Are Coupled File Changes Suggestions Useful?

Jasmin Ramadani* and Stefan Wagner

Institute of Software Technology, University of Stuttgart, Germany

Abstract

Background. Software maintenance is an important activity in the process of software engineering where over time maintenance team members leave and new members join. The identification of files being changes together frequently has been proposed several times. Yet, existing studies about these file changes ignore the feedback from developers as well as the impact on the performance of maintenance and rely on the analysis findings and expert evaluation.

Methods. We conducted an experiment with the goal to investigate the usefulness of coupled file changes during maintenance tasks when developers are inexperienced in programming or when they are new on the project. Using data mining on software repositories we can identify files that changed most frequently together in the past. We extract coupled file changes from the Git repository of a Java software system and join them with corresponding attributes from the versioning and issue tracking system and the project documentation. We present a controlled experiment involving 36 student participants where we investigate if coupled file change suggestions influence the correctness of the task solutions and the time to complete them.

Results. The results show that coupled file change suggestions significantly increase the correctness of the solutions. However, there is only a small effect on the time to complete the tasks. We also derived a set of the most useful attributes based on the developers feedback.

Discussion. Coupled file changes and a limited number of the proposed attributes are useful for inexperienced developers working on maintenance tasks whereby although the developers using these suggestions solved more tasks, they still need time to organize and understand and implement this information.

*Corr. author: Jasmin Ramadani, Universitätsstr. 38, 70569 Stuttgart, Germany, phone +49 711 685 884306, jasmin.ramadani@informatik.uni-stuttgart.de

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2492v1 | CC BY 4.0 Open Access | rec: 30 Sep 2016, publ: 30 Sep 2016

1 Introduction

Software maintenance represents a very important part in software product development (Abran & Nguyenkim, 1991). Maintenance is often performed by maintenance programmers. Over time teams change when members leave and others join (Hutton & Welland, 2007). The members cannot be immediately productively included to solve maintenance tasks, so they need some support to successfully perform their tasks.

Software development produces large amounts of data which is stored in software repositories. These repositories contain the artifacts developed during software evolution. After some time, this data becomes a valuable information source for solving maintenance tasks.

One of the most used techniques for analyzing software repositories is data mining. The term *mining software repositories (MSR)* describes investigations of software repositories using data mining (Kagdi et al., 2007).

Couplings have been 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 also described as files having the same commit time, author and modification description (Gall et al., 2003). Knowing, which files were frequently changed together 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 significant experience in software development.

1.1 Problem Statement

Several researchers have proposed approaches to identify coupled file changes to give recommendations to developers (Bavota et al., 2013; Kagdi et al., 2006; Ying et al., 2004; Zimmermann et al., 2004). Existing studies, however, focus on the presentation of the mining results and expert investigations, and neglect the feedback of developers on the findings as well as the impact on the performance on maintenance tasks.

1.2 Research Objectives

The overall aim of our research is to investigate the usefulness of coupled file change suggestions in supporting developers working which are inexperienced, new on the projects or work on unfamiliar parts of the project. We provide suggestions for likely changes so that we can explore how useful the suggestions are for the developers.

We identify frequent couplings between file changes based on the information gathered from the software project repository. We use the version control system, the issue tracking system and the project documentation archives as data sources for additional attributes. We join these additional information to the coupled changes we discover.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2492v1 | CC BY 4.0 Open Access | rec: 30 Sep 2016, publ: 30 Sep 2016

The usefulness of coupled file changes is defined by analyzing their influence on the correctness of the solutions and the effort for solving maintenance tasks.

1.3 Contribution

We present a controlled experiment on the usefulness of coupled change suggestions where each of the 36 participants try to solve 4 different maintenance tasks and report their feedback on the usefulness of the repository attributes.

2 Related Work

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

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

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

To the best of our knowledge, there are few 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 were 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

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2492v1 | CC BY 4.0 Open Access | rec: 30 Sep 2016, publ: 30 Sep 2016

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 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. In Ramadani & Wagner (2016), the feedback on the interestingness of coupled file changes and the attributes from the software repository have been investigated. In our experiment we extend the findings of this case study and investigate the usefulness of coupled file changes and the corresponding attributes.

Various experiments involving maintenance tasks have been described in the literature. Nguyen et al. (2011) deal with assessing and estimating of software maintenance tasks. De Lucia et al. (2002) investigate the effort estimation for corrective software maintenance. Ricca et al. (2012) perform an experiment on maintenance in the context of model driven development. Chan (2008), investigate the impact of programming and application specific knowledge on maintenance effort. In our experiment, we investigate how the coupled file changes suggestions influence the correctness of performing maintenance tasks and the time effort needed to solve the tasks.

3 Background

3.1 Software Maintenance

Software maintenance includes program or documentation changes to make the software system perform correctly or more efficiently (Shelly et al., 1998). Software maintenance has been defined in the IEEE 1219 Standard for Software Maintenance (IEEE, 1998) to be a software product modification after delivery to remove faults, improve performance or adapt the environment. In the ISO/IEC 12207 Life Cycle Processes Standard (ISO/IEC, 1995), the maintenance is described as the process where the software code and documentation modification is performed due to some problem or improvement.

3.1.1 Maintenance Categories

Swanson (1976) defined three different categories of maintenance: corrective, adaptive and perfective. The ISO/IEC 14764 International Standard for Software Maintenance (ISO/IEC, 2000) updates this list with a fourth category, the preventive maintenance so we have the following maintenance categories (Pigoski, 1996):

- Corrective Maintenance: This type of maintenance tasks includes correction of errors in systems. Here, software product modifications are performed after delivery to correct the discovered problems. It corrects

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2492v1 | CC BY 4.0 Open Access | rec: 30 Sep 2016, publ: 30 Sep 2016

design, source code and implementation errors.

- Adaptive Maintenance: It satisfies the changes in the environment and includes adding of new features or functions to the system. Software product modification are performed to ensure software product usability in changed environment.

- Perfective Maintenance: It involves changes in the system which influence its efficiency. Also it includes an software product modification after delivery to improve maintainability or performance.

- Preventive Maintenance: Here, the changes in the system have been perform to reduce the possibility of system failures in the future. It includes software product modification after delivery to detect and remove failures before they become effective.

3.2 Data Mining

3.2.1 Coupled File Changes

To be able to discover coupled file changes 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 item sets. 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 item set. An important property of an item set is the support count δ which is the number of transactions containing an item. We call the item sets frequent if they have a support threshold min_{sup} greater than a minimum specified by the user with

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

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2492v1 | CC BY 4.0 Open Access | rec: 30 Sep 2016, publ: 30 Sep 2016

3.2.2 Data Mining Algorithm

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

3.2.3 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 Experimental Design

In this section we define the research questions, hypotheses and metrics used in our analysis.

4.1 Study Goal

We use the GQM approach (Basili et al., 1994) and its MEDEA extension (Briand et al., 2002) to define the study goal. The *goal* of the study is analyzing the usefulness of coupled file change suggestions. The *objective* is to compare the correctness of the solution and the time needed for a set of maintenance tasks between the group using coupled change suggestions and the group which does not use this kind of help. The *purpose* is to evaluate how effective are coupled file change suggestions regarding the correctness of the modified source code and the time required to perform the maintenance tasks. The *viewpoint* is from a software developers and the targeted environment is open source systems.

4.2 Research Questions

We investigate the usefulness of coupled file change suggestions and the joined attributed from the software repository. For that purpose we define the following research questions:

RQ1: How useful are coupled file change suggestions in solving maintenance tasks?
This research question needs to be answered to define the usefulness of the coupled file changes concept. We investigate if the coupled file change suggestions

influence the correctness of the maintenance tasks and how fast these tasks have been accomplished.

RQ2: How useful are the attributes from the software repository in solving maintenance tasks?
The second research question deals with the attributes from the versioning system, the issue tracking system and the documentation. We investigate the perceived usefulness of each attribute in the proposed attribute set to understand which attributes are good candidates to be provided to the developers.

4.3 Hypotheses

We formulate the following hypotheses to answer the research questions in our study. For **RQ1** we define the following hypotheses:

$\mathbf{H_{0.1.1}}$: There is no significant difference in the correctness of maintenance tasks solutions between the developers which used coupled file change suggestions and the developers not using these suggestions.
$\mathbf{H_{A.1.1}}$: There is a significant difference in the correctness of maintenance tasks between the developers which used coupled file change suggestions and the one not using these suggestions.
$\mathbf{H_{0.1.2}}$: There is no significant difference in the time to solve maintenance tasks between the developers which used coupled file change suggestions and the developers not using these suggestions.
$\mathbf{H_{A.1.2}}$: There is a significant difference in the time to solve maintenance tasks between the developers which used coupled file change suggestions and the one not using these suggestions.

To answer **RQ2** we formulate the following hypotheses:

$\mathbf{H_{0.2}}$: There is no significant difference in the perceived usefulness among the attributes from the software repository in the current set of attributes.
$\mathbf{H_{A.2}}$: There is a significant difference in the perceived usefulness among the attributes from the software repository in the current set of attributes.

4.4 Experiment Variables

We have defined the following dependent variables: the correctness of solution after the execution of the maintenance task, the time spent to perform the maintenance task and the usefulness of the repository attributes. For the first variable, the correctness of the task solution, we assign scores to each developer solution of the maintenance tasks.

Our approach is similar to the one presented by Ricca et al. (2012) where the correctness of the solution for the maintenance task is manually assessed by defining scores from totally incorrect to completely correct task solution. We

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2492v1 | CC BY 4.0 Open Access | rec: 30 Sep 2016, publ: 30 Sep 2016

define three scores: 0, if the developers did not execute or did not solve the task at all, 1 if the task was partially solved and 2 if the developer performed a complete solution of the maintenance task. The solutions are tested using unit tests to ensure the correctness of the edited source code.

The second variable, the time for executing the maintenance tasks is measured by examining the screen recordings. We mark the start time and the end time for every task. We calculate the difference to compute the total time needed to solve each task.

For the third variable, the usefulness of the repository attributes, we use an ordinal scale to identify the feedback of the developers. The participants can choose between the following options for each attribute: very useful, somehow useful, neutral, not particularly useful and not useful. We code the usefulness feedback using the scoring presented in Table 1.

Table 1: Usefulness score

very useful	somehow useful	neutral	not particularly useful	not useful
5	4	3	2	1

4.5 Experiment Design

We distinguish two cases for the maintenance tasks: the first one includes tasks executed on Java Code in the Eclipse IDE without any suggestions and the second one includes tasks executed with additional coupled files suggestions and corresponding attributes from the repositories. We use a similar approach to the one presented by Ricca et al. (2012) and define two values: $-$ for Eclipse only and $+$ for the coupled file suggestions.

We use a counterbalanced experiment design as described in Table 2. This ensures that all subjects work with both treatments: without and with coupled change suggestions. We split the subjects randomly in two groups working in two lab sessions of two hours each. In each session, the participants work on two tasks only with the task description and on two tasks where they receive the coupled file changes suggestions and the related attributes. The participants in the second lab have swapped the order of the tasks used during the first lab.

Table 2: Experiment Design

Lab	Tasks	
Lab 1	Tasks 1-2 (–)	Tasks 3-4 (+)
Lab 2	Tasks 1-2 (+)	Tasks 3-4 (–)

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2492v1 | CC BY 4.0 Open Access | rec: 30 Sep 2016, publ: 30 Sep 2016

4.6 Objects

The object of the study is an open source Java software called ASTPA. The source code and the repository were downloaded from SourceForge.[1] The system is built mainly in Java by 12 developers at the University of Stuttgart during a software project between year 2013 and 2014. It represents an Eclipse based tool for hazard analysis.

4.7 Subjects

The experiment participants are 36 students from the Software Engineering course in their second semester at the University of Stuttgart (Germany). The students have basic Java programming and Eclipse knowledge and have not been related in any way with the software system investigated in the experiment.

4.8 Material, Procedure and Environment

All subjects received the following materials which can be found in the supplemental material of this paper.

- Tools and code: The participants received the Eclipse IDE to work with, the screen capturing tool and the source code they need to edit.

- Questionnaires: The first questionnaire is performed at the start of the experiment and it is related to their programming background. The second questionnaire performed at the end of the experiment is about their feedback on the usefulness of coupled changes and the additional set of repository attributes.

- Software documentation: We have provided the technical documentation for the software system including the data model and package descriptions.

- Setup instructions: The participants received the instruction steps how to prepare the environment, where to find the IDE, the source code and and how to perform the experiment.

- Maintenance tasks and description: Every participant received spreadsheets with four maintenance tasks and their free-text description.

- Coupled file changes: The files changed together frequently used to solve a similar tasks have been provided to the group which uses coupled file changes. These sets of file suggestions do not represent the solutions for a

[1] `https://sourceforge.net/projects/astpa/`

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2492v1 | CC BY 4.0 Open Access | rec: 30 Sep 2016, publ: 30 Sep 2016

particular task in the experiment and can contain more or less files than needed to solve the particular task.

- Repository Attributes: The attribute set from the versioning system, the issue tracking system and the documentation about similar tasks performed in the system.

The environment for the experiment tasks was Eclipse IDE on a Windows PC in both treatments. For each lab, we prepared an Eclipse project containing the Java source code of the ASTPA system. The project materials were made available to the subjects on a flash drive. The participants had a maximum of two hours to fill the questionnaires and perform the maintenance tasks.

4.9 Maintenance Tasks

The maintenance tasks represent quick program fixes that should be performed by the participants according to the maintenance requests (Basili, 1990). All four maintenance tasks are perfective and have been assigned to the participant groups in both groups. The tasks require the participants to add various enhancements to the system whereby the changes do not influence the structure or the functionalities of the application. The tasks are related to simple changes of the user interface of the system.

4.10 Maintenance Activities

After receiving the task description, the participants investigate the source code of the application, identify the files where the change is needed and perform the change according to the requirement. The scenario for solving the provided maintenance tasks includes the following activities (Nguyen et al., 2011):

- Task understanding: First of all, the participants need to read the task description and the instructions and prepare for the changes. They can ask if they need some clarification around the settings and the instructions.

- Change specification: During this step, the participants locate the source code they need to change, try to understand and specify the code change.

- Change design: This step includes the performing of the already specified source code changes and debugging the affected source code.

- Change test: To specify the successfulness of the performed code changes, a unit test needs to be performed. This step is performed by the experiment organizers after the lab sessions.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2492v1 | CC BY 4.0 Open Access | rec: 30 Sep 2016, publ: 30 Sep 2016

4.11 Data Collection Procedure

We collect data from several sources: the software repository of the system, the questionnaires, the provided task solutions and the screen capturing recordings.

4.11.1 Software Repositories

- Version Control System: The first data source we use is the log data from the version control system. The investigated project uses Git as a control management tool. It is an distributed versioning system allowing the developers to maintain their local versions of source code. 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 change set 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. From this data source we extract the coupled file changes and the commit related attributes.

Table 3: Repository Attributes Description

Attribute Name	Attribute Description
Commit ID	Unique ID of Git commit
Commit Message	Free-text comment of the commit in Git
Commit Time	Time-stamp of committed change in Git
Commit Author	Person which executed the commit
Issue Description	Free-text comment of issue to be solved
Issue Type	Type of the issue: bug, feature
Issue Author	Person who created the issue to be solved
Package Description	Free-text description of the package: layer, feature

- Issue Tracking System: In issue tracking systems, important information is stored about the software changes or problems. In our case, the developers used JIRA as issue tracking systems. The issue tracking systems data source is used to extract the issue related attributes.

- Project Documentation: The software documentation gathered during the development process represents a rich source of data. The documentation contains the data model and code descriptions. From these documents, we discover the project structure. For example in the investigated project, the package containing the files described by the following path: `astpa/controlstructure/figure/`, contains the Java classes responsible for the control diagram figures of this software. We use the documentation to identify the package description.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2492v1 | CC BY 4.0 Open Access | rec: 30 Sep 2016, publ: 30 Sep 2016

The complete set of attributes we extract from the software repository are presented in Table 3.

4.11.2 Questionnaire

The developers answer a number of multiple-choice questions. Using the first questionnaire, we investigate the developers' programming background. We use a second questionnaire after the tasks being solved in order to gather the feedback on the usefulness of coupled changes and the additional attributes[2].

4.11.3 Tasks completion

Similarly to other studies (Chan, 2008; Nguyen et al., 2011; Ricca et al., 2012), we define two factors which represent the completion of the maintenance tasks:

- Correctness of solution: We determine the correctness of the solution by examining the changed source code if the solution satisfies the change requirements. We use the scoring presented previously where we summarize the points each developer gathers for every of the four tasks. The score is added next to each of the participant for both treatments, with and without using coupled file changes.

- Time of task completion: It represents the total time in minutes spent solving the maintenance tasks. We use a screen capturing device to record the time for each participant that spend solving each of the four tasks. We record the time needed for each tasks in both treatments.

4.12 Data Analysis Procedure

To be able to test our hypotheses, we need to analyze the usefulness of the coupled file changes and the usefulness of the attributes from the software repository. We perform the analysis using SPSS statistical software.

4.12.1 Usefulness of Coupled File Changes

The main part of the analysis is the investigation of the usefulness of the coupled changes. For this purpose we compare the scores of each task solution and the amount of time needed for solving the tasks in both groups: without using coupled file suggestions and with using of coupled file suggestions. For the time needed for the solution, we use only the values for the accomplished tasks only. This way we assure that the values for the unsolved tasks do not corrupt the overall values for the time needed to successfully solve the tasks.

To achieve this, we test the overall difference in the correctness of solving the tasks using the two-tailed Mann-Whitney U test. It is used to test hypotheses where two samples from same population have the same medians or that one of

[2]The questionnaires are available in the supplemental material of this paper

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2492v1 | CC BY 4.0 Open Access | rec: 30 Sep 2016, publ: 30 Sep 2016

them have larger values so we test the statistical significance of difference between two value sets. Determining an appropriate significance threshold defines if the null hypothesis must be rejected or not (Nachar, 2008). If the p-value is small, the null hypothesis can be rejected meaning that the value sets are different. If the p-value is large, the values do not differ. Usually a 0.05 level of significance is used as threshold. The p-value is not enough to determine the strength of the relationship between variables. For that purpose we report the effect size estimate (Tomczak & Tomczak, 2014).

We use an conservative approach where we test the difference in the correctness of our tasks. Without differentiating the tasks, we compare all the solutions of the tasks using coupled file changes and the tasks performed without any suggestion. We repeat the same approach to test the overall difference between the time needed to solve the tasks using coupled change suggestions against the tasks solved without the help of coupled file changes.

We use the SPSS statistical software and its typical output for the Mann-Whitney U Test whereby the p-value of the statistical significance in the difference between the two groups is reported. The mean ranking determines how each group scored in the test. To support statistical difference between the samples, we calculate the r-value of the effect size proposed by (Cohen, 1977) using the z value from the SPSS output (Fritz et al., 2012). A value of 0.5 determines a large effect, 0.3 medium and 0.1 small effect (Coolican & Taylor, 2013).

4.12.2 Usefulness of Attributes

We analyze the feedback from the questionnaire investigating which attributes are useful. We investigate every attribute in the set extracted from the versioning system, the issue tracking system and the documentation as previously presented. For that purpose we use the Kruskal-Wallis H test, an extension of the Mann-Whitney U test. Using this test, we determine if there are statistically significant differences between the medians of more than two independent groups. We test the statistical significance between more than two value sets. The significance level determines if we can reject the null hypothesis. p-values bellow 0.05 it means that there is a significant difference between the groups (Pohlert, 2014). The effect size for the Kruskal-Wallis H test, we calculate the effect sizes for the pairwise Mann Whitney U test for each of the attributes using the z statistic. We individually calculate the effect size value r for each pair comparison. The r value is calculated using the following formula:

$$r = \frac{z}{\sqrt{N}} \tag{2}$$

Our approach tests the differences in the feedback about the usefulness between all the attributes for all 36 participants. This way we identify which attributes we should offer to the participants when solving their tasks together with the coupled file changes suggestions.

Using SPSS, we provide the statistical significance values of the difference between all eight attributes. The ranking of the means for the feedback on the usefulness values determine the most useful attributes.

4.13 Execution Procedure

- Log Extraction: We extract the information from the Git log containing the committed file changes and the attributes. The log data is exported as text file and the output is managed using proper log commands.

- 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.

- Support threshold: To be able to begin the investigation, we need to extract coupled file changes from the software repository. We extract the coupled changes by defining the threshold value of the support for the frequent item set 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 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 item sets. 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 item sets with a support below 0.2 meaning the coupled changes should have been found in 20 percent of the commits.

- Mining Framework: 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 item sets and association rules called the *SPMF-Framework*.[3] It consists of a large collection of algorithms supported by appropriate documentation.

- Experiment preparation: We prepare the environment by setting up the source code and the Eclipse where the participants will work on the tasks. We define the maintenance tasks and provide the free text description. We prepare the coupled file changes and the attributes from the software repository to be presented to the participants in the experiment.

- Solving tasks: The participants in both groups worked for two hours in two labs and provide solution for the maintenance tasks. The solution and the screen recording have been saved for further analysis.

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

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2492v1 | CC BY 4.0 Open Access | rec: 30 Sep 2016, publ: 30 Sep 2016

- Material gathering: We gather the questionnaires, the edited source codes and the video files of the participants screens for further analysis.

- Solution analysis: We analyze the scores for the correctness of the maintenance tasks, calculate the time needed for solving the tasks and determine the influence of the coupled file changes on the tasks solution.

5 Results and Discussion

The complete list of the maintenance tasks, the coupled file changes, the software repository attributes, the questionnaires and the analysis results can be found in the supplemental material of this paper.

Table 4: Descriptive Statistics (Attributes Usefulness)

Attribute	Median	MAD
Package Description	4	1
Issue Description	4	1
Commit Message	4	1
Issue Type	3	1
Commit ID	3	1
Commit Author	3	1
Issue Author	3	1
Commit Time	3	1

Table 5: Statistical Significance (Coupled changes)

Depend. Variable	p-value	r-value
Correctness	0.000	0.448
Time Effort	0.041	0.259

5.1 Usefulness of Coupled File Changes

As we already explained, we operationalize the usefulness of coupled file changes by their influence on the correctness of the solutions and the time needed to solve the tasks.

5.1.1 Correctness

We summarize the distribution of the correctness distribution using box-plots as presented in Figure 1. On the y-axis we have the correctness score for the successful solving of the tasks. Here the observations are grouped based on the presence of coupled changes suggestions during the maintenance tasks solution.

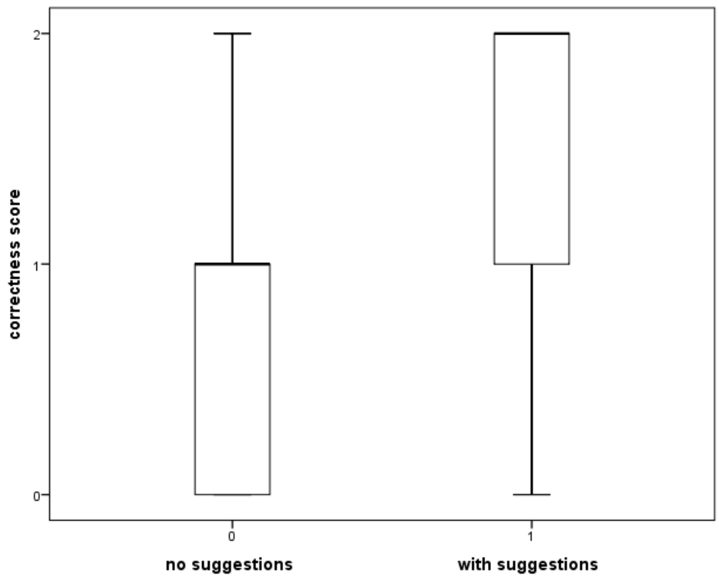

Figure 1: Correctness Boxplots

From this box-plot we see that the participants achieved better scores when solving the maintenance tasks using the coupled file changes suggestions we have provided.

We investigate the correctness difference of both groups by testing the first null hypothesis of the first research question claiming that there is no significant differences in the correctness of the task solutions.

Applying the Mann-Whitney U Test results in a p-value of 0.000 as presented in Table 5. This result has to be lower than the threshold of 0.05, so this null hypothesis can be rejected. This means that there is a statistically significant difference in the correctness of the solution for the provided tasks when using coupled file changes suggestions against the correctness of the solutions only using the provided task description. The r-value of the effect size for the correctness is 0.448 which describes a strong statistical difference in the correctness of the maintenance tasks solutions between the groups with and without using coupled change suggestions.

In Table 6 we represent the descriptive statistics for the correctness of the tasks solutions. The participants which used the suggestions solved 63.8% of the tasks completely, whereby the participants not using suggestions solved only 22% of the tasks. This shows an significantly higher score for the group using coupled changes suggestions.

The median absolute deviation (MAD) value for the group using coupled changes is 0, whereby the value for the group not using coupled changes is 1. These values show that the correctness score is spread very close to the median for the score of the first group. The statistical results provide an evidence

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2492v1 | CC BY 4.0 Open Access | rec: 30 Sep 2016, publ: 30 Sep 2016

Table 6: Descriptive statistics for the correctness of the tasks

Without Suggestions (–)			With suggestions (+)		
Completely solved tasks	Median	MAD	Completely solved tasks	Median	MAD
22 %	1	1	63.8 %	2	0

that the coupled file changes significantly influence the correctness of the maintenance tasks in the experiment. Inexperienced developers solves more tasks when using our suggestions which means they uses the benefit of hints related to similar tasks. The coupled change suggestions allow the developer to follow a set of files and remind him/her that similar tasks include changes in various locations in the source code.

The improvement in the number of solved tasks for the group using the coupled change suggestions shows that developers have used the benefits of additional help in locating the features and the files to be modified to solve their tasks successfully. The group which did not use this kind of help did has not succeeded to solve the same or higher number of tasks which points to the usefulness of our approach.

The use of coupled file changes has been especially noticed in cases where the developer needs to perform a similar changes in several locations, like editing different views of the application GUI. Here, the developers not using coupled change suggestions missed to implement the change in all the files where the change should be performed. Coupled file suggestions help the developers not to miss other source code locations they need for their task.

5.1.2 Time

We have analyzed the influence of the coupled file change suggestions on the time needed to successfully perform the tasks when using coupled versus not using the coupled file change suggestions. The distribution of the values for both groups is presented in Figure 2. We see that the distributions are similar with a slight tendency to more time without suggestions. We test the second null hypothesis which claims that there is no influence of the coupled file changes on the time needed to solve the tasks.

The p-value for the two tailed test is 0.041. This value is slightly below the threshold of 0.05 for the significance of the difference in the time needed to solve the tasks using coupled file changes versus the group without using the coupled file changes. Therefore, we have to reject the null hypothesis. The r-value for the time needed to solve the maintenance tasks is 0.259 which shows a relatively small statistical difference between the group which used coupled change suggestions and the group without suggestions compared to the r-value for the correctness of the solution.

The descriptive statistic values in Table 7 for the time variable report a decrease of the means for the time needed to solve the tasks by 26% for the group

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2492v1 | CC BY 4.0 Open Access | rec: 30 Sep 2016, publ: 30 Sep 2016

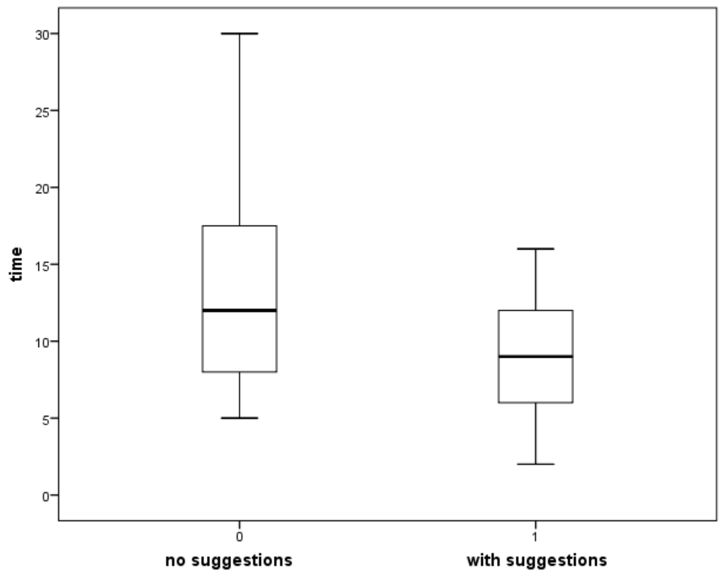

Figure 2: Time Boxplots

not using coupled change suggestion. The means ranking reports slightly better results for the group using coupled file changes, meaning the the participants of this group solved their tasks faster. The standard deviation for the group using coupled changes is twice lower than for the group not using coupled changes which shows a higher spread-out for the first group. These results show an

Table 7: Descriptive statistics for the time needed in minutes

Without Suggestions (−)			With suggestions (+)		
Median	Mean	Stand. Dev.	Median	Mean	Stand. Dev.
12	14.060	8.925	9	9.230	4.158

improvement with a statistical significance. This still provides some benefit by the coupled file changes approach for faster solving of maintenance tasks. The time effort drops because developers using the coupled change suggestions needed less time to find the files to change instead to search for the features and files in the source code they need to edit.

The improvement in the time needed to solve the tasks for the group using the coupled file changes is not that strong as the improvement in the correctness of the task solution which leads us to the point that although our approach helps the developers to locate the files needed to be changed. However, it does not eliminate the time they need to understand the features and the changes they need to perform in the source code. They still need time to organize this information and use it. Furthermore, they need to read and understand the suggestions. This means that the change suggestions do not provide an automatic solution for solving their tasks.

5.2 Usefulness of software repository attributes

The distribution of each attribute usefulness is presented in Figure 3 where the usefulness distribution for each of the repository attributes is presented based on the feedback of all participants in the experiment.

We test the third null hypothesis which claims that there is no difference in the usefulness between the attributes using the p-value of the Kruskal-Wallis H Test. In our case, the p-value for this test is 0.000 which is lower than the 0.05 threshold. This result leads us to rejecting the null hypothesis.

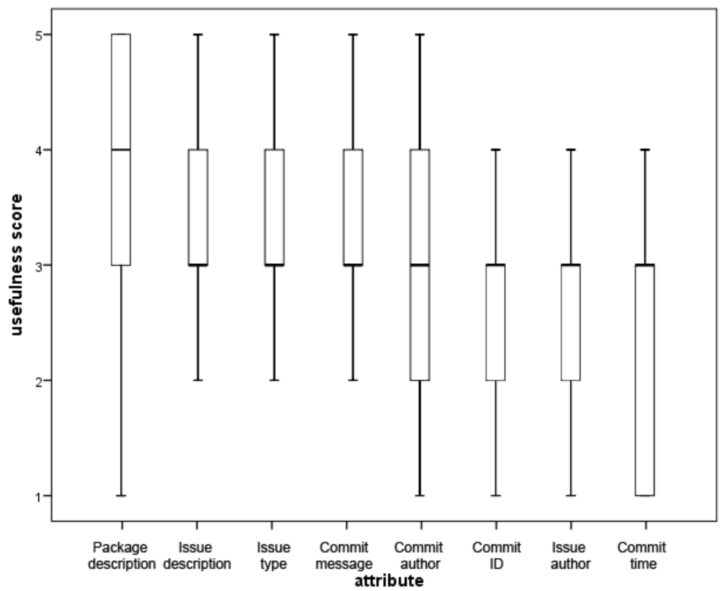

Figure 3: Time Boxplots

We reported a set of various software attributes from the software repository. The participants reported their feedback on their usefulness at the end of the experiment lab after the tasks has been performed. We calculated the r-value of the size effect for the repository attributes by creating pairs of each of the attributes where we determined the z-value of the Mann-Whitney test 8 for each pair. We have 28 pairs of attributes.

The greatest difference in the usefulness is between the commit time and the issue description where the r-value is 0.566, followed by the difference between the commit time and the package description with an r-value of 0.557. This indicates a high statistical significance between these pairs of attributes. The lowest difference is between the commit id and the commit author, here the r-value is 0.004, followed by the difference between the commit id and the issue author

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2492v1 | CC BY 4.0 Open Access | rec: 30 Sep 2016, publ: 30 Sep 2016

Table 8: Statistical Significance (Coupled changes)

p-value	r-value	Repository Attribute pairs	
0.180	0.279	Commit ID	Commit Message
0.972	0.004	Commit ID	Commit Author
0.249	0.136	Commit ID	Commit Time
0.000	0.467	Commit ID	Issue Description
0.108	0.190	Commit ID	Issue Type
0.624	0.058	Commit ID	Issue Author
0.000	0.465	Commit ID	Package Description
0.022	0.270	Commit Message	Commit Author
0.001	0.400	Commit Message	Commit Time
0.048	0.233	Commit Message	Issue Description
0.582	0.065	Commit Message	Issue Type
0.004	0.336	Commit Message	Issue Author
0.220	0.269	Commit Message	Package Description
0.228	0.142	Commit Author	Commit Time
0.000	0.459	Commit Author	Issue Description
0.122	0.182	Commit Author	Issue Type
0.599	0.062	Commit Author	Issue Author
0.000	0.464	Commit Author	Package Description
0.000	0.566	Commit Time	Issue Description
0.008	0.311	Commit Time	Issue Type
0.476	0.084	Commit Time	Issue Author
0.000	0.557	Commit Time	Package Description
0.118	0.279	Issue Description	Issue Type
0.000	0.526	Issue Description	Issue Author
0.530	0.074	Issue Description	Package Description
0.039	0.244	Issue Type	Issue Author
0.009	0.308	Issue Type	Package Description
0.000	0.515	Issue Author	Package Description

with an r-value of 0.9058. This shows that there are significant differences between the attributes usefulness. We have also gathered the descriptive statistics for the participants feedback on the usefulness of each attributes presented in Table 4. The median values vary from 3 for the commit id, the commit author, the commit time, the issue author and the issue time, and 4 for the commit message and the package description. This places the cutoff between "neutral" and "somehow interesting" for most of the attributes. The MAD value for all attributes is 1, which shows a low spread out of the usefulness values around the median.

We determined that the attributes have different usefulness according to the feedback of the participants. The median ranking defines which of the attributes are most useful. As most useful attribute we identify the package description followed by the issue description and the commit message. This leads us to the

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2492v1 | CC BY 4.0 Open Access | rec: 30 Sep 2016, publ: 30 Sep 2016

conclusion that the inexperienced developers seek for help about the features of the source code they need to edit and the task they have to complete.

The issue type and the commit time are in the middle of the list. The most useless attribute is the commit author followed by the issue author and the commit id. Here, we suppose that the developers are not interesting about the information who performed the changes because they do not know this person. This could change if the developers were included in the project for a longer time.

Although we enlisted a list of typical repository attributes, the participants have identified a smaller set of attributes to be useful for them than we provided in this experiment. This means that we don't have to not present all the attributes for the reason that different developers can happen to find some attributes as obsolete to be included in the coupled file change suggestions. The individual choice of useful attributes will avoid a confusion of developers. Reporting an individual set of attributes can increase the acceptance of coupled file change suggestions concept.

5.3 Threats to Validity

- Internal Validity: Possible internal validity threats can rise from the experiment design. To limit this possibility and the learning effect, we use a counterbalanced design where every developer solves four different tasks whereby each of them solves two tasks without and two tasks using coupled change suggestions. This way the results are not directly influenced by the task supported with the coupled file suggestions. The judgment of correctness of the task solutions represents another internal threat whereby we test the solutions to determine the level of correctness.

- Construct Validity: The greatest construction threat for the study are the coupled file changes we have extracted. The coupled files we extracted using a relatively high threshold which limits the possibility to provide suggestions for coupled changes that happened by chance. Also the metrics we have used to measure to determine the usefulness can represent a threat. The subjective usefulness usefulness rating represents another construct validity whereby we evaluate the provided tasks solutions pairwise to minimize the errors in conducting the score distribution. For the time needed to solve the tasks we play the captured screens of the participants to calculate the time effort needed for the tasks.

- External Validity: The external validity threat concerns the generalization of the experiment. The main threat here arises from the type of maintenance tasks, the participants and the system we investigate. We use four different perfective tasks which are supported by a free text description without any other adaptation or external help. This way we limit the possibility to create artificial conditions specially tailored for our participants.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2492v1 | CC BY 4.0 Open Access | rec: 30 Sep 2016, publ: 30 Sep 2016

The system we have used for the experiment is an open source Java project with a clear project structure. We have used data mining technique that can be easily performed on other Git repositories to extract coupled file changes.

6 Conclusion and Future Work

From the provided results and hypotheses tests we can summarize that the coupled changes approach was successfully tested in the performed experiment. The participants working with coupled change suggestions provided significantly more correct solutions than the participants without these suggestions.

The participants which used coupled file changes suggestions finished their tasks slightly faster compared to the participants group which was working only using the issue description.

We can conclude that the coupled file change suggestions can be positively judged to be useful for inexperienced developers working on maintenance tasks. The influence is particularly positive on the correctness level of the tasks solutions, meaning that it helps them to solve more tasks.

The influence of the coupled change suggestions on the time effort for solving the tasks is lower than on the correctness of the solutions.

We have extended the findings of Ramadani & Wagner (2016) where the participants in their feedback reported the coupled file changes and the attributes as neutral to use in maintenance tasks. Our experiments outcomes are more positive compared to the results of Ramadani & Wagner (2016). Working on real maintenance tasks using the tasks of the working software product increases the acceptance of coupled change suggestions by the developers. Also we rounded up the set of useful attributes based on the set of attributes presented in this study.

The next steps would be to transform the results and the findings in a tool implementation to support the developers working on maintenance tasks using visual presentation of suggestions which set of files they should also change. The final set of attributes presented in the tool should be adjustable for the reason not to flood the developer with information which can cause a negative effect on their usefulness.

7 Acknowledgments

We would like to thank Asim Abdulkhaleq for his help in the tasks scoring and the analysis of the questionnaires.

References

Abran, A., & Nguyenkim, H. (1991). Analysis of maintenance work categories through measurement. In *Software Maintenance, 1991., Proceedings. Confer-*

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2492v1 | CC BY 4.0 Open Access | rec: 30 Sep 2016, publ: 30 Sep 2016

ence on, (pp. 104–113).

Agrawal, R., & 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, (pp. 487–499). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Basili, V. R. (1990). Viewing maintenance as reuse-oriented software development. *IEEE Softw.*, *7*(1), 19–25.

Basili, V. R., Caldiera, G., & Rombach, H. D. (1994). *The Goal Question Metric Approach*. Wiley.

Bavota, G., Dit, B., Oliveto, R., Di Penta, M., Poshyvanyk, D., & 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, (pp. 692–701). Piscataway, NJ, USA: IEEE Press.

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

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

Briand, L., Morasca, S., & Basili, V. (2002). An operational process for goal-driven definition of measures. *IEEE Transactions on Software Engineering*, *28*, 1106–1125.

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

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

Chan, T. (2008). Impact of programming and application-specific knowledge on maintenance effort:a hazard rate model. In *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*, (pp. 47–56).

Cohen, J. (1977). In *Statistical Power Analysis for the Behavioral Sciences*, (pp. 469 – 474). Academic Press, revised edition ed.

Coolican, H., & Taylor, F. (2013). *Research methods and statistics in psychology*. Routledge.

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

De Lucia, A., Pompella, E., & Stefanucci, S. (2002). Effort estimation for corrective software maintenance. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering*, SEKE '02, (pp. 409–416). New York, NY, USA: ACM.

Fischer, M., Pinzger, M., & 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, (pp. 23–). Washington, DC, USA: IEEE Computer Society.

Fluri, B., Gall, H., & Pinzger, M. (2005). Fine-grained analysis of change couplings. In *Source Code Analysis and Manipulation, 2005. Fifth IEEE International Workshop on*, (pp. 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/`.

Fritz, C. O., Morris, P. E., & Richler, J. J. (2012). Effect size estimates: Current use, calculations, and interpretation. *Journal of Experimental Psychology : General*, *141*(1), 2–18.

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

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

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

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

Han, J., Pei, J., Yin, Y., & 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., & Holt, R. C. (2004). Predicting change propagation in software systems. In *Proceedings of the 20th IEEE International Conference on Software Maintenance*, ICSM '04, (pp. 284–293). Washington, DC, USA: IEEE Computer Society.

Hattori, L., dos Santos Jr, G., Cardoso, F., & 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, (pp. 210–223). Porto Alegre, Brazil, Brazil: Sociedade Brasileira de Computação.

Hutton, A., & Welland, R. (2007). An experiment measuring the effects of maintenance tasks on program knowledge. In *Proceedings of the 11th International Conference on Evaluation and Assessment in Software Engineering*, EASE'07, (pp. 43–52). Swinton, UK, UK: British Computer Society.

IEEE (1998). *STD 1219: Standard for Software Maintenance*.

ISO/IEC (1995). 12207: Information technology-software life cycle processes.

ISO/IEC (2000). 14764: Software engineering-software maintenance.

Kagdi, H., Collard, M. L., & 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., & 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, (pp. 47–53). New York, NY, USA: ACM.

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

Nachar, N. (2008). The mann-whitney u: A test for assessing whether two independent samples come from the same distribution. *Tutorials in Quantitative Methods for Psychology*, *4*(1), 13–20.

Nguyen, V., Boehm, B., & Danphitsanuphan, P. (2011). A controlled experiment in assessing and estimating software maintenance tasks. *Inf. Softw. Technol.*, *53*(6), 682–691.

Pigoski, T. M. (1996). *Practical Software Maintenance: Best Practices for Managing Your Software Investment*. Wiley Publishing, 1st ed.

Pohlert, T. (2014). *The Pairwise Multiple Comparison of Mean Ranks Package (PMCMR)*. R package.

Ramadani, J., & Wagner, S. (2016). 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).

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

Ricca, F., Leotta, M., Reggio, G., Tiso, A., Guerrini, G., & Torchiano, M. (2012). Using unimod for maintenance tasks: an experimental assessment in the context of model driven development. In *Proceedings of the 4th International Workshop on Modeling in Software Engineering, MiSE 2012, Zurich, Switzerland, June 2-3, 2012*, (pp. 77–83).

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2492v1 | CC BY 4.0 Open Access | rec: 30 Sep 2016, publ: 30 Sep 2016

Shelly, G. B., Cashman, T. J., & Rosenblatt, H. J. (1998). Systems analysis and design.

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

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

Swanson, E. B. (1976). The dimensions of maintenance. In *Proceedings of the 2Nd International Conference on Software Engineering*, ICSE '76, (pp. 492–497). Los Alamitos, CA, USA: IEEE Computer Society Press.

Tomczak, M., & Tomczak, E. (2014). The need to report effect size estimates revisited. an overview of some recommended measures of effect size. *The need to report effect size estimates revisited. An overview of some recommended measures of effect size*, *TRENDS in Sport Sciences*, 19–25.

van Rysselberghe, F., & 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., & 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, (pp. 15–25). New York, NY, USA: ACM.

Ying, A. T. T., Murphy, G. C., Ng, R. T., & 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., & Whitehead, E. J., Jr. (2006). Mining version archives for co-changed lines. In *Proceedings of the 2006 International Workshop on Mining Software Repositories*, MSR '06, (pp. 72–75). New York, NY, USA: ACM.

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

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2492v1 | CC BY 4.0 Open Access | rec: 30 Sep 2016, publ: 30 Sep 2016