Bachelor's Proposal

COMMIT FEATURE INTERACTIONS

SIMON STEUER
(2579492)

May 15, 2023

Advisor:
Sebastian Böhm Chair of Software Engineering

Examiners:
Prof. Dr. Sven Apel Chair of Software Engineering

Chair of Software Engineering
Saarland Informatics Campus
Saarland University

PRESENTATION ABSTRACT

Short summary of the contents in English…a great guide by Kent Beck how to write good abstracts can be found here:

https://plg.uwaterloo.ca/~migod/research/beckOOPSLA.html

INTRODUCTION

Goal of this Thesis

The primary focus of this thesis is to gain an overview of how Commits interact with Features in Software Projects. Our goal is to lay basic groundwork regarding this subject, while leaving more detailed questions to future research. As previously mentioned we investigate two types, strutctural and dataflow-based Commit-Feature Interactions. While using both types separately can already answer many research questions, we will also show applications utilizing a combination of both.

Overview

BACKGROUND

Commit Feature Interactions
Broadly speaking Commits interact with Features, when one affects the other in any way imagineable. Obviously there is more than one such type in a Software Project. One type that we won't focus on, but is still worth mentioning, are control flow interactions, which occur when a commit decides wether parts of a feature get executed and vice versa. We will instead lay our focus on two other types, which we will now discuss in detail.
Structural Interactions
We define structural Commit-Feature Interactions as the occurence of syntactical overlap between commits and features. The syntax of a commit is comprised of commit code, namely code that was last changed by said commit. Feature syntax is defined in the same way as being comprised of feature code. Identifying feature code is more complex however, as it is any code whose execution depends on a feature being active or directly uses some feature variables. Now, we say that a commit and a feature interact structurally when commit code is part of feature code and vice versa.
Dataflow-based Interactions
We define dataflow-based Commit-Feature Interactions as data that was changed by a commit being accessed by a feature later in the program flow and vice versa. For programmers it can be difficult to be aware of these dependencies as they might be hidden and span over several files. We provide software that can automatically discover these, which can aid a programmer's ability to find errors and remove bugs. For instance detecting the latest commits that might have caused a bug in a certain feature can be enhanced by looking at its recent datflow-based Commit-Feature Interactions. In this thesis, we will research the direction of Commit -> Feature, e.g. a Commit affecting a Feature through dataflow. This direction promises more use-cases, like the example shown above, and interesting Research Questions.

RELATED WORK

Sattler et al. [2] combined low-level program analysis and high-level repository mining techniques in the form of Commit Interactions. They computed Commit Interactions using mapped repository information and a static taint analysis. They can be utilized to answer practical as well as scientific questions, which neither analysis can answer on their one.

What use are Commit Interactions
How were they computed
what are results

Lillack et al. [1] first implemented functionality to track Feature taints along program flow...

EXAMPLE CHAPTER

This chapter gives you some examples how to include graphics, create tables, or include code listings. Examples on how to cite papers from the literature can be found in .

Graphics

In Figure 1, we give a small example how to insert and reference a figure.

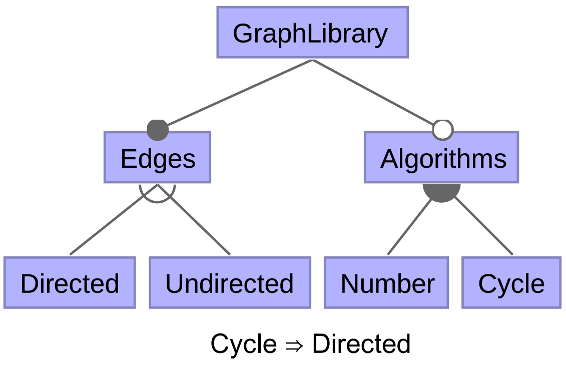

Figure 1: A feature model representing a graph product line

Tables

Table 1 shows the result of a simple tabular environment.

Table 1: Mapping a feature model to a propositional formula

Group Type	Propositional Formula
And	$(P \Rightarrow C_{k_1} \wedge \ldots \wedge C_{k_m}) \wedge (C_1 \vee \ldots \vee C_n \Rightarrow P)$
Or	$P = C_1 \vee \ldots \vee C_n$
Alternative	$(P = C_1 \vee \ldots \vee C_n) \wedge \text{atmost1}(C_1, \ldots, C_n)$

METHODOLOGY

The detection of structural as well as dataflow-based Commit-Feature Interactions is implemented in VaRA. VaRA offers two main functionalities for this. The first one is the detection of feature- and commit-code, which is accomplished by being able to receive all commits and features an llvm-IR instruction belongs to. Thus we can collect all structural Commit-Feature Interactions by iterating over all instructions in the code space. On instruction level we save every combination of commits and features as a CFI. It follows that, in order for an instruction to have a single interaction, it needs to be part of at least one commit as well as one feature. Besides that VaRA is also able to track taints of values along program flow, where taints essentially carry information on which commit or feature affected that specific value. Similarly to structural Interactions, dataflow-based Commit-Feature Interactions are iteratively collected on instruction level. In this case we speak of an dataflow interaction when an instruction both has either a commit or feature taint and belongs to either a commit or feature region. Consequently this instruction uses a value that was changed by a commit or feature earlier in the program while constituting feature or commit code. For our research we will examine numerous software projects to get a wide range of reference data, as Commit-Feature Interactions might vary greatly between different code spacess. Accordingly, the VaRA-Tool-Suite was extended making it possible to generate a report comprising all found CFIs of an according type in a software project. This will aid us in examining several Software Projects to gain sufficient and sensible data about Commit Feature Interactions. The created reports will also be evaluated VaRA-Tool-Suite, which offers support to process statstics of the generated data.

Research Questions

RQ1: What are the characteristics of structural Commit-Feature Interactions?

How many commits does a feature interact with structurally? In how many Features can we find a certain commit? These and many more questions surround the development process of Features and Software Projects in general, for which we aim to present the statistical evidence found in our research. With the acquired the data, we can answer many interesting assumptions about said development process. For example one might assume that, because they generally change a specific functionality, commits mostly are feature-specific, e.g. only affect the code of a single feature. This can give further insight on what purpose commits serve in a Software Developer's work.

RQ2: How do Commits interact with Features through Dataflow?

Previous research on dataflow interactions has been about interactions between commits and interactions between features. This has already shown the importance of research in the area and why further research is necessary. That's why we want to provide fist insights on the properties of dataflow-based Commit-Feature Interactions. One could form several statements about them based on a basic understanding of commits and features in Software Engineering. Showing wether these can be based on statiscal evidence will either secure or change the way we view their usage. For example one wouldn't normally want features to interact with large parts of the program, as they mostly serve a specific functionality, but to what extend does this get applied in coding? Additionally, analyzing the amount of features a commit usually interacts with through dataflow will display how common interactions really are. Since it's obvious that commits constituting code of a feature might be more likely to change data of said feature, we want to differintiate between these types of commits here. This can be accomplished with the help of our structural Commit-Feature Interaction analysis.

RQ3: How do Authors implement Features?

Usually there are many programmers working on the same Software Project, implementing different features, sometimes alone, sometimes with the help of colleagues. We want to shine some light on the exact statistics of this by using CFIs. One major question is how many authors implement a feature on average, where considering the size of the feature could help put this data into perspective. Digging deeper, we aim to gauge the amount of code each developer of a feature contributes. Is the distribution rather equal or does it vary greatly between programmers? The collected results could serve as advice for software companies on how to balance workload on to-be implemented features.

Operationalization

RQ1: What are the characteristics of structural Commit-Feature Interactions?

RQ2: How do Commits interact with Features through Dataflow?

RQ3: Focusing on Authors, what properties of Feature development can be discovered using Commit-Feature Interactions?

Expectations

In this section, discuss the results you expect to get from your evaluation.

Threats to Validity

In this section, discuss the threats to internal and external validity you have to be aware of during the evaluation.

CONCLUSION

BIBLIOGRAPHY

[1] Max Lillack, Christian Kästner, and Eric Bodden. "Tracking load-time configuration options." In: *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*. 2014, pp. 445–456.

[2] Florian Sattler, Sebastian Böhm, Philipp Dominik Schubert, Norbert Siegmund, and Sven Apel. *SEAL: Integrating Program Analysis and Repository Mining*. 2023.