Bachelor's Proposal

COMMIT FEATURE INTERACTIONS

SIMON STEUER
(2579492)

May 23, 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

UNIVERSITÄT
DES
SAARLANDES

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. Obviously there are many possibilities for such interactions in a Software Project. Since looking at all possible types would go beyond the scope of this work, we will instead concentrate on two of the most important ones. To properly define both types we will first introduce some necessary definitions.
Sattler et al. [3] explains that a program P can be viewed as a composition of instructions i_1, ..., $i_n \in$ P. We say that an instruction is part of a commit when the code that instruction belongs to was last changed by said commit.
Definition 1. Let $Commit(i)$ be a function that maps an IR instruction i $\in$ P to the commit it belongs to.
Secondly, we say that an instruction is part of a feature when its execution depends on said feature being activated or directly uses said feature variable. Naturally, there can exist more than one such feature.
Definition 2. Let $Features(i)$ be a function that maps an IR instruction i $\in$ P to all the features it is part of.
An instruction is tainted by a commit when it uses values that were changed by an instruction belonging to said commit earlier in the program.
Definition 3. Let $Taints(i)$ be a function that maps an IR instruction i $\in$ P to all the commits that taint it.

Structural Interactions

We speak of a structural commit-feature interaction when there is an occurence of syntactical overlap between commit- and feature-code.
Definition 4. Let $StructuralInteractions(c, f)$ be a function that returns all instructions i $\in$ P containing a structural interaction between commit c and feature f:
$StructuralInteractions(c, f) = \{\ i \mid c \in Commit(i) \land f \in Features(i)\ \}$
Consequently we say that a commit c and a feature f interact structurally, if and only if $StructuralInteractions(c, f) \geq 1$.

```
1. int calc(int val) {                          ▷ d93dj4gr
2.     int ret = val + 5;                       ▷ 7shd28dj
3.     if (FeatureDouble) {                     ▷ fu3w17ds    ▷ FeatureDouble
4.         ret = ret * 2;                       ▷ fu3w17ds    ▷ FeatureDouble
5.     }                                        ▷ fu3w17ds    ▷ FeatureDouble
6.     return ret;                              ▷ d93dj4gr
7. }                                            ▷ d93dj4gr
```

Listing 1: Commit Feature Interactions

The code example contains both structural as well as dataflow-based commit feature interactions. A structural CFI can be found between the commit with hash "fu3w17ds" and the "Double"-Feature, as commit fu3w17ds implements the functionality of FeatureDouble for this function. Commit 7shd28dj introduces a new variable that is later changed inside the "Double"-Feature. This accounts for a CFI through dataflow, as data that was changed by the aforementioned commit is used by a feature later on in the program.

Dataflow-based Interactions

We speak of dataflow-based commit-feature interactions when an instruction that is part of a feature uses values that were changed by an instruction belonging to a commit earlier in the program. **Definition 5.** Let $DataflowInteractions(c, f)$ be a function that returns all instructions i ∈ P containing a dataflow-based interaction between commit c and feature f:
$DataflowInteractions(c, f) = \{\, i \mid c \in Taint(i) \land f \in Features(i) \,\}$.
Similarly to structural interactions we say that a commit c and a feature f interact through dataflow, if and only if $DataflowInteractions(c, f) \geq 1$.
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 looking at a features's dataflow-based interactions with recent commits can help detect the commit that might have caused a bug in said feature.

RELATED WORK

Interactions between Features and Interactions between Commits have already been used to answer many research questions surrounding software projects. However investigating Feature Interactions has been around for a long time whereas examining Commit Interactions is a more recent phenomenon.
In an article published in 2023, Sattler et al. [3] analysed several open-source projects with their novel approach, SEAL. SEAL merges low-level data-flow with high-level repository information in the form of Commit Interactions. The paper shows the importance of a combination of low-level Program Analysis and high-level Repository Mining techniques by discussing research problems that neither analysis can answer on its own. For example SEAL is able to detect commits that are central in the dependency structure of a program. This was used to identify small commits affecting central code that would normally not be considered impactful to a program. Furthermore they investigated author interactions at a dataflow level with the help of commit interactions. Thus they can identify interactions between developers

that cannot be detected by a purely syntactical approach. They found that, especially in smaller projects, there often exists one main developer authoring the majority of commits and thus, logically, accounting for most author interactions. It was also explained how SEAL makes it possible to relate occurences of bad programming practices to developers. This is accomplished by SEAL enriching program analyses with computed repository information.
Lillack et al. [2] first implemented functionality to automatically track load-time configuration options along program flow. Said configuration options can be viewed analogously to feature variables in our research. Their analysis tool Lotrack can detect which features, here configuration options, must be activated in order for certain code segments to be executed. They evaluated Lotrack on numerous real-world Android and Java applications and observed a high accuracy for the predicted code execution constraints.
Referencing this paper Kolesnikov et al. [1] published a case study on the relation of external and internal feature interactions. Internal feature interactions are control-flow feature interactions that can be detected through static program analysis as mentioned above. They concluded that considering internal feature interactions could potentially help predict external, performance feature interactions.

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 and display 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] Sergiy Kolesnikov, Norbert Siegmund, Christian Kästner, and Sven Apel. "On the relation of external and internal feature interactions: A case study." In: *arXiv preprint arXiv:1712.07440* (2017).

[2] 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.

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