Ömer Furkan Tercan

Software Refactoring Rationale in Agile Development: A Case Study in a Lean Startup

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 31.03.2015

Thesis supervisor:

Assoc. Prof. Casper Lassenius

Aalto University
School of Science

AALTO UNIVERSITY
SCHOOL OF SCIENCE

ABSTRACT OF THE
MASTER'S THESIS

Author: Ömer Furkan Tercan

Title: Software Refactoring Rationale in Agile Development: A Case Study in a Lean Startup

Date: 31.03.2015 Language: English Number of pages: 7+10

Department of Computer Science and Engineering

Professorship: Code:

Supervisor and advisor: Assoc. Prof. Casper Lassenius

What is refactoring? What is the goal of refactoring?

-

What is the role of software refactoring in agile development? What is the developer's perception of refactoring? What does the literature say about refactoring rationale? How does these to views match? What is the scope of this study?

-

How does this thesis conduct a study to answer the research questios? What is the research context in a sentence? What is the research methodology in summary?

-

What are the key findings? Limitations and future work?

Keywords: refactoring, code smells, anti-patterns, agile, lean

Contents

Glossary

LOC Lines of code. vii, 4, 5

RCS Revision control system. 5, 6

List of Tables

1 Introduction

In this chapter section 1.1 presents the background and motivation of this study. Following section 1.2 presents the scope of the thesis. Section 1.3 introduces the objective and research questions. Finally, section 1.4 summarizes the structure of this document.

1.1 Background and motivation

What is refactoring? Why and how critical is it for software development. What does the literature say about it's importance. If we scope it a bit down, what is the view of agile development to these questions?

What is the tricky part when deciding on what and how much to refactor? Is this hard to define. Does it highly depend on the context? How credible is what the literature say about drivers on refactoring decisions. Is there a research gap to reveal empirical findings in order to contribute to the scientific knowledge regarding refactoring rationale?

What would be the benefits, once we know more about refactoring rationale? What does the literature say about it's possible benefits? Take into account different viewpoints (customers, developers, etc.).

Discuss the relevant literature. Have this specific topic been investigated? What are the touching points? Which areas of the topic will be overlapped (and why) in this study? What would be the additional contribution? Why does this contribution make sense?

How do you claim to achieve this contribution? Summarize the research process. Why does it make sense to conduct the study in such a research context?

Motivated by the findings in earlier studies, this thesis investigates the refactoring rationale in a lean startup ICT company. Attention has been paid to understand the drivers behind refactoring decisions in the company's agile development process. The results of the case study is analyzed, discussed and the most significant findings are compared with the scientific knowledge regarding software refactoring rationale. We will also discuss the issues which needs more thorough investigation and research in the future.

This study also contains a literature review chapter. In chapter 3, we will discuss software refactoring and refactoring rationale as it is described in the literature. Then, we will explore refactoring drivers defined by the scientific knowledge, including anti-patterns and code smells.

1.2 Scope of the thesis

1.3 Objective and research questions

According to the motivation stated in section 1.1, there is need to investigate and reveal the correlation between how literature have conceptualized software refactoring rationale and how it is actually percieved by developers. Accordingly, an empirical study can be conducted investigating refactoring decisions in a case company. The empirical results can then be further analyzed and compared with the scientific knowledge regarding software refactoring rationale.

This motivation leads to the main study objective:

Objective: Study refactoring rationale, particularly how it is perceived by the scientific knowledge versus agile software developers, to identify neglected threats to software evolution.

The following research questions were derived based on the main objective:

RQ1 What does literature say about refactoring and when it is needed?

This research question tries to investigate relevant literature to form a basis on refactoring, in particular to present a refactoring rationale taxonomy based on related work.

RQ2 How does developers perceive refactoring rationale?

This research question is of explorative nature and it tries to find empirical evidence on refactoring rational, taking into account the human factor.

RQ3 What does the correlation of the two reveal in terms of threats to software evolution?

This research question aims to discuss how well the developer's perception fits to the presented taxonomy. Based on this discussion, it aims to identify neglected threats to software evolution.

1.4 Structure of the thesis

2 Research setting

The main objective of this study is to gain an understanding on refactoring rationale, in particular how it is described in the relevant literature versus how it is perceived by agile software developers. Thus, identify neglected threats to software evolution.

The objective is targeted in terms of a) studying relevant literature to form a basis on refactoring, particularly on refactoring rationale, b) finding empirical evidence on how refactoring rationale is percieved by developers, and c) discussing the inclusionary nature of the literature study and empirical findings. The relation of these studies to the research questions is presented in Table 1.

Table 1: Relation between studies and research questions

Research question	**Literature**	**Empirical**
RQ1: What does literature say about refactoring and when it is needed?	X	-
RQ2: How does developers perceive refactoring rationale?	-	X
RQ3: What does the correlation of the two reveal in terms of threats to software evolution?	x	x

Legend: "X" indicates higher emphasis then "x", while "-" indicates inessential

Finding an answer to RQ1 requires a literature study. This research question aims to present a software refactoring rationale taxonomy, which will act as a building stone when answering RQ3.

Finding an answer to RQ2 through a real world investigation overlaps with Yin's (2003) definition of a case study — an empirical method aimed at investigating contemporary phenomena in their context. Similarly, the findings will act as a building stone when answering RQ3.

RQ3 aims to study the correlation between RQ1 and RQ2. In addition, it conducts a minor literature study on software evolution, to reveal neglected threats by taking the studied correlation into account.

2.1 Case company

The case study was conducted in a lean start-up ICT company based in Helsinki, Finland. The company is producing telecommunication application services enabling video calling from multiple endpoints, including web browsers, smart TV's and mobile devices.

Software under study

The service platform consists of a server backend and multiple client endpoints. However, the scope of this study includes the server backend and two of the clients as these three components were actively developed within the data collection period.

C1 The server component presented in Table 2, was implemented using javascript, particularly using nodejs.

C2 An android client component presented in Table 3, was implemented using java.

C3 A web client component presented in Table 4, was implemented using javascript, particularly using angularjs.

Table 2: Server component; number of files and LOC

Language	Files	Blank	Comment	Code
Javascript	36	582	131	4059
Json	4	1	0	251
Python	5	72	46	213
Html	1	0	1	43
Shell	2	21	5	39
Sum	48	676	183	4605

Table 3: Android client component; number of files and LOC

Language	Files	Blank	Comment	Code
Xml	707	2687	4921	25950
Java	143	2699	5873	15078
Shell	2	50	21	237
Sum	852	5436	10815	41265

Development process

The organization is a lean startup, having a lean development flow employing most of the agile development practices. Practices worth mentioning includes; continuous integration and deployment, continuous improvement (process and software

Table 4: Web client component; number of files and LOC

Language	Files	Blank	Comment	Code
Javascript	37	542	161	3631
Less	15	275	10	1359
Html	24	58	17	540
Json	2	0	0	0
Sum	78	875	188	5606

quality), efficient and face-to-face communication, peer reviews, one peace flow, pair programming, testing as an integral part of development, and collective code ownership.

As it is a key principle of lean development, in ensuring continuous integration and deployment, continuous software quality improvement takes place routinely but critically within the organization. In particular, software refactoring serves as the main approach in this context.

The empirical study was introduced to the developers during continuous process improvement meetings, named as retrospectives. As part of the study, developers were requested to put addition attention in reporting their refactoring related tasks. Collective code ownership and peer reviews were routine practices at present. Therefore, developers were already used to report their daily tasks using the source-code revision control system RCS.

Developers

All developers of the organization participated in the empirical study. During this period, 6 developers reported their refactoring tasks in the source code revision control system.

«A figure presenting developers and their work experience levels»

2.2 Data collection

Data was gathered from a single source. The single source of information was stored in the source-code RCS. Developers have reported their refactoring related tasks via the source-code RCS. Since collective code ownership and peer reviews where routine practices already, this extra duty required almost no extra effort from developers.

Prior to the empirical data collection kick-off, several group discussions were conducted to motivate developers to highlight their refactoring tasks in their notes and to place useful conventions for data collection and analysis. In order to ease the data collection and analysis process, a convention of labeling their reports was introduced to developers.

Developers were using git as the source-code RCS. Therefore, developers highlighted their refactoring tasks using git commit messages and git code line comments. Subsequently, a public API offered by the cloud-based git repository service was used to retrieve the commit messages, commit changes, and code line comments. The API offered json as a response format. Therefore, it made it trivial to collect, filter, further analyze and categorize refactoring tasks based on developer notes.

In the cases were developer notes lacked sufficient amount of detail on the rational behind a refactoring decision, informal communication was used to resolve the ambiguity. Thanks to a) peer reviews with short feedback loops, b) efficient and face-to-face communication possibility, and c) technical reviews following short iterations, ambiguous developer notes were minimized.

2.3 Data analysis

How was the data filtered, cleaned, and categorized? What tools and analysis methods were used?

3 Literature study

In addition to the references mentioned in the sub-sections, take a look at the following papers:

- Yamashita and Leon (2012)
- Yamashita and Leon (2013)
- Mäntylä (2009)
- Arcoverde et al. (2011)
- Mäntylä (2005)
- Mäntylä and Lassenius (2006)

3.1 Software refactoring

Define refactoring referring mainly to Brown et al. (1998) and Fowler (2000).

Explain the importance and benefits of refactoring referring to at least Khomh et al. (2009), Cusumano et al. (1997), Cusumano et al. (1997),

3.2 Refactoring rationale

Admit the basis of the presented taxonomy comes from Fowler (2000) but it is inspired (or taken from) Mäntylä et al. (2003).

Attempt to discuss and improve the taxonomy with at least Brown et al. (1998), Mäntylä and Lassenius (2006), Yamashita et al. (2013)

3.3 Software evolution

Refer to Lehman's (1980) software evolution study and to the described harmful side-effects of software evolution. The idea is to understand if there is neglected threats based on the difference between how literature describe refactoring rational and developer's view based on the empirical study.

4 Case study

Present findings based ond data collection and analysis.

5 Discussions

Discuss correlation between RQ1 and RQ2. Correlation shall reveal findings on neglected threats to software evolution

6 Conclusions

6.1 Threads to Validity

6.2 Future Work

References

- Arcoverde, Roberta, Alessandro Garcia, and Eduardo Figueiredo. "Understanding the longevity of code smells: preliminary results of an explanatory survey." Proceedings of the 4th Workshop on Refactoring Tools. ACM, 2011.
- Cusumano, Michael A., and Richard W. Selby. "Microsoft secrets." (1997).
- Cusumano, Michael, and David Yoffie. "Competing on Internet time." (1998).
- Fowler, Martin. "Refactoring: Improving the Design of Existing Code. 2000."
- Khomh, Foutse, Massimiliano Di Penta, and Y. Gueheneuc. "An exploratory study of the impact of code smells on software change-proneness." Reverse Engineering, 2009. WCRE'09. 16th Working Conference on. IEEE, 2009.
- Lehman, Meir M. "Programs, life cycles, and laws of software evolution." Proceedings of the IEEE 68.9 (1980): 1060-1076.
- McCormick, Hays W., Thomas J. Mowbray, and Raphael C. Malveau. "AntiPatterns: refactoring software, architectures, and projects in crisis." (1998).
- Mäntylä, Mika V., Jari Vanhanen, and Casper Lassenius. "A taxonomy and an initial empirical study of bad smells in code." Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on. IEEE, 2003.
- Mäntylä, Mika V. "An experiment on subjective evolvability evaluation of object-oriented software: explaining factors and interrater agreement." Empirical Software Engineering, 2005. 2005 International Symposium on. IEEE, 2005.
- Mäntylä, Mika V., and Casper Lassenius. "Drivers for software refactoring decisions." Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering. ACM, 2006.
- Mäntylä, Mika V., and Casper Lassenius. "Subjective evaluation of software evolvability using code smells: An empirical study." Empirical Software Engineering 11.3 (2006): 395-431.
- Mäntylä, Mika. "Software evolvability-empirically discovered evolvability issues and human evaluations." (2009).
- Yamashita, Aiko, and Leon Moonen. "Do developers care about code smells? An exploratory survey." Reverse Engineering (WCRE), 2013 20th Working Conference on. IEEE, 2013.
- Yamashita, Aiko, and Leon Moonen. "Do code smells reflect important maintainability aspects?." Software Maintenance (ICSM), 2012 28th IEEE International Conference on. IEEE, 2012.

- Yamashita, Aiko, and Leon Moonen. "Exploring the impact of inter-smell relations on software maintainability: an empirical study." Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, 2013.
- Yin, Robert K. "Applications of case study research (applied social research methods)." Series, 4th. Thousand Oaks: Sage Publications (2003).