How Do Coupled File Changes Influence How Developers Seek Help During Maintenance Tasks?

Jasmin Ramadani
University of Stuttgart
jasmin.ramadani@informatik.uni-stuttgart.de
ORCID: 0000-0001-7180-9964

Stefan Wagner
University of Stuttgart
stefan.wagner@informatik.uni-stuttgart.de
ORCID: 0000-0002-5256-8429

***Abstract*—Software repositories contain a lot of information that can be transformed into suggestions other files they need to modify during maintenance tasks (so-called "coupled changes"). Existing studies however ignore developer feedback and their influence on the developer strategy for getting help during maintenance tasks. We used the Grounded Theory approach to investigate screen capture videos from an experiment to find which information sources developers use to find help and what is their relevance. We compared the frequency as well as the sequence patterns of used information sources both for the developers using coupled change suggestions and those not using them. We found a set of information sources where the developers seek for help and identified two categories of relevance. Also, we discovered that for the tasks using coupled change suggestions, the developers used mostly the internal IDE elements as an information source whereby the developers not using coupled change suggestions often used external sources like the documentation or the web. Coupled change suggestions influence the strategy how the developers seek for help by reducing the search for information on external locations which makes the process of solving maintenance tasks more compact.**

I. Introduction

Software development produces a rapidly increasing amount of data related to the changes in the source code. It contains valuable information which can be used to support developers in solving maintenance tasks. Software maintenance is performed by maintenance programmers whereby members leave and others join the team [19]. The new members could use a support to successfully perform their tasks. They need to understand how a specific software system has been implemented [31].

Maintenance tasks on unfamiliar parts of the system require additional knowledge which can lead to a situation where developers cannot solve all tasks or need more effort to understand them [38]. They usually seek for help to increase their knowledge. Help seeking constitutes an important skill in the strategies how to solve a problem [25].

Data mining represents a very popular technique for analyzing software repositories. The investigations of software repositories using data mining is known as *mining software repositories (MSR)* [20]. Using it we can extract change couplings which represent a set of files having the same commit time, author and modification description [13]. Frequently changed files can support developers in their maintenance tasks when they are new on the project, the project started a long time ago or if the developers do not have sufficient experience in software development.

A. Problem Statement

There are many studies dealing with coupled changes and data mining [4], [21], [45], [47]. However, they ignore the feedback of developers and the impact on the developer strategies for solving maintenance tasks.

B. Research Objective

The aim of our research is to explore how coupled change suggestions influence how developers seek for help during maintenance tasks. We investigate the developers' information sources using the Grounded Theory method [15] on videos captured on the developer's screen. We explore which information sources are used by the developers and what kind of information both the group not using coupled change suggestions and the group using this kind of help look for.

C. Contribution

We present an exploratory study based on the data from a controlled experiment where each of the 36 participants try to solve 4 different maintenance tasks. The experiment results show that the use of coupled file change suggestions significantly increases the correctness of the solutions. The original experiment analysis is available in [32]. In this study, we found that the participants' information sources choice and search strategy differs depending on the use of coupled change suggestions.

II. Background

A. Coupled Changes

The changes in software repositories performed by various developers on different occasion are usually stored in the version history. These changes in particular source code areas repeat with various frequency in different time periods. For the first time logical couplings were introduced as evolutionary couplings [1] or logical dependencies [12]. This kind of dependencies can be detected by analyzing the version history of the software. In a situation where the developer changes a file and also changes another file shortly after, we say that we have *Coupled File Changes*.

B. Data Mining

The term *mining software repositories (MSR)* describes investigations of software repositories using data mining [20]. Using the data from the software repository of the system, we extract coupled changes and build suggestions including related attributes from the versioning system, the issue tracking system and the documentation. For that purpose, we use an algorithm for frequent item set discovery called FP-Tree-Growth algorithm. It is fast algorithm which operates without using candidate itemset generation [16]. This algorithm is considered to be faster and more memory efficient than the very often used Apriori algorithm. [17]. We also use a heuristic that groups the changes committed by a single developer meaning that the changes performed by the same developer are considered as related [21].

Sequential pattern mining is used to discover interesting subsequences in a set of sequences. One of the measures for their interestingness is the occurrence frequency of the ordered elements [11]. For this purpose we use the Prefix-Span algorithm, a fast and and memory efficient sequential pattern mining algorithms [28]. It uses the pattern-grown paradigm and outperforms fast algorithms like GSM or SPADE as well as the BIDE algorithm for greater support levels [42].

C. Task Relevant Information Sources

Working on a maintenance task, when developers are facing difficulty or need an additional help for a solution often seek for a relevant information on various locations [22]. We can have several information sources like the IDE, in our case, it is Eclipse with its functionalities like the *search function*, the *source code editor* or the *package explorer*. There are also other sources like the *software documentation*, *web search engines*, *source code examples* or *programming tutorials*. Developers can seek for source code to find files or code locations for further modifications [34] or look for a code description to better understand the part of the code they need to edit based on a similar situation or a problem.

D. Grounded Theory

The Grounded theory data analysis approach has been firstly described in [15]. To analyze the data and build the theory, we use the following activities: open, axial and selective coding [40]. Afterwards, we perform the theoretical coding and create the conceptual model.

III. Experimental Design

A. Study Goal

We perform our exploratory study using a mixed-method approach on top of a maintenance tasks experiment where the impact of coupled change suggestions on the time and the effort needed to solve maintenance tasks has been investigated [32]. We define the *goal* of the study using the GQM approach [3] and its MEDEA extension [6]. We analyze the influence of coupled file change suggestions on the location and the purpose the developers search for help. The *objective* is to compare the information sources the developers access for the tasks using coupled change suggestions and the those without using them. The *purpose* is to evaluate how effective are coupled file change suggestions related to the location of the task relevant information, the relevance of the information they look for, the frequency as well as the patterns of information sources.

B. Research Questions

RQ1: Where do developers look for task relevant information? We want to identify the information sources used for a particular maintenance tasks solution. This will enlist the usual locations to seek for help during the tasks.

RQ2: What kind of relevance have the information sources for their tasks? The answer of this research question will expose how information sources contribute to the tasks solution.

RQ3: How do coupled file changes influence the search for task relevant information during maintenance tasks? We investigate their influence related to the following subquestions:

RQ3.1: How frequently do developers use particular information sources? We identify the most popular sources for seeking help used by the developers in this study to identify common information sources.

RQ3.2: What information sources patterns developers use to find task relevant information? We explore if there are patterns of information sources to identify the difference in the strategy how developers seek for help.

C. Overview

1) Experiment Design: We use a counterbalanced experiment design similar to the one presented by Ricca et al. [36]. It ensures that all subjects work on tasks without and with coupled change suggestions. We split the subjects randomly in two lab sessions having a maximum of two hours to solve the tasks.

We distinguish two groups of maintenance tasks: the first one includes tasks executed in Eclipse IDE without using suggestions and the second one includes additional coupled files suggestions and the corresponding attributes from the repositories. In each session, the subjects work on two tasks without coupled suggestions using only the task description and on two tasks using suggestions consisted of coupled file changes and the related attributes we deliver together with the task description. The participants in the second lab swap the order of the tasks used during the first lab.

2) Objects: The study object is A-STPA, an open source Eclipse based Java tool for hazard analysis built at the

TABLE I
EXPERIMENT DESIGN

Lab	Tasks	
Lab 1	Tasks 1-2 (without suggestions)	Tasks 3-4 (with suggestions)
Lab 2	Tasks 1-2 (with suggestions)	Tasks 3-4 (without suggestions)

University of Stuttgart[1] in 2013. It has been chosen for the analysis because of the availability of the source code, the git repository, the complete list of the issues and the project documentation. The source code contains 16012 lines of code and 178 classes organized in 37 packages. The Git repository of the project contains 1106 commits from which we have extracted 205 coupled changes.

3) Subjects: The participants are 36 undergraduate students from the Introduction to Software Engineering course in their 1st and 2nd study year at the University of Stuttgart. The students had the chance to apply for the experiment using the first come first served principle. They have not been related in any way with the software system investigated in the experiment. The students have already passed the Java programming and software development course and have basic Java and Eclipse knowledge.

4) Environment, Materials: The participants worked on a Windows PC with an Eclipse IDE. Their actions were recorded using a full screen video capturing tool. We have provided the source code, the technical documentation for the software system as pdf document including the data model and package descriptions, the instructions for the experiment and the maintenance tasks free-text description.

5) Tasks: The maintenance tasks represent program fixes needed to be performed by the participants according to the maintenance requests [2]. All four maintenance tasks are perfective and enhance the software usability without influencing the system structure. The tasks are related to simple changes of the user interface of the system. The complete set of tasks and coupled changes is available on-line[2].

6) Coupled Files: We have provided a set of files which changed together frequently to the group which uses coupled file change suggestions. These coupled files do not represent the solutions for a particular task in the experiment and usually contain a subset of the solution file set. We have joined to the coupled changes a set of attributes from the versioning system, the issue tracking system and the project documentation to build the suggestions. Using the related information about the context of the change, the developers can decide if the coupled files are suitable for their tasks.

7) Maintenance Activities: The maintenance tasks solving process includes the following activities: task understanding where the participants read the task description and the instructions, change specification where they locate the source code to be changed, change design where they perform the modification and change testing where the successfulness of the changes is tested.

8) Data Collection: The first data source we use is the log data from the Git version control system. Git preserves the changes in a single change set or a so-called *atomic commit*. Here, the data is organized in a transaction form where every transaction represents the files which changed together in a single commit. The issue tracking system data source is used to extract the issue related attributes. The software documentation represents a rich information source related to the the data model or the project structure.

We collected the data how developers search for help using full screen video capturing of their actions during the experiments including the mouse movement and the keyboard input they performed. We have recorded the screen including their actions when using the IDE they work with as well as the applications and the documents the developers used or opened on the screen.

D. Data Analysis

We analyze the captured videos to follow the information sources the developers used during their search for help.

1) Information Sources: Before we use the Grounded Theory method, we define the focus involving the locations the developers use to search for help to avoid confusion during the coding. We distinguish different actions related to the locations where the information sources have been found. We transcribe the captured videos to isolate the locations as concepts. Afterwards, we start with the coding process to identify the information sources.

2) Relevance of the Information: The relevance of the information is important to determine what is the purpose of the help search. We aim to find out how the task-relevant information contributes to the solution. We differentiate between resolving the source code location and the source code description.

3) Frequency of Using Information Sources: We explore how popular various information sources are. We analyze the frequency of use of a particular source of help to see what the developers find most useful as a task relevant information source.

[1]https://sourceforge.net/projects/astpa/
[2]https://peerj.com/preprints/2492/

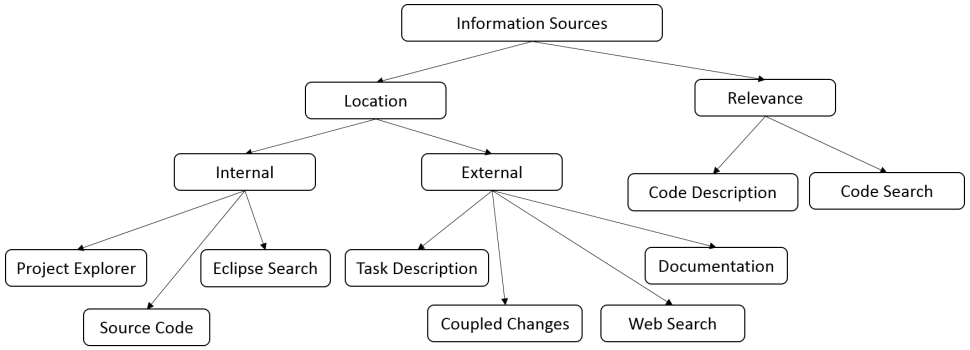

Fig. 1. Information Sources

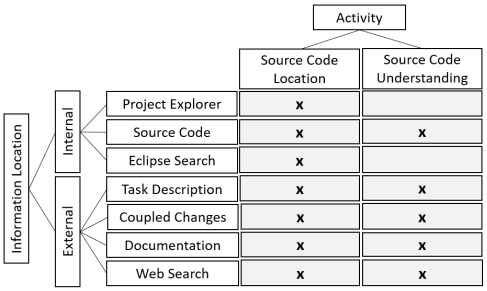

Fig. 2. Information Sources Relevance

TABLE II
INFORMATION SOURCES FREQUENCY

Information Source	Source ID	No Suggestions		With Suggestions	
		Freq.	%	Freq.	%
Task description	1	65	100	67	100
Project explorer	2	65	100	67	100
Documentation	3	21	32.31	7	10.44
Eclipse search	4	26	40	28	41.79
Web search	5	11	16.92	1	1.49
Source code	6	65	100	67	100

4) Patterns of Information Sources: Using sequential pattern mining, we explore common patterns in the information sources. We take all unique information sources for a particular task as items and order them in subsets whereby the items in a subset are not repeating. All subsets for a particular task build a transaction. Analyzing the transactions, we explore the most frequent patterns of information sources.

IV. RESULTS AND DISCUSSION[3]

A. Information Sources

1) Identified Sources: We identified a number of task relevant information sources using open coding. The main concepts were determined using a set of open codes. We continued with the axial coding to relate the concepts and their categories as well as the relations among them. Further, using the theoretical coding, we established the relationships between the categories and the subcategories and delivered the information sources as concepts for the theory as presented in Figure 1.

We identified a number of information sources categorized in two categories: *internal locations* including the *project explorer*, the *source code editor* and the *search window* in Eclipse and *external locations* like the *task description*, the *documentation* and the *web search*.

[3]The study results are available at: http://dx.doi.org/10.5281/zenodo.291865.

2) Relevance of Information Sources: Using the Grounded Theory method, we have defined two general categories of information relevance:

- *Source code location*: Here the developers try to locate the part of the source code they have to edit like packages, classes or methods.
- *Source code description*: The information the developer try to find to understand particular functionality or code segment.

We use a relevance matrix as presented in Figure 2 to demonstrate the relation of the information sources and their relevance for the task solution similar to the maintenance matrix presented in [8].

The internal task relevant information sources like the project explorer and the Eclipse search are used to find the source code location to be modified. However, the source code window can be also used to find code comments, classes, methods or other parts of the source code related to the task solution. The external sources of information like the documentation and the web search have been used in both cases: to find the source code location to be edited and to find additional information or examples for the task solution.

The task description and the coupled changes are used to understand the tasks. Coupled file change suggestions can be also used to identify the files as locations where the code modifications need to be performed. This exposes that the developers use the various information sources for their strategy, by excavating the search for additional information and the search for the source code to modify.

TABLE III
INFORMATION SOURCE PATTERNS

Support	Sequence patterns without suggestions (Source ID)	Sequence patterns using suggestions (Source ID)
0.1	1 2, 1 2 4, 1 2 4 6, 1 2 3, 1 2 6, 1 3, 1 2 3 4, 1 2 3 4 6,1 2 3 6, 1 2 5, 1 4, 1 4 6, 1 3, 1 3 4, 1 3 4 6, 1 6, 2 4, 2 3, 2 3 4, 2 3 4 6, 2 3 6, 2 5, 2 6, 2 4 6,2 5 6, 3 6, 3 4 6, 4 6, 4 5, 5 6,	1 2, 1 2 4, 1 2 4 6, 1 2 3, 1 2 6, 1 4, 1 4 6, 1 6, 2 6, 3 6, 4 6
0.2	1 2, 1 2 6, 1 2 4, 1 2 4 6, 1 2 4, 1 6, 1 4, 1 4 6, 1 5, 2 4, 2 6, 2 4 6, 2 5, 3 6, 4 6	1 2, 1 2 6, 1 4, 1 4 6, 1 6, 2 4, 2 6
0.4	1 2, 1 2 6, 1 6, 1 4, 2 6	1 2, 1 2 6, 1 6, 2 6

B. Influence of Coupled Change Suggestions

1) Frequency of Use of Information Sources: Table II presents how frequently each of the information sources has been used by the developers for the tasks both using coupled file change suggestions and without using these suggestions. We identify that the task description, the project explorer and the source code have been used for all the tasks in the experiment in both groups. The documentation has been used three times more for the tasks not using coupled change suggestions than for the tasks using the suggestions. The Eclipse search has been almost equally used in both groups. The web search has been used ten times more by the group without using coupled change suggestions than by the group not using them.

The results unveil that we have two different situations describing the influence of coupled change suggestions on information sources being used during the maintenance tasks. The first one sets out that coupled change suggestions do not influence the use of the internal information sources, both groups used almost equally the Eclipse functionalities. Very frequently they used the task description. This means that coupled change suggestions do not affect how the developers use the IDE and do not reduce the need for the task description. Moreover, the tendency of the group without coupled change suggestions is to use more external help like the documentation or search for help on the web using Google search, Stack Overflow, tutorials and videos.

This demonstrates that the developers not using the suggestions tried to find help outside their workspace. For the tasks using coupled change suggestions, developers used mostly the internal source code locations and made very little use of the documentation and almost no use of web search for their tasks. This indicates that using coupled change suggestions, the developers do not spread out their search for help on locations outside the IDE.

2) Patterns of Information Sources: We have extended our investigation of the used task relevant information sources by looking for patterns of information sources both for the tasks when using coupled change suggestions and those not using the suggestions. The results in Table III show the pattern sequences for both groups and various support values which represent the frequency of occurrence of these patterns. Here, the source code locations are represented by their IDs as described in Table II.

We can see that for the highest support value of 0.4 which includes the patterns repeating in 40% of the tasks, a typical pattern starts with the task description, continues with the project explorer and the Eclipse search and ends with the source code window. This seems to represent to be a common pattern in most of the cases for both for not using and using coupled change suggestions for the tasks.

For lower support values including 20% and 10% of the tasks respectively, we have specific patterns for the group not using coupled change suggestions where they start with the task description, look up in the project explorer and jump for the web search or use the documentation before they edit the code in the source code window.

In summary, these results present that the most frequent patterns are similar for both of the groups. However, in many cases their strategy for looking for help during their maintenance tasks varies. Some of the developers not using coupled change suggestions extended their pattern of help-seeking strategy with external sources before accessing the source code they need to edit. This shows that without the coupled change suggestions, the developers frequently need more information sources and have a different strategy in help-seeking than the group which uses coupled change suggestions which concentrates on the IDE elements to accomplish the source code modification.

V. THREATS TO VALIDITY

The main internal validity threat is that most of the analysis in this study is based on the subjective actions by the researchers. Transcribing the videos and the coding process can be error prone whereby the researchers can eventually miss actions by the developers. For that reason, we include a third party in the transcribing of the videos for a random set of videos.

The work in a controlled experiment using inexperienced 1st and 2nd year students instead of a real development process in company represents a major threat to the external validity. Perhaps, developers with more experience use other patterns of help-seeking. Especially developers familiar with a system might show a different behavior. We have designed simple maintenance tasks to increase the generalization possibility. We used an open source project for the study and a well known data mining technique which can be performed on other repositories.

VI. Related Work

Various studies use some kind of data mining to investigate software repositories [14], [18], [21], [37], [41], [45], [47]. However, they do not include the developers' feedback on their findings. Very often, the association rules technique has been used to identify frequent changes in the system [21], [45], [47]. Most of the studies employ the Apriori [21], [47] or the FP-growth algorithm [45]. In our study we also use the frequent itemsets analysis with the FP-growth algorithm to extract the coupled file changes.

Many studies investigated software repositories to find logically coupled changes, e.g. [5], [10], [13]. Some of them examined the couplings based on the file level [21], [33], [45], other identified logical dependencies between parts of files like classes, methods or modules [10], [20], [46], [47]. Our approach uses couplings on a file level.

The feedback on couplings has been investigated in various studies. Revelle et al. [34] inspected a set of coupling metrics on the structural and the semantic level where the developers answered that feature couplings on a higher level of abstraction than classes are useful. Bavota et al. [4] explored the developers' perceptions of software couplings. The authors examined how semantic and logical metrics align with the developers perception of couplings. Here, the semantic couplings have received the best rating of all types of couplings.

The interestingness of coupled changes is also studied by Ying et al. [45]. They define interestingness categories related to the source code changes based on expert knowledge and not using developers' feedback.

We investigated the feedback on the interestingness of coupled file changes from software repository [33]. We also performed an experiment on the usefulness of coupled file change suggestions [32]. Based on the outcomes of this study, we went beyond the interestingness of coupled changes and explored their influence on the maintenance tasks. We did not directly ask the developers to give their feedback on couplings; we recorded their actions on the screen.

The fault localization as technique for finding the location of defects given the program failures has been investigated in many occasions. Kocchar et al. [23] explored the developers expectations on fault localization. They reported the importance of data availability, granularity, reliability efficiency as well as an IDE integration of fault localization techniques. The study on the use of a spectra-based fault localization techniques by Xia et al. [44] shows that it saves significantly saves debugging time. Parnin et al. [27] empirically investigated the usability of a technique for spectrum based fault localization, with and without using this technique whereby the ranking and the search for defects has been identified as important.

We do not localize faults or bugs, we provide suggestions for potential file changes to solve an issue or a defined maintenance task.

The program comprehension including the understanding of the structure of a program and the functional dependencies has been investigated by Perez et al. [30]. They proposed a tool-set which improves the location of software components needed to be inspected. In [29], they introduced a spectrum based approach borrowed from fault localization technique. Based on test case executions, it identifies important components related to a given feature. Lawrence et al. [24] investigated how developers navigate during debugging using modern programming environment proposing to include the information foraging theory when searching and fixing a bug. The use of the same theory has been proposed by Fleming et al. [9] to support developer activities for information seeking needed to understand software engineering tasks.

We do not use testing or debugging, we provide the coupled files based on the changes in the version history.

Several studies explored how developers seek for help during their development or maintenance tasks. Singer et al. [38] examined software engineering work practices of developers. Li et al. [25] identified a number of help-seeking activities in software engineering both from static and dynamic perspective as well as relying on human factors. Ko et al. [22] defined various activities of seeking, relating and collecting developers perform to understand unfamiliar code. Wang et al. [43] investigated the actions, phases and patterns to locate the source code for a feature in maintenance tasks. Revelle et al. [35] introduced a combination of textual, dynamic and static analysis of feature location that needs to be implemented in the source code.

We use a similar approach by identifying the help-seeking activities, sources of information and patterns from a static perspective to describe how the developer seek for help for the maintenance tasks.

Various studies involve maintenance tasks related experiments. Nguyen et al. [26] describe the assessment and estimation of software maintenance tasks. De Lucia et al. [7] investigate the effort estimation for corrective software maintenance. We use a similar experiment design and investigation in our study.

A number of studies involve analysis of screen capturing videos. Developer actions in the investigation of help-seeking studies have been described in [22], [25]. Salinger et al. [39] identifies the problems in the coding of video data using Grounded Theory for qualitative analysis. They recommend practices to support video data analysis like perspectives and concepts. We follow the suggestions and focus the search on the information sources before we transcribe the videos.

VII. Conclusion

A. Summary

Coupled change suggestions demonstrate that they influence the strategy of inexperienced developers for searching task relevant information sources by reducing the variety of information sources. Our study shows that developers using coupled change suggestions mostly concentrate on the IDE features like the project explorer, the source code window and the Eclipse search. Without using this kind of help, developers often accessed also external information sources like the product documentation or searched for code examples

or descriptions to find the needed information for their task solution on the web.

B. Impact/Implications

The main effect of coupled change suggestions is that it makes the process of help-seeking compacter by reducing the need for additional sources of information related to the concept location and the source code comprehension. The suggestions influence the strategy of using various information sources and help the developers to reduce the effort for solving maintenance tasks.

C. Future Work

Our next research steps will be to expand the analysis on several projects using a larger number of maintenance tasks and participants to perform a deeper investigation of the influence of coupled change suggestions on the choice of information sources during maintenance tasks.

REFERENCES

[1] T. Ball, J. min Kim, A. A. Porter, and H. P. Siy. If your version control system could talk, 1997.

[2] V. R. Basili. Viewing maintenance as reuse-oriented software development. *IEEE Software*, 7(1):19–25, Jan. 1990.

[3] V. R. Basili, G. Caldiera, and H. D. Rombach. *The Goal Question Metric Approach*. Wiley, 1994.

[4] G. Bavota, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia. An empirical study on the developers perception of software coupling. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 692–701, 2013.

[5] J. Bieman, A. Andrews, and H. Yang. Understanding change-proneness in oo software through visualization. In *Proceedings of the 11th IEEE International Workshop on of the Program Comprehension*, pages 44–53, 2003.

[6] L. Briand, S. Morasca, and V. Basili. An operational process for goal-driven definition of measures. *IEEE Transactions on Software Engineering*, 28:1106–1125, 2002.

[7] A. De Lucia, E. Pompella, and S. Stefanucci. Effort estimation for corrective software maintenance. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering*, pages 409–416, 2002.

[8] F. Deissenboeck, S. Wagner, M. Pizka, S. Teuchert, and J. F. Girard. An activity-based quality model for maintainability. In *Proceedings of the 2007 IEEE International Conference on Software Maintenance*, pages 184–193, 2007.

[9] S. D. Fleming, C. Scaffidi, D. Piorkowski, M. Burnett, R. Bellamy, J. Lawrance, and I. Kwan. An information foraging theory perspective on tools for debugging, refactoring, and reuse tasks. *ACM Transactions on Software Engineering and Methodology*, 22:14:1–14:41, 2013.

[10] B. Fluri, H. Gall, and M. Pinzger. Fine-grained analysis of change couplings. In *Proceedings of the fifth IEEE International Workshop on Source Code Analysis and Manipulation*, pages 66–74, 2005.

[11] P. Fournier-Viger, J. C.-W. Lin, R. U. Kiran, and Y. S. Koh. A survey of sequential pattern mining. *Data Science and Pattern Recognition*, 1(1):54–77, 2017.

[12] H. Gall, K. Hajek, and M. Jazayeri. Detection of logical coupling based on product release history. In *Proceedings of the International Conference on Software Maintenance*, pages 190–, 1998.

[13] H. Gall, M. Jazayeri, and J. Krajewski. Cvs release history data for detecting logical couplings. In *Proceedings of the sixth International Workshop on Principles of Software Evolution*, pages 13–23, 2003.

[14] D. M. German. Mining cvs repositories, the softchange experience. In *Proceedings of the 1st International Workshop on Mining Software Repositories*, pages 17–21, 2004.

[15] B. Glaser and A. Strauss. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Observations (Chicago, Ill.). Aldine Publishing Company, 1967.

[16] C. Győrödi and R. Győrödi. A comparative study of association rules mining algorithms, 2004.

[17] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Journal of Data Mining and Knowledge Discovery*, 8(1):53–87, Jan. 2004.

[18] L. Hattori, G. dos Santos Jr, F. Cardoso, and M. Sampaio. Mining software repositories for software change impact analysis: A case study. In *Proceedings of the 23rd Brazilian Symposium on Databases*, pages 210–223, 2008.

[19] A. Hutton and R. Welland. An experiment measuring the effects of maintenance tasks on program knowledge. In *Proceedings of the 11th International Conference on Evaluation and Assessment in Software Engineering*, EASE'07, pages 43–52, 2007.

[20] H. Kagdi, M. L. Collard, and J. I. Maletic. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of Software Maintenance and Evolution*, 19(2):77–131, Mar. 2007.

[21] H. Kagdi, S. Yusuf, and J. I. Maletic. Mining sequences of changed-files from version histories. In *Proceedings of the 2006 International Workshop on Mining Software Repositories*, pages 47–53, 2006.

[22] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *IEEE Transactions on Software Engineering*, 32(12):971–987, Dec. 2006.

[23] P. S. Kochhar, X. Xia, D. Lo, and S. Li. Practitioners' expectations on automated fault localization. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, ISSTA 2016, pages 165–176, 2016.

[24] J. Lawrance, C. Bogart, M. Burnett, R. Bellamy, K. Rector, and S. D. Fleming. How programmers debug, revisited: An information foraging theory perspective. *IEEE Transactions on Knowledge and Data Engineering*, 39:197–215, 2010.

[25] H. Li, Z. Xing, X. Peng, and W. Zhao. What help do developers seek, when and how? In *2013 20th Working Conference on Reverse Engineering (WCRE)*, pages 142–151, 2013.

[26] V. Nguyen, B. Boehm, and P. Danphitsanuphan. A controlled experiment in assessing and estimating software maintenance tasks. *Information and Software Technology*, 53(6):682–691, June 2011.

[27] C. Parnin and A. Orso. Are automated debugging techniques actually helping programmers? In *Proceedings of the 2011 International Symposium on Software Testing and Analysis*, ISSTA '11, pages 199–209, 2011.

[28] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1424–1440, Nov. 2004.

[29] A. Perez and R. Abreu. A diagnosis-based approach to software comprehension. In *Proceedings of the 22Nd International Conference on Program Comprehension*, ICPC 2014, pages 37–47, 2014.

[30] A. Perez and R. Abreu. Framing program comprehension as fault localization. *Journal of Software Evolution and Process*, 28:840–862, 2016.

[31] M. Petrenko, V. Rajlich, and R. Vanciu. Partial domain comprehension in software evolution and maintenance. In *Proceedings of the 16th IEEE International Conference on Program Comprehension*, pages 13–22. IEEE, June 2008.

[32] J. Ramadani and S. Wagner. Are coupled file changes suggestions useful? In *PeerJ Preprint*, 2016.

[33] J. Ramadani and S. Wagner. Are suggestions of coupled file changes interesting? In *Proceedings of the 11th International Conference on Evaluation of Novel Software Approaches to Software Engineering*, pages 15–26, 2016.

[34] M. Revelle, M. Gethers, and D. Poshyvanyk. Using structural and textual information to capture feature coupling in object-oriented software. *Empirical Softw. Engg.*, 16(6):773–811, Dec. 2011.

[35] M. Revelle and D. Poshyvanyk. An exploratory study on assessing feature location techniques. In *Proceedings of the 17th International Conference on Program Comprehension*, pages 218–222, 2009.

[36] F. Ricca, M. Leotta, G. Reggio, A. Tiso, G. Guerrini, and M. Torchiano. Using unimod for maintenance tasks: an experimental assessment in the context of model driven development. In *Proceedings of the 4th International Workshop on Modeling in Software Engineering*, pages 77–83, 2012.

[37] J. Shirabad, T. Lethbridge, and S. Matwin. Mining the maintenance history of a legacy software system. In *Proceedings of the 2003 International Conference on Software Maintenance*, pages 95–104, 2003.

[38] J. Singer, T. Lethbridge, N. Vinson, and N. Anquetil. An examination of software engineering work practices. In *Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research*, pages 21–, 1997.

[39] L. P. Stephan Salinger, Laura Plonka. The mann-whitney u: A test for assessing whether two independent samples come from the same distribution. *Human Technology: An Interdisciplinary Journal on Humans in ICT Environments*, 4(1):9–25, 2008.

[40] A. Strauss and J. M. Corbin. *Basics of Qualitative Research : Techniques and Procedures for Developing Grounded Theory*. SAGE Publications, September 1998.

[41] F. van Rysselberghe and S. Demeyer. Mining Version Control Systems for FACs (frequently Applied changes). In *Proceedings of the International Workshop on Mining Repositories*, 2004.

[42] J. Wang and J. Han. Bide: Efficient mining of frequent closed sequences. In *Proceedings of the 20th International Conference on Data Engineering*, ICDE '04, pages 79–, 2004.

[43] J. Wang, X. Peng, Z. Xing, and W. Zhao. An exploratory study of feature location process: Distinct phases, recurring patterns, and elementary actions. In *Proceedings of the 27th IEEE International Conference on Software Maintenance*. IEEE, 2011.

[44] X. Xia, L. Bao, D. Lo, and S. Li. Automated debugging considered harmful: A user study revisiting the usefulness of spectra-based fault localization techniques with professionals using real bugs from large systems. In *Proceedings of the 2016 IEEE International Conference on Software Maintenance and Evolution*, pages 267–278, 2016.

[45] A. T. T. Ying, G. C. Murphy, R. T. Ng, and M. Chu-Carroll. Predicting source code changes by mining change history. *IEEE Transactions on Software Engineering*, 30(9):574–586, 2004.

[46] T. Zimmermann, S. Kim, A. Zeller, and E. J. Whitehead, Jr. Mining version archives for co-changed lines. In *Proceedings of the 2006 International Workshop on Mining Software Repositories*, pages 72–75, 2006.

[47] T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller. Mining version histories to guide software changes. In *Proceedings of the 26th International Conference on Software Engineering*, pages 563–572, 2004.