Mining Github Repositories

Joakim Leed, jolee18@student.sdu.dk
OnlyFans Hansen, ohans21@student.sdu.dk
SpankBanko, emspa15@ßtudent.sdu.dk Jectho, jetof20@student.sdu.dk

March 14, 2025

Introduction

- What field are we in
- What problems is the field facing
- Any other important points for contextualizing this project
- What are our contributions

INSERT introduction to the general field of mining software repositories

A multitude of information on bugs occurance and their fixes can be deduced from software repositories. This information have been used to categorize bugs and patches and further to evaluate their mutual relevance. The use cases are expanded with the improvements of automated program repair (APR) techniques. The APR approaches are used to automatically generate a patch solution to a bug-fix. In contrary this paper presents a tool to infer bug-types based on a given patch automatically. The solution is a dataset for mapping bug type to code patch semantics and frequencies. The dataset enables users to evaluate and identify statistical information for a specific patch. The mapping dataset is constructed based on historical data from mining software repositories. The targeted mined repositories consists of popular open source python projects. A rule-based approach is leveraged to analyze the data by applying a simple regex-based filtering on the commit messages to extract bug-fix patches and identify the related bugs.

Analysis

- The reasons why a rich data set would benefit the field
- Research questions
- Our plan to answer these research questions
-

Design

- Challenges to solve the issues
- How we address these challenges
- Diagrams of our data stages
- Design of our resulting rich data matrix

Implementation

- Overview of the implementation steps
- Highlights constraints of our implementations (e.g we shouldn't scrape too often from github)
- Share experiences of discovered pitfalls in the implementation
- Briefly document that our implementation works and where the code/dockerfile can be found and how to execute it

Related works

- Brief summary of related works
- Draw similarities between the related works and our approahc
- Postulate how our data could improve the related works
- automatically retrieve commit histories
- extract metadata
- rule-based commit analysis
  - regex-based filtering to identify bug type
  - manual analysis to classify filtered commit messaged to bug type
- use diff tools or AST parsers to analyze code changes in corresponding patches

PrevaRank a heuristic straight forward approach to rank patches for bug fixes in relation to historical proven solutions. A similar ranking solution could be developed with machine learning algorithms. However it is lightweight and effective and can be implemented as part of the post-processing step of any APR pipeline without considerable overhead. It showcased an improvement of 29% from outside to inside the top 3-ranks relative to the original tool. And only a 2% deterioration of ranking correct patches from inside to outside the top-3 ranks.

PrevaRank (Bhuiyan et al., 2024) ranks plausible patches for automated program repair (APR) based on their similarity to historical fixes, and then improves the prioritization of correct patches. This project differ as it does not focus on automated patch ranking, however both approaches leverage commit histories of targeted git repositories to extract relevant bug-fix patterns.

Our project focuses on what instead of APR-generated patches? deterministic heuristics? static code analysis? could PrevaRank be used to enhance our rule-based approach for example by utizing its insights into historical fix frequencies by incorporating patterns of common bug resolutions?

Classification of bugs into categories

Conclussion

- summarize:
  - the field and its issues
  - How we contribute with our data set
  - Shortcomings of our data set
  - Where to go next