How Interesting Are Suggestions of Coupled File Changes for Software Developers?

Jasmin Ramadani(✉) and Stefan Wagner

Institute of Software Technology, University of Stuttgart,
Universitätstraße 38, Stuttgart, Germany
{jasmin.ramadani,stefan.wagner}@informatik.uni-stuttgart.de
http://www.iste.uni-stuttgart.de/en/se.html

Abstract. Software repositories represent a data source from which we can extract interesting information to be presented to the developers working on their maintenance tasks. Various studies use the software repositories to extract sets of files that changed frequently in the past. However, they do not consider feedback from developers on whether they would like to use this kind of information. The aim of our research is to support developers in maintenance tasks using suggestions which other files they should also change. We investigate three software repositories to find coupled file changes to support the software developers. We also propose a set of attributes from the versioning system, the issue tracking system and the project documentation. We contrast our findings with the feedback gathered using survey and interviews with the developers. According to our results, small repositories make an insightful analysis difficult. Both from experienced and inexperienced developers, the feedback was mostly neutral. Most of the attributes we proposed were accepted as interesting by the developers. Furthermore, developers also suggested other additional issues to be relevant, e.g. the context of the coupled changes. Generally, developers did not reject the coupled file changes suggestions. However, the presentation form of coupled changes and context information need to be taken into account.

Keywords: Data mining · Coupled file changes · Usefulnesses

1 Introduction

Software product development produces large amounts of data which is stored in software repositories. They contain the artifacts developed during software evolution. These repositories 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

L.A. Maciaszek and J. Filipe (Eds.): ENASE 2016, CCIS 703, pp. 201–221, 2016.
DOI: 10.1007/978-3-319-56390-9_10

software repositories. The term *mining software repositories (MSR)* describes investigations of software repositories using data mining [19].

To help the developers to identify the files to be changed during maintenance tasks, a mining software repositories approach has been proposed [33]. 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" [29]. Change couplings are described as files having the same commit time, author and modification description [12]. 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 [20,33,35]. 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.

We define *interestingness* as the subjective measure of the developers' opinion on how useful findings (here: coupled change suggestions) are for 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. The version control system, the issue tracking system and the project documentation archives are used as data sources for additional repository attributes we join 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.

This paper is an extended version of our case study on interestingness of coupled file changes [23]. This extended version adds an additional research question investigating the influence of the involvement of the developers in the project on the interestingness of coupled change suggestions. We have also extended the discussion about the research questions and the related conclusions.

2 Interestingness

Coupled changes suggestions can be provided to the developers with an intention to help them by providing suggestions about other changes. There is no guarantee that they would like to use this kind of help.

Our approach is based on determining the interestingness of these coupled changes. We consider interestingness as a subjective measure which is derived from the user's beliefs or expectations [22]. Information is defined to be *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 [11]. The interestingness of coupled changes is represented by the possibility that they will use it during their maintenance tasks. To determine the level of interestingness of the coupled changes and the repository attributes we conclude questionnaire and interviews to measure the feedback from the developers included in our case study.

We measure the interestingness using three levels: *interesting*, *neutral* and *not interesting*. Two categories of interestingness has been identified. The first category is the interestingness of coupled file changes. The second category is the interestingness of the repository attributes we extract from the version control system, the issue tracking system and the project documentation. We join this repository attributes to the coupled file changes.

3 Data Mining Background

To be able to extract 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 [15]. 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

$$0 \leq \mathrm{min}_{\mathrm{sup}} \leq |F| \tag{1}$$

4 Related Work

Many studies investigated software repositories to find logically coupled changes, e.g. [3,9,12]. We identify two granularity levels, the first one [20,33] investigates the couplings based on the file level, the second [9,19,34,35] identifies coplings between parts of files like classes, methods or modules.

Most of the studies dealing with identifying coupled changes use some kind of data mining for this purpose [13,18,20,27,31,33,35]. Especially the association rules technique is often used to identify frequent changes [20,33,35]. This data mining technique uses various algorithms to determine the frequency of these changes. Most of them employ the Apriori algorithm [20,35], however other algorithms like the FP-Tree algorithm are also in use [33].

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 [4,6,17]. Other studies combine more than one data source to be investigated, like a version control system and an issue tracking system [5,7,8,32] 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 [24]. 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. Here, the developers' perceptions of software couplings are investigated in [2]. 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 [33]. 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.

5 Case Study Design

The structure of our case study is based on existing guidelines [25].

[1] http://git-scm.com/.
[2] http://mercurial.selenic.com/.
[3] https://www.atlassian.com/software/jira.
[4] http://www.redmine.org/.

5.1 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 would like to use the suggested couplings.
RQ3: Does the experience of developers influences the interestingness of coupled changes? We expect that inexperienced developers would be more interested in coupled file suggestions considering their possible problems understanding the system [26]. Therefore, we investigate the developer's programming and project experience.
RQ4: Does the involvement in the project of developers influences the interestingness of coupled changes? We include both developers who were involved and those not involved in the development of the software products used in the case study. Although our goal is to support inexperienced or developers not involved in the projects, we expand the investigation on developers which were included in the software products, we want to get their feedback on the coupled changes.
RQ5: 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.
RQ6: 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.

5.2 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.

5.3 Data Collection Procedure

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

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 files changed. A commit snapshot represents the total set of modified files and directories [21]. We organize the data in a transaction form where every transaction represents a set of 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 [20]. We use a heuristic considering the file changes done by a single committer as related. We group the transactions of files committed only by a particular author. We do not relate the changes done by other committers.

Issue Tracking System. Issue tracking systems store important information 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 following issue attributes: *issue titles*, *issue descriptions* and *issue messages*. The issue tracking systems support spreadsheet export containing the considered issue attributes.

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 package 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.

Joining Collected Data. After the mining process is finished and we have identified the coupled changes, we join them with the attributes from the version control system, the issue tracker and the project documentation. In [8], 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 sub-projects. 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.

Survey and Interviews. We investigate the developers' feedback on the interestingness of coupled changes and the additional attributes by conducting a survey and performing interviews[5] with the developers.

Survey: The developers answer a list of multiple-choice questions on-line. 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 choose 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* to evaluate the interestingness of coupled changes and repository attributes.

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.

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

5.4 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.

5.5 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.

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 [1,14,16]. We use the FP-Tree-Growth algorithm to find the frequent change patterns. As opposed to the Apriori algorithm [1] which uses a bottom up generation of frequent itemset combinations, the FP-Tree algorithm uses partition and divide-and-conquer methods [14]. This algorithm is faster and more memory efficient than the Apriori algorithm used in other studies and allows frequent itemset discovery without candidate itemset generation.

Support Level: 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 [10] 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.

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

Analysis of Questionnaires and Interviews. To answer RQ2–RQ6, 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 experience. 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* [30]. 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 [30]. 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 [30] 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.

5.6 Validity Procedure

Internal Validity: We use widely known techniques and algorithms for repository mining. We extract data from a repository systems used among a high 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

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

interviewing developers. We collect the answers and compare the results related to the research questions to identify if these reflect the investigated information [25]. This way we avoid to rely on a possible lack of precision in the answers on the questionnaires by the developers concerning the interestingness.

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.

6 Results and Discussion

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.

6.1 Case Description

The cases in this study are three software projects. The first 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 is a web-based software written in Java and supplied by an industrial partner. The repository of this project contains 1,610 commits performed by 26 developers during 2 years of development. The software changes are stored in Git and the issues are tracked using JIRA.

The second project is a C# software supplied by another partner from the IT industry. The repository contains 159 commits performed by 5 developers during 1 year of development. The project used Mercurial as version control tool and Redmine for issues management.

The third project is a Java open source software which was developed at the University of Stuttgart by student developers. The repository contains 752 commits, committed by 9 developers during 1 year. It uses Git for versioning and Redmine as issue tracking system. Certain project documentation archives of the projects were available from where we extract the information about the software structure and the naming conventions.

6.2 Number of Couplings (RQ 1)

In Table 1, we summarize the analyzed information from the repositories. Referring to the first project, the data from 22 out of 26 developers was relevant for the study. For the second project, the data from 4 out of 5 developers was taken into account. For the third project, the data committed by all 9 developers was

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

Table 1. Results based on repository analysis, table reproduced from [23].

	Project1	Project2	Project3
No. of relev. dev	22	4	9
No. of commits	1610	138	752
No. of couplings	205	13	200
Freq. itemset supp	0.2	0.2	0.1

Table 2. Interestingness of coupled changes, table reproduced from [23].

	Involved	Not involved	All
Interesting	2	2	4
Neutral	9	10	19
Not interesting	0	0	0
Sum	11	12	23

suitable for analysis. The rest of the developers reported a low number of commits so we did not consider their change commits. We excluded their commits as unsuitable for the reason that they did not reach the minimum support for the frequency of the changes we defined previously.

The number of commits represents the size of the projects followed by the number of change couplings we have extracted. The number of coupled changes represents the basis of our analysis. We were able to extract 205 couplings from the first repository. From the second, a smaller repository, we report only 13 coupled changes. The third repository delivered 200 coupled changes. These results show that we need larger project repositories containing high number of commits to be able to deliver a high number of couplings.

6.3 Interestingness of Coupled Changes (RQ 2)

The participants were asked to give their feedback on how interesting coupled changes for maintenance tasks are. Most of the developers (19 of 23) reported a neutral opinion for the concept of coupled changes. A small group of four participants noted coupled changes as interesting. None of the developers rejected the idea as not interesting (Table 2).

The fact that the developers did not reject coupled changes allows us to continue our analysis. These results allow us to continue investigating the next research questions. We proceed our analysis and investigate how coupled changes is influenced by the developers' programming and project experience. Taking into account our small sample size, we refrain from formal hypotheses testing.

6.4 Influence of Developer Experience on Interestingness (RQ 3)

Both experienced and inexperienced developers were similarly interested in coupled changes which is in contrast to our expectations. In Table 3 we present the distribution of the interestingness of coupled changes in relation to the programming experience of the developers. What we can see is that regardless of their expertise level, none of the developers rejected the coupled changes. Very few developers have accepted the coupled changes as interesting, yet most of the developers took a neutral position toward the coupled change suggestions.

6.5 Influence of Developer Involvement in the Project on Interestingness (RQ 4)

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. Continuing with the developers involved in the project development, we group their answers based on their project experience. Table 4 shows the distribution of the developers by their programming experience. Again in all three groups from beginners to developers knowing the system, most of them have answered neutrally, not rejecting the coupled change suggestions.

6.6 Interestingness of Additional Information (RQ 5)

After the investigation of the coupled changes, we continued examining the interestingness of the repository attributes we have joined to the coupled files presented in Table 5. To support the coupled changes, we reported a set of common meta-data attributes [28] which allow us to find more information about the commits, the issues and the product itself. The repositories offer various attributes related to the committed changes, the issues found and the project structure.

Table 3. Couplings and developer's experience, table adapted from [23].

Programming experience	Freq	Freq. [%]	Interesting	Neutral	Not interesting
<1 year	2	9	0	2	0
1–3 years	4	17	2	2	0
3–5 years	9	39	1	8	0
>5 years	8	35	1	7	0

Table 4. Couplings and developer's project involvement.

Project involvement	Freq	Freq. [%]	Interesting	Neutral	Not interesting
<6 months–1 year	3	27	0	3	0
1–2 years	3	27	1	2	0
>2 years	5	46	1	4	0

Table 5. Interesting attributes, table reproduced from [23].

Attribute	Frequency	Frequency [%]
Commit message	22	95
File name	18	78
File type	9	39
Commit time	8	34
Commiter	6	26
Commit id	2	9
Issue title	21	91
Issue status	15	65
Issue type	14	60
Issue time	6	26
Project structure	20	86
Naming conv	15	65

We asked the participants about their feedback on the interestingness of each of the provided repository attributes. The results show that most of the offered attributes were rated by the developers as interesting.

Considering the commit related attributes, most of the developers found the commit message to be the most interesting attribute followed by the file name. The developers did not show much interest for the commit time, the committer and the file type. The commit id as attribute did not attract the developers' interest.

Regarding the issue related attributes, most of the developers were interested in the issue description. Some of the developers also found the issue status and type to be interesting. The issue time was not interesting for the developers.

From the documentation related attributes, the developers reported that both naming convention and the project structure information are interesting.

6.7 Influence of Developer Experience on Interestingness of Additional Information (RQ 6)

We examined the distribution of interestingness of the repository attributes according to developers' experience level. Based on this distribution we created two general groups of developers in this context: the first group called experienced, includes the developers having more than 5 years experience and the second group called inexperienced, includes developers having less than 5 years of experience. The results show that the experienced developers have a more clear picture of the set of interesting repository attributes. They have chosen a lower number of attributes compared to the inexperienced developers. The inexperienced developers have marked various commit and issue attributes being interesting for them. The more experienced developers' choice is more narrow

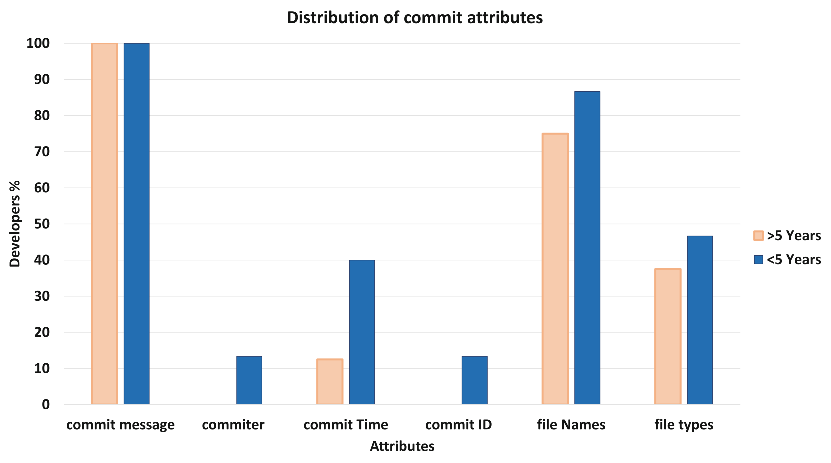

Fig. 1. Commit attributes and experience, figure reproduced from [23].

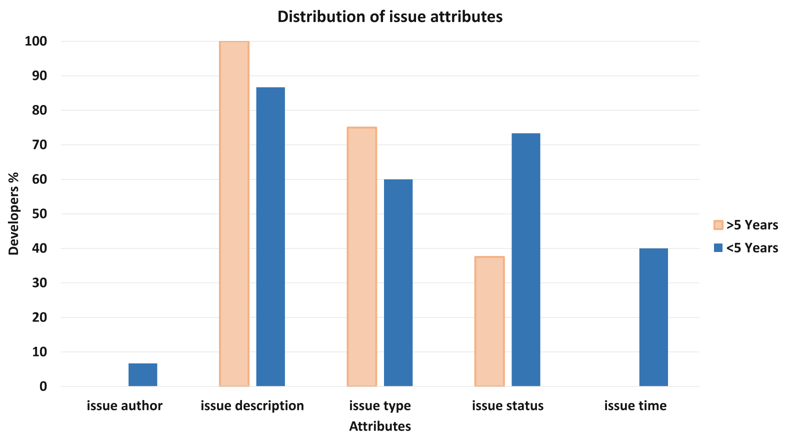

Fig. 2. Issue attributes and experience, figure reproduced from [23].

than the one for the inexperienced ones. The distribution of commit attributes is shown in Fig. 1. The distribution of issue attributes is presented in Fig. 2.

6.8 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 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.

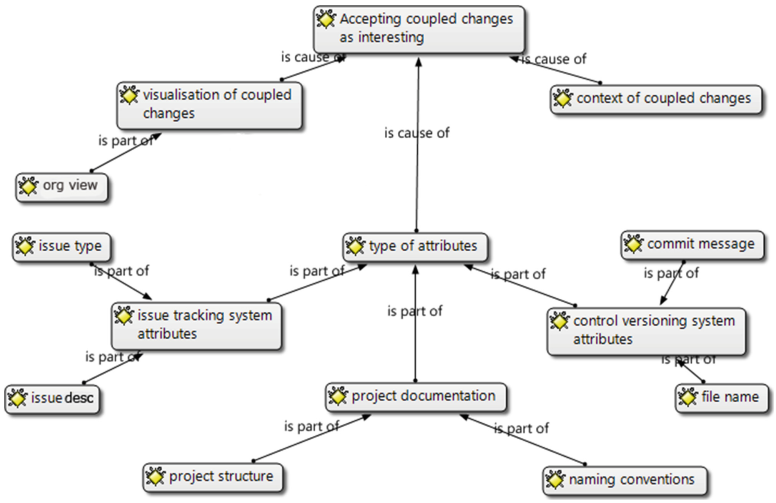

Fig. 3. Theoretical Framework, figure reproduced from [23].

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 Fig. 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 Fig. 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 Fig. 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.

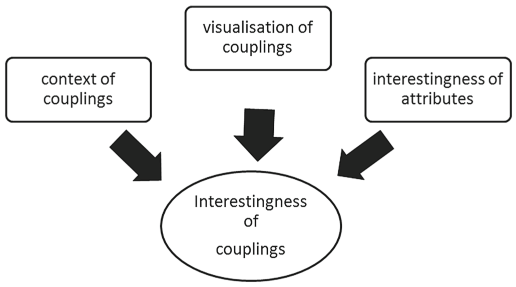
visualisation of couplings
context of couplings
interestingness of attributes
Interestingness of couplings

Fig. 4. Conceptual Model from grounded theory, figure reproduced from [23].

6.9 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 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, the setup of our analysis identifies a number of strongly coupled changes which limits the possibility they have happened by chance. We could reduce the support level of the data mining algorithm to provide a higher number of coupled changes, however, this could produce a threat for their accuracy.

The results for RQ2 report that the developers weakly support that coupled changes are interesting. The general concept of coupled changes was received mostly as neutral. The developers did not judged the coupled change suggestions very positive for the reason that they were not solving real maintenance tasks. We beleive that working with coupled change suggestions related to real maintenance tasks would increase the acceptance of coupled file changes.

The fact that none of the developers rejected the coupled changes, gave 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, actual examples of coupled changes were presented to the developers which increased their acceptance.

Considering RQ3, we expected that the coupled changes would be interesting for developers having a lack of programming experience. Our results at contrary show that also the experienced developers are similarly interested in coupled changes. The developers higher experience does not eliminate the possibility that the coupled suggestions could be helpful when working on an unknown source code, software structure or on an older project. The fact they did not reject the coupled changes reports that the benefit from them is not limited on novice developers which makes the coupled changes attractive for a broader audience.

About the results related to RQ4, both uninvolved developers in the project development of the investigated software products and those who worked on the projects provided a neutral feedback. They fact that they did not reject the coupled changes increases the target group for our coupled changes suggestions. These unexpected results show that also the developers working on a particular part of the source code could use some help when working on other parts of the system. These findings encourages us to include the coupled changes as a part of an integrated tool support for developers.

Answering RQ5, our results show that most of the attributes from the provided set were interesting for the developers. These results were also validated by the interviews. Using the commits, the questionnaire and the interviews, we reported that the commit message and the file names are the most interesting attributes. This shows that the developers found the information about the files being changed and the description of these changes to be interesting.

For the issue attributes, the developers reported that the issue description and the issue type are interesting, meaning that they were looking for the information which describes the problem to be solved and the importance of the issue.

For the documentation attributes, the project structure and the naming convention were both interesting for the developers. This shows that they were looking for the information that could help them to find the location in the system to begin with their source code changes.

We reported a set of repository attributes used by well known versioning and issue tracking systems involved in the 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 graphical representation of the coupled changes. 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 RQ6 show that experienced developers know well what kind of repository attributes they want to see. Their choice is more precise compared to the inexperienced developers. The inexperienced developers did not have a clear picture which attributes to choose from the provided set. The fact that developers with different programming experience considered various attributes to be interesting brings us to the conclusion that we should not make a fixed choice of attributes for all developers. We can offer a flexible way for the developers to choose the attributes individually. This way, we support the developers which are not experienced and would like to have an overview of the provided set of repository attributes. On the other side we would like to offer the experienced developers to hide the unnecessary information including the not interesting attributes during maintenance tasks.

The results of the grounded theory show that the interestingness of coupled file changes is influenced by their presentation form and the related information such as the description of the change context. Providing a good visual concept is inevitable for a successful visual representation. Also the repository attributes influence their interestingness. Choosing wrong or not useful attributes can drop the acceptance of coupled change suggestions.

6.10 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 used 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.

7 Conclusion and Future Work

7.1 Summary of Conclusions

The study results show that smaller software repositories do not provide a meaningful number of coupled file changes.

The feedback of developers on the interestingness of coupled changes is mostly neutral. Our results lead to the conclusion that the couplings were weakly accepted by developers having various programming experience and level of involvement in the project. Working on real maintenance tasks would increase the acceptance of coupled change suggestions.

The developers accepted most of the proposed software repository attributes joined to the couplings as interesting. Experienced developers report a narrower choice of attributes as opposed to the inexperienced developers.

The Grounded Theory shows that the our set of repository attributes influence the interestingness of coupled changes. Although we provided a number of repository attributes, the developers suggested additional aspects concerning the coupled change suggestions and the repository attributes. They would like to see more information about the change context and the visual presentation of the coupled changes. We conclude that the we have to develop a visualization concept for the coupled change suggestions and provide the possibility that the developers can individually adjust their choice of repository attributes.

7.2 Relation to Existing Evidence

Revelle et al. [24] investigated source code features coupling using structured and textual features. Here the developers are surveyed to determine if the metrics align with the developers' opinion. Their results show that the developers support the proposed coupling metrics. Our results show that the developers weakly accept the concept of coupled changes and the corresponding attributes from the repository.

Ying et al. [33] investigated the interestingness of coupled changes, whereby the authors used open-source projects and categorized 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.

7.3 Impact/Implications

This case study gives evidence that the coupled file changes are interesting to the developers during maintenance tasks. Yet, the interest is rather weak overall. Therefore, other contextual information should be investigated in future research to increase the interestingness. Using proper visualization, coupled file suggestions could be incorporated in a tool to support the developers during maintenance tasks.

7.4 Limitations

As it is case study research, we cannot guarantee the generalizability of the study. The data comes from two commercial and one open-source project. However, the procedure should be similar for other projects for the reason that we use well defined data mining techniques and commonly used repositories as data sources.

The number of coupled changes we found is limited by the support value of the frequent itemsets algorithm. Our results preserve relative small number of the most frequent and most valid couplings. The size of our sample limits the possibility for a deeper statistical analysis. Yet, our findings constitute a first insight into developers' opinions on coupled file changes.

7.5 Future Work

The next step is to perform an experiment to investigate coupled changes by directly observing their use for a real maintenance tasks. This could be visualized in 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 using additional context description.

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

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proceedings of the 20th International Conference on Very Large Data Bases, VLDB 1994, pp. 487–499 (1994)
2. Bavota, G., Dit, B., Oliveto, R., Di Penta, M., Poshyvanyk, D., De Lucia, A.: An empirical study on the developers perception of software coupling. In: Proceedings of the 2013 International Conference on Software Engineering, ICSE 2013, pp. 692–701 (2013)
3. Bieman, J., Andrews, A., Yang, H.: Understanding change-proneness in OO software through visualization. In: 11th IEEE International Workshop on Program Comprehension, pp. 44–53, May 2003
4. Bird, C., Rigby, P.C., Barr, E.T., Hamilton, D.J., Germán, D.M., Devanbu, P.T.: The promises and perils of mining git. In: MSR, pp. 1–10 (2009)
5. Canfora, G., Cerulo, L.: Impact analysis by mining software and change request repositories. In: 11th IEEE International Symposium on Software Metrics, p. 29, September 2005
6. Carlsson, E.: Mining git repositories: an introduction to repository mining (2013)
7. D'Ambros, M., Lanza, M., Robbes, R.: On the relationship between change coupling and software defects. In: WCRE, pp. 135–144 (2009)
8. Fischer, M., Pinzger, M., Gall, H.: Populating a release history database from version control and bug tracking systems. In: Proceedings of the International Conference on Software Maintenance, ICSM 2003, p. 23 (2003)
9. Fluri, B., Gall, H., Pinzger, M.: Fine-grained analysis of change couplings. In: Fifth IEEE International Workshop on Source Code Analysis and Manipulation, pp. 66–74, September 2005
10. Fournier-Viger, P.: How to auto-adjust the minimum support threshold according to the data size (2013). http://data-mining.philippe-fournier-viger.com/
11. Frawley, W.J., Piatetsky-shapiro, G., Matheus, C.J.: Knowledge discovery in databases: an overview (1992)
12. Gall, H., Jazayeri, M., Krajewski, J.: CVS release history data for detecting logical couplings. In: Proceedings of Sixth International Workshop on Principles of Software Evolution, pp. 13–23, September 2003
13. German, D.M.: Mining CVS repositories, the softchange experience. In: 1st International Workshop on Mining Software Repositories, pp. 17–21 (2004)
14. Győrödi, C., Győrödi, R.: A comparative study of association rules mining algorithms (2004)
15. Han, J., Mining, D.: Concepts and Techniques. Morgan Kaufmann Publishers Inc., San Francisco (2005)
16. Han, J., Pei, J., Yin, Y., Mao, R.: Mining frequent patterns without candidate generation: a frequent-pattern tree approach. Data Min. Knowl. Discov. **8**(1), 53–87 (2004)
17. Hassan, A.E., Holt, R.C.: Predicting change propagation in software systems. In: Proceedings of the 20th IEEE International Conference on Software Maintenance, ICSM 2004, pp. 284–293 (2004)
18. Hattori, L., dos Santos Jr., G., Cardoso, F., Sampaio, M.: Mining software repositories for software change impact analysis: a case study. In: Proceedings of the 23rd Brazilian Symposium on Databases, SBBD 2008, pp. 210–223 (2008)
19. Kagdi, H., Collard, M.L., Maletic, J.I.: A survey and taxonomy of approaches for mining software repositories in the context of software evolution. J. Softw. Maint. Evol. **19**(2), 77–131 (2007)

20. Kagdi, H., Yusuf, S., Maletic, J.I.: Mining sequences of changed-files from version histories. In: Proceedings of the 2006 International Workshop on Mining Software Repositories, MSR 2006, pp. 47–53 (2006)
21. Loeliger, J.: Version Control with Git - Powerful Techniques for Centralized and Distributed Project Management. O'Reilly, New York (2009)
22. McGarry, K.: A survey of interestingness measures for knowledge discovery. Knowl. Eng. Rev. **20**(1), 39–61 (2005)
23. Ramadani, J., Wagner, S.: Are suggestions of coupled file changes interesting? In: Proceedings of the 11th International Conference on Evaluation of Novel Software Approaches to Software Engineering, pp. 15–26 (2016)
24. Revelle, M., Gethers, M., Poshyvanyk, D.: Using structural and textual information to capture feature coupling in object-oriented software. Empirical Softw. Engg. **16**(6), 773–811 (2011)
25. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Softw. Engg. **14**(2), 131–164 (2009)
26. Sayles, J., et al.: z/OS Traditional Application Maintenance and Support. IBM Redbooks (2011)
27. Shirabad, J., Lethbridge, T., Matwin, S.: Mining the maintenance history of a legacy software system. In: Proceedings of International Conference on Software Maintenance, ICSM 2003, pp. 95–104, September 2003
28. Steven, J., Zach, W.: Bad commit smells (2013). http://pages.cs.wisc.edu/~sjj/docs/commits.pdf
29. Stevens, W.P., Myers, G.J., Constantine, L.L.: Structured design. IBM Syst. J. **13**(2), 115–139 (1974)
30. Strauss, A., Corbin, J.M.: Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory. SAGE Publications, USA (1998)
31. van Rysselberghe, F., Demeyer, S.: Mining version control systems for FACs (frequently applied changes). In: the International Workshop on Mining Repositories, Edinburgh, Scotland, UK (2004)
32. Wu, R., Zhang, H., Kim, S., Cheung, S.-C.: 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 2011, pp. 15–25 (2011)
33. Ying, A.T.T., Murphy, G.C., Ng, R.T., Chu-Carroll, M.: Predicting source code changes by mining change history. IEEE Trans. Softw. Eng. **30**(9), 574–586 (2004)
34. Zimmermann, T., Kim, S., Zeller, A., Whitehead, Jr., E.J.: Mining version archives for co-changed lines. In: Proceedings of the 2006 International Workshop on Mining Software Repositories, MSR 2006, pp. 72–75 (2006)
35. Zimmermann, T., Weisgerber, P., Diehl, S., Zeller, A.: Mining version histories to guide software changes. In: Proceedings of the 26th International Conference on Software Engineering, ICSE 2004, pp. 563–572 (2004)