Are Suggestions of Coupled File Changes Interesting?

Jasmin Ramadani and Stefan Wagner
University of Stuttgart, Stuttgart, Germany

Keywords: Data Mining, Repository Mining, Coupled Changes, Interestingness.

Abstract: Software repositories include information which can be made available for bug fixing or maintenance using repository mining. The identification of coupled changes have been proposed several times. Yet, existing studies focus on the found couplings and ignore feedback from developers. We investigate three development projects and their repositories to find files that frequently change together to support the software developers. We complement the coupled files information with details from the issue tracking system and the project documentation. We contrast our findings with feedback from the developers about how interesting our findings are for them. We found that the small size of the repositories made an insightful analysis difficult. The response to coupled changes both from experienced and inexperienced developers was mostly neutral. They accepted most of the additional attributes we presented. Furthermore, developers also suggested other additional issues to be relevant, e.g. the context of the coupled changes and the way they are presented, which we did not cover in this study. Therefore, coupled change analysis research will need to take the presentation and context information into account.

1 INTRODUCTION

Software product development produces large amounts of data which is stored in software repositories. These repositories contain the artifacts developed during software evolution. They include different data sources like version control systems, issue tracking systems and project documentation archives. After some time, this data becomes a valuable information source for bug fixing or maintenance tasks.

To learn from it, we need a technique to extract relevant details from the source code history and search for valuable information. One of the most used techniques is data mining which has become popular for analyzing software repositories. The term *mining software repositories (MSR)* has been coined to describe investigations of software repositories using data mining (Kagdi et al., 2007).

To help the developers to identify the files to be changed during bug fixing or maintenance tasks, a mining software repositories approach has been proposed which finds files that have changed together frequently (Ying et al., 2004). These files can be used to recommend coupled file changes. Couplings are defined as "the measure of the strength of association established by a connection from one module to another" (Stevens et al., 1974). Change couplings are described as files having the same commit time, author and modification description (Gall et al., 2003). Frequently changed files can support developers in dealing with the large amount of information about the software product, especially if the developer is new on the project, the project started a long time ago or if the developer does not have much experience in software development.

1.1 Problem Statement

Several researchers have proposed approaches to identify coupled files to give recommendations to developers during a change (Bavota et al., 2013). Existing studies, however, focus on the presentation of the mining results and ignore the feedback of developers on the findings.

1.2 Research Objectives

The overall aim of our research is to support the developers in common maintenance tasks. In this paper, we concentrate on applying MSR to provide suggestions for likely changes so that we can investigate how interesting the suggestions are for the developers and what further information besides version histories might increase the interestingness.

Ramadani J. and Wagner S.
Are Suggestions of Coupled File Changes Interesting?.
DOI: 10.5220/0005854400150026
In *Proceedings of the 11th International Conference on Evaluation of Novel Software Approaches to Software Engineering (ENASE 2016)*, pages 15-26
ISBN: 978-989-758-189-2

We define *interestingness* as the subjective measure of the developers' opinion on how useful findings (here: coupled change suggestions) are for bug fixing or maintenance tasks.

1.3 Contribution

We present an industrial case study on the interestingness of coupled change suggestions. We identify frequent couplings between file changes based on the information gathered from three software project repositories and investigate the interestingness of these couplings. We use the version control system, the issue tracking system and the project documentation archives as data sources for additional information. We join this additional information to the coupled changes we discover. In particular, we investigate the feedback of the developers about the interestingness of our findings by conducting a survey. We evaluate the answers by performing additional interviews and analyze them using the *Grounded Theory* method.

2 DATA MINING BACKGROUND

To be able to discover coupled file changes by using data mining, we introduce the data technique that we employ in our study. One of the most popular data mining techniques is the discovery of frequent itemsets. To identify sets of items which occur together frequently in a given database is one of the most basic tasks in data mining (Han, 2005). Coupled changes describe a situation where someone changes a particular file and also changes another file afterwards. Let us say that the developer changes file f_1 and then also frequently changes file f_3. By investigating the transactions of changed files in the version control system commits we identify a set of files that changed together. Let us have the following three transactions: $T_1 = \{f_1, f_2, f_3, f_7\}$, $T_2 = \{f_1, f_3, f_5, f_6\}$, $T_3 = \{f_1, f_2, f_3, f_8\}$. From these three transactions, we isolate the rule that files f_1 and f_3 are found together: f_1 and f_3 are coupled. This means that when the developers changed file f_1, they also changed file f_3. If these files are found together frequently, it can help other persons by suggesting that if they change f_1, they should also change f_3. Let $F = \{f_1, f_2, ..., f_d\}$ be the set of all items (files) f in a transaction and $T = \{t_1, t_2, ..., t_n\}$ be the set of all transactions t. As transactions, we define the commits consisting of different files. Each transaction contains a subset of chosen items from F called itemset. An important property of an itemset is the support count δ which is the number of transactions containing an item. We call the itemsets frequent if they have a support threshold min_{sup} greater than a minimum specified by the user with

$$0 \leq \min_{\text{sup}} \leq |F| \quad (1)$$

3 RELATED WORK

There are many studies dedicated to investigating software repositories to find logically coupled changes, e.g. (Bieman et al., 2003; Gall et al., 2003; Fluri et al., 2005). We identify two granularity levels, the first one investigates the couplings based on the file level (Ying et al., 2004; Kagdi et al., 2006) and the second one is a finer granularity level where the coupled changes are identified between parts of files like classes, methods or modules (Zimmermann et al., 2004; Zimmermann et al., 2006; Fluri et al., 2005; Kagdi et al., 2007).

Most studies dealing with identifying coupled changes use some kind of data mining for this purpose (Zimmermann et al., 2004; Ying et al., 2004; Kagdi et al., 2006; German, 2004; Hattori et al., 2008; Shirabad et al., 2003; van Rysselberghe and Demeyer, 2004). Especially the association rules technique is often used to identify frequent changes (Zimmermann et al., 2004; Ying et al., 2004; Kagdi et al., 2006). This data mining technique uses various algorithms to determine the frequency of these changes. Most of the studies employ the Apriori algorithm (Zimmermann et al., 2004; Kagdi et al., 2006), however other algorithms like the FP-Tree algorithm are also in use (Ying et al., 2004).

Most of the studies use a single data source where a kind of version control system is investigated, typically CVS or Subversion. To our knowledge there are few studies which investigate a Git version control system (Bird et al., 2009; Hassan and Holt, 2004; Carlsson, 2013). Other studies combine more than one data source to be investigated, like a version control system and an issue tracking system (Fischer et al., 2003; Canfora and Cerulo, 2005; Wu et al., 2011; D'Ambros et al., 2009) where the data extracted from these two sources is analyzed and the link between the changed files and issues is determined.

To the best of our knowledge, there are only three studies investigating how couplings align with developers' opinions or feedbacks. Coupling metrics on the structural and the semantic level are investigated in (Revelle et al., 2011). The developers are asked if they find these metrics to be useful. They show that feature couplings on a higher level of abstraction than classes are useful. The developers' perceptions of

software couplings are investigated in (Bavota et al., 2013). Here the authors examine how class couplings captured by different coupling measures like semantic, logical and others align with the developers perception of couplings. The semantic couplings have received the best rating of all types of couplings. The interestingness of coupled changes is also studied in (Ying et al., 2004). This study defines categorization of coupled changes interestingness according to the source code changes.

We focus on the interestingness of coupled file changes and attributes involving the developers' feedback on our findings using the following data sources: Two of the projects use Git[1] and the third one uses Mercurial.[2] The first industrial project uses JIRA as issue tracking system,[3] the open source project and the second industrial project use Redmine.[4] We use the available product documentation of the projects as additional source of information.

4 CASE STUDY DESIGN

The structure of our case study is based on existing guidelines (Runeson and Höst, 2009).

4.1 Research Questions

Our case study addresses five research questions:

RQ1: How Many Coupled Changes Can We Extract from Software Repositories?

This research question provides the basis for our research. It is relevant to investigate for the reason that the number of coupled changes affects the outcome of the repository data analysis.

RQ2: How Interesting Are Coupled Change Suggestions for Developers?

This is the central question of this study which decides if developers will use the suggested couplings.

RQ3: Does the Experience of Developers Influences the Interestingness of Coupled Changes?

We expected that inexperienced developers would be more interested in coupled file suggestions considering their possible problems understanding the system (Sayles et al., 2011). Therefore, we investigate the developer's programming and project experience.

RQ4: How Interesting Is Additional Information from Other Related Project Artifacts?

After we determine the interestingness of the couplings, we will investigate if adding additional data sources influences the interestingness. First, we examine the version control system that is related to the changes, e.g. commit ids where the couplings were found, commit messages, commit dates and authors of the commits. Second, the information stored in the issue tracking system is investigated, attributes like issue description, issue date and issue status. Third, we look into the project documentation archive for information about the project structure and naming conventions.

RQ5: Does the Experience of Developers Influences the Interestingness of Additional Information from Other Related Project Artifacts?

We investigate if the choice of the attributes from the version control system and the issue tracking system depends on the developer's programming experience.

4.2 Interestingness

In our study, we consider interestingness as a subjective measure which is derived from the user's beliefs or expectations (McGarry, 2005). Information is *interesting* if it is novel, useful and nontrivial to compute. Here, *useful* means that it can help to achieve a goal of the system or the user (Frawley et al., 1992). The interestingness of the information represents the possibility that developers will use this information during their maintenance or bug fixing tasks.

We measure interestingness of couplings using three levels: interesting, neutral and not interesting. We identify two categories of interestingness. The first category is the interestingness of coupled changes.

The second category is the interestingness of the additional attributes we extract from the version control system, the issue tracking system and the project documentation. We join this attributes to the coupled file changes.

4.3 Case Selection

The case selection is based on their availability and the suitability for our research. We select cases from industry as a part of our cooperation with our industrial partners as well as from the available open source projects developed at the University of Stuttgart. Hence, our subjects will be practitioners as well as students.

4.4 Data Collection Procedure

The case study uses two main data sources to investigate the coupled file changes. As first data source,

[1] http://git-scm.com/

[2] http://mercurial.selenic.com/

[3] https://www.atlassian.com/software/jira

[4] http://www.redmine.org/

we use the artifacts from the software product development archived in software repositories. We did not have any direct contact with the development process of the product. Instead, we examine the repositories of the software product being developed or maintained. The second data source consists of surveys and interviews with the project stakeholders providing direct information. We divide the data collection procedure into five parts.

4.4.1 Version Control System

The first unit of data we use is the log data from the version control system. Two software projects used Git, while the third project uses Mercurial as a control management tool. Both are distributed version control systems allowing the developers to maintain their local versions of source code. The data collection from the version control system consists of four steps which lead to the extraction of the information we need.

- Log Extraction: We extract the information from the log file containing the committed file changes and the commit attributes. The log data is exported as text file.
- Data Preprocessing: After the text files with the log data have been generated, we continue with the preparation of the data for data mining. Various data mining frameworks use their own format, so the input for the data mining algorithm and framework needs to be adjusted.
- Identifying Atomic Change Sets: We divide the data into a collection of atomic change sets. Version control systems deal with this issue differently. In our case, the version control systems preserve the possibility to group changes into a single change set or a so-called *atomic commit*. It represents an atomic changeset regardless of the number of directories, files or lines of code that change. A commit snapshot represents the total set of modified files and directories (Loeliger, 2009). We organize the data in a transaction form where every transaction represents the files which changed together in a single commit.
- Data Filtering: We filter the file names and the following commit attributes: *commit id*, *commit message*, *commit* date and *commit author*. We deal with empty entries and outliers and we prepare the log entries for data mining.
- Change Grouping Heuristic: There are different heuristics proposed for grouping file changes (Kagdi et al., 2006). We use a heuristic considering the file changes done by a single committer are related. We group the transactions of files committed only by a particular author. We do not relate the changes done by other committers.

4.4.2 Issue Tracking System

In issue tracking systems, important information is stored about the software changes or problems. In our case, the companies chose to use JIRA and Redmine as issue tracking systems. The students also track their issues using Redmine. We investigate the follwing issue attributes: *issue titles*, *issue descriptions* and *issue messages*. The issue tracking systems support spreadsheet export containing the considered issue attributes.

4.4.3 Project Documentation

The software documentation gathered during the development process represents a rich source of data. The documentation consists of file naming conventions, directory paths and the project structure description. From these documents, we discover the project structure. For example in the last project, the subproject containing the files described by the path `astpa/controlstructure/figure/` contains the Java classes responsible for the control diagram figures of this software.

4.4.4 Joining Collected Data

After the mining process is finished and we have identified the coupled changes, we will join them with the attributes from the version control system, the issue tracker and the project documentation. In (Fischer et al., 2003), the authors create a release history database where they import the data from the version control systems and the issue tracking systems. Similarly, we create a database containing all file changes and the corresponding attributes from the repositories.

Every commit has it own hash value which represents the commit id. It is a unique value which identifies all the commits in the database. The issues are identified by their keys. We use the issue keys to follow down the commit where the change took place using the merge points of issues with the commit messages. We use the path information of the changed files to enlist the subproject. As a result we have a list of the most frequently changed files accompanied by the information about the commit attributes, issue attributes and the project structure.

4.4.5 Survey and Interviews

We investigate the developers' feedback on the interestingness of coupled changes and the additional at-

tributes by conducting a survey and performing interviews[5] with the developers.

Survey. The developers answer a list of multiple-choice questions online. We investigate the background of the developers by asking their programming and project experience. The developers give us feedback on the concept of coupled changes, not on particular couplings. We chose this setup as a first means to get as many opinions as possible. Only few developers were available for in-depth interviews on specific findings. The developer can choose between: interesting, neutral and not interesting.

- What is your programming/project experience?
- How interesting are suggestions for coupled changes for you?
- What commit/issue/documentation information for bug fixing or maintenance tasks is interesting for you?

Interviews. We perform semi-structured interviews to get more in-depth feedback from the developers. This way, we ensure that the developers did not answer the surveys by randomly choosing the options. We ask the available developers who worked on the projects and other uninvolved developers about the interestingness of the file changes and the attributes. We present them actual coupled file changes extracted from the repositories.

4.5 Ethical Considerations

The data delivered by the companies is confidential. Therefore, we preserve the anonymity of the stakeholders and the companies during this study. The confidentiality and the publication is regulated by a non-disclosure agreement between the researchers and the companies. All personal information extracted from the repositories, the survey and the interviews is anonymized and is not presented in the study.

4.6 Analysis Procedure

The data analysis is a combination of quantitative and qualitative methods. We use quantitative methods to find the number of couplings. We augment the results with a qualitative and quantitative analysis of the survey and the interviews with the developers.

[5] All questions are available on http://dx.doi.org/10.5281/zenodo.15065

4.6.1 Analysis of Repository Data

We analyze the repository data to answer RQ1. We run the mining algorithm to discover frequently coupled file changes. We investigate the additional attributes we gather from the commit logs, the issue tracking export and the project documentation.

Data Mining Algorithm. Various algorithms for mining frequent itemsets and association rules have been proposed in literature (Agrawal and Srikant, 1994; Han et al., 2004; Győrödi and Győrödi, 2004). We use the FP-Tree-Growth algorithm to find the frequent change patterns. As opposed to the Apriori algorithm (Agrawal and Srikant, 1994) which uses a bottom up generation of frequent itemset combinations, the FP-Tree algorithm uses partition and divide-and-conquer methods (Győrödi and Győrödi, 2004). This algorithm is faster and more memory efficient than the Apriori algorithm used in other studies. This algorithm allows frequent itemset discovery without candidate itemset generation.

We analyze the coupled changes by defining the threshold value of the support for the frequent itemset algorithm. We use the thresholds that give us a frequent yet still manageable number of couplings. This threshold is normally defined by the user. We use the technique proposed by Fournier-Viger presented in (Fournier-Viger, 2013) to identify the support level. These values vary from developer to developer, so we test the highest possible value that delivers frequent itemsets. If for a particular developer, the support value does not bring any useful results, we continue dropping the value of the threshold. We did not consider itemsets with a support below 0.2 for the first two projects and 0.1 for the third project.

There is a variety of commercial and open-source products offering data mining techniques and algorithms. For the analysis, we use an open-source framework specialized on mining frequent itemsets and association rules called the *SPMF-Framework*.[6] It consists of a large collection of algorithms supported by appropriate documentation.

4.6.2 Analysis of Questionnaires and Interviews

To answer RQ2–RQ5, we analyze the questionnaires and the outcomes of the interviews.

Survey Analysis. We start by investigating the background of the developers by checking their answers about their programming and project experi-

[6] http://www.philippe-fournier-viger.com/spmf

ence. We analyze the answers from the questionnaire by calculating the distribution of the frequency of their answers. We put the main focus on the answers of the participants about the interestingness of coupled changes and the answers about the additional attributes.

Interview Analysis. We examine the interviews with the developers to validate the outcomes of the questionnaires and to understand the context of their answers. We analyze the interviews by using *Grounded Theory* (Strauss and Corbin, 1998). The goal is to generate a theory that emerges from the data being comparatively analyzed. To analyze the data and build the theory, we use the following types of coding activities in sequence: open, axial and selective coding (Strauss and Corbin, 1998). After these codings, we perform the theoretical coding and create the conceptual model. We use the analysis software *Atlas.ti*[7] to link the codes and create a network diagram.

- *Open Coding*: In the open coding we have a line-by-line examination of the interview transcripts to identify the main concepts and categories together with their dimensions and properties. We code the data from interview answers with a set of open codes derived from our research questions. Before we continue, we write a memo consisting of the hypotheses and ideas noted during the analysis.
- *Axial Coding*: After the open coding is performed, we continue with the axial coding where we relate the categories, concepts and codes by identifying the relations among them. This is done using the paradigm model (Strauss and Corbin, 1998) and considering the relationships between contexts, interactions, conditions and consequences.
- *Selective Coding*: The selective coding formulates a core category to which all other categories and codes can be related and includes all of the data.
- *Theoretical Coding*: After finishing the open and axial coding, this coding involves the relationships between categories and subcategories and gives meaning to the theory.
- *Conceptual Mapping and Model*: We express the concepts of our theory and present their relations. We draw a category map which emerges from the analysis.

[7]http://www.atlasti.com/index.html

4.7 Validity Procedure

4.7.1 Internal Validity

We use widely known techniques and algorithms for repository mining. We extract data from a repository systems used among a large number of companies. We analyze the data from the software repository, perform a survey among the developers and we validate the answers given in the questionnaires by interviewing developers. We analyze the answers and compare the results related to the research questions to see if these reflect the investigated information (Runeson and Höst, 2009). This way we avoid to rely on a possible lack of precision in the answers on the questionnaires by the developers concerning the interestingness.

4.7.2 External Validity

We choose representative cases with high standards considering software development and standardized development techniques. We use an independent party to record the memos for the interviews and code the information to increase the objectivity of the analysis results.

5 CASE STUDY RESULTS

We report the results of the analysis of the software repository data, the questionnaires and the interviews in relation to the interestingness of coupled changes and attributes.[8] We discuss the analysis outcomes and evaluate the validity of our results by taking into account the feedback from the developers.

5.1 Case Description

The cases we investigate in this study represent three software projects. Two projects were provided by IT companies from the area of Stuttgart, Germany. The third one is an open-source project developed at the University of Stuttgart. The first project under analysis is a web-based software written in Java and supplied by an industrial partner. The project repository contains 1,610 commits out of which we consider the commits made by 26 developers during 2 years of development. The commit messages are stored in Git and the issues are tracked using JIRA. The second project is a C# software supplied by another company.

[8]The analysis results are available at http://dx.doi.org/10.5281/zenodo.15065

It has 159 commits created by 5 developers during 1 year. The project used Mercurial as version control tool and Redmine for issues management. The third project is a Java open source software supplied by student developers. It contains 752 commits, committed by 9 developers during 1 year. It uses Git for the versioning and Redmine as issue tracking system. Certain project documentation archives were available from where we extract the information about the software structure and the naming conventions for the projects.

5.2 Findings

5.2.1 Number of Couplings (RQ1)

We summarize the analyzed information form the our repository data. Referring to the first project, we identify relevant data from 22 out of 26 developers. For the second project, the data from 4 out of 5 developers is taken into account. For the third project, all 9 developers committed suitable data for analysis. The rest of the developers reported a low number of commits. We exclude their commits as unsuitable to successfully perform the data mining analysis. Their data did not reach the established minimum support we defined previously. These results are presented in Table 1.

The number of commits shows the size of the projects followed by the number of change couplings found during the data mining process. The number of coupled changes represents the basis of our analysis. The analysis of the data from the first repository delivers 205 couplings. The second, a very small repository, reports only 13 coupled changes. The third repository delivered 200 coupled changes.

Table 1: Results based on repository analysis.

	Project 1	Project 2	Project 3
No. of relevant developers	22	4	9
No. of commits total	1,610	138	752
No. of couplings found	205	13	200
Freq. items support	0.2	0.2	0.1

The results report that larger or older projects deliver more data to analyze, delivering larger number of couplings.

5.2.2 Interestingness of Coupled Changes (RQ2)

We asked the participants how interesting they find that coupled changes are for bug fixing or maintenance tasks. The results report that 19 of 23 developers had a neutral opinion for the concept of coupled changes while 4 of the participants noted coupled changes as interesting. None of the developers reject the idea as not interesting (Table 2).

Table 2: Interestingness of coupled changes.

	involved	not involved	all
interesting	2	2	4
neutral	9	10	19
not interesting	0	0	0
sum	11	12	23

The fact that the developers do not reject coupled changes allows us to continue our analysis. This research question is the basis for further interestingness analysis. We proceed our analysis and investigate how coupled changes depend on the developers characteristics like their programming and project experience. Taking into account our small sample size, we refrain from formal hypothesis testing.

5.2.3 Influence of Developer Experience on Interestingness (RQ3)

The results in Table 2 show that there is no difference based on the involvement of the developers in the projects. Both involved and uninvolved developers did not reject coupled changes. We group the developers answers based on their programming experience. Table 3 shows the distribution of the developers by their programming experience. In contrast to our expectations, both experienced and inexperienced developers are interested in coupled changes. In table 4 we present the distribution of the interestingness of coupled changes related to the programming experience of the developers. None of the developers reject the coupled changes. The sample size of the results is not suitable to formally test a hypothesis for this research question.

Table 3: Developers experience.

progr. experience	frequency	frequency [%]
<1 year	2	9
1–3 years	4	17
3–5 years	9	39
>5 years	8	35

Table 4: Couplings and developer's experience.

programming experience	interesting	neutral	not interesting
<1 year	0	2	0
1–3 years	2	2	0
3–5 years	1	8	0
>5 years	1	7	0

5.2.4 Interestingness of Additional Information (RQ4)

We investigate the interestingness of additional attributes from the repository we have joined to the coupled files. To support the coupled changes, we report common meta-data attributes (Steven and Zach, 2013) which allow us to find more information about the commits, the issues and the product itself. The repositories offer various attributes concerning the commits performed, the issues found and the project structure. We asked the participants to report which attributes they find interesting. The results show that most of the offered attributes are rated by the developers as interesting. From our commit attributes, most of the developers find the commit message to be the most interesting attribute followed by the file name. The developers do not show much interest for the the commit time, the commiter and the file type. The commit id as attribute does not attract the developers' attention. From our issue attributes, most of the developers are interested in the issue description. Some of the developers find the issue status and type to be interesting. The issue time does not interest the developers. From our documentation, the naming convention and the project structure information is more interesting than the naming convention as shown in Table 5.

Table 5: Interesting attributes.

type	*question*	*frequency*	*frequency [%]*
commits	attributes		
	commit message	22	95
	file name	18	78
	file type	9	39
	commit time	8	34
	commiter	6	26
	commit id	2	9
issues			
	issue title	21	91
	issue status	15	65
	issue type	14	60
	issue time	6	26
docu			
	project struct.	20	86
	naming conv.	15	65

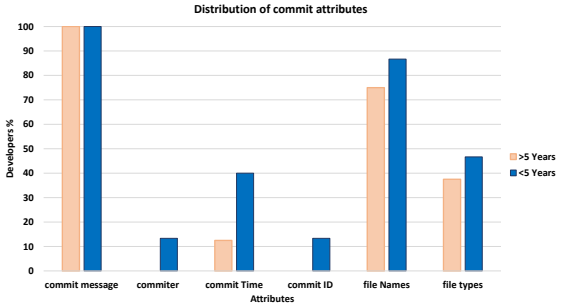

Figure 1: Commit attributes distribution for high and low experience.

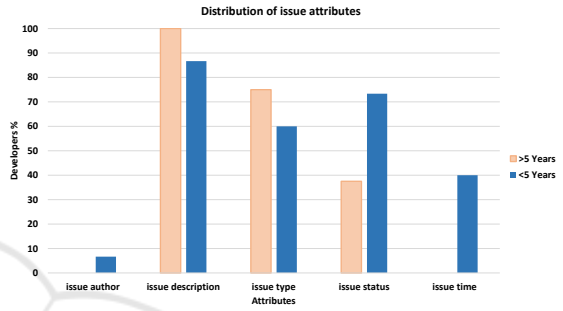

Figure 2: Issue attributes distribution for high and low experience.

5.2.5 Influence of Developer Experience on Interestingness of Additional Information (RQ5)

We examined the distribution of interestingness of various commit, issue and project attributes according to developers' experience. We created two groups of developers in this context: experienced, having more than 5 years experience and inexperienced, having less than 5 years of experience. We continue using these two categories of developers. The results show that the experienced developers have a more clear picture about which attributes are interesting for them. They have chosen a lower number of attributes. The inexperienced developers have marked many commit and issue attributes being interesting for them.

The more experienced developers' choice is more narrow than the one for the inexperienced ones. The distribution of commit attributes is shown in Figure 1. The distribution of issue attributes is presented in Figure 2.

5.2.6 Validation and Theory

After the data mining analysis, we performed the interviews with developers who were active on the projects. For the first project, we managed to enlist 2 of the developers for interviewing. For the second

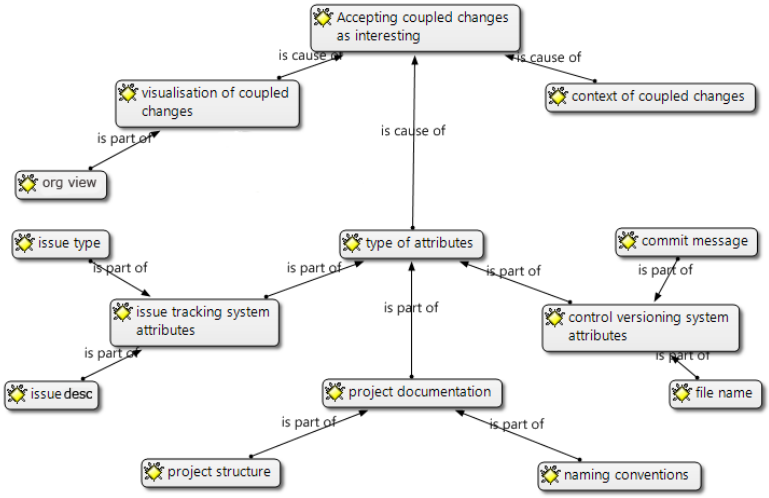

Figure 3: Theoretical Framework.

project, we interviewed 2 developers and from the third project, we interviewed 4 out of 9 developers. They had been involved in the project from the beginning and have the most knowledge about the software. We also interviewed 4 developers not involved in any of the projects.

Using Grounded Theory analysis on the interview transcripts, we derived a corresponding theory. We created the codes using an open coding procedure of the memos we created. They represent the answers of our participants to interview questions. We extracted the codes by identifying common issues in their answers.

We continued with the axial coding where we identified several categories as presented in Figure 3. The core category we identified after the selective coding is *Interestingness of couplings and software repository information*. The results from the theoretical code show the core category, the subcategories and the relationships presented as a diagram in Figure 3. We have categories covering the attributes we found to be interesting: version control attributes, issue attributes and project documentation. They are respectively divided in these subcategories: commit message, file names, issue titles, issue types, project structure and naming conventions. They represent the most interesting attributes which affect the interestingness of coupled changes.

The next categories are the visualization of coupled changes, consisting of the sub-category *organized view*, and the category *context of coupled changes*. The last two categories represent an additional feedback given by the interviewed developers where they would like to see an organized representation of changed files with a possibility to filter the information about them. They would also like to have information about the context of the changes.

We present the key concepts of the theory together with their relations in Figure 4. We see that the interestingness of the coupled changes also depends on the chosen repository attributes. Furthermore, it is also important to develop an organized presentation of coupled changes to the developers and to describe the context of these changes.

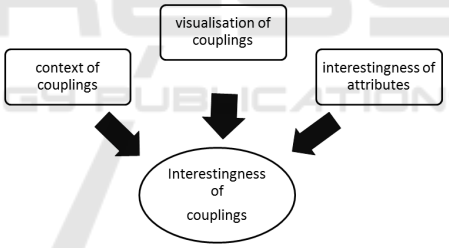

Figure 4: Conceptual Model from Grounded Theory.

5.3 Discussion

The results related to RQ1 show that large repositories deliver more couplings compared to the smaller or younger repositories. Projects with a low number of commits which lasted for a shorter time, do not provide enough data for a broader analysis. The number of commits and their size limit the output of our analysis. Our results lead to the conclusion that we need a relatively high number of couplings to be able to present a more exhaustive support for the developers in their tasks. Still, our analysis discovers that in our projects we can identify frequent coupled file changes using the proposed mining technique.

The results for RQ2 from the analysis of the questionnaires, weakly support that coupled changes are

interesting. The general concept of coupled changes was received mostly as neutral. The fact that none of the developers rejected the coupled changes, gives us an impulse to investigate other attributes related to the coupled changes. We proceeded with the analysis of the interestingness based on the developers' experience. During the interviews, the acceptance increased when actual examples of coupled changes were presented to the developers.

For RQ3, we expected that the coupled changes would be interesting for developers having a lack of programming or experience on the project. Our results at contrary show that experienced developers are also interested in coupled changes. Also the fact that both involved and uninvolved developers gave similar results, makes the coupled changes attractive for a broader audience. This encourage us that the coupled changes should be part of an integrated tool support for developers.

Answering RQ4, our results show that most of the attributes we reported are considered interesting by the developers. These results were also validated by the interviews. From our commit attributes, the questionnaire and the interviews, we report that the commit message and the file names are the most interesting attributes. For the issue attributes, these are the issue description and the issue type, and for the documentation the project structure and the naming convention are most interesting for the developers. We report a number of attributes used by the tools involved in all three projects. The attributes we defined are known and common in software development. During the analysis of the interviews, however, we found that the developers want a clear presentation of the couplings. They also reported that they would like to see the context of the coupled changes. This brings additional aspects to be considered in further research about coupled changes.

The results for RQ5 show that experienced developers know well what kind of attributes they want to see. Their choice is more precise. In contrary, the inexperienced developers do not have a clear picture what attributes they want to add to the coupled changes. The fact that the developers consider various attributes to be interesting brings us to the conclusion that we do should not make a fixed choice of attributes for all developers. We can offer the possibility for the developers to choose the attributes individually. This way, we support the developers which are not sure what attributes can be helpful to accomplish bug fixing or maintenance tasks.

5.4 Evaluation of Validity

We validated the results of our study by checking all the steps in the procedure of gathering and transforming the data from the repository, the analysis methods and the results.

In our study, we use a single data mining technique for the reason that the frequent itemsets technique is most appropriate for investigating frequent couplings. We investigated products built with common technologies and the repositories are maintained by well known and commonly used products.

We tested different threshold values for the support and the confidence of the algorithm to produce a sufficient number of frequent itemsets. The relatively low support threshold signalizes that there is not much space for a greater reduction of the value. However, it also reports a relatively low number of frequent couplings which reduces the possibility that these couplings happened by chance.

We validated the outcomes of the questionnaire answers by asking the developers again in the interviews about the interestingness of the couplings and attributes. The interview transcript was coded by two persons after we compared the notes. This way we checked whether we understood the developer's answers correctly. We interviewed both involved and not involved developers on the projects. We also performed double checks of the coding and the outcomes of the Grounded Theory analysis.

6 CONCLUSION AND FUTURE WORK

We summarize the conclusions from the case study, set them into relation with existing evidence, discuss possible implications and limitations and describe future research directions.

6.1 Summary of Conclusions

According to our results, we need relatively large repositories to be able to perform a successful analysis and report useful numbers of coupled changes. We conclude that coupled changes are to some degree interesting for the developers. The software repository attributes we joined to the couplings were accepted by the developers as interesting. Although we provided a number of attributes and aspects following the coupled changes, the developers suggested additional accompanying information. They would like to see the context of the changes and they also care about that how the suggestions should be presented to them.

Our results lead to the conclusion that the couplings are also interesting for more experienced developers. Developers with various level of programming experience find different coupled change attributes as interesting. We conclude that the list of attributes to be presented should be individually adjusted by the developers. The Grounded Theory shows that these attributes affect the interestingness of coupled changes.

6.2 Relation to Existing Evidence

The authors in (Revelle et al., 2011) investigate source code features coupling using structured and textual features where the developers are surveyed to determine if the metrics align with the developers' opinion. Their results show that the developers support the feature coupling metrics they propose indicating that the measures capture couplings between features. Our results show that the developers accept the concept of coupled changes and the corresponding attributes from the repository.

We investigate the interestingness of coupled changes as explained in (Ying et al., 2004), whereby the authors use open-source projects and categorize the interestingness of couplings according to their criteria. We studied the coupled file changes in relatively small projects: two industrial projects and one open source project which reduces the number of couplings found. We used the developer's feedback to determine the interestingness of coupled changes instead of statically defining the interestingness of couplings.

6.3 Impact/Implications

This case study gives evidence that the analysis of coupled changes is interesting to the developers during bug fix or maintenance tasks. Yet, the interest is rather weak overall. Therefore, other contextual information should be investigated in future research to increase the interestingness. This kind of suggestions could be incorporated in a tool to support the developers during maintenance or bug fixing tasks.

6.4 Limitations

As it is case study research, we cannot guarantee the generalizability of the study. The data comes from two software development companies and one open-source project. However, the procedure should be similar for other projects for the reason that we use well defined data mining algorithms, techniques and commonly used data sources. We can perform our analysis on similar projects or data sources using these techniques. The number of coupled changes we found is limited by the support value of the frequent itemsets algorithm. We do not report a large number of couplings. Our results preserve the most frequent couplings, however, which are the most valid ones. We have a small sample which is not suitable for a deeper statistical analysis. Yet, our findings constitute a first insight about developers' opinion on coupled file changes.

6.5 Future Work

We plan to continue our research on coupled changes by directly observing their use for a real bug fix or maintenance tasks. The next step is to perform an experiment to investigate the usefulness of coupled changes to and create a tool to present this changes to the developers. Furthermore, based on our findings, we believe more research should look into complementing the reporting of coupled changes with information from additional, related data sources.

ACKNOWLEDGMENT

The authors would like to thank Asim Abdulkhaleq for his help in the interview transcripts and coding for the Grounded Theory analysis.

REFERENCES

Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Bavota, G., Dit, B., Oliveto, R., Di Penta, M., Poshyvanyk, D., and De Lucia, A. (2013). 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, Piscataway, NJ, USA. IEEE Press.

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

Bird, C., Rigby, P. C., Barr, E. T., Hamilton, D. J., Germán, D. M., and Devanbu, P. T. (2009). The promises and perils of mining git. In *MSR*, pages 1–10.

Canfora, G. and Cerulo, L. (2005). Impact analysis by mining software and change request repositories. In *Software Metrics, 2005. 11th IEEE International Symposium*, pages 9 pp.–29.

Carlsson, E. (2013). Mining git repositories : An introduction to repository mining.

D'Ambros, M., Lanza, M., and Robbes, R. (2009). On the relationship between change coupling and software defects. In *WCRE*, pages 135–144.

Fischer, M., Pinzger, M., and Gall, H. (2003). Populating a release history database from version control and bug tracking systems. In *Proceedings of the International Conference on Software Maintenance*, ICSM '03, pages 23–, Washington, DC, USA. IEEE Computer Society.

Fluri, B., Gall, H., and Pinzger, M. (2005). Fine-grained analysis of change couplings. In *Source Code Analysis and Manipulation, 2005. Fifth IEEE International Workshop on*, pages 66–74.

Fournier-Viger, P. (2013). How to auto-adjust the minimum support threshold according to the data size. http://data-mining.philippe-fournier-viger.com/.

Frawley, W. J., Piatetsky-shapiro, G., and Matheus, C. J. (1992). Knowledge discovery in databases: an overview.

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

German, D. M. (2004). Mining cvs repositories, the softchange experience. In *1st International Workshop on Mining Software Repositories*, pages 17–21.

Győrödi, C. and Győrödi, R. (2004). A comparative study of association rules mining algorithms.

Han, J. (2005). *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Han, J., Pei, J., Yin, Y., and Mao, R. (2004). Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Min. Knowl. Discov.*, 8(1):53–87.

Hassan, A. E. and Holt, R. C. (2004). Predicting change propagation in software systems. In *Proceedings of the 20th IEEE International Conference on Software Maintenance*, ICSM '04, pages 284–293, Washington, DC, USA. IEEE Computer Society.

Hattori, L., dos Santos Jr, G., Cardoso, F., and Sampaio, M. (2008). Mining software repositories for software change impact analysis: A case study. In *Proceedings of the 23rd Brazilian Symposium on Databases*, SBBD '08, pages 210–223, Porto Alegre, Brazil, Brazil. Sociedade Brasileira de Computação.

Kagdi, H., Collard, M. L., and Maletic, J. I. (2007). A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *J. Softw. Maint. Evol.*, 19(2):77–131.

Kagdi, H., Yusuf, S., and Maletic, J. I. (2006). Mining sequences of changed-files from version histories. In *Proceedings of the 2006 International Workshop on Mining Software Repositories*, MSR '06, pages 47–53, New York, NY, USA. ACM.

Loeliger, J. (2009). *Version Control with Git - Powerful techniques for centralized and distributed project management.* O'Reilly.

McGarry, K. (2005). A survey of interestingness measures for knowledge discovery. *Knowl. Eng. Rev.*, 20(1):39–61.

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

Runeson, P. and Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Softw. Engg.*, 14(2):131–164.

Sayles, J. et al. (2011). *z/OS Traditional Application Maintenance and Support*. IBM Redbooks.

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

Steven, J. and Zach, W. (2013). Bad commit smells. http://pages.cs.wisc.edu/ sjj/docs/commits.pdf.

Stevens, W. P., Myers, G. J., and Constantine, L. L. (1974). Structured design. *IBM Syst. J.*, 13(2):115–139.

Strauss, A. and Corbin, J. M. (1998). *Basics of Qualitative Research : Techniques and Procedures for Developing Grounded Theory*. SAGE Publications.

van Rysselberghe, F. and Demeyer, S. (2004). Mining Version Control Systems for FACs (frequently Applied changes). In *the International Workshop on Mining Repositories*, Edinburgh, Scotland, UK.

Wu, R., Zhang, H., Kim, S., and Cheung, S.-C. (2011). Relink: Recovering links between bugs and changes. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, pages 15–25, New York, NY, USA. ACM.

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

Zimmermann, T., Kim, S., Zeller, A., and Whitehead, Jr., E. J. (2006). Mining version archives for co-changed lines. In *Proceedings of the 2006 International Workshop on Mining Software Repositories*, MSR '06, pages 72–75, New York, NY, USA. ACM.

Zimmermann, T., Weisgerber, P., Diehl, S., and Zeller, A. (2004). Mining version histories to guide software changes. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, pages 563–572, Washington, DC, USA. IEEE Computer Society.