CS202: Software Tools and Techniques for CSE

Lab Report 2

Pathan Mohammad Rashid (22110187)

February 2025

Abstract

This lab report documents the process of mining software repositories for bug fixes using the MineCPP tool and analyzing real-world open-source projects. The study focuses on extracting and evaluating bug-fix patterns, commit structures, and the overall impact on software maintenance. In this report, detailed methodology, observations, and discussions are provided to highlight the insights gained from the experiment.

Introduction

Objective: The objective of this lab is to mine software repositories for bug fixes using MineCPP and analyze real-world OSS projects to understand bug-fix patterns and their impact on software maintenance.

Setup and Tools: For this lab, I utilized MineCPP on an Ubuntu system running in an Oracle VM. The primary repositories analyzed include:

- **Tesseract-OCR/tesseract** (GitHub link)
- **conference-deadline (trial)** (GitHub link)
- **SensAI (trial)** (GitHub link)

These repositories were chosen using the SEART GitHub Search Engine, based on their star ratings and relevance to software maintenance and AI applications.

Methodology and Execution

The lab was divided into several key phases, each aimed at ensuring a systematic approach to repository mining and analysis.

1. Software Tool Setup

- Installed MineCPP via pip along with its necessary dependencies.

- Spent considerable time resolving version conflicts and module errors—this process took nearly one day, underscoring the importance of proper dependency management.

2. Repository Selection and Criteria

- Utilized the SEART GitHub Search Engine to filter repositories with star ratings between 50k-100k.
- Repositories were further screened for their relevance to software maintenance issues and AI applications.
- The selected repositories were then queued for analysis using MineCPP.

3. Running MineCPP The following commands were executed to mine each repository:

```
# Installing MineCPP
$ pip install minecpp

# Running MineCPP on selected repositories
$ minecpp -u https://github.com/tesseract-ocr/tesseract
$ minecpp -u https://github.com/abhshkdz/conference-deadlines
$ minecpp -u https://github.com/jbennet/SensAI
```

4. Bug Type Analysis

- Analyzed 15 bug-fix pairs for each repository using descriptions generated by an LLM.
- Each bug-fix pair was categorized with labels: `TOOL`, `DEV`, `BOTH`, or `NONE`, indicating the origin of the fix.

Results and Analysis

The results of the mining process provided several key insights into the repositories analyzed.

1. Commit Analysis

- The repositories exhibited distinct commit patterns. For instance, **Tesseract-OCR/tesseract** showed frequent minor commits associated with rapid bug fixes, while **SensAI** demonstrated fewer, but larger, commits.
- Detailed analysis of commit history revealed that repositories with high commit frequencies tend to adopt continuous integration practices, which aid in rapid debugging.

2. Visualization and Metrics

- The codebase metrics, particularly within the `src/` directories, showed significant coding efforts.

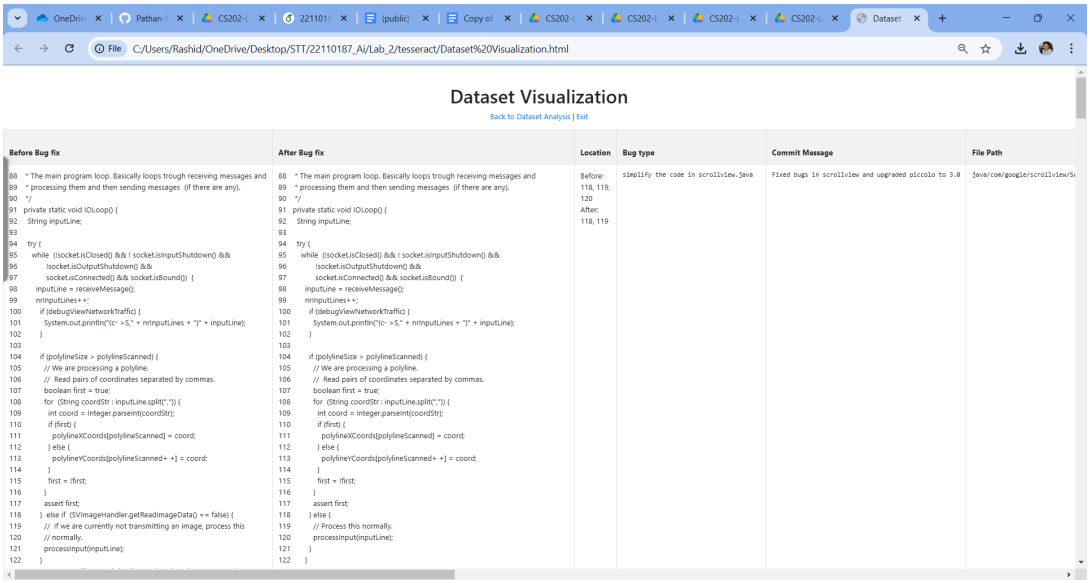

Figure 1: Commit History Overview

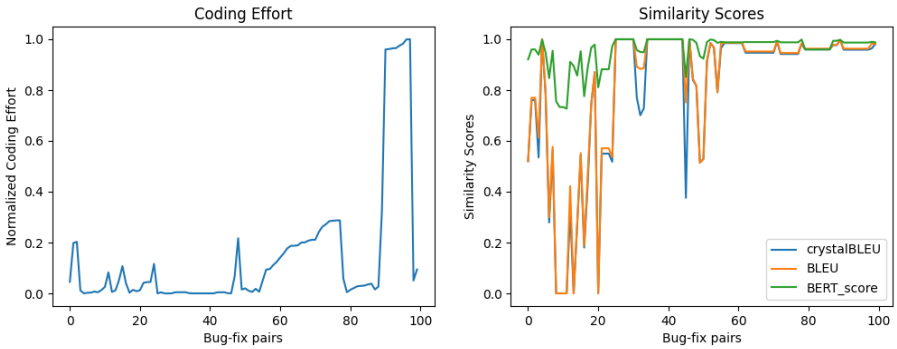

Figure 2: Analysis after 100 Commits

- Visualizations across different commit intervals (100, 500, 1000, and maximum commits) indicated that major bug fixes correlated with large-scale commits. This observation suggests that critical issues were often bundled together rather than addressed incrementally.

Observations: The results indicate a couple of notable trends:

- **Variation in Commit Behavior:** Repositories with higher activity demonstrated more frequent, but smaller, bug-fix commits. In contrast, those with lower activity tended to accumulate fixes in larger, more consolidated commits.

- **Impact on Software Quality:** The visual metrics suggest that repositories employing continuous integration and automated testing frameworks have a more structured approach to handling bugs, leading to more predictable commit patterns.

- **Correlation with Repository Size:** Larger repositories showed significant variations in commit frequency, which could indicate a more complex development process and a higher likelihood of bundled bug fixes.

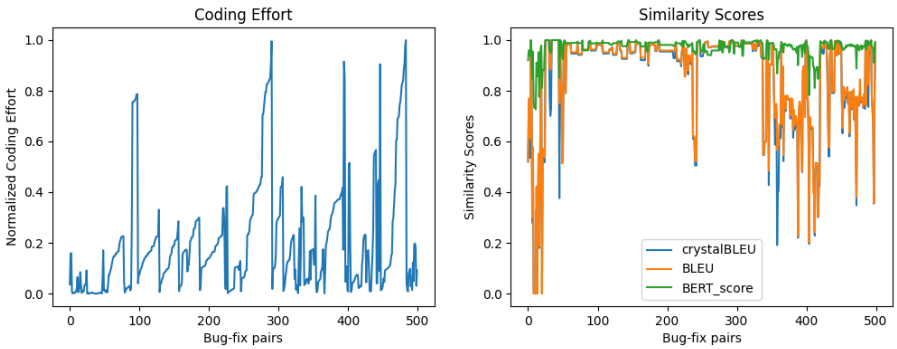

Figure 3: Analysis after 500 Commits

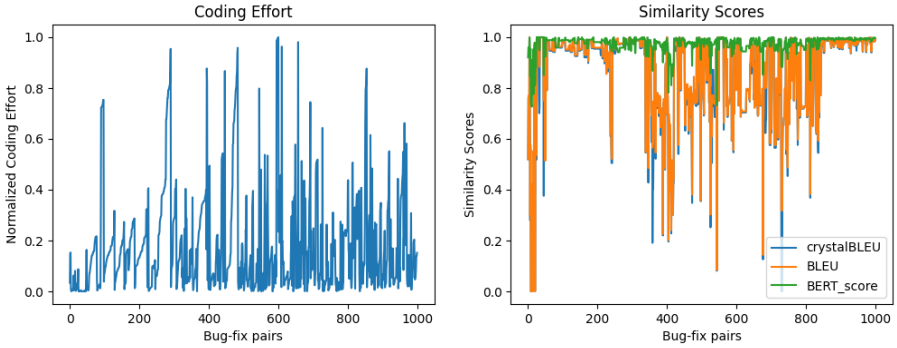

Figure 4: Analysis after 1000 Commits

Discussion and Conclusion

Challenges Faced:

- Resolving version conflicts during MineCPP installation was time-consuming.
- Large repositories took significantly longer to process, affecting overall runtime.

Lessons Learned:

- The importance of maintaining well-documented repositories is paramount, as it directly impacts the efficiency of bug-fix mining.
- Automated mining tools like MineCPP can reveal hidden patterns in code evolution, which is crucial for long-term software maintenance.

Final Thoughts: This lab provided valuable insights into the dynamics of bug fixes in large-scale OSS projects. The structured analysis of commit histories and bug-fix patterns highlights the importance of continuous integration and robust version control practices. Such tools not only assist in maintaining code quality but also offer a window into the evolution of software projects over time.

References and Resources

- **Full Report, Results, and Analysis:** Drive Link
- **Code Repository:** GitHub Repository

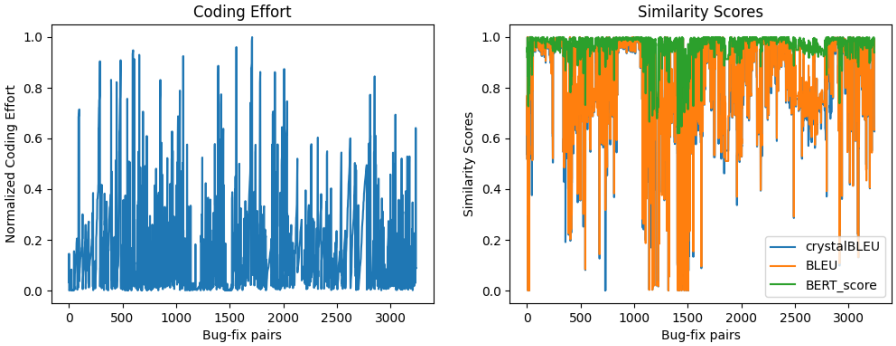

Figure 5: Analysis at Maximum Commit Count