WHAT CAN WE LEARN ABOUT

LOCATING REFACTORYING OPPORTUNITIES

FROM

DECOMPOSING SOFTWARE TO MICROSERVICES

SIU/CAPES bilateral project "Modern Refactoring"
EXTRACT AND MOVE METHOD

LOCATING REFACTORING OPPORTUNITIES

MICROSERVICES

SOFTWARE DECOMPOSITION
ME: PAST-CURRENT

- B.S. University of Bergen, computer science
- M.S. joint University of Bergen & Western Norwegian University of Applied Sciences (Volker Stolz)
- PhD University of Bergen, Language / Processor Co-Evolution (Anya Bagge)
- currently on research stay at (the very empirical) Software Engineering group at University of British Columbia, Canada (Gail C. Murphy)
(EXTRACT AND) MOVE METHOD
(EXTRACT AND) MOVE METHOD

Extract method

Move method

class A

class A

class B
(EXTRACT AND) MOVE METHOD

public class Customer {

    public double getOwing(List<Invoice> invoices) {
        double outstanding = 0;
        for(Invoice invoice : invoices)
            outstanding+=invoice.getOutstanding();
        return outstanding + outstanding * 0.2;
    }
}

public class Invoice {

    public double getOutstanding() {
        return outstanding;
    }
    public double getInterest() {
        return outstanding * 0.2;
    }
}

Extract method

Move method
LOCATING REFACTORYING OPPORTUNITIES

```java
public class Customer {
    public double getOwing(List<Invoice> invoices) {
        double outstanding = 0;
        for(Invoice invoice : invoices)
            outstanding+=invoice.getOutstanding();
        return outstanding + outstanding * 0.2;
    }
}
```

Extract method

```java
public class Invoice {
    double outstanding = 0;
    public double getOutstanding() {
        return outstanding;
    }
    double getInterest() {
        return outstanding * 0.2;
    }
}
```

Move method
LOCATING REFACTOIRNG OPPORTUNITIES

When you delete a block of code that you thought was useless
LOCATING REFACTORING OPPORTUNITIES

- all possible
- smell-reducing
- machine learning
- metrics-aware
- heuristics

https://refactoring.guru/refactoring/catalog
LOCATING REFACTORING OPPORTUNITIES
Use, Disuse, and Misuse of Automated Refactorings

Mohsen Vakilian, Nicholas Chen, Sia Negah, Baliy Aweilb, Rajkumar Bhargava, Bruce Bailey, Ralph E. Johnson
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA

Abstract—Though refactoring tools have been available for
more than a decade, research has shown that programmers
rarely use such tools. However, little is known about why
refactorings are not used. In this study, we performed a field
study on programmers to determine what causes programmers
to not use refactoring tools. In our sample of 24 developers,
we discovered that the main reasons programmers do not
use refactoring tools are disinterest in the technology, not
knowing how to use it, and not seeing the value in using
it. These findings have important implications for the
industry: (a) refactoring tools need to be more intuitive
and usable, (b) refactoring tools need to be marketed
more effectively, and (c) refactoring tools need to be
integrated into larger tool suites.

1. INTRODUCTION

Refactoring is defined as changing the design of self
without altering its observable behavior [1]. Refactor-
ings, rename, move, split, and join programs elements such
as fields, methods, packages, and classes. Agile software
processes such as Extreme Programming (XP) prescribe
refactoring [2], because it enables evolutionary software
design and is key to maintainable and reliable code [3].
Programmers refactor their code frequently [4], [5].
Some refactoring tools are also available and are free to
perform manually. Thus, automated refactoring tools are
widely used in industry. They are designed to make code
easier to test, maintain, and maintainable. However, many
refactoring tools do not provide support for the refactoring
process or are not integrated into larger tool suites.

Recently, there has been much interest in improving
the reliability of existing automated refactoring tools and
building new ones to automate sophisticated refactorings.
This is not surprising, given the inherent
and more precise of some refactorings and the perceived
benefits of their automation. In spite of the growing interest
in improving the reliability of automated refactoring tools [11],
[12], this aspect of refactoring has not received enough
attention. For example, the user interface of refactoring
tools have changed little since they were first introduced,
and recent studies suggest that programmers greatly underestimate
the existing refactoring tools [8]. We need to understand
the problem in order to improve today’s refactoring tools.

We conducted a study consisting of both quantitative
and qualitative data collection. We studied 24 developers
working in their natural settings on their code for a total
of 360 programming hours over three months, and collected
data about their interactions with automated refactorings.
We observed patterns of interaction in our quantitative data
and interviewed our participants to take a more detailed qualitative look at our behavioral data. Then, we
adapted a general framework of human-computer inter-
action [5] to frame the use, disuse, and misuse of au-
tomated refactoring tools. Use of automated refactoring tools
in programmers applying automated refactoring to perform
code changes they might otherwise do manually. Misuse of
automated refactoring tools in programmers applying automated refactoring tools as refactoring tools (Section 5). Finding users deploy to minimize the main property of automated refactoring tools, namely, refactoring speed. In addition, we observed some unexpected uses of the refactoring tool (Section 5). Finally, we have proposed alternative ways
of designing refactoring tools based on the findings of our
study (Not Software Refactoring Tools (V), Version Control (Y), and VCS.)

Our interviewees did not use automated refactorings that
they had found to have complex user interfaces and unclear
benefits. In general, if the benefits of automation are not
readily apparent, humans are less likely to use the automation
because of the cognitive overhead involved in evaluating
and using the automation.

On the other hand, programmers appreciate the
tools that propose applicable refactorings, and are willing
to use automated refactorings even when they may change
the program’s behavior.
ABSTRACT
In which refactoring improves software quality and developer productivity. However, few empirical studies quantitatively assess refactoring benefits in software development practices toward these benefits. This paper presents a field study of refactoring benefits and challenges. It involves a 328 engineers participating in a survey, semi-structured interviews with professional software engineers and practitioners, and quantitative analysis of team history data. The survey finds that the refactoring definition is functionally defined and utilizes software development practices, and that refactoring reduces maintenance cost and effort. It also shows that some refactoring activities have a measurable cost and effort. We also report on interviews with a designated refactoring team that has a methodology, a central effort on refactoring Windows. The qualitative analysis of Windows 7 development team reveals that the team experienced significant reduction in the number of defects and changes in their development practices, reflecting a positive benefit of refactoring.

1. INTRODUCTION
It is widely believed that refactoring improves software quality and developer productivity by making it easier to maintain and understand software systems. However, it is not clear that lack of refactoring leads technical debt to be resolved through incremental and non-incremental refactoring techniques. This paper presents a case study of Microsoft Windows software development processes, which uses a large-scale refactoring effort. The refactoring activities are associated with a large-scale refactoring effort, which is focused on improving software quality and developer productivity. The paper investigates the benefits of refactoring in practice and in the development of Windows 7. A team of software developers conducted a survey with over 328 software engineers who described their refactoring activities in software development. The survey found that refactoring significantly reduces the number of defects and changes in their development practices, reflecting a positive benefit of refactoring.

The value of refactoring is difficult to measure. How do you measure the value of a bug that never existed, or the time saved on a later undetermined feature? How does this value bubble up to management? Because there's no way to place immediate value on the practice of refactoring, it makes it difficult to justify to management.

These (Fowler’s refactoring types or refactoring types supported by Visual Studio) are the small code transformation tasks often performed by Visual Studio, but they are unlikely to be performed alone. There’s usually a bigger architectural change behind them.

I’d love a tool that could estimate the benefits of refactoring.
SOFTWARE DECOMPOSITION
SOFTWARE DECOMPOSITION

“Finding, or creating, ‘seams’ in your code base”

- Michael Feathers, Working Effectively with Legacy Code
MICROSERVICES

- Clear areas of responsibility
- Strong encapsulation
- Individually deployable
DECOMPOSING MONOLITHS TO MICROSERVICES
DECOMPOSITION TO MICROSERVICES

Re-architecting OO into Microservices: A Quality-Centred Approach

Andel Selmadji, Mohamed-Ely Essam Serag, Christophe Doisy, and Haimin Ounj

Abstract. Due to its tremendous advantages, object-oriented (OO) style has become an essential element for the development of applications deployed on the cloud and for those adopting cloud computing. Migrating existing applications to microservices from these advantages. Thus, in this paper, we focus on automatically identifying microservices from an existing OC system. We propose a new approach to service decomposition based on 16 criteria and the literature and industry experience. This approach considers the Service Cutter, our method and tool framework, to identify the target applications. This tool is unique and effective in identifying microservices from the given system by considering the requirements of the target applications.

Keywords: Object-Oriented; Migration; Identification

1 Introduction

Recently, microservice architectural style has become the development of applications deployed on the DevOps practices [5, 16]. In this style, an app is made up of services which are independently deployable, and manage its own data [10, 12]. These services exploit mechanisms and they are deployed using container clouds. The microservices facilitate the monitoring according to the changes that may occur at runtime related to cloud resources (e.g., resources allocated on availability, scalability guarantee, etc.) or any other event that microservices facilitate a continuous integration tasks [5].

Besides the adoption of microservices architecture of new applications, the migration of existing ones [3, 12] is an important issue.

Microservices Identification through Interface Analysis

Luiz Fernando Cotta Barros, Marla Gomes, and Alex De Sá

Abstract. In this work, the microservices architectural style is a guiding star and more essential for the development of applications as the role of cloud computing and microservices, which are more feasible to understand, deploy, and scale. One of today’s problems is finding the appropriate granularity and composition of microservices, both in existing software and in the literature and industry experience. This approach considers the Service Cutter, our method and tool framework, to identify the target applications. This tool is unique and effective in identifying microservices from the given system by considering the requirements of the target applications.

Keywords: Microservices, Microservice architecture, monolithic decomposition

1 Introduction

Microservices is a new architectural style that seeks to overcome the shortcomings of monolithic, monolithic architectures [13], in which the application logic is encapsulated in big deployment units. The most widely adopted definition of a microservices architecture is an approach for developing a single application as a set of small services, each running in its own process and communicating with lightweight inter-process communication; often a RESTful API [9]. In contrast to monolithic microservices, service independence is achieved, which can be developed using different technologies, etc. [5].

Although microservices can be seen as an evolution of Service-Oriented Architecture (SOA), they are (usually) different regarding scaling and governance issues, which is considered more often than less often [1]; instead of having scale at a coarse-grained level, Micr...
DECOMPOSITION TO MICROSERVICES

- MVC-based [Levcovitz2016]
- resource based [Levcovitz2016, Mazlami2017, Gysel2016]
- metrics-based, source code analysis (k-clustering) [Mazlami2017, Gysel2016, Selmadji2018]
- team structure [Mazlami2017]
- interface analysis (semantic) [Baresi2017]
LOCATING REFACTORING OPPORTUNITIES?

- MVC-based [Levcovitz2016]
- resource based [Levcovitz2016, Mazlami2017, Gysel2016]
- metrics-based, source code analysis (k-clustering) [Mazlami2017, Gysel2016, Selmadji2018]
- team structure [Mazlami2017]
- interface analysis (semantic) [Baresi2017]
LOCATING REFACTORING OPPORTUNITIES
Anna Maria Eilertsen

University of Bergen, Norway

https://annaei.github.io/

@sowhow

anna.eilertsen@uib.no
REFERENCES

[Gysel2016]
Service Cutter: A Systematic Approach to Service Decomposition
Michael Gysel1, Lukas Kölbener1, Wolfgang Giensche2
and Olaf Zimmermann1
IFIP International Federation for Information Processing 2016

[Mazlami2017]
Extraction of Microservices from Monolithic Software Architectures
Genc Mazlami, Ju’rgen Cito, Philipp Leitner
2017 IEEE 24th International Conference on Web Services

[Selmadji2018]
Re-architecting OO Software into Microservices A Quality-Centred Approach
Anfel Selmadji(B), Abdelhak-Djamel Serial, Hinde Lilia Bouziane, Christophe Dony, and Rahina Oumarou Mahamane
IFIP International Federation for Information Processing 2018

[Baresi2017]
Microservices Identification through Interface Analysis
Luciano Baresi1, Martin Garriga1, and Alan De Renzis2
Conference Paper · September 2017

[Levcovitz2016]

[Yarygina2018]