Contributing Guidelines, and Fork), we also find that others (i.e. Wiki, Issue Tracker, and License) are still consistently used by most projects (Cluster 1).



(a) Projects created in 2015



(b) Projects created in 2016



(c) Projects created in 2017

Figure 19: Generated topologies for projects created in (a) 2015, (b) 2016 and (c) 2017

Table 17: Evolution of Externalization and Combination between 2015 and 2017

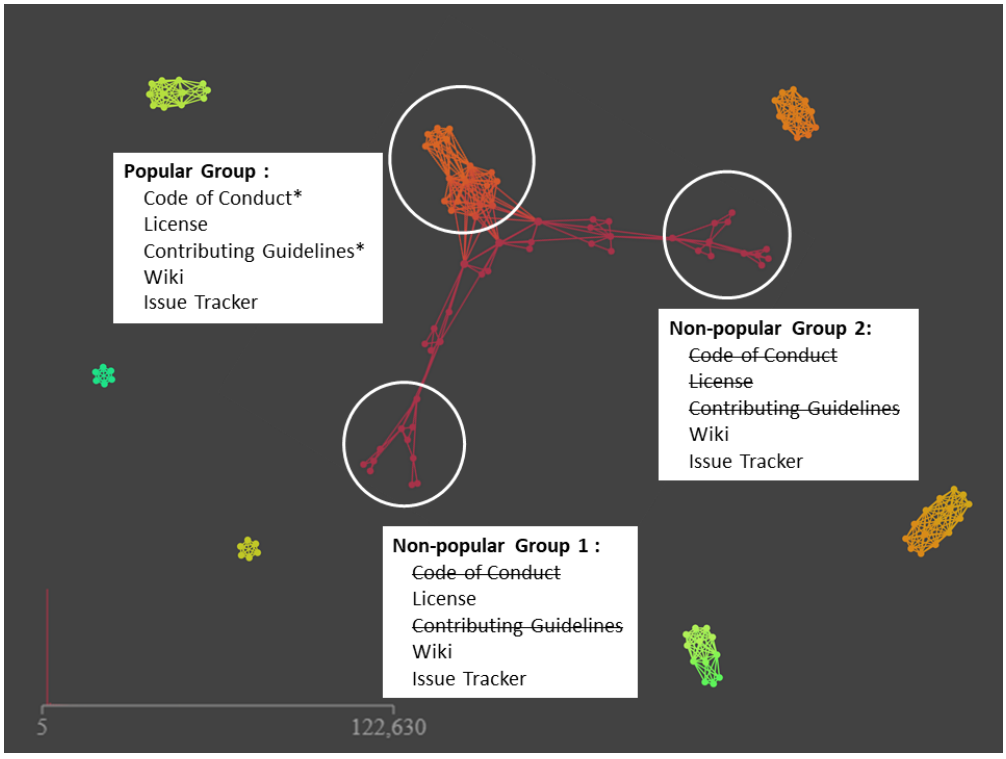
Period	Cluster	#Nodes	#Points	Externalization			Combination				
				GitHub Pages	Security Audit	Wiki	Changelog	Contributing Guidelines	Fork	Issue Tracker	License
2015	1	376	16,906	✓	-	✓	✓	✓	✓	✓	✓
	2	4,943	15,638	-	-	✓	-	-	-	✓	✓
	3	3,257	5,138	✓	-	✓	-	-	-	✓	✓
2016	1	6,289	46,800	✓	✓	✓	✓	✓	✓	✓	✓
	2	1,377	7,650	-	-	✓	-	-	-	✓	-
	3	4,038	3,088	✓	-	✓	-	-	-	✓	✓
2017	1	973	14,098	-	-	✓	-	-	-	✓	✓
	2	1,595	8,794	-	-	✓	-	-	-	✓	-
	3	354	5,046	✓	-	✓	✓	✓	✓	✓	✓

RQ_2: Do communication channels differ within ecosystems?

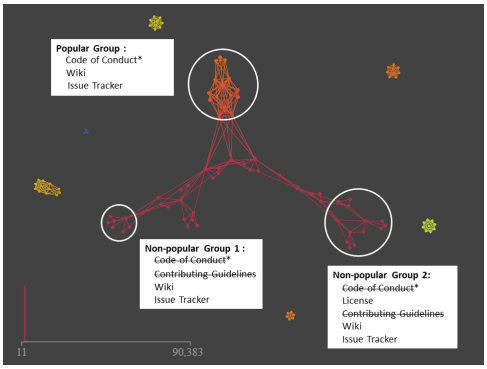
> *'Library ecosystems employ channels that capture new knowledge (i.e., externalization). Channels updating existing knowledge (i.e., Combination knowledge) varies from one ecosystem to another'*

In terms of the topology shape, Figure 20 depicts ecosystems (i.e. Bower, PyPI and RubyGems) having triangular shape topology with three main group points. This is consistent for the rest of the studied ecosystems. Under further investigation, we see that one of the group represents the popular projects (i.e., popular), while the other two group points were the non-popular data points (i.e., non-popular 1 and 2).

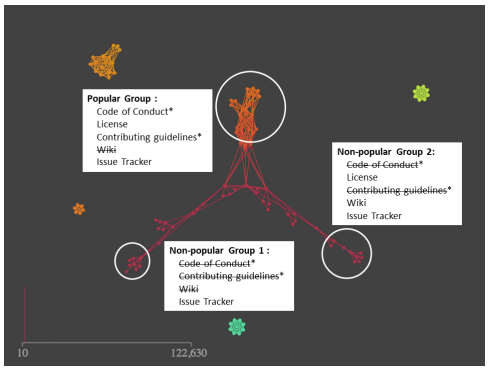
Table 18 shows two results. For both popular and non-popular projects, we find that the issue tracker has been a consistent communication channel for applying explicit knowledge (i.e., combination). Combining with the results of RQ_1, one explanation could be that these are older projects. Second, each ecosystem depicts a different set of explicit dominant features. For example, the Bower ecosystem includes combination knowledge transfer forms (i.e., Code of Conduct, License, Contributing Guidelines, Wiki, Issue Tracker), while PyPI projects are less likely to include a license or contributing guidelines. One reason could be that the license information is embedded in other locations, such as a webpage. For example, the python library scikit-learn has its license information on the



(a) Topology for Bower libraries



(b) Topology for PyPI libraries



(c) Topology for RubyGems libraries

Figure 20: Topology for three of the seven ecosystems (a) Bower, (b) PyPI, and (c) RubyGems

Table 18: Dominant Extracted Features Topologies across the Ecosystems

Topology Cluster	Features	Dimensions	Bower	PyPI	Go	npm	RubyGems	Packagist	Maven
Popular	Code of Conduct	Combination	✓	✓	-	-	✓	-	-
	Contributing Guidelines	Combination	✓	-	✓	✓	✓	✓	-
	Issue Tracker	Combination	✓	✓	✓	✓	✓	✓	✓
	License	Combination	✓	-	✓	✓	✓	✓	✓
	Wiki	Externalization	✓	✓	-	-	-	-	✓
Non-popular	Code of Conduct	Combination	-	-	✓✓	-	-	-	-
	Contributing Guidelines	Combination	-	-	-	-	-	-	-
	Issue Tracker	Combination	✓✓	✓✓	✓	✓✓	✓✓	✓✓	✓
	License	Combination	✓	✓	✓	✓	-	✓	✓
	Wiki	Externalization	✓✓	✓✓	✓	-	✓	✓✓	-

python ecosystem website.[34]

Our study results also confirm that issue tracker as an important communication channel is common for both popular and non popular projects, as described in Figure 20 and Table 18. Issue trackers serve not only as part of the workflow and process (code maintenance and evolution) for software development, but also plays a significant role of communication in a software development process that store a large amount of data, such as discussion during triage meetings, reproduction step clarifications between the person who created an issue and its owner, etc [9]. Other work such as Dingsøyr and Røyrvik [28] confirms that issue tracker builds up a substantial amount of information concerning the issue reports from customers, partially complete feature ideas, and the communication surrounding the software development. This large amount of information is often beneficial for both the organization and the software project team on a number of different levels.

3.5 Topology Evaluation

As mentioned in Section 3.3.1, TDA has been proven to deal with both small and large scale patterns that often fail to be detected by other methods. Other more traditional methods of analysis are the principal component analysis (PCA), multidimensional scaling (MDS), and cluster analysis. Unlike traditional statistical methods, TDA does not provide any statistical test that is performed to support the observation. To evaluate and validate our use of TDA, we compared our

[34]license at `https://pypi.org/project/scikit-learn/` (accessed in September 2018)

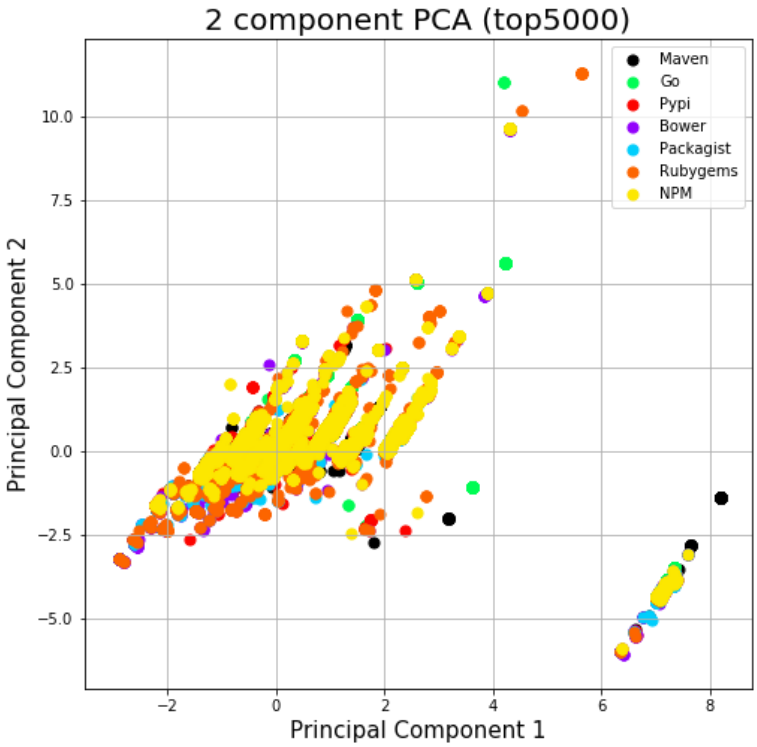


Figure 21: A replication of RQ_1 using PCA. PCA does show location of the ecosystems of different platforms, but since the features are combined, we cannot identify the dominant channels.

method to the PCA method. The main different with PCA is that it simplifies the complexity in high-dimensional data by transforming the data into fewer dimensions (i.e., usually into a x and y axis, depicted by a scatterplot), which act as summaries of features.

In our approach, we apply the PCA method using the `sklearn.decomposition` python library[35] to visually examine the results. For the evaluation, we will regenerate the results for RQ_2 and determine if we can visually identify dominant features within each ecosystem.

Figure 21 shows the results of evaluating the TDA technique against the statistical Principle Component Analysis (i.e., PCA) method. The PCA method

[35] documentation at `https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html` (accessed in July 2018)

shows the location of the ecosystems of different platforms and is able to summarize the features into two principle components. However, the analysis is unable to show details of each feature, outlining (i) which features are dominant and (ii) how the features are different to each other.

3.6 Implications

Based on our results, we discuss three implications of the results in relation to the nature of communication channels for both researchers and practitioners.

1. *Contemporary GitHub Projects will continue to adopt multiple Communication Channels.* Results indicate that GitHub projects are adopting 13 communication channels. Thus the topological mapping is able to cluster together projects with similar channels. As shown in the topological evaluation, other techniques is able to map these relationships. The implication for researchers and practitioners is that knowledge is not stored in one channel, thus multiple channels must be considered to fully understand the knowledge shared in software projects.

2. *Communication Channels will change and evolve over time.* Results indicate that communications are constantly changing. For instance in RQ_1, channels like contributing guidelines have changed over time compared with the consistent ones like issue tracker. Furthermore, in RQ_2, we find that there are differences between popular and non-popular projects in different ecosystems. Results indicate that some channels are ecosystem specific. For example, contributing guidelines are commonly used by seven targeted ecosystems, except PyPI and Maven. While PyPI and Maven use wiki pages alongside Bower where this channel is not prevalence for the other ecosystems. This means the higher starred projects tend to move from externalization to combination. The implication for researchers and practitioners is that understanding the popular channels will help understand where knowledge is shared. For instance, we envision researchers should keep up with the newer channels to understand knowledge sharing within younger projects.

3. *Knowledge in Communication Channels is both external and combination.* Communication channels are used to capture new knowledge (i.e., externalization knowledge). For example, the externalization of the Wiki is very popular. As shown in the preliminary study, the Wiki has some tacit features of being personalized to match the individual project.

   In contrast, updating existing knowledge in communication channels is common. As mentioned by GitHub, contributing guidelines *help them (developers) verify that they're submitting well-formed pull requests and opening useful issues.*[36] GitHub projects are also encouraged to use the platform workflow, with the Issue Tracker becoming a popular tool and communication channel for developers. The final example is the License channel. Putting a license has become increasingly important, especially for projects intended for library reuse. This practice may also be community driven. For example, according to Lertwittayatrai et al. [64], npm projects tend to use the MIT license in their projects. The implication for researchers and practitioners is that understanding where new knowledge is shared. This information could be very useful, for instance, especially for newbies to a project.

3.7 Threats to Validity

We discuss three key threats to the validity of the study. The first relates to the categorization of knowledge. Nonaka and Takeuchi's categorization has been contested in CSCW [97], especially in terms of the tacit knowledge. By adopting the SECI model, we identify channels that have a possibility to capture tacit knowledge. Furthermore, we focus on channels and how they are important for project attractiveness and sustainability.

The second threat is related to the experiment setup and methodology. In this work, we extract common collaborative channels as shown in prior works [13, 4, 45, 113, 16], providing confidence in our channel selection. To reduce feature bias, we used a normalized score in formulating features for the TDA.

[36] `https://help.github.com/en/articles/setting-guidelines-for-repository-contributors`

One key threat is the quality of the channel. For example, the existence of readme files in a project doesn't mean it is used or contains valuable information. This is outside the current scope of work, however, future investigations will focus on quality of these channels and how much knowledge they contain. Finally, we use the star count rating in our project selection, yet this metric has been related to skewedness and not being normalized. Since we assume that social coding is related to the social sharing nature of communication channels, we believe our use of star count is a useful proxy of projects that are more likely to actively use communication channels.

The third threats to validity are the accuracy and the limitation of the tools, especially whether results will change according different sample sizes. As such, we use the largest sample of 10,000 points to ensure confidence in our result. As shown by Lertwittayatrai et al. [64], the result tends to stabilize as more points are added.

3.8 Related Work

In this section, we present related works that complement this study organized into these (i) Communication Channels, (ii) Sharing Architectural Knowledge and (iii) the use of Topological Data Analysis.

3.8.1 Communication Channels

Several studies in other fields analyzed channels as the exchange of information. In organization management, communication methods, whether verbal or non-verbal messages to produce meanings in heterogeneous contexts, cultures and media [56]. Channels are practical in a complex network of relationships where messages are created, delivered and received by individuals, as well as other communication practices that allow larger democracy [56]. A study by Wang et al. [116] investigated the usability, purposes and challenges of channels in industry during safety analysis. Related to software development, communication between developers is possible to augment through collaborative programming [117], or direct communication between team members [98]. Therefore, the channels design or the necessary of social skills in organization management receive more atten-

tion from researchers. Lindsjørn et al. [66] analyzed communication technique to measure the teamwork quality in influencing the performance of software teams and the successful of their team members. The finding indicates that the quality of teamwork in agile teams does not tend to be higher than traditional teams in other similar survey. Team performance is the only effect of teamwork quality which is greater for agile teams than traditional teams. Our study complements these studies, showing how communication channels are indicators of knowledge in a software organization.

In the field of Software Engineering, research into channels is based on social practices. Social practice characterizes the existence of activities which are related to each other [29]. These collaborative works are conducted through (i) distributed teleo-affective structures for software design and development, (ii) shared common or specific knowledge of the software development requirements, and (iii) clear procedures and regulations governing people to accomplish specific activities.[37] Social practices are not confined to only industry-related practices, but more broadly, they can be implemented in open source software projects. In addition to requiring a shared understanding of the requirements to become a project member, a development of open source products performed by complying with common rules as well, and by using a shared teleo-affective. Therefore, the activities of each individual can be connected from the initial of the development to the end of the project. The example of the requirements in the open source projects is also described by Scacchi [96]. This study analyzes channels from a knowledge perspective instead of social collaborations. Our topology confirms that the collaborative and participatory nature of software development continues to evolve, shape, and be shaped by communication channels that are used by development-related communities of practice [63]. A study undertaken by Treude and Storey [111] shows that different media artifacts and channels used for knowledge sharing have different implications for software development. We believe that the methods such as ecosystem topology can provide us a more empirical means to assess inconspicuous patterns within an ecosystem. For example, the topology can reveal the type of channels that were used by most projects.

There is related work that specifically studied GitHub projects, especially

[37] `https://en.wikipedia.org/wiki/Practice_theory` (accessed in July 2019)

library ecosystems and their social collaborations. The social features used in a social coding platform, such as GitHub, has attracted many researchers to analyze. The collaborative features used in their studies, including open bug repositories [4], project fork [13, 45], the usage of a software license [113, 119], and the use of wiki [16]. Open bug repositories, such as the Bugzilla,[38] are mostly managed by open source projects to allow users to be more contributing. Anvik et al. [4] stated that even though these repositories are often used as a reference by open source developers, however the data availability on how they interact with the issues tracking systems is limited. A work carried out by Borges et al. [13] studied that the popularity of a project on GitHub relies on some factors such as the language that developers used to program and the domain of the application. These main elements were presumed to impact on the number of stars of a project. A prior study also analyzed the evolution of software licenses empirically [113]. To complement prior work, this work looks at all channels to provide a holistic viewpoint of all the different channels.

3.8.2 Sharing Architectural Knowledge

The impact of communication channels in Sharing Architectural Knowledge has been highlighted in several studies outside of Software Engineering. For instance, Borrego et al. [14] conducted an empirical study to investigate agile methodologies articulation in unstructured and textual electronic media (such as emails, forums, chats etc.) in global software development. The findings show the involvement of aspects in architectural knowledge in the unstructured and textual electronic media in the teams. Architectural knowledge in the unstructured media is also perceived as important, regardless the interaction frequency.

In a software engineering context, other work studied how knowledge in communication channels impact project and their ecosystem success. Failing FLOSS projects provide insights into some of the outside forces that detract developers from making contributions. A study by Coelho et al. [22] found the following reasons for failing projects: usurped by competitor, obsolete project, lack of time and interest, outdated technologies, low maintainability, conflicts among developers, legal problems, and acquisition. To mitigate these reasons, projects need to

[38] `https://www.bugzilla.org/` (July 2018)

attract as well as retain its existing base of contributors. In fact, Hata et al. [43] suggests that improving the code writing mechanisms (i.e., wikis, official webpage, contributing and coding guidelines and using multi-language formats) leads to more sustainable projects. A study by Storey et al. [106] showed that ecosystems of FLOSS projects are shaped through social and communication channels (sometimes referred to as social coding). Recently, Aniche et al. [3] confirmed that news channels also play an important role in shaping and sharing knowledge among developers. Hence, owners of projects could boost their social presence through participation on recent topics from news aggregators such as `reddit`,[39] `Hacker News`[40] and `slashdot`.[41] In addition, a study conducted by Tamburri et al. [107] described that the characteristics of ecosystem measurement can also be utilized to explain the structure of open-source ecosystem pattern. Our results complement these work and have the similar goal of understanding how projects can attract developer contributions.

3.8.3 Topological Data Analysis (TDA)

The TDA technique has been applied in different research fields outside of software engineering. For instance, a study by Lum et al. [68] used TDA to investigate three different cases, namely, patient identification in breast cancer, implicit networks of the US House of Representatives, and NBA team stratification. The study shows that TDA can handle various types and high-dimensional datasets using three real world examples. From the analysis, the TDA shows the shapes of the breast cancer gene expression networks that allow to identify subtle but potentially biologically relevant subgroups, the shapes of the networks formed across the years about the voting patterns of the members of The US House of Representatives, and the playing styles of the NBA players.

In the software engineering context, the TDA topology has also been applied in such studies. Lertwittayatrai et al. [64] use topological methods to visualize the high-dimensional datasets from a software ecosystem. In the study, the TDA allows the analysis of relationships between six related dataset features

[39] `https://www.reddit.com`

[40] `https://news.ycombinator.com`

[41] `https://slashdot.org`

of a package, that is, author, author domain, license, tagged keywords, version released, and number of dependencies. In our work, we combine all communication channels to understand at a higher level how projects in the ecosystem use communication channels to capture and share knowledge.

3.9 Section Summary

To understand what knowledge sharing occurs in communication channels, we conducted an analysis of channels in 70 thousand GitHub projects. First we conducted a preliminary study to identify and map what knowledge exists and is transferred through the 14 channels. We then used the topological mapper to provide a high-dimensional visual shape of the communications over time and for different library ecosystems. Our work shows that GitHub projects tend to adopt multiple channels. Furthermore, these channels changing over time and can be classified as either capturing new knowledge or updating the existing knowledge.

Based on this work, which established the role of multiple communication channels with knowledge sharing, there are many open avenues for future work: understanding the role and the different combination usage of channels, further studies into cross-channel knowledge, and tool support for channel recommendations, to name a few.

4 Human Aspects in Eclipse Community Forums: Participation, Discussion, and Interaction

4.1 Background

Connectivity and clear communication channels play an essential role in collaborative software development environments by enabling developers to engage with, learn from, and co-create with other contributors. Web-based discussion forums serve as a reliable implementation of such communication channels for software development. As one of discussion groups channels, forums support mass communication and coordination among distributed software development contributors [106]. Forums are considered to be an improvement over mailing lists in that it provides browse and search functions, which are especially helpful for repetitive questions and answers [103].

Recent studies have been conducted that compare forums to communication channels that seem to have similarities to a forum. For instance, there have been studies on mailing lists [40, 122], question and answer sites (Stack Overflow) [120, 115, 126, 18], Microblogs [39, 72], and News aggregators [3]. Kahani et al. analyzed discussion topics using a topic modeling technique [51]. Squire studied the transition from self-supported forums to Stack Overflow [103]. It has been reported that generic question and answer platforms such as Stack Overflow have taken over the roles of forums. For instance, many software development projects had closed their self-supported forums and moved to Stack Overflow [103]. Furthermore, gamification strategies such as awarding of badges

or a voting system of Stack Overflow are considered to be incentives designed to participate and improve answer quality [103].

As a long-living free and libre and open-source software (FLOSS) project, the Eclipse project still maintains an established and active forum. The dataset itself was selected as (part of) targeted data source for three MSR Mining Challenges.[42,43,44] Recently Eclipse has integrated multiple software development systems and has been providing functionalities for analytical and visual information. In December 2016, Eclipse launched the Eclipse User Profile, which shows an overview of all contributor activities within the Eclipse ecosystem, such as contributed projects, reviews in Gerrit, and topics in forums.[45] For this service, contributor accounts in different development systems are integrated to unique profiles, and we can easily see summaries of contributors' various activities in the Eclipse development ecosystem. From the discussion in 2014, donation contributors have been recognized with badges on Bugzilla and Eclipse Community Forums.[46] With such connective functionalities, users can be easily aware of Eclipse community members' status and activities. Nakasai et al. reported that badges for donation contributors in Eclipse's Bugzilla have practical impact on decreasing response time of bug reports, and badges can be considered to be an effective signalling system [77].

In this paper, we would like to investigate how the Eclipse project has successfully maintained its forum, thus creating a healthy ecosystem. We first perform a preliminary study to analyze the participation of users in the Eclipse ecosystem, the characteristics of threads, and the classification of memberships. Then, we set out to empirically analyze over 1 million forum threads and 2,170 connected contributions to other four systems (i.e., around 416 thousand profiles, 120 thousand code review submissions, 532 thousand bug reports with 2,883 commits from multiple projects) within the Eclipse ecosystem. We use the following research questions as a guide:

[42]MSRMiningChallenge2007: `http://2007.msrconf.org/challenge/`.

[43]MSRMiningChallenge2008: `http://2008.msrconf.org/challenge/`.

[44]MSRMiningChallenge2011: `http://2011.msrconf.org/msr-challenge.html`.

[45]Antoine Thomas, The Eclipse User Profile, `http://blog.ttoine.net/en/2016/12/01/the-eclipse-user-profile/`, December 1, 2016.

[46]Bug 434249 - Add decorator for Friends of Eclipse: `https://bugs.eclipse.org/bugs/show_bug.cgi?id=434249`

- RQ1: What is the participation based on the membership?
- RQ2: What kind of discussion are communicated based on the membership?
- RQ3: What is the sentiment of interactions based on the membership?

Our results show that forum members actively participate in posting and responding to the threads equally. We found that the Eclipse forums are dominated by question and answer threads, especially discrepancies, which most of them were posted by junior members. We also found that the status of users is likely relate to the topics discussed in the forums. Finally, sentiment among developers are consistent, with interaction among developers that are neutral or positive having more chances to receive a response compare to other types of interactions. The results of our study provides three sets of recommendations:

- *Users*. Joining a forum discussion is beneficial as forums play a role as the center of knowledge sharing platform. Since forum has various discussion topics and as a bridge to communicate between people, users should identify the appropriate topics and share knowledge in positive manner to maximize the probability of getting responses.
- *Software development projects*. Considering to prepare project-specific discussion forums is recommended for software development projects since forums are not only limited to question-and-answer based discussion, but also enable developers to share such project-specific information or announcement.
- *Researchers*. The study shows that there are many open issues for future work: understanding and supporting forum discussions and further studies of activities in multiple software development systems.

Furthermore, contributions are as follows:

- a comprehensive study of threads in Eclipse community forum, that covers over 1 million threads that have linkage to profiles, code review, bugs and project systems. In detail, we also analyze the category of link targets that were referenced by forum users in the threads as the external sources of knowledge;

- an identification of membership in the Eclipse forums and users contribution in the ecosystem using topological analysis;
- manual labelling of the discussion types communicated between forum members;
- manual analysis on social interactions between individuals within an ecosystem to share knowledge in the forum.

The rest of this chapter is structured as follows. Section 4.2 describes our preliminary study, including the motivation, data collection, an online appendix, approach and the results of preliminary study. Section 4.3 presents our empirical analysis of Eclipse community forums. In details, we explain the motivation that includes three main research questions, and describe the study results. Section 4.4, 4.5 and 4.6 describe our recommendation, threats to validity and provide related works. Finally, we conclude this chapter in Section 4.7.

4.2 Preliminary Study

Before we proceed with the study, we first carried out a preliminary study to describe the users' participation, thread characteristics, and users' classification in the Eclipse forums.

4.2.1 Motivation

The motivation of this preliminary study is to describe statistically the Eclipse community forums from the perspective of threads and users. Unlike the other online question and answer platforms such as Stack Overflow, Eclipse forums serve a communication channel that is not only for a question and answer interaction, but also for general discussion, sharing information, or even announcing such events. Furthermore, Eclipse forums also manage their users by assigning a membership status per user. Therefore, in detail, we would like to quantitatively investigate the contributors' participation in the Eclipse ecosystem, what is being discussed by users (i.e., hot categories of threads) in the Eclipse forums, how the organization communicate with members (i.e., webmaster's message), to what extent are links used as the expert knowledge sources in forums, and classify the

membership types of forum users. To understand the contributors' participation, forum characteristics and users' categories, we carry out an analysis to answer the following preliminary questions:

- ***PS*$_1$*: What is the participation of users in the Eclipse ecosystem?***

  This study is conducted to answer the extent to which users in the ecosystem are (1) using forums and (2) using various systems.

- ***PS*$_2$*: What are the characteristics of threads in the forums?***

  This analysis is to understand the patterns of messages posted by both organization and users, the most frequent discussion topics in the forums, and how users reference the information sources to support their answers.

- ***PS*$_3$*: Are we able to classify the membership of forum users?***

  The classification of forum users is important to understand the impact of membership on the participation, discussion and interaction between Eclipse forum users.

4.2.2 Data Collection

In this section, we describe the data sources and the data extraction used in the preliminary study. As shown in Figure 22, we collected the dataset from multiple sources in the Eclipse ecosystem. To answer the participation of Eclipse contributors in different systems (PS_1) in the preliminary study, we used a topological data analysis (TDA) technique to analyze the dataset from multiple sources (see Section 4.2.2.2). The characteristics of forum threads (PS_2) and membership classification (PS_3) are analyzed using the data of forum threads (see Section 4.2.2.3). The details of the data sources and the data extraction are presented as follows.

4.2.2.1 Data Sources

Figure 22 illustrates the schema that connects the five software development datasets within the Eclipse ecosystem. This connected data enables the integration of contributors' various activities. Since the Eclipse REST API requests are limited to 1,000 an hour, we collect data not only from API but also from individual systems.

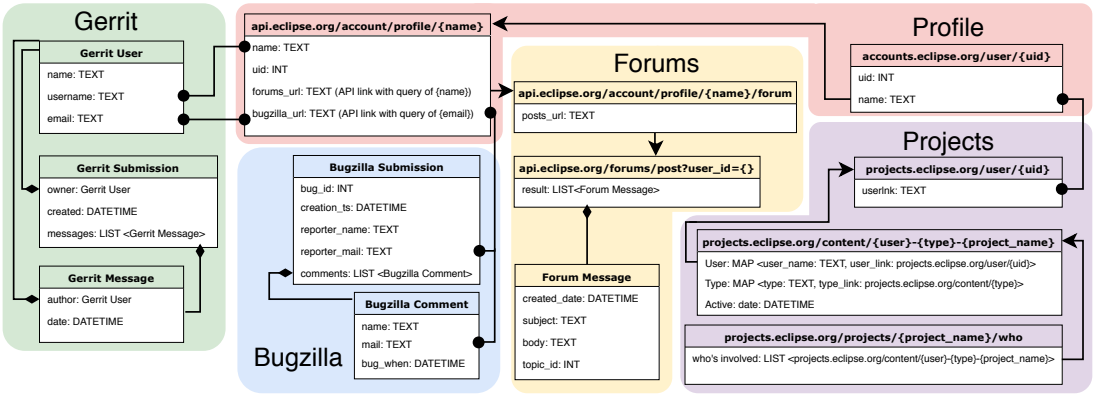


Figure 22: Connected data from Gerrit, Bugzilla, Forums, and Projects via REST API in the Eclipse ecosystem.

- **Profile Dataset**. Each contributor in the Eclipse ecosystem is required to register for a profile.[47] The profile contains basic contributor information and can be used to track contributor activities.
- **Bugzilla Dataset**. The Bugzilla dataset contain the tracking of bug reports and their fixes within the Eclipse ecosystem.[48]
- **Projects Dataset**. To track social roles (committer, review, mentor, etc.) of contributors, we include the project dataset.[49]
- **Gerrit Dataset**. Gerrit is a review tool that facilitates collaboration between committers and contributors of within Eclipse.
- **Forum Threads**. Eclipse community forums, a user-to-user interaction site for Eclipse users, have a hierarchical structure. It contains a number of sub-forum categories which may have several topics. Within a forum's problem-related topic, each new initial thread posted by a user can be responded by other users in the community.

[47] `https://accounts.eclipse.org/user/register`
[48] `https://bugs.eclipse.org/bugs/`
[49] A detailed list of Eclipse projects is available from `https://projects.eclipse.org/`.

Table 19: Connected data extraction from five data sources within the Eclipse ecosystem

step	**dataset**	**quantity**	
step 1:	Profile	416,126	profiles
	Bugzilla	531,752	bug reports
	Projects	2,883	committers
	Gerrit	120,165	submissions
	Forums	1,097,174	threads
step 2:	Connected Gerrit contributors	2,170	contributors
	Connected forum messages	467	messages

4.2.2.2 Contributors from all Eclipse Community Systems

The participation of contributors within the Eclipse ecosystem can be analyzed using TDA from the connected data of five datasets, as described in Section 4.2.2.1. As presented in Table 19, the data used in this analysis are extracted through several steps.

Step 1: In this step, we extracted the dataset from five different sources, that are, Profile, Bugzilla, Projects, Gerrit, and Forums. For Profile dataset extraction, 416,150 *uid*s and *name*s were collected on September 14, 2018 from the profile system.[50] Using the Profile API, 416,126 Profile data were obtained (24 users were not found) from November 5 to 13, 2018. To connect the activities in Bugzilla with the Profile dataset, we used the *email* addresses. We extracted 531,752 bug reports from October 10, 2001 to January 13, 2019. From the Projects data source, we then collected a total of 7,082 commits from the obtained 438 projects (on January 16, 2019). There were 2,942 distinct committers in the records. Among them, the information of 59 committers could not be accessed because of the deletion of accounts. Hence, the remaining 2,883 committers can be connected with Profile dataset through the *uid*. We collected 120,165 submissions from October 1, 2009 to October 31, 2018. We link users with the Profile dataset, using the *name* or *email* to match. To download the fo-

[50] `https://accounts.eclipse.org/user/`

rum threads, we extract the data from its API. We downloaded 1,097,174 threads (topics) available in the Eclipse community forums until January 9, 2019.[51]

Step 2: The connected forum data are collected in this step. On November 23, 2018, we extracted forum message data from Gerrit contributors, via the Eclipse REST API. As part of the data collection process, we applied a pre-processing to detect and remove duplicated accounts. We report 39 pairs of Gerrit and Eclipse forum accounts that have same *name*s but different *username*s. Through manual examination, we verified identities within those pairs; 27 pairs were found to be identical and merged. After the duplicate removal and linking to Profile data, we obtained 2,170 Gerrit contributors and extracted 467 forum message data that connect to these Gerrit contributors.

The collected 2,170 contributors from this extraction are then used to investigate the participation of contributors within the Eclipse ecosystem using a TDA technique. To build the topology, we construct the metrics of each contributor from four data sources, namely, Bugzilla, Gerrit, Forums, and Projects (see the details in Section 4.2.4.1). The metrics describe the connected activities of the contributors within the Eclipse ecosystem.

4.2.2.3 Threads from the Eclipse Community Forum

Eclipse community forum, a user-to-user interaction site for Eclipse users, has a hierarchical structure. It contains a number of sub-forum categories which may have several topics. Within a forum's problem-related topic, each new initial thread posted by a user can be responded by other users in the community. Figure 23 depicts different elements of a thread, including (i) the topic category and subcategory, (ii) the member status and (iii) the link that was posted in the thread.

To improve the quality of our dataset for both threads, as shown in Table 20, the forum threads underwent three stages of filtering.

1. **Step 1**. Raw extraction. In this step, we use all 1,097,174 collected threads yielded from *Step 1* in Section 4.2.2.2.

[51] `https://www.eclipse.org/forums/index.php/t/`

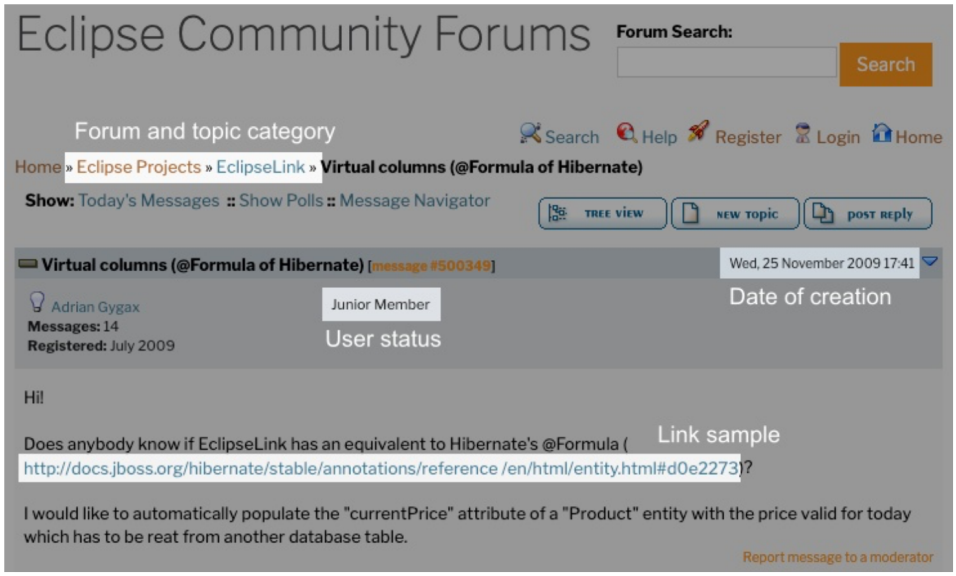


Figure 23: Example of a forum thread

Table 20: Outputs of the pre-processing of the Forum Dataset

step		**# threads**
step 1:	raw extraction	1,097,174
step 2:	remove duplication	832,058
step 3:	separation:	
	(i) threads by webmaster	542,997
	(ii) threads by non-webmaster users	289,061

2. **Step 2**. Remove duplication. In the Eclipse community forums, we found that some threads had been duplicated by the system several times. To avoid redundancies, we removed such threads based on message identity numbers. We were able to reduce the number of threads to 832,058.

3. **Step 3**. Separation. To investigate how the organization communicate with the members and analyze the characteristics of forum usages, discussion topics, etc., we separate those threads posted by the webmasters and the non-webmaster users that include the threads from users that contribute in the other systems. From the total number of threads resulted in *Step 2*,

we were able to separate (i) 542,997 threads posted by the webmasters and (ii) 289,061 threads posted by non-webmaster users.

4.2.3 Online Appendix

Our online appendix contains three data files used in this study: (i) 289,061 threads associated with the information of thread URL, message identity number, forum name and topic, thread title, name of initial-post user, user identity number, and member status; (ii) 216,864 links with the description of thread url, thread title, user's name, extracted link, the link with first directory, domain name, and top level domain; (iii) 2,170 contributors with the metric values described in Table 21; (iv) 1,149 samples of the annotated threads to answer the types of discussion; and (v) 1,149 pairs of main post and first response that are labeled based on the polarity and interaction sentiment. The appendix is available at `https://github.com/yusufsn/EclipseForumData`.

4.2.4 Approach

In this section, we describe our statistical approach to answer the preliminary study questions. We use empirical evidence from the collected threads to discover the thread characteristics, while the sequence number of messages posted by each user is used to classify the type of users.

4.2.4.1 Users' participation (PS_1)

This section is to analyze how contributors participate to each system (i.e., gerrit, bugzilla, projects and forums). We would like to understand how many of the contributors are using multiple systems, and whether or not they are forum users.

To show how contributors use the multiple systems we adopt a topological data analysis technique (i.e., TDA), applies (algebraic) topology and computational geometry to create a shape of a high-dimensional dataset [68]. TDA has been applied in various fields [50, 65], most recently in software engineering by Lertwittayatrai et al. [64] to distinguish characteristics of JavaScript npm libraries (such as licence usage, dependency usage and so on) and Tantisuwankul et al. [108] to analyze the implementation of such communication channels over

Table 21: Contributor metrics

metric	description	system
bug_subm	# of submitted bug reports	Bugzilla
bug_comm	# of bug reports to which only put comments	Bugzilla
review_subm	# of submitted patches for review	Gerrit
review_comm	# of review to which only put comments	Gerrit
thread	# of threads participated in	Forums
committer	# of committer roles	Projects

open source software projects. We use TDA to provide a visual representation that shows (i) the activity levels for each feature and (ii) show the contributor activity accross systems, especially in respect to forums.

As shown in Table 21, we first prepare a contributor metrics data that will be used as features for the TDA mapper. This data contains 6 metrics that pertain to the different activities for each system from 2,170 Gerrit contributors described in Section 4.2.2.2. We use the Knotter tool,[52] which is an implementation of mapper algorithm and the t-Distributed Stochastic Neighbor Embedding (t-SNE) [11], a technique for dimensional reduction and clustering, and our defined features as the filters for the visualization construction.

For evaluation, we present the typologies of the four systems, highlighting the most active contributors. To show this relationship in terms to their contributions to the forum, we will annotate the most active contributors of forums.

4.2.4.2 Characteristics of threads (PS_1)

This analysis includes the investigation of message patterns posted by the organization, the hottest topic of threads discussed in the forums, and the forum linkage to the external sources.

Threads by organization. In the Eclipse community forums, a large number of messages were posted by the organization that deals with the servers and software that runs the eclipse.org site.[53] When the organization posts a message

[52] `https://github.com/rosinality/knotter`

[53] `https://wiki.eclipse.org/WebMaster`

in the forums, they employ the same identity, that is "Eclipse Webmaster". To understand how the Eclipse Webmaster manages its community forums, we used 542,997 messages collected in Section 4.2.2.3. After removing duplicated message contents, we classified webmaster's 934 distinct messages using keywords and manual inspection.

Forums and topic categories. To identify the forums and topics that are mostly discussed by the users, we quantitatively analyzed the messages based on their forum names and topic categories that are embedded in the collected threads. We investigated 289,061 threads that were posted by the non-webmaster users, as resulted in Table 20.

Forums linkage. To understand how forum links to the external sources in the Eclipse ecosystem, we manually analyzed the representative sample of links. The link targets are extracted from 289,061 threads posted by the users (see Table 20). We first prepared a statistically representative sample from 216,864 collected links by computing a random sampled data with a confidence level of 95% and the interval of 5.[54] The calculation of the sample size yields 383 links.

To extract the usage of the link targets, similar to Hata et al. [44], we performed a manual labeling to all link targets from our sample. At this stage, the authors of this chapter specified the code for categorizing the usage of the links. The initial codes used to categorize the links were imported from the study by Hata et al. [44]. However, we dropped several labels because they are unrelated to our research. We also combined and adjusted some codes to make them appropriate to our study. After this step, four authors of this chapter coded the first 30 links from the representative sample independently using the designed codes. The kappa agreement from the four raters is 0.81 or 'almost perfect' [114]. Based on this encouraged agreement, the remaining link targets were then coded by the first author.

The codes used to characterize the link targets including the description are as follows:

- *404*: the link target cannot be accessed or missed
- *bug report or Bugzilla*: a specific bug report or a Bugzilla top page

[54] `https://www.surveysystem.com/sscalc.htm`

- *other documentation*: documentation of a product or project in general except for API documentation
- *personal or organization homepage*: a web page of an individual or organization
- *product or project homepage*: a web page of a product or project
- *API documentation*: specific documentation of an API component
- *tutorial or article*: a tutorial or technical article without comments
- *thread*: thread in the forums
- *blog post*: informational website that displays postings by one or more individuals and usually has commenting section
- *release*: a web page informing the release of new files, new versions of a software, new packages, etc.
- *code*: a web page of a source code file
- *book or research paper*: a web page of a book or entire book or academic paper
- *licence*: licence of a software project
- *other*: anything that does not fit the other labels, or a web page requiring sign-in

4.2.4.3 Membership classification (PS_3)

This section describes our techniques to classify the membership for each message posted by the users.

Unlike the other Q&A online forums such as Stack Overflow, in the Eclipse community forum, all registered users are assigned into three statuses of membership, that are, (1) Junior, (2) Member, and (3) Senior. These user statuses are included in our collected data resulted from *Step 4* in Section 4.2.2.3 which can be seen in every post of a user, as shown in Figure 23. The status of each

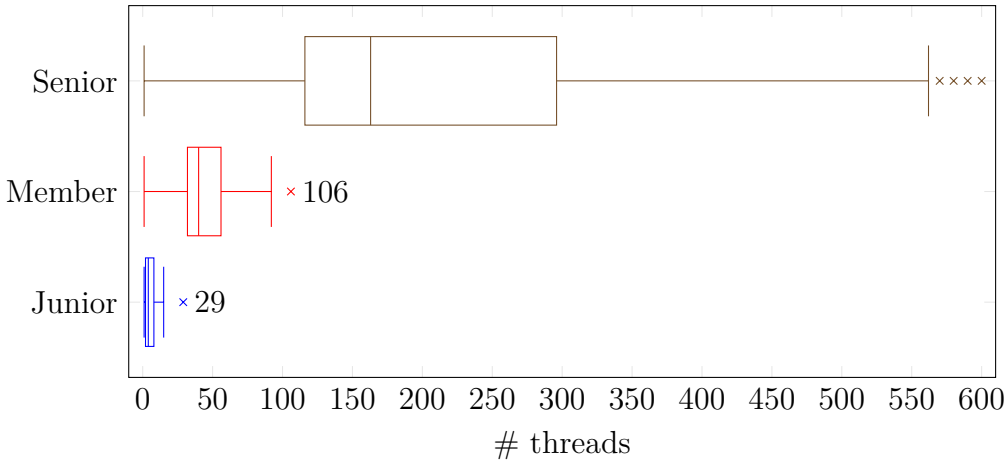


Figure 24: Frequency of messages per user status. The maximum number of posts for each type of users is used to define the threshold of post-based membership. The threshold for Juniors and Members are 29 and 106, respectively. Although Seniors have posted more than 28 thousands messages, we limit up to 600 in the figure.

user may changed from the lowest level (i.e. Junior) into the highest one (i.e. Senior) depends on the contributions of the user in the community. However, in the forum, we could not differentiate which posts that were posted by users when they were a junior, member or senior. This is because once the status of a user has changed, it will replace the old status in all posts of a user with the latest status, including their first posts. Furthermore, we also did not find any information about the time when the status of a user changed.

To define the member status of each registered user, we attempted to calculate the total number of posts of every user. From this amount of threads, we summarized the quantity of posts per author based on the user identity number. The total number of posts per author varies, from less than ten to more than one thousand posts. In this step, we found the maximum number of posts of each user if we consider the latest status of users as collected in the dataset, as shown in Figure 24. The maximum numbers of messages posted by Juniors and Members are accounting for 29 and 106 respectively, while the Seniors have posted the threads up to more than 28 thousands messages. Based on this finding, we used these maximum numbers as the thresholds to determine the user status for

each message based on the sequence number of a post. The sequence number of a post is specified depends on its creation date in order. The earliest post will be assigned as the first post, then followed by the other posts ordered by date of creation.

4.2.5 Results of Preliminary Study

We now present the results of thread characteristics and membership classification.

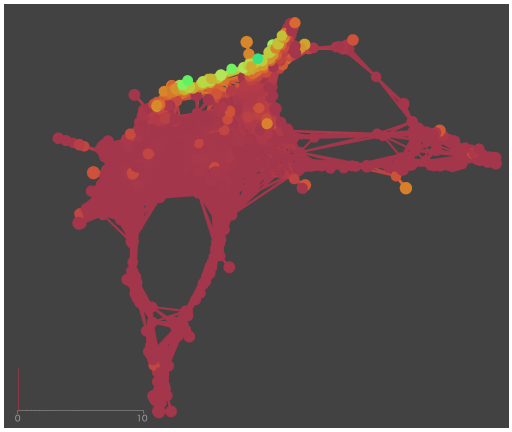
4.2.5.1 *PS_1: What is the participation of users in the Eclipse ecosystem?*

Figure 25 shows the topology of active Eclipse contributors in all other systems. Each node in the visualization represents similar sets of contributors. In general, the map is read as follows:

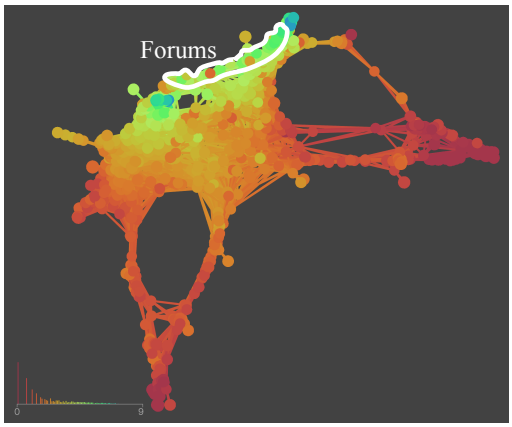
1. *Topology cluster* - a cluster of nodes represent contributors that shared similar features (i.e., share similar contributions per systems). Hence, closely clustered nodes indicate these contributors share the same attributes.

2. *Topology color activity* - the color represent the density of each feature. Starting from red to blue, the red color (i.e., red=low activity) indicates a low activities of the feature, while green to blue color represented high activities (i.e., blue=high activity). For example, with the review comments feature, contributors that contributed many reviews to the Gerrit system were clustered in the green nodes, while those contributors with almost no activity are assigned red color.

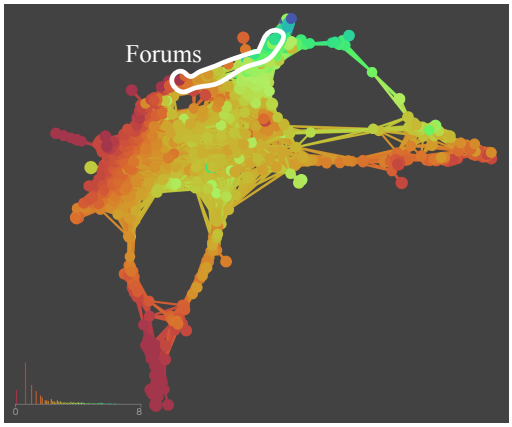
Shape of contributors' participation. Figure 25 shows that the Eclipse contributors with high activities in all other systems are more likely to be active in the forums. As shown in Figure 25b and Figure 25c, Eclipse contributors tend to work on Gerrit and Bugzilla systems to report bug and submit reviewed. This is because the permission may be required for submission and committing of patches, hence it is not easily accessible. Another interesting observation is



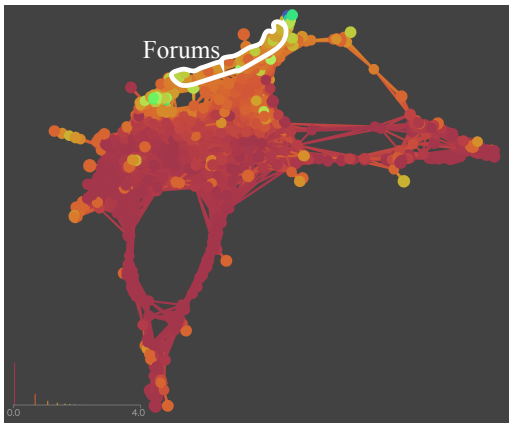
(a) Color activity by `threads` (Forums)



(b) Color activity by `bug_subm` (Bugzilla)



(c) Color activity by `review_subm` (Gerrit)



(d) Color activity by `committer` (Projects)

Figure 25: The topology shows that Eclipse contributors with high activities is all systems (i.e., (a) Forums, (b) Bugzilla, (c) Gerrit, (d) Projects) are active in forums. Note that the metrics are taken from Table 21. The white area represents the contributors that actively participate in the forums.

that forums are used by the more socially active contributors. Thus, this provides evidence that the forum is a source where expert knowledge can be shared.

Summary: We find that active contributors are likely to be also active in forums, making it a source of expert knowledge for all systems.

4.2.5.2 *PS_2: What are the characteristics of threads in the forums?*

The Eclipse webmaster. Table 22 shows the message pattern posted by the Eclipse webmaster. We see that the frequency of the reply and welcome threads have extremely surpassed other classes. The `Re:` messages are the most posted message patterns by the webmaster to indicate the responses of previous messages. The Eclipse webmaster was also frequently sending a welcome message to the newly registered members in a particular community discussions group. This type of message is used to provide a brief statement of the purpose of the newsgroups.

Despite the occurrences of the closure threads are not as frequent as the top 2 patterns, but they were posted by the organization occasionally to inform the contributors that the related forums had been closed. The other pattern of messages that have an adjacent number of threads posted in the forum are test and archive, as many as 14 and 13 message contents, followed by guideline, announce and move, with 6, 5 and 4 messages, respectively. Finally, the prevalence of the pattern "other" in the findings is an indicator of the various message patterns in the threads posted by the Eclipse Webmaster.

Forum and topic categories. Table 23 shows that the Eclipse platform is the most common topic discussed by Eclipse users. This may imply that the threads are more generic, rather than being very specific. The next topic is followed by BIRT, i.e., an open source software project to create data visualizations and reports. Both topics are in the forum of Eclipse projects. In our statistical records, Newcomers has become the third most topic in the discussion list of Eclipse forum. This conversation subject is used dominantly by those who are new in the Eclipse community. The other prevalent topics are EMF of Modeling, C/C++ IDE (CDT) and Java Development Tools (JDT) of Language IDEs, to

Table 22: Webmaster's message patterns

pattern	**description (keywords)**	**#**
re:	a reply for a previous message ("*Re:*")	538
welcome	a greeting to a newcomers ("*Welcome to*")	270
closure	notification of closing threads ("*closure*", "*forum closed*")	25
test	a test message ("*test*", empty message)	14
archive	notification of archiving threads ("*been archived*")	13
guideline	rules, intention, or suggestions for users ("*posting guidelines*", "*please read before posting*")	6
announce	notification of specific events or internal conditions ("*outage*", "*we're hiring!*", "*available*", "*shutting down*")	5
move	notification of moving threads ("*this group has moved to*", "*forum move*")	4
other	a message that does not fit the above	59
sum		**934**

complete the top six topics discussed in the Eclipse community forums.

Taxonomy of link targets. Table 24 shows the result of our qualitative analysis. From the table, we see that a bug-related link target from online issue trackers, such as Bugzilla, is the most links inserted in the messages as a supplement to the answer, accounting for 15%. It indicates that facing problems or finding a software defect was frequently reported by the users in the forum. Other documentations are also prevalent, nearly the same as personal or organizational homepages, accounting for 11% and 10% respectively. The common use of the label 'other' in the link target types represents the heterogeneous of links present in the Eclipse discussion forum. Lastly, we also see that there were more than 28% of link targets referenced by the Eclipse forum members are currently inaccessible (i.e. 404).

Frequently linked resources. Table 25 shows that the top 10 most prevalent referenced domains from the collected 216,864 links. We see that the Eclipse organizational homepage is the most popular link to refer to, which is separated

Table 23: Top 10 forum and topic categories in the Eclipse community forums

forum	**topic**	**# threads**
Eclipse Projects	Eclipse Platform	31,795
	BIRT	29,019
Newcomers	Newcomers	23,054
Modeling	EMF	16,171
Language IDEs	C / C++ IDE (CDT)	13,449
	Java Development Tools (JDT)	13,193
Eclipse Projects	Standard Widget Toolkit (SWT)	11,307
	Rich Client Platform (RCP)	10,842
Modeling	TMF (Xtext)	9,909
	GMF (Graphical Modeling Framework)	8,484

into four most common directories: Forums, and some Eclipse projects (Modeling, EMF, and BIRT). Bugzilla, a web-based general-purpose bug tracker, is the second most prevalent domain to be referenced, followed by the wiki pages and the download pages of Eclipse. Reporting newly discovered issues, or referencing the existing bugs in the Bugzilla is very common amongst the Eclipse forum discussions. The domain of `dev.eclipse.org`, which is currently not available, was also frequently referenced, especially for CVS repositories and news bulletins. The most communal external links posted in the discussions are from `xtext.itemis.com`, i.e. the originator of the Xtext framework, `www.w3.org`, i.e. the world wide web standard organization, `github.com`, i.e. distributed version-control platform, and `twitter.com`, i.e. online news and social network. In addition, the Eclipse Git repositories website (`git.eclipse.org`) remains hot in the forum discussion to complement the top 10 referenced domains.

Summary: Specific projects of `Eclipse Platfrom` and `BIRT`, and the forums for `Newcomers` are most active categories. We found that referencing to other resources, like bug reports, documentation, etc., is common in forum discussions, which indicates that forums are essential platform for linking various resources in Eclipse ecosystem.

Table 24: Frequency of link target types in our sample

availability	target	# links	(%)
available			(72%)
	bug report/Bugzilla	58	(15%)
	other documentation	41	(11%)
	personal/organizational homepage	38	(10%)
	product/project homepage	28	(7%)
	API documentation	20	(5%)
	tutorial or article	16	(4%)
	thread	15	(4%)
	blog post	10	(3%)
	release	8	(2%)
	code	4	(1%)
	book or research paper	1	(0%)
	licence	1	(0%)
	other	34	(9%)
not available			(28%)
	404	109	(28%)
sum		**383**	**(100%)**

4.2.5.3 *PS_3: Are we able to classify the membership of forum users?*

Users' classification. As shown in Table 26, we were able to assign the status of the users in all threads based on the sequence number of the posts and able to distinguish the messages that were posted by the users when they were junior, member and senior. Questions or answers that are assigned from 1 to 29 are considered as posts that were posted by Juniors, while posts from number 30 to 106 were posted by Members, and Seniors posted questions or answers from 107 up to the remaining posts.

In the Eclipse forums, we also found that there are a lot of threads posted by authors with the username "Eclipse User", accounting for 30% of the forum users, as described in Table 26. In the threads, this username is automatically assigned by the system and could be from anyone. It does not show the status

Table 25: Frequently referenced domains in the Eclipse community forums

domain		**description**	**# links**	
www.eclipse.org			39,917	
	/forums/	Eclipse Community Forums		(5,723)
	/modeling/	Eclipse Modeling Project		(4,076)
	/emf/	Eclipse Modeling Framework (EMF) (moved to under the above /modeling/)		(3,119)
	/birt/	Eclipse BIRT (Business Intelligence and Reporting Tools) Project		(2,391)
bugs.eclipse.org		Eclipse Bugzilla	30,135	
wiki.eclipse.org		Eclipse wiki pages	22,920	
download.eclipse.org		download page for Eclipse product	9,343	
dev.eclipse.org		(currently not available or redirected to wiki.eclipse.org/Development_Resources)	9,297	
	/viewcvs/	Eclipse CVS repositories (not available)		(4,528)
	/newslists/	Eclipse news bulletin (not available)		(2,533)
xtext.itemis.com		Xtext framework for programming language development	3,459	
www.w3.org		international standard organization of world wide web	3,323	
github.com		web-based hosting service for version control using Git	3,298	
twitter.com		an online social networking site	3,096	
git.eclipse.org		Eclipse Git repositories	2,123	

explicitly whether Junior, Member, or Senior. Furthermore, it does not have a user identity number and a link that connects to the user's profile. Since we only focus to identify the three statuses of the user in the analyses, the messages that were posted by "Eclipse User" were excluded. After the exclusion process, we obtained 202,602 threads that were only posted by Juniors, Members and Seniors.

Message distribution. Figure 26 shows the contributions of forum members in the Eclipse forums. Even though the number of junior members is higher, they had posted a smaller quantity of posts, compared with the two other user categories. In contrast, although the total number of senior members is lower than the other forum members, most seniors have contributed in the forum multiple times.

Table 26: Frequency of posted threads based on the latest status of user

status	**# threads**	**(%)**
Junior Member	144,440	(50%)
Member	31,719	(11%)
Senior Member	26,443	(9%)
Eclipse User	86,459	(30%)
sum	**289,061**	**(100%)**

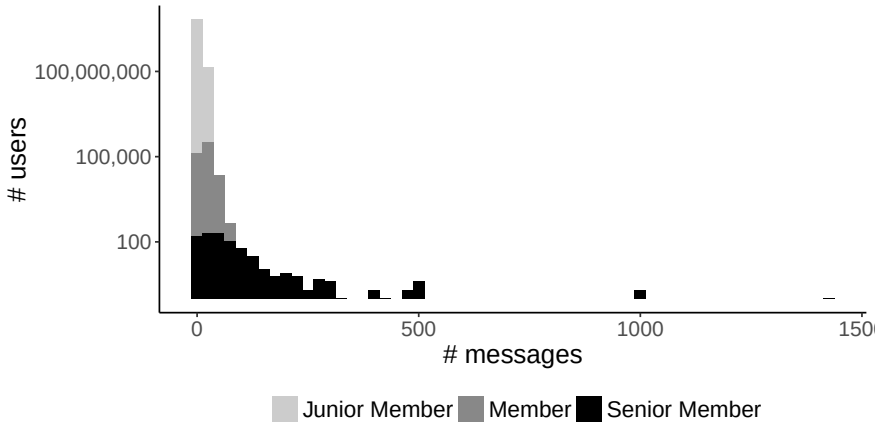


Figure 26: Distribution of messages per user

Summary: We are able to classify the membership of forum users based on the sequence number of messages posted by the users. Although the majority of users are junior members, senior members tend to use the forum more frequently.

4.3 An Empirical Analysis of Eclipse Community Forums

Taking the findings from the preliminary study, we are now able to study the membership-based participation, discussion, and interaction in the Eclipse community forums.

4.3.1 Motivation

Our motivation for the empirical analysis is to present the nature of participation, discussion, and interaction between users in the Eclipse forums. To guide the study to achieve these goals, we constructed the following research questions.

- ***RQ_1: What is the participation based on the membership?***

  The motivation of RQ_1 is to analyze the extent to which type of developers in the ecosystem that frequently responds the posts of a particular type of user.

- ***RQ_2: What kind of discussion are communicated based on the membership?***

  The key motivation for RQ_2 is to investigate the different kinds of discussion that users share in forums. We conjecture that different memberships share different types of discussion.

- ***RQ_3: What is the sentiment of interactions based on the membership?***

  The motivation of RQ_3 is to get deep insight on developers sentiment while sharing knowledge. Throughout this RQ, we aim to test the degree of consistency while interacting with each other.

4.3.2 Results

In this section, we present the answer to the research questions and describe the results.

4.3.2.1 *RQ_1: What is the participation based on the membership?*

To study how forum users participate in the discussion, we divided the type of threads into two categories, (i) bug-related threads, and (ii) non-bug-related threads. In this analysis, we targeted all 202,602 threads that were posted by the non-Eclipse Users (combination of Juniors, Members and Seniors). To distinguish the category of threads, we filtered the threads using the Eclipse bug-report URL as the keyword, that is, `bugs.eclipse.org`. If the threads contain at least

Table 27: Frequency of developers responding bug-related and non-bug-related threads

Askers / Answerers	Bug-related Threads			Non-bug-related Threads		
	Juniors	Members	Seniors	Juniors	Members	Seniors
Juniors	19,769	3,507	1,736	7,583	972	1,296
Members	15,097	2,180	1,346	1,740	1,262	809
Seniors	65,731	13,698	5,634	4,803	1,443	5,144

one keyword of the bug-report URL, the threads are categorized as bug-related threads, otherwise, non-bug-related threads. After this process, we subsequently investigated 113,592 bug-related threads, and 89,010 non-bug-related threads.

For both categories of threads, we separated the targeted threads into three classes based on the status of the askers, that is, Juniors' thread, Members' thread, and Seniors' thread. In each thread, we then classified the response messages into the same classes as the askers to investigate the user types of the answerers. To reduce bias, the answers from the same authors who ask the questions in the threads were excluded. We also removed the response messages with duplicate identification numbers of the answerers in each thread. Finally, we quantified the frequency of the answerers grouped by their membership statuses.

As shown in Table 27, Seniors seem to be the most active forum users in responding any category of questions, followed by Juniors and Members respectively. In the category of bug-related threads, all types of users are more interested in answering questions that were posted by Junior members. It is shown by the large number of authors replied Juniors' questions. Table 27 also hints that Seniors are more interested in answering the bug-related threads in comparison with Juniors and Members. In the non-bug-related threads, the type of answerers likely to have the same types as the askers, except the Members. Junior users tend to answer juniors' questions, and Senior users seem to be more interested in answering seniors' questions.

Although the number of responses is not as high as in the questions posted by Juniors and Members in both categories of threads, the Juniors and Members seem to reply the Seniors moderately. It indicates that the communication between forum users is not limited by the type of users.

Summary: Compared with Juniors and Members, Seniors tend to respond to bug-related threads more frequently. All forum users, irrespective of membership type, actively participate in posting and responding to the threads equally.

4.3.2.2 *RQ_2: What kind of discussion are communicated based on the membership?*

Our approach to answer RQ_2 is through manual analysis of a representative sample to extract the category of discussion among the forum users. Since this analysis only focus on the threads that were posted by three types of registered users (i.e. Juniors, Members and Seniors), we excluded all threads that were posted by "Eclipse Users". Similar to the procedure of forum linkage in Section 4.2.4.2, the statistically representative samples from the 202,602 threads were prepared. The calculation of the sample size yields 383 threads for each status of the askers.

To extract the discussion types, we conducted an interactive process of coding. In this process, three authors of this chapter firstly discussed the initial coding guide from previous work [11]. To make the coding guide fit with our study, the initial coding guide were adjusted and the other codes were added for labeling the threads. Then, the authors coded 30 main posts independently for each type of askers in the sample using the designed labels. The kappa scores[55] for each status of askers were calculated to see the level of agreement between three authors. We gained 0.76 and 0.72 for both Juniors and Members which mean 'acceptable', and 0.82 for Seniors' threads which indicates 'almost perfect' [114]. Based on these motivated agreement scores, the coding task for the remaining main posts from the sample were undertaken only by the first author.

The following terms list shows the labels used in our analysis to code the threads discussed including the descriptions:

1. **Q&A threads**: if the thread is triggered by a question for asking a solution, reason, advice, clarification, or guidance even if the questions are unanswered, then we used the following coding guide:

[55]http://justusrandolph.net/kappa/

- *Errors*: post contains **exceptions or the stack trace** and/or asks for help in fixing an error or understanding what the exception means.
- *Review*: this category merges the categories Decision Help and Review [110], the category Better Solution [12], and What [93], as well as How or Why something works [2]. Questioners of these posts ask for better solutions or reviewing of their **code snippets**. Often, they also ask for best practice approaches or ask for help to make decisions, for instance, which API to select.
- *Plans*: main question is asking for the future plans or processes of the development.
- *Learning*: this category merges the categories Learning a Language or Technology [2] and Tutorials or Documentation [10]. In these posts, the **questioners ask for documentation, tutorials, or examples** to learn a tool or language. In contrast to the first category, they do not aim at asking for a solution or instructions on how to do something. Instead, they aim at asking for support to learn on their own.
- *Usage*: this category subsumes questions of the types **How to implement something** and Way of using something [2], as well as the category How-to [12, 110], and the Interaction of API classes [10]. Specific functionalities are mentioned in the question. The questioner is asking for concrete instructions.
- *Versions*: this question category is equivalent to the categories Version [12] and API Changes [10]. The questions arise due to the changes in an API or due to compatibility issues between different **versions** of an API. Specific versions are mentioned in the questions.
- *Conceptual*: High-level question. The posts of this category are equivalent to the category Conceptual [110] and subsumes the categories **Why...?** and **Is it possible...?** [12]. Furthermore, it merges the categories **What** [93] and **How/Why something works** [2]. The posts consist of questions about the limitations of an API and API behaviour, as well as about understanding concepts, such as design patterns or architectural styles, and background information about

some API functionality.

- *Discrepancy*: Low-level question. This question category contains the categories **Do not work** [2], Discrepancy [110], **What is the Problem...?** [12], as well as **Why**. The question is asking about problems or other types of question, but the questioner has **no clue how to solve it**.

2. **non-Q&A threads**: if the thread is triggered by a non-question post, we categorized the topics using the following labels:

   - *Misuse*: a post is identified out of the scope, unrelated to the community forum, or difficult to understand.
   - *Announcement*: a post provides an announcement from system or core developers about specific events (e.g. future updates, file release).
   - *Information*: a post provides a general information. It could be from anyone.
   - *Recruitment*: indicates an offer of a job vacancy or recruiting people.
   - *Test*: a post is used for a test.
   - *Other*: anything that does not fit the above labels, including posts that answering questions from different threads (the post is not an original question from the asker).

Taxonomy of discussion type. The results of our manual classification show that the Eclipse forum is very much similar to other community-based question-and-answer (Q&A) sites, dominated by the users that share problems through questions to get the answers from the community. As shown in Table 28, discrepancy is the most common type of problems shared by the forum members, especially from junior members. Review, conceptual, usage, and errors complete the top five types of discussion that dominate in the Eclipse forum shared by all forum users, which accounts for more than 10% for each type. For the non-Q&A threads, information and announcement are two types of discussion that were most frequently shared between the forum members.

Table 28: Frequency of knowledge type and the number of links in our sample (gray color represents Q&A threads, white color represents non-Q&A threads)

Topic	Juniors		Members		Seniors		Total	
	#	%	#	%	#	%	#	%
Discrepancy	153	(39.9%)	116	(30.3%)	59	(15.4%)	328	(28.5%)
Review	51	(13.3%)	64	(16.7%)	39	(10.2%)	154	(13.4%)
Conceptual	31	(8.1%)	50	(13.1%)	62	(16.2%)	143	(12.4%)
Usage	45	(11.7%)	36	(9.4%)	45	(11.7%)	126	(11.0%)
Errors	49	(12.8%)	41	(10.7%)	32	(8.4%)	122	(10.6%)
Information	6	(1.6%)	17	(4.4%)	46	(12.0%)	69	(6.0%)
Learning	19	(5.0%)	19	(5.0%)	13	(3.4%)	51	(4.4%)
Versions	9	(2.3%)	22	(5.7%)	18	(4.7%)	49	(4.3%)
Announcement	3	(0.8%)	5	(1.3%)	22	(5.7%)	30	(2.6%)
Plans	8	(2.1%)	4	(1.0%)	9	(2.3%)	21	(1.8%)
Misuse	1	(0.3%)	1	(0.3%)	6	(1.6%)	8	(0.7%)
Recruitment	2	(0.5%)	1	(0.3%)	2	(0.5%)	5	(0.4%)
Test	0	(0.0%)	2	(0.5%)	3	(0.8%)	5	(0.4%)
Other	6	(1.6%)	5	(1.3%)	27	(7.0%)	38	(3.3%)
sum	**383**	**(100%)**	**383**	**(100%)**	**383**	**(100%)**	**1,149**	**(100%)**

Our other findings on the discussion types that shared amongst the forum members at least hint that the forum is not only utilized for asking solutions or responses from other users, but also to announce some specific events or updates, distributing general information, offering some job vacancies, or even just testing the forum as well.

Membership based discussion types. Our findings also highlight that there are several distinguished types of discussion that were communicated between the members based on their statuses. As illustrated in Figure 27, discrepancy is the most common type that was shared by Junior members. However, the number of this type reduces once the status of the user levels up into higher status. Conversely, the quantity of conceptual discussions increases inline with the changes of the user statuses, from the lowest level of membership (Juniors) to the highest one (Seniors). This shows that the higher the membership status of a user, the higher the level of the problems discussed in the forums. Further-

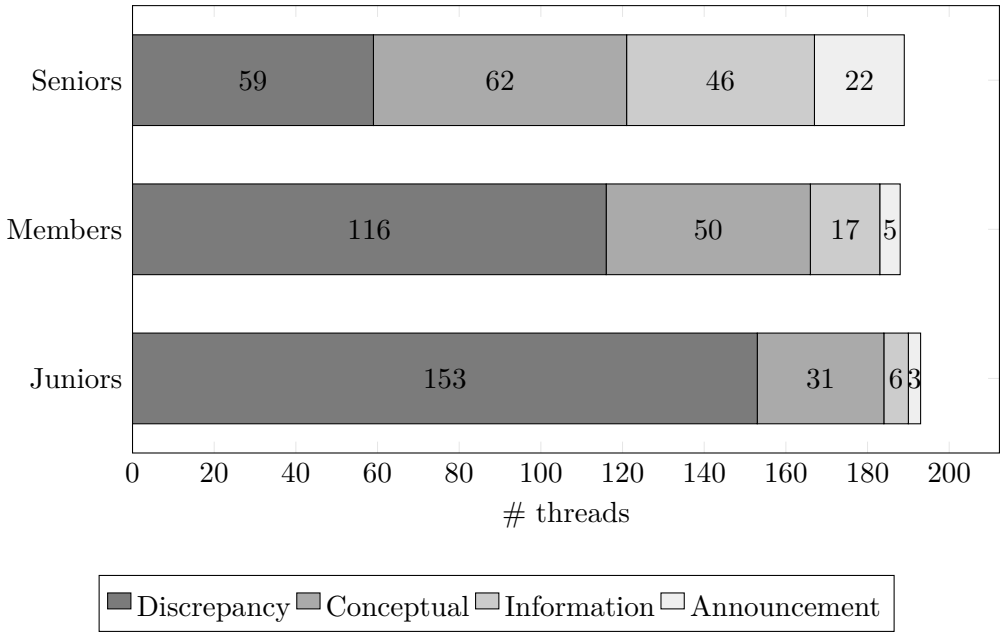


Figure 27: Four discussion types communicated in the forums

more, senior members posted general information and announcement more often than the other types of users. The information shared in the forums is commonly general events, such as a seminar, presentation or competition to motivate users' participation. While the announcement describes specific events related to products, for example, file release, future updates, etc. This result at least hints that most of the senior members are core members of a software development company who act as sources of information and convey the information to inspire and motivate other people [121].

Summary: The Eclipse forums are dominated by question-and-answer threads, especially discrepancies, which most of them were posted by junior members. Furthermore, the status of users is likely relate to the topics discussed in the forums. The higher the status of users, the more conceptual the discussion type they communicate in forums.

4.3.2.3 *RQ_3: What is the sentiment of interactions based on the membership?*

Our approach to answer RQ_3 is through manual analysis of a representative samples to understand the sentiment of forum users while sharing knowledge. To analyze the sentiment, we only focus on the questions that were posted by three types of users (i.e., Juniors, Members and Seniors) and their first replies of the collected questions. This is because the first reply in each thread shows the tangible feeling of a user reaction in responding to the main question. Similar to Section 4.3.2.2, we excluded all the questions from "Eclipse User". We then prepared statistically representative samples for each users type with 95% confidence interval.[54] The calculation of the sample size yields 383 pairs of question and first response for each status of user.

To find the types of sentiment among users, we conducted an interactive process of coding. In this process, three authors of this chapter discussed the initial coding guide from previous work [53]. To make the coding guide fit with our study, the initial coding guides were adjusted. We then independently applied the adjusted question coding rules on both questions and the first responses of the first 30 samples of thread. We used the kappa score calculator[55] to check the agreement level and find the score 0.78. According to [114], this kappa agreement score is 'acceptable'. Based on the agreement level, the coding tasks for the remaining threads from samples were undertaken by the three authors, which each author classified 383 different samples of threads.

Our sentiment analysis is composed of two different analysis, that is, (i) polarity, and (ii) interaction. The following terms list shows the labels used in our analysis to code the type of polarity and interactions while sharing knowledge through forums.

1. **Polarity analysis**: To analyze the polarity of messages, we used the following three labels:

    - *Positive*: It includes posts that has positive feeling while reading and also include positive words (i.e., good, nice, working example, thanks a lot etc). For example:

        Thanks, you have been a great help. It is enough for me.

- *Neutral*: It includes posts that has neutral feeling (e.g. started with positive then ended with negative talks) while reading or do not include any biasing words (i.e., positive, negative). For example:

  > *See the Task List's view menu's "Show UI Legend" action for an explanation of the colors and a link to how to change them.*

- *Negative*: It includes posts that has negative feeling while reading and also include negative words (i.e., error, bug, not working etc). For example:

  > *I've added some custom key binds in M8 and sometimes they don't work. They still show up in the preferences dialog, but they are not indicated on the menu items they are assigned to. Closing Eclipse and restarting fixed the problem. Anyone else seen this? I don't know of any repro steps at this point.*

2. **Interaction analysis**: To analyze the interactions between forum members in the threads, we used the following seven statements:

   - *Positive procedural (PPc)*: It includes procedural statements that are goal oriented (i.e., pointing out or leading back to the topic), procedural suggestion (i.e, suggestions for further procedure), procedural questions (i.e., questions about further procedure), economical thinking (i.e, weighing costs/benefits), etc.
   - *Positive socioemotional (PS)*: It includes socioemotional talks like encourage participation, agreeing suggestions, offering praise, etc.
   - *Positive proactive (PPa)*: It includes proactive statements that discuss about interesting ideas, plan actions, agreeing upon tasks to be carried out, etc.
   - *Neutral*: It includes statements that can't fall in the above categories or posts that are neutral in nature.
   - *Negative procedural (NP)*: It includes procedural statements that talks more about failure of procedure, complaining the procedure (i.e., behaviour of API, code), irrelevant things, etc.

Table 29: Polarity of communication among developers

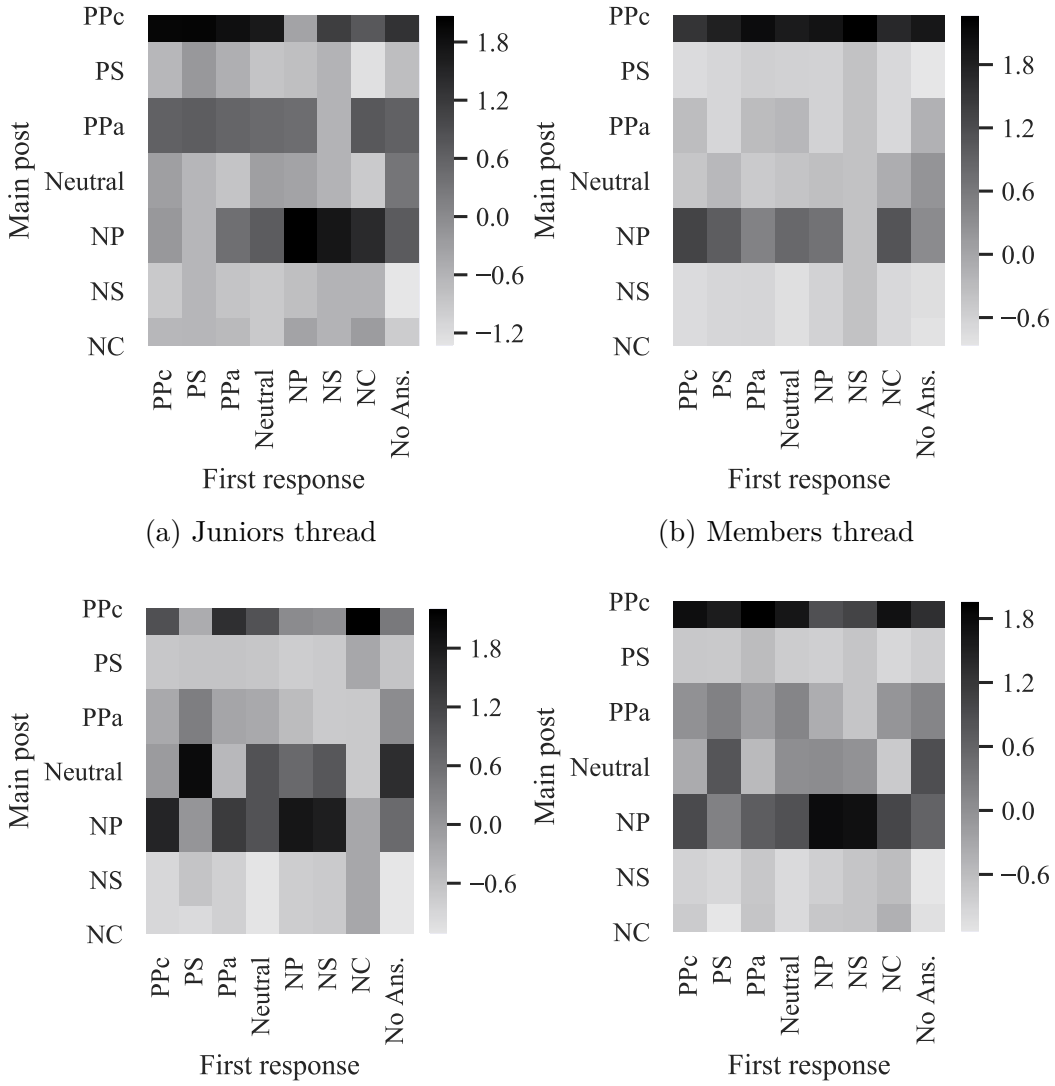
users' type	**main post**	**first response**				**sum**
		positive	neutral	negative	no response	
Juniors	positive	93	67	22	53	**235**
	neutral	8	12	3	17	**40**
	negative	23	32	28	25	**108**
Members	positive	139	19	15	63	**236**
	neutral	10	2	2	20	**34**
	negative	71	9	8	25	**113**
Seniors	positive	56	24	14	75	**169**
	neutral	19	16	9	53	**97**
	negative	41	17	18	41	**117**
sum		**460**	**198**	**119**	**372**	**1,149**

- *Negative socioemotional (NS)*: It includes socioemotional talks like criticizing, disparaging comments about others, etc.
- *Negative counteractive (NC)*: It includes counteractive statements that cover irrelevant problems, no action plan, terminating discussions, etc.

Polarity detection. Table 29 shows the polarity results of main posts and first responses among developers. We found that Juniors and Members post more positive messages than Seniors. On the other hand, compared with the other types of developers, neutral and negative messages were mostly posted by senior developers We also observed that overall positive posts tend to receive more replies while negative posts seem to receive less responses.

Social interaction. Figure 28 shows the nature of first responses in the interaction among different type of developers. From Figure 28a and 28b, we observed that social interaction by juniors and members are overall positive. We found that juniors and members tend to post a thread in a more positive procedural way. For this positive procedural type of interaction, juniors receive more positive responses while members receive both positive and negative responses.

In contrast with the other types of developers, Figure 28c shows that seniors post more neutral and negative messages. Complement to the RQ_2, the neutral threads mostly include an announcement relates to a release of a new product,



(a) Juniors thread

(b) Members thread

(c) Seniors thread

(d) Combination of all user types

Figure 28: Nature of first responses in the interaction between forum members. The color scale represents the frequency of the first responses. The darker the area in the heatmap, the more frequent the first responses in the threads.

and general information such as advertisement, seminar, etc. In addition, seniors did not receive responses in a large number of threads. This hints that seniors tend to post more complex questions than juniors and members that may be difficult to answer. If we combine all types of developers together as illustrated in Figure 28d, we observed that the social interaction among developers are generally similar to the polarity. The result shows that, procedural posts tend to receive more responses than the other types of posts. This is indicated by positive and negative procedural messages that receive the highest number of first responses. Furthermore, posting a message in a positive procedural way is likely to have higher chances of getting positive replies. In contrast, negative responses are frequently addressed to negative procedural messages. In sum, the results suggest that developers are encouraged to post procedural threads and thus increasing the probability of receiving responses.

Summary: We found that sentiment among developers are consistent while knowledge sharing. Interaction among developers shows that procedural threads overall have more chances to receive first responses than the other types of interaction.

4.4 Recommendation

Based on our study results, we provide three sets of recommendation for forum users, software development projects, and researchers.

For forum users, the recommendation regarding the threads and referencing links are as follows:

- *Join forum discussions*, as forums play a role as the center of knowledge sharing platform. Not only knowledge from community the users will receive, but also from external sources.

- *Identify appropriate forum topic*. As found in this chapter, some amount of users posted their threads in the incorrect discussion groups. Sharing problems in a compatible topic category would maximize the probability to

get responses and solutions. Otherwise, the questions might be marked out of scope.

- *Share knowledge in positive manner.* As shown in the results, users are likely to respond to a positive and procedural thread. Since the clarity of knowledge shared in the forums might increase the chances to get responses, describing the problems in a procedural way is essential.

The findings of this study also recommend to software development projects as follows:

- *Consider preparing project-specific discussion forums.* As observed in the study, Eclipse forums is not only limited to Q&A based discussion, but also enable developers to provide Eclipse community-related information or announcement. Since the information or announcement that specifically relates to the software projects is important to share amongst the community, providing a project-specific forum is necessary. GitHub Discussions, a new feature of GitHub for asking questions or discussing topics outside of specific Issues or Pull Requests,[56] could be an option for projects hosted on GitHub.

For researcher, we can consider future works with the following possible challenges:

- *Further studies of activities in multiple software development systems.* Our results show that forums are being actively used by contributors that are also active in the other systems. This is indeed evidence that supports the claim that forums do work as a center of knowledge-sharing. Possible future directions could include tracking the actual threads to understand whether the discussions in forums cross-over into the review and bugs systems, and to what effect.

- *Causal inference on promoting career.* From Eclipse historical data of social roles, we can also collect contributor metrics of leader. Combined with

[56] https://github.blog/2020-05-06-new-from-satellite-2020-github-codespaces-github-discussions-securing-code-in-private-repositories-and-more/\#discussions

the metrics of committer, we can address the impact of the forum on the social leadership, that is, the causal effect of forum participation on being promoted from committers to leaders.

4.5 Threats to Validity

Several threats to the *construct validity* emerge in our study. Related to the PS_1 and RQ_1, it is possible that there are identical users that have different identity number and different status in the forums. We also found in several threads, a message posted by a user had been duplicated with different message identity number. Since we distinguished the users and the messages by their identity number, in our analysis, we recorded them differently even though they are same. In relation with forums linkage, we found that not all links in the threads can be extracted since they are written in plain text by users. However, the number of these issues were small. Thus, we consider that the impact of the missing links is not significant.

Threats to the *external validity* appear in our data preparation. Even though we investigated a large scale of discussion forum in Eclipse, the findings could not be generalized to other organizations.

We diminished the threats to *reliability* by preparing the online appendix of our collected threads, links, contributors, and the results of our manual annotation for RQ_2 and RQ_3 (see Section 4.2.3).

4.6 Related Work

In this section, we present some related works to analyze the forum discussions.

Stack Overflow. A number of studies on a web-based question-and-answer communication channel, Stack Overflow (SO), has shown its importance. Zagalsky et al. [122] reported that SO with R-tag has became one of the two questions and answers (Q&A) communication channels that have a relationship with the users' discussions in R software development community forum. The collaboration between members and the independent individuals work have shaped the knowledge characteristics of the community. Squire [103] showed that the quality measurements, participation of users, and the effectiveness of responding time in

the SO forum have been the main factors that induced the developers' movement from self-supported forums and mailing lists. Ye et al. [120] analyzed the reason behind the inclusion of web links in SO forum. It has been reported that the forum users share the URLs in SO for various purposes. Referencing sources to provide the solutions is the most prevalent purposes for the users who posted the web links. The guidelines for users in making a successful question in SO forum has been proposed by Calefato et al. [18], whilst Wang et al. [115] investigated the factors related to the time of getting an accepted answer in four Stack Exchange websites. The factors of potential-to-success questions are conciseness, completeness, and the exactness of the characters usage [18]. In line with that, after controlling other factors, the accepted answers has been affected strongly by the dimension of the answerers [115]. The more frequent a user in answering the questions, the faster to receive an accepted answer. The study by Zou et al. [126] describes the insights of developers' requirements through SO. The authors found that the usability and reliability are the most common topics discussed in the SO forum and also the most unresolved problems faced by the users. By visualizing the development activities evolution over time, most software developers interested in the functionality and reliability, while the usability still becoming the trends.

In our study, the basic idea of the thread codes basically referred to the artifacts types proposed by Zagalsky et al. [122]. Since we analyzed the entire discussion for each topic (not per message), we modified the label used in our study to code the types of thread (see Section 4.3.2.2).

Twitter and news aggregators. Communication between programmers not limited only in mailing lists and Q&A channels, but also frequently shared on social media (i.e. Twitter) and news aggregators. The information hidden in the tweets posted by Twitter users have been attracting for researchers to explore in a number of studies. Mezouar et al. [72] investigated the tweets posted by users to relate to the bug reports after removing the noisy tweets. Guzman et al. [39] analyzed the characteristics of the usage of Twitter, information in the messages, and classified automatically the potential messages about software applications. The understanding of developers' motivation to contribute, and how the knowledge shaped in the community has also been explored from two modern

news aggregator sites: Reddit and Hacker News [3]. The authors highlighted that the community size, the interaction scope between users, and the social features might change the shape of knowledge and the sharing method of the development communities.

Senior contributors. Software development could not be separated from users' participation in a discussion forum. Their contributions are not always related to writing code. Recent studies show that every individual has an opportunity to become a valuable contributor. Zhou et al. [124] built a model to analyze the users' chances of becoming a senior contributor depend on her competence, passion, and first-time contribution opportunity. Zhou et al. [125] found that the participation of new members in the issue tracking system environment might impact to their status of becoming a long term contributor.

Junior contributors. Despite most valuable information for the improvement of a software quality come from the experienced members in a software project [65], software developers should not underestimate the newcomers' contributions in a discussion forum. However, the support from core developers are needed to motivate the young members to actively participate [82], even though their contributions unrelated with programming. Steinmacher et al. [105] identified that the lack of social interaction with the community, having unanswered questions or receiving delayed answers, and their technical experience backgrounds are some difficulties of new members faced when they make contributions to an open source software project. To support the newcomers to overcome their obstacles since their first participation in an OSS project, Steinmacher et al. [104] designed and built a conceptual model. The proposed model has succeeded in guiding the young members and minimizing their problems during the contribution process. The contribution characteristics of new members in OSS development has also been studied by Middleton et al. [73]. The authors identified that the participation forms of the newly members, such as pull request and how they comment in the discussion influence the decision to join OSS teams.

4.7 Section Summary

To understand the impact of Eclipse community forums related to the linkage of the ecosystem and the social leadership, we conducted (i) a preliminary statistical

study of 1,097,174 threads to analyze the characteristics of threads, contributions of Eclipse users, and users' classification; and also (ii) a large-scale study on the membership-based participation, discussion, and a sentiment investigation on the social interaction.

Our preliminary study has shown that forums are essential platform for linking various resources in Eclipse ecosystem and a source of expert knowledge for other development systems. The results of our analysis reveal that membership plays an important role in shaping the community through participation, discussion and interaction between developers. Thus, maintaining project-specific forums is essential since it has an impact on the health of the ecosystem. Based on this work that clarify the roles of forums in ecosystem linkage and the impact of membership in shaping the knowledge sharing activities in the Eclipse community forums, there are many open issues for future work: understanding and supporting forum discussions, further studies of activities in multiple software development systems, and causal inference for assessment, to name a few.

5 Conclusions

This thesis has addressed the impact of uncommunicated update on relevant domains, the implementation of communication channels within software ecosystems, and the human aspects for sharing knowledge in Eclipse community forums. The summaries of this dissertation are as follows:

- **Systematic mapping study**. This work was initially carried out to investigate how previous studies implemented the `git diff` command. 52 papers were selected from 3 top journals and 8 high ranking international conference proceedings that were published between 2013 and 2017. The key outcome of this mapping study is a set of indicators for conducting three comparison analyses.

- **An empirical study of different `diff` algorithms in Git**. Based on three most common purposes for using `git diff` command that were found in the systematic mapping study, three comparison analyses between two `diff` algorithms were undertaken, that are, collection of metrics, identification of bug-introduction using SZZ algorithm, and manual comparison of patches. The main outcomes of these comparisons are two-fold. First, there are 1% to 8% of the changed lines of code produced by *Myers* and *Histogram* are identified differently in the number and the position, and 2% to 7% files were detected differently in both metrics collection and SZZ algorithm implementation. Second, based on the result of manual comparison of patches, the *Histogram* `diff` algorithm is found to be more appropriate than the *Myers* in 62% files to describe the changes of code. Since the implementation of different `diff` algorithms has an impact on a study result, considering different `diff` algorithms is an important knowledge. Thus, various types of knowledge, either tacit or explicit, are necessary to document and communicate.

- **A topological analysis of communication channels for knowledge sharing**. The first finding of this study is the classification of 13 communication channels adopted in 7 different ecosystems into two different knowledge forms, that are, externalization and combination. To understand

how software projects share knowledge within the ecosystems, an analysis of various communication channels using topological approach was then performed. The results of the analysis show that communication channels change over time. Furthermore, GitHub projects tend to adopt multiple channels for either capturing new knowledge (externalization) or updating the existing knowledge (combination).

- **An investigation on the human aspects of forums with such connective information**. This work is conducted to investigate the human aspects in the Eclipse community forums. The results of the preliminary study show that Eclipse manages the forums in a various way. Forums are also essential platform for linking various resources in Eclipse ecosystem. This is demonstrated by a multitude types of links referenced by forum users in the discussion threads. In addition, Eclipse forums have been a source of expert knowledge for other development systems, since active contributors in the ecosystem tend to be also active in forums. Finally, the main findings of the analysis indicate that human factors play an important role in the Eclipse community forums. The membership-based participation and interactions between individuals within the Eclipse forums have an effect on shaping the knowledge sharing activities. Different statuses of users tend to post different types of messages and behavioural expression while knowledge sharing.

This thesis shows that problem of communication, such as uncommunicated update, in software development processes might leverage the results of the relevant works. Thus, various types of knowledge, either tacit or explicit within an organization is necessary to document, communicate and share between individuals. To achieve the success of knowledge sharing, software projects are required to start adopting multiple communication channels for both capturing new knowledge and updating the existing knowledge. Project-specific communication channels, such as Eclipse forums, can also be beneficial as a center of knowledge sharing platforms since it links to numerous resources in the ecosystem and as a source of expert knowledge for other development systems. In addition, contributing to forums has an impact on shaping the community.

In sum, the results of this dissertation highlight that since maintaining knowledge sharing in software development environment is challenging, therefore, a better understanding of where and how knowledge is captured and shared in software development is necessary. This is significant to both researcher and practitioners. To the researcher, this dissertation confirms that the notion of knowledge sources should be treated as distinct for knowledge sharing success, while the establishment of the technological and human aspects is important to enhance the knowledge sharing outcome for practitioners.

References

[1] Rabe Abdalkareem, Olivier Nourry, Sultan Wehaibi, Suhaib Mujahid, and Emad Shihab. Why do developers use trivial packages? an empirical case study on npm. In *Proceedings of the 2017 11th on Foundations of Software Engineering*, pages 385–395, 2017.

[2] Miltiadis Allamanis and Charles Sutton. Why, when, and what: Analyzing stack overflow questions by topic, type, and code. In *Proceedings of the 2013 10th Working Conference on Mining Software Repositories*, pages 53–56, 2013.

[3] Maurício Aniche, Christoph Treude, Igor Steinmacher, Igor Wiese, Gustavo Pinto, Margaret-Anne Storey, and Marco Aurélio Gerosa. How modern news aggregators help development communities shape and share knowledge. In *Proceedings of the 40th International Conference on Software Engineering*, pages 499–510, 2018.

[4] John Anvik, Lyndon Hiew, and Gail C. Murphy. Coping with an open bug repository. In *Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology eXchange*, pages 35–39, 2005.

[5] Rayhab Anwar, Mobashar Rehman, Khor Siak Wang, Manzoor Ahmed Hashmani, and Amjad Shamim. Investigation of knowledge sharing behavior in global software development organizations using social cognitive theory. *IEEE Access*, 7:71286–71298, 2019.

[6] Linda Argote and Ella Miron-Spektor. Organizational learning: From experience to knowledge. *Organization science*, 22(5):1123–1137, 2011.

[7] Earl T. Barr, Yuriy Brun, Premkumar Devanbu, Mark Harman, and Federica Sarro. The plastic surgery hypothesis. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 306–317, 2014.

[8] Irma Becerra-Fernandez and Rajiv Sabherwal. *Knowledge management: Systems and processes*. Routledge, 2014.

[9] Dane Bertram, Amy Voida, Saul Greenberg, and Robert Walker. Communication, collaboration, and bugs: The social nature of issue tracking in small, collocated teams. In *Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work*, pages 291–300, 2010.

[10] Stefanie Beyer, Christian Macho, Massimiliano Di Penta, and Martin Pinzger. Analyzing the relationships between android api classes and their references on stack overflow. Technical report, Technical Report. University of Klagenfurt, University of Sannio, 2017.

[11] Stefanie Beyer, Christian Macho, Massimiliano Di Penta, and Martin Pinzger. What kind of questions do developers ask on stack overflow? a comparison of automated approaches to classify posts into question categories. *Empirical Software Engineering*, 25:2258–2301, 2020.

[12] Stefanie Beyer and Martin Pinzger. A manual categorization of android app development issues on stack overflow. In *Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution*, pages 531–535, 2014.

[13] Hudson Borges, Andre Hora, and Marco Tulio Valente. Understanding the factors that impact the popularity of github repositories. In *Proceedings of the 2016 IEEE International Conference on Software Maintenance and Evolution*, pages 334–344, 2016.

[14] Gilberto Borrego, Alberto L. Morán, Ramón Palacio, and Oscar M. Rodríguez. Understanding architectural knowledge sharing in agsd teams: An empirical study. In *Proceedings of the 2016 IEEE 11th International Conference on Global Software Engineering*, pages 109–118, 2016.

[15] David Budgen, Mark Turner, Pearl Brereton, and Barbara Kitchenham. Using mapping studies in software engineering. In *PPIG*, volume 8, pages 195–204, 2008.

[16] Andrew Lincoln Burrow. Negotiating access within wiki: A system to construct and maintain a taxonomy of access rules. In *Proceedings of the 15th ACM Conference on Hypertext and Hypermedia*, pages 77–86, 2004.

[17] Elizabeth F. Cabrera and Angel Cabrera. Fostering knowledge sharing through people management practices. *The International Journal of Human Resource Management*, 16(5):720–735, 2005.

[18] Fabio Calefato, Filippo Lanubile, and Nicole Novielli. How to ask for technical help? evidence-based guidelines for writing questions on stack overflow. *Information and Software Technology*, 94:186–207, 2018.

[19] Suranjan Chakraborty, Saonee Sarker, and Suprateek Sarker. An exploration into the process of requirements elicitation: A grounded approach. *Journal of the Association for Information Systems*, 11(4), 2010.

[20] Thomas Chau and Frank Maurer. Knowledge sharing in agile software teams. In *Logic versus Approximation: Essays Dedicated to Michael M. Richter on the Occasion of his 65th Birthday*, pages 173–183. Springer, Berlin, Heidelberg, 2004.

[21] Ai Ling Chua and Shan L. Pan. Knowledge transfer and organizational learning in is offshore sourcing. *Omega*, 36(2):267–281, 2008. Special Issue on Knowledge Management and Organizational Learning.

[22] Jailton Coelho and Marco Tulio Valente. Why modern open source projects fail. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 186–196, 2017.

[23] Kieran Conboy, Sharon Coyle, Xiaofeng Wang, and Minna Pikkarainen. People over process: Key challenges in agile development. *IEEE Software*, 28(4):48–57, 2011.

[24] Daniel Alencar da Costa, Shane McIntosh, Weiyi Shang, Uirá Kulesza, Roberta Coelho, and Ahmed E. Hassan. A framework for evaluating the results of the szz approach for identifying bug-introducing changes. *IEEE Transactions on Software Engineering*, 43(7):641–657, 2017.

[25] Thomas H. Davenport and Lawrence Prusak. Working knowledge: How organizations manage what they know. *Ubiquity*, 2000(August):6, 2000.

[26] Monika Dávideková and Jozef Hvorecký. Collaboration tools for virtual teams in terms of the seci model. In *Proceedings of the International Conference on Interactive Collaborative Learning*, pages 97–111, 2017.

[27] Alexandre Decan, Tom Mens, and Eleni Constantinou. On the impact of security vulnerabilities in the npm package dependency network. In *Proceedings of the 15th International Workshop on Mining Software Repositories*, pages 181–191, 2018.

[28] Torgeir Dingsøyr and Emil Røyrvik. An empirical study of an informal knowledge repository in a medium-sized software consulting company. In *Proceedings of the 25th International Conference on Software Engineering*, pages 84–92, 2003.

[29] Yvonne Dittrich. What does it mean to use a method? towards a practice theory for software engineering. *Information and Software Technology*, 70:220 – 231, 2016.

[30] Georg Dotzler and Michael Philippsen. Move-optimized source code tree differencing. In *Proceedings of 2016 31st IEEE/ACM International Conference on Automated Software Engineering*, pages 660–671, 2016.

[31] Ekwa Duala-Ekoko and Martin P. Robillard. Tracking code clones in evolving software. In *Proceedings of the 29th International Conference on Software Engineering*, pages 158–167, 2007.

[32] Jean-Rémy Falleri, Floréal Morandat, Xavier Blanc, Matias Martinez, and Martin Monperrus. Fine-grained and accurate source code differencing. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, pages 313–324, 2014.

[33] Beat Fluri, Michael Wuersch, Martin PInzger, and Harald Gall. Change distilling: Tree differencing for fine-grained source code change extraction. *IEEE Transactions on Software Engineering*, 33(11):725–743, 2007.

[34] Nicolai J. Foss and Torben Pedersen. Transferring knowledge in mncs: The role of sources of subsidiary knowledge and organizational context. *Journal of International Management*, 8(1):49–67, 2002.

[35] Shahla Ghobadi. Challenges of cross-functional software development teams: A conceptual study. *Journal of Information Technology Management*, 22(3):26–35, 2011.

[36] Jesus M. Gonzalez-Barahona, Paul Sherwood, Gregorio Robles, and Daniel Izquierdo. Technical lag in software compilations: Measuring how outdated a software deployment is. In *Proceedings of the International Conference on Open Source Systems*, pages 182–192, 2017.

[37] Georgios Gousios, Eirini Kalliamvakou, and Diomidis Spinellis. Measuring developer contribution from software repository data. In *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, pages 129–132, 2008.

[38] Robert M. Grant. Toward a knowledge-based theory of the firm. *Strategic Management Journal*, 17(S2):109–122, 1996.

[39] Emitza Guzman, Rana Alkadhi, and Norbert Seyff. A needle in a haystack: What do twitter users say about software? In *Proceedings of the 2016 IEEE 24th International Requirements Engineering Conference (RE)*, pages 96–105, 2016.

[40] Anja Guzzi, Alberto Bacchelli, Michele Lanza, Martin Pinzger, and Arie van Deursen. Communication in open source software development mailing lists. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 277–286, 2013.

[41] Masatomo Hashimoto and Akira Mori. Diff/ts: A tool for fine-grained structural change analysis. In *Proceedings of the 2008 15th Working Conference on Reverse Engineering*, pages 279–288, 2008.

[42] Hideaki Hata, Osamu Mizuno, and Tohru Kikuno. Bug prediction based on fine-grained module histories. In *Proceedings of the 34th International Conference on Software Engineering*, pages 200–210, 2012.

[43] Hideaki Hata, Taiki Todo, Saya Onoue, and Kenichi Matsumoto. Characteristics of sustainable oss projects: A theoretical and empirical study. In

Proceedings of the 2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering, pages 15–21, 2015.

[44] Hideaki Hata, Christoph Treude, Raula Gaikovina Kula, and Takashi Ishio. 9.6 million links in source code comments: Purpose, evolution, and decay. In *Proceedings of the 41st International Conference on Software Engineering*, ICSE '19, pages 1211–1221, 2019.

[45] Claudia Hauff and Georgios Gousios. Matching github developer profiles to job advertisements. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, pages 362–366, 2015.

[46] Joseph Hejderup, Arie van Deursen, and Georgios Gousios. Software ecosystem call graph for dependency management. In *Proceedings of the 2018 IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results*, pages 101–104, 2018.

[47] Yoshiki Higo, Akio Ohtani, and Shinji Kusumoto. Generating simpler ast edit scripts by considering copy-and-paste. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, pages 532–542, 2017.

[48] Kaifeng Huang, Bihuan Chen, Xin Peng, Daihong Zhou, Ying Wang, Yang Liu, and Wenyun Zhao. Cldiff: Generating concise linked code differences. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 679–690, 2018.

[49] James W. Hunt and M. Douglas MacIlroy. *An algorithm for differential file comparison.* Computing science technical report. Bell Laboratories Murray Hill, 1976.

[50] Abasiofiok M. Ibekwe, Jincai Ma, David E. Crowley, Ching-Hong Yang, Alexis M. Johnson, Tanya C. Petrossian, Pek Y. Lum, Eelco Franz, and Antje Flieger. Topological data analysis of Escherichia coli O157:H7 and non-O157 survival in soils. *Frontiers in Cellular and Infection Microbiology*, 4:1–10, 2014.

[51] Nafiseh Kahani, Mojtaba Bagherzadeh, Juergen Dingel, and James R. Cordy. The problems with eclipse modeling tools: A topic analysis of eclipse forums. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, pages 227–237, 2016.

[52] Yasutaka Kamei and Emad Shihab. Defect prediction: Accomplishments and future challenges. In *Proceedings of the 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering*, volume 5, pages 33–45, 2016.

[53] Simone Kauffeld and Nale Lehmann-Willenbrock. Meetings matter: Effects of team meetings on team and organizational success. *Small Group Research*, 43(2):130–158, 2012.

[54] Karlheinz Kautz and Annemette Kjærgaard. Knowledge sharing in software development. *P.A. Nielsen and K., Kautz (Editions), Software Processes & Knowledge. Beyond Conventional Software Process Improvement*, pages 43–68, 2008.

[55] R. Kavitha. Collection development in digital libraries: trends and problems. *Indian Journal of Science and Technology*, 2(12), 2009.

[56] Joann Keyton. Communication in organizations. *Annual Review of Organizational Psychology and Organizational Behavior*, 4(1):501–526, 2017.

[57] Riivo Kikas, Georgios Gousios, Marlon Dumas, and Dietmar Pfahl. Structure and evolution of package dependency networks. In *Proceedings of the 14th International Conference on Mining Software Repositories*, pages 102–112, 2017.

[58] Miryung Kim, David Notkin, Dan Grossman, and Gary Wilson. Identifying and summarizing systematic code changes via rule inference. *IEEE Transactions on Software Engineering*, 39(1):45–62, 2013.

[59] Miryung Kim, Vibha Sazawal, David Notkin, and Gail Murphy. An empirical study of code clone genealogies. In *Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT*

International Symposium on Foundations of Software Engineering, pages 187–196, 2005.

[60] Julia Kotlarsky, Man Oshri, Jos van Hillegersberg, and Kuldeep Kumar. Globally distributed component-based software development: An exploratory study of knowledge management and work division. *Journal of Information Technology*, 22(2):161–173, 2007.

[61] Marco Kuhrmann, Fernandez, Daniel Mendez, and Maya Daneva. On the pragmatic design of literature studies in software engineering: An experience-based guideline. *Empirical Software Engineering*, 22(6):2852–2891, 2017.

[62] Raula Gaikovina Kula, Daniel M. German, Ali Ouni, Takashi Ishio, and Katsuro Inoue. Do developers update their library dependencies? *Empirical Software Engineering*, 23(1):384–417, 2018.

[63] Filippo Lanubile. Social software as key enabler of collaborative development environments, 2013. [Online]. Available: `https://www.slideshare.net/lanubile/lanubilesse2013-25350287`.

[64] Nuttapon Lertwittayatrai, Raula Gaikovina Kula, Saya Onoue, Hideaki Hata, Arnon Rungsawang, Pattara Leelaprute, and Kenichi Matsumoto. Extracting insights from the topology of the javascript package ecosystem. In *Proceedings of the 2017 24th Asia-Pacific Software Engineering Conference*, pages 298–307, 2017.

[65] Paul Luo Li, Andrew J. Ko, and Jiamin Zhu. What makes a great software engineer? In *Proceedings of the 37th International Conference on Software Engineering*, volume 1, pages 700–710, 2015.

[66] Yngve Lindsjørn, Dag I.K. Sjøberg, Torgeir Dingsøyr, Gunnar R. Bergersen, and Tore Dybå. Teamwork quality and project success in software development: A survey of agile development teams. *Journal of Systems and Software*, 122:274–286, 2016.

[67] Xiang Lingzi and Lin Zhi. An overview of source code audit. In *Proceedings of the 2015 International Conference on Industrial Informatics - Computing Technology, Intelligent Technology, Industrial Information Integration*, pages 26–29, 2015.

[68] Pek Y. Lum, Gurjeet Singh, Alan Lehman, Tigran Ishkanov, Mikael Vejdemo-Johansson, Muthu Alagappan, John Carlsson, and Gunnar Carlsson. Extracting insights from the shape of complex data using topology. *Scientific Reports*, 3(1236), 2013.

[69] Lech Madeyski and Marian Jureczko. Which process metrics can significantly improve defect prediction models? an empirical study. *Software Quality Journal*, 23(3):393–422, 2015.

[70] Xiaozhu Meng, Barton P. Miller, William R. Williams, and Andrew R. Bernat. Mining software repositories for accurate authorship. In *Proceedings of the 2013 IEEE International Conference on Software Maintenance*, pages 250–259, 2013.

[71] Tom Mens. An ecosystemic and socio-technical view on software maintenance and evolution. In *Proceedings of the 2016 IEEE International Conference on Software Maintenance and Evolution (Invited Paper)*, 2016.

[72] Mariam El Mezouar, Feng Zhang, and Ying Zou. Are tweets useful in the bug fixing process? an empirical study on firefox and chrome. *Empirical Software Engineering*, 23(3):1704–1742, 2018.

[73] Justin Middleton, Emerson Murphy-Hill, Demetrius Green, Adam Meade, Roger Mayer, David White, and Steve McDonald. Which contributions predict whether developers are accepted into github teams. In *Proceedings of the 15th International Conference on Mining Software Repositories*, pages 403–413, 2018.

[74] Samim Mirhosseini and Chris Parnin. Can automated pull requests encourage software developers to upgrade out-of-date dependencies? In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, pages 84–94, 2017.

[75] Eugene W. Myers. An o(nd) difference algorithm and its variations. *Algorithmica*, 1:251–266, 1986.

[76] Nachiappan Nagappan and Thomas Ball. Use of relative code churn measures to predict system defect density. In *Proceedings of the 27th International Conference on Software Engineering*, pages 284–292, 2005.

[77] Keitaro Nakasai, Hideaki Hata, and Kenichi Matsumoto. Are donation badges appealing? a case study of developer responses to eclipse bug reports. *IEEE Software*, 2018.

[78] Ikujiro Nonaka. *Management of knowledge creation: A theory of organizational knowledge creation*. Tokyo: Nihon Keizai Shinbun-sha, 1990.

[79] Ikujiro Nonaka and Hirotaka Takeuchi. *The Knowledge-creating Company: How Japanese Companies Create the Dynamics of Innovation*. Everyman's library. Oxford University Press, 1995.

[80] Saya Onoue, Hideaki Hata, Raula Gaikovina Kula, and Kenichi Matsumoto. Human capital in software engineering: A systematic mapping of reconceptualized human aspect studies. *arXiv preprint arXiv:1805.03844*, 2018.

[81] Marc Palyart, Gail C. Murphy, and Vaden Masrani. A study of social interactions in open source component use. *IEEE Transactions on Software Engineering*, 44(12):1132–1145, 2018.

[82] Sebastiano Panichella. Supporting newcomers in software development projects. In *Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution*, pages 586–589, 2015.

[83] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic mapping studies in software engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, pages 68–77, 2008.

[84] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18, 2015.

[85] Joao Pita Costa and Tihana Galinac Grbac. The topological data analysis of time series failure data in software evolution. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*, pages 25–30, 2017.

[86] Michael Polanyi. Tacit knowing: Its bearing on some problems of philosophy. *Review of Modern Physics*, 34:601–616, 1962.

[87] Michael Polanyi. The logic of tacit inference. *Philosophy*, 41(155):1–18, 1966.

[88] Foyzur Rahman and Premkumar Devanbu. Ownership, experience and defects: A fine-grained study of authorship. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 491–500, 2011.

[89] Thomas Rausch, Waldemar Hummer, Philipp Leitner, and Stefan Schulte. An empirical analysis of build failures in the continuous integration workflows of java-based open-source software. In *Proceedings of the 14th International Conference on Mining Software Repositories*, pages 345–355, 2017.

[90] Baishakhi Ray, Meiyappan Nagappan, Christian Bird, Nachiappan Nagappan, and Thomas Zimmermann. The uniqueness of changes: Characteristics and applications. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, pages 34–44, 2015.

[91] Gema Rodriguez-Perez, Gregorio Robles, and Jesus M. Gonzalez-Barahona. Reproducibility and credibility in empirical software engineering: A case study based on a systematic literature review of the use of the szz algorithm. *Information and Software Technology*, 99:164–176, 2018.

[92] David Rooney, Greg Hearn, and Abraham Ninan. *Handbook on the knowledge economy*. Edward Elgar Publishing, 2005.

[93] Christoffer Rosen and Emad Shihab. What are mobile developers asking about? a large scale study using stack overflow. *Empirical Software Engineering*, 21:1192—1223, 2016.

[94] Shinobu Saito, Yukako Iimura, Aaron K. Massey, and Annie I. Antón. Discovering undocumented knowledge through visualization of agile software development activities. *Requirements Engineering*, 23(3):381–399, 2018.

[95] Saonee Sarker, Suprateek Sarker, Darren B. Nicholson, and Kshiti D. Joshi. Knowledge transfer in virtual systems development teams: an exploratory study of four key enablers. *IEEE Transactions on Professional Communication*, 48(2):201–218, 2005.

[96] Walt Scacchi. Understanding the requirements for developing open source software systems. *IEE Proceedings - Software*, 149(1):24–39, 2002.

[97] Kjeld Schmidt. The trouble with 'tacit knowledge'. *Computer Supported Cooperative Work*, 21(2-3):163–225, 2012.

[98] Ken Schwaber and Mike Beedle. *Agile Software Development with Scrum*, volume 1. Prentice Hall Upper Saddle River, 2001.

[99] Yonghee Shin, Andrew Meneely, Laurie Williams, and Jason A. Osborne. Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. *IEEE Transactions on Software Engineering*, 37(6):772–787, 2011.

[100] Gurjeet Singh, Facundo Mémoli, and Gunnar Carlsson. Topological methods for the analysis of high dimensional data sets and 3d object recognition. *Eurographics Symposium on Point-Based Graphics*, 91, 2007.

[101] Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. When do changes induce fixes? *SIGSOFT Software Engineering Notes*, 30(4):1–5, May 2005.

[102] John-Christopher Spender and Robert M. Grant. Knowledge and the firm: Overview. *Strategic management journal*, 17(S2):5–9, 1996.

[103] Megan Squire. Should we move to stack overflow?: Measuring the utility of social media for developer support. In *Proceedings of the 37th International Conference on Software Engineering*, volume 2, pages 219–228, 2015.

[104] Igor Steinmacher, Tayana Uchoa Conte, Christoph Treude, and Marco Aurélio Gerosa. Overcoming open source project entry barriers with a portal for newcomers. In *Proceedings of the 38th International Conference on Software Engineering*, pages 273–284, 2016.

[105] Igor Steinmacher, Marco Aurelio Graciotto Silva, Marco Aurelio Gerosa, and David F. Redmiles. A systematic literature review on the barriers faced by newcomers to open source software projects. *Information and Software Technology*, 59:67 – 85, 2015.

[106] Margaret-Anne Storey, Alexey Zagalsky, Fernando Figueira Filho, Leif Singer, and Daniel M. German. How social and communication channels shape and challenge a participatory culture in software development. *IEEE Transactions on Software Engineering*, 43(2):185–204, 2017.

[107] Damian A. Tamburri, Fabio Palomba, Alexander Serebrenik, and Andy Zaidman. Discovering community patterns in open-source: a systematic approach and its evaluation. *Empirical Software Engineering*, 24:1369–1417, 2019.

[108] Jirateep Tantisuwankul, Yusuf Sulistyo Nugroho, Raula Gaikovina Kula, Hideaki Hata, Arnon Rungsawang, Pattara Leelaprute, and Kenichi Matsumoto. A topological analysis of communication channels for knowledge sharing in contemporary github projects. *Journal of Systems and Software*, 158:110416, 2019.

[109] Páivi M. Tikka, Markku T. Kuitunen, and Salla M. Tynys. Effects of educational background on students' attitudes, activity levels, and knowledge concerning the environment. *The Journal of Environmental Education*, 31(3):12–19, 2000.

[110] Christoph Treude, Ohad Barzilay, and Margaret-Anne Storey. How do programmers ask and answer questions on the web? (nier track). In *Proceedings of the 33rd International Conference on Software Engineering*, pages 804—807, 2011.

[111] Christoph Treude and Margaret-Anne Storey. Effective communication of software development knowledge through community portals. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, pages 91–101, 2011.

[112] Laurens Van Der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[113] Christopher Vendome, Mario Linares-Vasquez, Gabriele Bavota, Massimiliano Di Penta, Daniel German, and Denys Poshyvanyk. License usage and changes: A large-scale study of java projects on github. In *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension*, pages 218–228, 2015.

[114] Anthony J. Viera and Joanne M. Garrett. Understanding interobserver agreement: The kappa statistic. *Family Medicine*, 37(5):360–363, 5 2005.

[115] Shaowei Wang, Tse-Hsun Chen, and Ahmed E. Hassan. Understanding the factors for fast answers in technical q&a websites. *Empirical Software Engineering*, 23(3):1552–1593, 2018.

[116] Yang Wang, Daniel Graziotin, Stefan Kriso, and Stefan Wagner. Communication channels in safety analysis: An industrial exploratory case study. *Journal of Systems and Software*, 153:135–151, 2019.

[117] Laurie Williams and Robert Kessler. *Pair Programming Illuminated*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[118] Claes Wohlin, Per Runeson, Paulo Anselmo da Mota Silveira Neto, Emelie Engstrom, Ivan do Carmo Machado, and Eduardo Santana de Almeida. On the reliability of mapping studies in software engineering. *Journal of Systems and Software*, 86(10):2594–2610, 2013.

[119] Yuhao Wu, Yuki Manabe, Tetsuya Kanda, Daniel M. German, and Katsuro Inoue. Analysis of license inconsistency in large collections of open source projects. *Empirical Software Engineering*, 22(3):1194–1222, 2017.

[120] Deheng Ye, Zhenchang Xing, and Nachiket Kapre. The structure and dynamics of knowledge network in domain-specific q&a sites: A case study of stack overflow. *Empirical Software Engineering*, 22(1):375–406, 2017.

[121] Yunwen Ye and K. Kishida. Toward an understanding of the motivation of open source software developers. In *Proceedings of the 25th International Conference on Software Engineering*, pages 419–429, 2003.

[122] Alexey Zagalsky, Daniel M. German, Margaret-Anne Storey, Carlos Gómez Teshima, and Germán Poo-Caamaño. How the r community creates and curates knowledge: An extended study of stack overflow and mailing lists. *Empirical Software Engineering*, 23(2):953–986, 2018.

[123] Mansooreh Zahedi, Mojtaba Shahin, and Muhammad Ali Babar. A systematic review of knowledge sharing challenges and practices in global software development. *International Journal of Information Management*, 36(6):995–1019, 2016.

[124] Minghui Zhou and Audris Mockus. What make long term contributors: Willingness and opportunity in oss community. In *Proceedings of the 2012 34th International Conference on Software Engineering*, pages 518–528, 2012.

[125] Minghui Zhou and Audris Mockus. Who will stay in the floss community? modeling participant's initial behavior. *IEEE Transactions on Software Engineering*, 41(1):82–99, 2015.

[126] Jie Zou, Ling Xu, Mengning Yang, Xiaohong Zhang, and Dan Yang. Towards comprehending the non-functional requirements through developers eyes. *Information and Software Technology*, 84:19–32, 2017.