Federal University of Campina Grande

SPLAB
SOFTWARE PRACTICES LABORATORY

Refactoring-aware Code Review: An Exploratory Case Study

PROPOSAL OF THE CASE STUDY DESIGN
Q2 2019

Abstract

This exploratory case study design proposal is based on the guidelines specified by Runeson and Höst [1]. In short, it consists of the research questions and procedures for data collection, qualitative and quantitative analysis, and validity issues of the data interpretation. Lastly, we also suggest a working schedule.

1 Case study design

1.1 Research objective

The main goal of this study is to establish an in-depth basis for understanding the refactoring-aware code review process.

1.2 Research questions

RQ1. What is the relationship between refactoring-awareness and code review quality?

RQ1.1. What is the relationship between the number of refactorings and the number of review rounds?

RQ1.2. What is the relationship between the complexity of refactorings and the number of patch sets?

RQ1.3. What is the relationship between the groups A/B and the number of patch sets?

RQ1.4. Is there any difference between a refactoring-aware and a no refactoring-aware change description for code review?

RQ1.5. What drives review positivity (proportion of accepted patches)?

RQ1.6. What drives abandonment (discard of changes)?

RQ2. What does happen with patches that contain *Extract Method* refactorings?

1.3 Case study and unit of analysis

In this work, the selected case is the *Gerrit* code review system, because it is an open source web-based tool, which provides repository management (a useful feature to the data collection – see subsection 3.3) for the *Git* version control system[1], and it is employed by large scale projects, such as Android, Chrome OS, Eclipse foundations, LibreOffice, MediaWiki, OpenStack, among others.

[1] `https://git-scm.com/`

At Gerrit, developers can push a commit, as a change, so reviewers can post comments directly on Gerrit's diff utility. After being checked, reviewed, and approved, a change can be applied to the code base. The approval or rejection of a change is performed by voting (see Appendix A for more details).

The *Eclipse*[2] and *Chromium*[3] projects are the selected units of analysis, which have respectively over 1,680 and 1,480 open source subprojects available on Gerrit.

1.4 Data collection procedures

1.4.1 Brief "case history"

In May 2019, before data collection, we carried out a brief "case history" on the unit of analysis, consisting of the following steps, in order to get a better understand of the data. First, we defined two groups (Table 1) of check-in/commit messages (step 1).

After that, we randomly selected one project from the Eclipse repository at Gerrit (step 2). From Gerrit, we can collect data such as the number of reviewers of a change, and the number of patch sets. Each patch set (or changeset) contains one check-in message, the modified files, the number of additional and deleted lines by file, and the reviewers' comments. Then, we ran *RefactoringMiner* (step 3) and randomly selected a few instances (step 4) for a short investigation (Table 2).

Lastly, we explored two instances from each group (step 5) and could realize the existence of poor messages in the refactoring-awareness context (Table 3). Indeed, for instance, the impact of 56 refactorings not mentioned in a commit message to the code review practice is, at least, interesting for further investigation.

Step 1. Definition of groups

Table 1: Groups definition

Group	Description
A	commit message with refactoring information
B	commit message with no refactoring information

Step 2. Selection of one subproject at a code review repository

Code review system	Repository selected	Subproject selected
Gerrit [4]	Eclipse [5]	tracecompass/org.eclipse.tracecompass

[4] Changes are classified in open, merged, or abandoned.

[5] `https://git.eclipse.org/r`

Step 3. *RefactoringMiner* execution

Project selected	#Commits	#Refactorings
tracecompass/org.eclipse.tracecompass	7,225	12,717

Step 4. Selection of instances

Step 5. Selection of critical instances

1.4.2 Procedures

Based on the outcomes of the brief "case history", we purpose the procedures for the collection of the data types (Table 4), as follows.

[2] `https://git.eclipse.org/r`

[3] `https://chromium-review.googlesource.com`

Table 2: Selected instances

Group	URL	#Refactorings [6]	#Patch Sets
A	https://git.eclipse.org/r/#/c/99698/	3	19
	https://git.eclipse.org/r/#/c/122968/	32	5
	https://git.eclipse.org/r/#/c/122045/	72	9
B	https://git.eclipse.org/r/#/c/140712/	1	2
	https://git.eclipse.org/r/#/c/104124/	27	47
	https://git.eclipse.org/r/#/c/126405/	23	51
	https://git.eclipse.org/r/#/c/134350/	48	19
	https://git.eclipse.org/r/#/c/123949/	56	2

[10] Number of refactorings detected by *RefactoringMiner*. Thus, we selected only *Merged* changes from Gerrit.

Table 3: Outline of commit messages and the refactorings detected by instance

URL	Description	Refactoring Type
https://git.eclipse.org/r/#/c/99698/	Mentioning refactorings, without specified types	2 Rename Attribute 1 Extract Variable
https://git.eclipse.org/r/#/c/122968/	Description of functionalities, mentioning Move	31 Move Class 1 Extract Variable
https://git.eclipse.org/r/#/c/104124/	Only the change title	9 Rename Variable 2 Rename Method 1 Extract Method 1 Inline Variable 2 Rename Parameter 1 Rename Attribute 1 Extract Class 6 Move Attribute 3 Extract and Move Method 1 Move Method
https://git.eclipse.org/r/#/c/123949/	Only the description of improvements	2 Rename Class 3 Extract Method 30 Rename Variable 1 Merge Variable 3 Rename Method 17 Rename Attribute

Table 4: Type of data to be collected

Data type	Description
Qualitative	check-in/commit messages
Quantitative	Measurements related to a change review history, such as the number of patch sets, and the number of inserted and deleted lines into files of a patch set

In order to clarify, the attributes listed in Table 4 are derived from our research questions and available to be collected from Gerrit. In particular, these attributes may be considered in an initial proposal and are subject to modification.

We propose the use of an independent method for data collection, because we are going to directly collect raw data without actually interacting with the subjects during this activity. The method consists of the subsequent steps.

Step 1. Extraction of the raw review data from the Eclipse code review repository[7], in JSON format. This format is a well-established, widely used, simple text-based, and lightweight data standard; thus, a reasonable choice to be a component of our replication kit.

Step 2. Pre-processing of raw review data for producing a refined **code review dataset**, in an appropriate manner to further the data analysis; for instance, getting specific attributes for characterizing change reviews.

Step 3. Execution of *RefactoringMiner* exactly for the Eclipse projects listed in the refined code review dataset (step 2 output), resulting in a **dataset of applied refactorings**.

Step 4. Execution of a string matching procedure for seeking refactoring-awareness patterns, in which input is the applied refactorings dataset, on the code review dataset; thus, identifying the instances of groups A and B (Table 1).

Step 5. Updating of the code review dataset by the addition of instances' categories (A or B).

1.5 Analysis procedures

In this point, it is clear that the conclusions should be based on a chain of evidence and purpose recommendations and implications for both practice and further research. For this reason, initially, we plan the procedures, stated in Table 5, for qualitative and quantitative data analysis.

Table 5: Data analysis procedures

Analysis type	Description
Qualitative	Criteria definition for categorizing of instances into groups A and B, and build the chain of evidence based on the identification of patterns, that show traceable inferences, from the data to our research questions
Quantitative	Analysis of descriptive statistics and correlation, and hypothesis testing

1.6 Validity procedures

The following procedures (Table 6) are proposed to deal with the issues of validity and reliability of the data interpretation. In order to reduce researcher bias, the analysis procedures need to be conducted by multiple researchers, which cooperation scheme will be properly reported.

2 Working schedule

In Table 7, we propose a research schedule for the next months.

References

[1] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Softw. Engg.*, vol. 14, pp. 131–164, Apr. 2009.

[7] `https://git.eclipse.org/r`

Table 6: Validity procedures

Type	**Description**
Construct validity	Use of multiple sources of evidence and establishment of a chain of evidence
Internal validity	In order to maintain consistency during the data collection, from each unit of analysis, we will follow the same procedures and tools
Reliability	This case study design has been developed, with efforts towards clarifying the data collection and analysis procedures

Table 7: Proposal for working schedule 2019

June/July	**August**	**September**
Data collection	Data analysis	Reporting

Appendices

A Basics on Gerrit Code Review System

On Gerrit, submitters push commits and reviewers can do revision tasks supported by its *diff* utility[8]. Each *change* has a *Change-Id*, which represents a single *commit under review*. In this context, as outlined in Figure 1, multiple git commits, which share the same Change-Id, are singly referred to as *patch sets*, i.e., a patch set is a git commit. Thus, each patch set has a unique *commit SHA* and a *change message*, which contains user-supplied information about what the change does. But, when a change is approved, only the latest version of a commit is submitted to the repository.

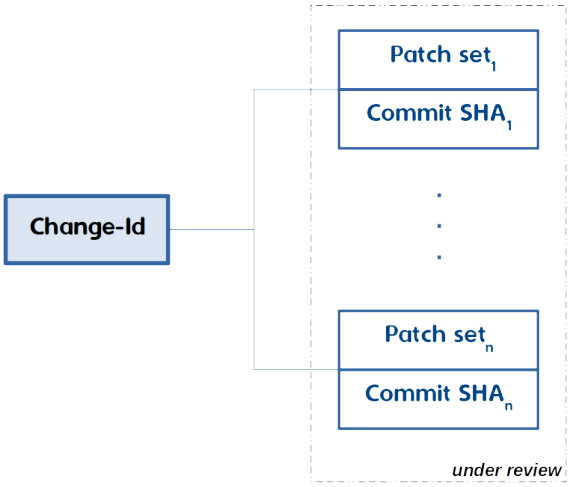


Figure 1: The relationship between a commit under review and its patch sets at Gerrit

In order to illustrate, as shown in Figure 2, a change under review has a unique numeric identifier on a Gerrit server, and a Change-Id, which is unique per repository and branch and identifies which patch sets belong to the same review.

In figure 3, we highlighted the patch sets number 7 (a) and 27 (b) from an Eclipse project on Gerrit, and their respective change messages and commit SHA. Observe that the change's numeric identifier, and the Change-Id are the same for both patch sets.

[8]For more details, visit `https://gerrit-review.googlesource.com/Documentation/concept-changes.html`

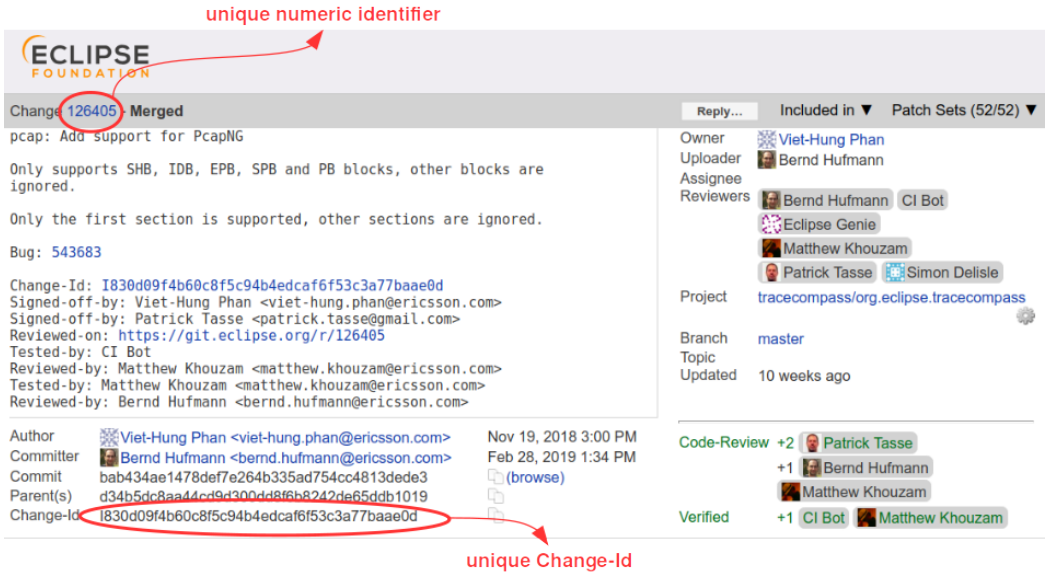
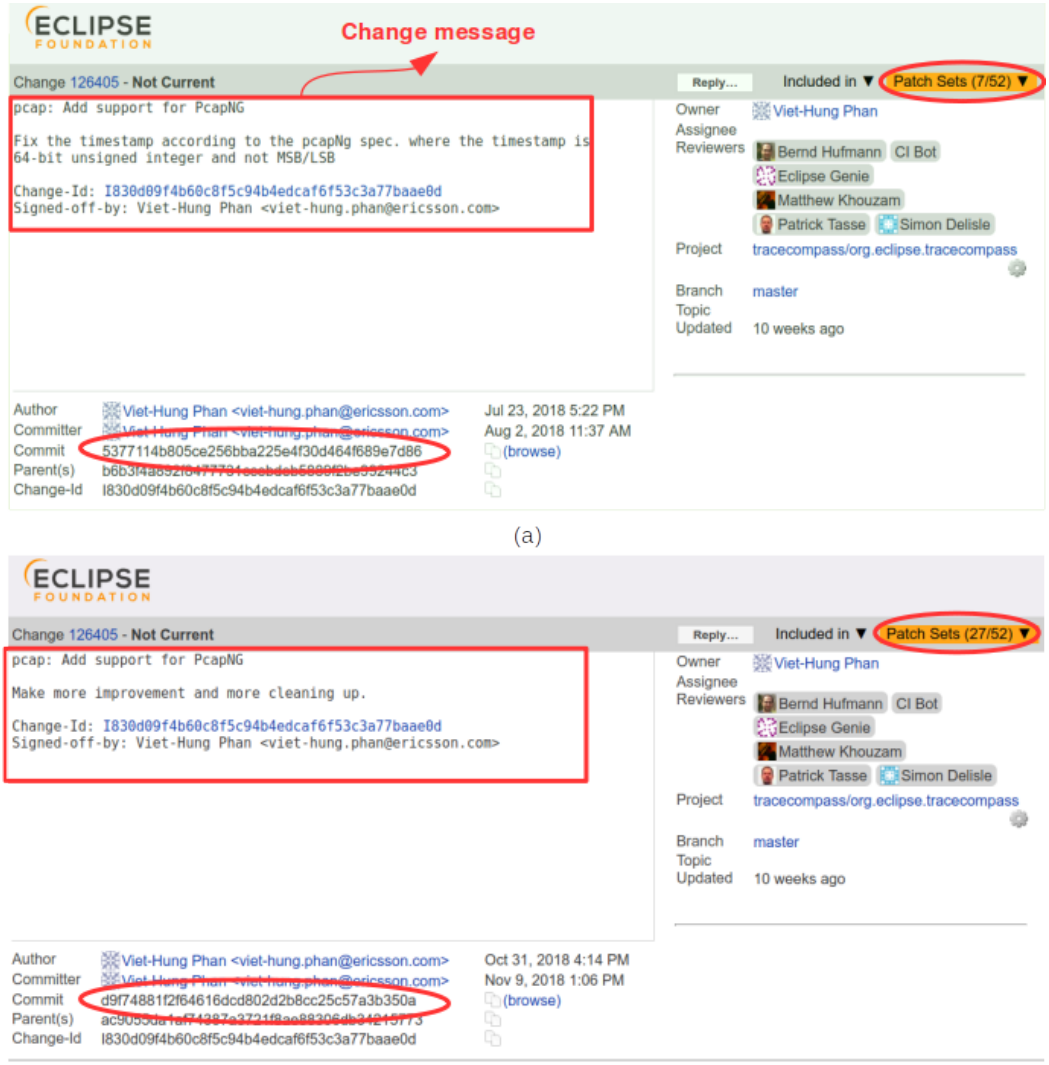


Figure 2: Gerrit's user interface of one change from Eclipse tracecompass/org.eclipse.tracecompass project



(a)

(b)

Figure 3: Patch sets number 7 (a) and 27 (b), and respective commit SHA and message of one change from Eclipse tracecompass/org.eclipse.tracecompass project