A Study of Conventional Commits – (SCC)

— Proposal for a Bachelor's Thesis —

Anna Freidl
Advised by: Alexander Schultheiß

Topic Description

In software development, effective communication and code maintainability are essential for successful projects [1]. Conventional commits ('CC') [2], a standardized message format for documenting code changes, provide a structured approach to improving code understanding and readability for both humans and automated tools. However, the practical adoption and tangible influence of CCs on software projects remains largely unexplored. This thesis aims to analyse the application consistency, distribution, and impact of conventional commits on software projects.

1 State of the art and preliminary work

The field of commit messaging practices is rich and varied, with roots tracing back to the early days of software engineering. In 1976, Swanson [3] established a basic framework by categorising software maintenance activities into three distinct types: corrective (fixing errors), adaptive (accommodating changes), and perfective (enhancing the software). This typology has proved to be remarkably durable, and subsequent research[4] [5] has consistently returned to these categories as a way of understanding and classifying the nature of changes to code. Current work in the field can be divided into 3 groups of general approaches: Categorizing Commits [5], Developing Commit Conventions [6], Automated Tools and Advanced Applications [7] [8].

1.1 Categorizing Commits

Building on Swanson's work[3], Mockus and Votta [5] took a closer look at the relationship between maintenance activities and commit messages. They identified three reasons for code changes that overlapped with Swanson's categories and thus identified specific keywords that often signalled the nature of a change. For example, terms such as "fix", "bug" and "error" often indicate corrective commits, while "add" and "new" often indicate adaptive changes. This research highlighted the potential for using commit messages as a valuable source of information about the reasons behind code modifications.

1.2 Developing Commit Conventions

Commit messages play a crucial role in software development, serving as a historical record of changes, facilitating collaboration between developers, and aiding in debugging and mainte-

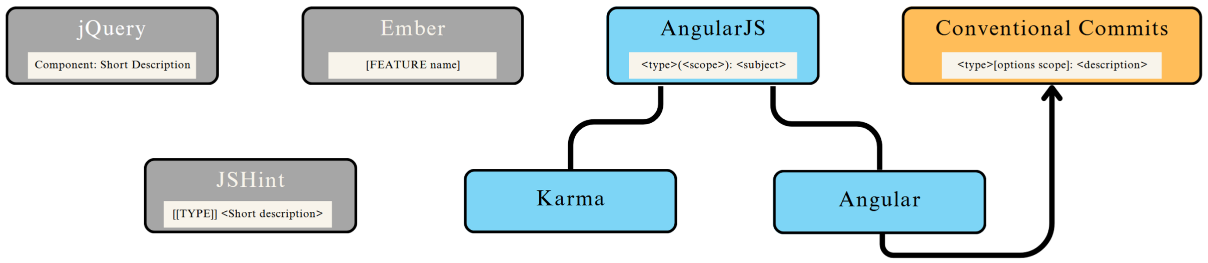
jQuery
Component: Short Description
Ember
[FEATURE name]
AngularJS
<type>(<scope>): <subject>
Conventional Commits
<type>[options scope]: <description>
JSHint
[[TYPE]] <Short description>
Karma
Angular

Figure 1: Commit Convention Variations

nance [9]. A well-written commit message provides context for why a change was made and what problem it solves. This not only makes it easier for others to understand the codebase, but also helps future developers understand the rationale behind past decisions [6].

Several projects have recognized the importance of clear and informative commit messages, and have developed their own guidelines and conventions. Early efforts, such as the Angular [10], jshint [11], and ember guidelines [12], laid the groundwork by suggesting structured formats, typically including a type, scope, and short summary. These formats aimed to make commit messages more concise, consistent, and easier to parse for both humans and machines.

Figure 1 illustrates the variation in these commit conventions. This evolution led to the development of the Conventional Commits specification, a more general and widely used standard that simplifies the Angular conventions in practice, and simply specifies the essentials of commit message conventions. The diversity of commit conventions is definitely a valid reason to have an official specification.

In their 2015 study Dyer et al. [13] found that about 14% of commit messages in over 23,000 OSS projects were completely blank, 66% of messages contained only a few words, and only 10% of commits contained messages with descriptive English sentences. Although there has been some research since then on how to improve the quality of commit messages [6], Ma and Lopez found in a recent study [14] that about 95% of commit messages in student projects had less than one descriptive sentence (less than 15 words), which is even worse than in Dyer et al.'s paper (90%) [13]. In this thesis, we will analyse the use of conventional commits in large open source projects to discover what, if any, changes are associated with the use of conventional commits.

1.3 Automated Tools and Advanced Applications

The emergence of standardised commit message formats has led to the development of automated tools such as semantic-release [7], which uses structured commit messages to automate semantic versioning and release processes. In addition, the Commit Conventions Specification has already been further developed in the security domain into so-called Security Commit Messages (SECOM) [8] to include security information that is essential for vulnerability detection and assessment based on commit messages. However, the effectiveness of such tools and their impact on real-world development workflows has not been fully explored.

This thesis examines how automated tools and conventional commits interact to demonstrate the need for a standardised structure for commit messages. By reviewing existing tools, the positive and negative impact of this standardised structure on the work of project managers is evaluated. In addition, we will analyse the changes in commit patterns before and after the switch to conventional commits, examining metrics such as the number of files changed, additions, deletions and the specific files affected. This analysis can be used to determine whether conventional commits result in more targeted and manageable changesets.

2 Objectives and work program

2.1 Anticipated total duration of the project

We anticipate a total duration of 4 months.

2.2 Objectives

This thesis aims to investigate the application, distribution and impact of conventional commits (CC) on software projects. By analyzing real-world code repositories and developer behavior, this study will address the following key objectives:

- **Analysing the consistency of CC applications and the distribution and frequency of commit types.**

  The overarching goal of this objective is to examine the real-world adoption of Conventional Commits (CC) within open source software repositories. It seeks to answer the following questions:

  - **Adoption Rate**: To what extent are CC guidelines being followed in different open source projects? Are there differences in adoption rates based on factors such as programming language, project size, or community activity?
  - **Consistency**: How consistently are CC guidelines applied within projects that have adopted them? Are there discrepancies in usage patterns among individual developers within the same project?
  - **Commit Type Distribution**: What is the frequency distribution of different commit types (e.g., bug fixes, feature additions, documentation updates) within projects? Are there patterns related to project type, development phase, or team size?
  - **Impact on Project Management**: How does the distribution of commit types influence project management tasks such as effort estimation and resource allocation? Is it possible to identify correlations between commit patterns and project outcomes?

  By addressing these questions, this thesis aims to shed light on the current state of CC adoption in the open source community and the factors that influence its effective use. The findings will contribute to a deeper understanding of the practical implications of

CCs, and inform the development of better tools, guidelines and practices to promote consistent and effective adoption of this standardised commit message format.

- **Change in commits before and after the transition to conventional commits.**

  This objective aims to provide a detailed understanding of how the transition to conventional commits affects the structure and content of commit messages and commited changes. By analysing repositories that have explicitly adopted conventional commits, we will compare commit patterns before and after the transition.

  Specifically, this investigation will address the following questions

  - How does the average number of files changed per commit change after the adoption of conventional commits? Are commits more focused, with fewer files changed per commit, or do they become broader in scope?
  - Does the number of insertions and deletions per commit increase, decrease, or stay the same after the switch? This will help determine whether conventional commits encourage more granular changes, or simply a change in the way changes are grouped within commits.
  - Are there specific types of files (e.g. source code, configuration files, documentation) that are more or less likely to be modified after the introduction of conventional commits? This might reveal changes in development practices or priorities.
  - Does the adoption of conventional commits lead to more frequent commits, or does it simply change the way changes are structured within existing commit frequencies?

  By answering these questions, this thesis aims to quantify the tangible impact of conventional commits on the way developers structure and document their code changes. This understanding can inform best practices for adopting conventional commits, help teams assess the potential benefits, and guide the development of tools to support the transition to structured commit messages.

By achieving these goals, this research will contribute to a deeper understanding of the practical effectiveness of CC in software development. The results can inform the development of tools and guidelines to promote consistent and effective adoption of CC, ultimately leading to better communication and project management practices.

2.2.1 This research will use the following key techniques:

- **Automated Commit Message Parsing**: Using scripts to automatically extract commit messages from a large database of open source repositories. These scripts will work with the Git version control system, and take care of potential variations in commit message formatting.

- **Conventional Commit Adherence Assessment**: The extracted commit messages will be analysed using tools such as the conventional-commits library to assess their compliance with the conventional commits specification. This includes identifying commit types, scope and other relevant metadata.

- **Statistical analysis**: Descriptive and inferential statistics will be used to analyse the collected data, identify trends and patterns in CC usage, and test hypotheses about their impact on project management and code maintainability.
- **Data visualisation**: Visualisations such as bar charts, histograms and scatter plots will be created to effectively communicate the results of the statistical analysis and highlight key findings related to CC adoption and its consequences.

2.2.2 Dataset Construction

The following multi-step process is used to construct the dataset:

- **Identifying Popular Programming Languages**
  The 10 most used programming languages are determined based on established indices that determine the popularity of languages based on factors such as stars on projects, pull request and issues. One possible programme for this task is Githut [15].
- **Selecting Open-Source Projects:**
  At least 200 leading open source projects will be selected for each language identified. Selection criteria will include factors such as number of stars, forks, contributors and recent activity. This approach will ensure that the selected projects are active, well maintained and have a significant user base.
- **Clean up dataset**
  Clean the data set of duplicate projects and bot-generated commits as well as non-English commit messages.

2.3 Work programme incl. proposed research methods

WP1 Data Collection and Preparation.

Challenges and motivation.

- The main challenge lies in identifying a diverse and representative set of open-source repositories that accurately reflect the variety of programming languages and development practices in use.
- Ensuring the quality and reliability of the collected data is crucial, as it will form the foundation for all subsequent analyses.

Research plan and individual research tasks. see 2.2.2 Dataset Construction

WP2 Conventional Commits Adoption and Consistency Analysis.

Challenges and motivation.

- Developing a reliable method to automatically assess CC adherence is crucial, as manual inspection of a large dataset is not realistic.

		W1-2	W3-4	W5-6	W7-8	W9-10	W11-12	W13-14	W15-16
WP1	**Data Collection**								
T 1a	Language Identification	●	•						
T 1b	Project Selection	●	•						
T 1c	Dataset Cleaning			●	•				
WP2	**CC Analysis**								
T 2a	Tool Exploration		●	•					
T 2b	Script Development			●	•				
T 2c	Quantitative Analysis				●	●			
T 2d	Qualitative Analysis				●	●			
WP3	**Commit Change Analysis**								
T 3a	Project Identification			●	•				
T 3b	Commit Data Collection				●	•			
T 3c	Changes Analysis					●	●	•	
WP4	**Thesis Writing**								
T 4a	Literature Research	•	•						
T 4b	Literature Review	•	•	•					
T 4c	Results Interpretation & Writing	•	•	•	•	•	•	●	●
T 4d	Drafting and Finalization						•	●	●

● major focus on this WP • minor focus on this WP

Table 1: Gantt Chart of Work Packages

Research plan and individual research tasks.

- Explore existing tools and libraries for assessing CC compliance.
- Develop custom scripts or adapt existing tools to handle potential variations in commit message formatting.
- Analyse the extracted commit messages to quantify the adoption rate of CCs in the selected repositories.
- Perform a qualitative analysis of commit messages that deviate from the CC policy, to identify common patterns and reasons for non-compliance.

WP3 Analyzing Changes in Commit Practices Before and After CC Adoption.

Challenges and motivation.

- Building an accurate classification model for commit types requires careful feature engineering and selection.
- Establishing a clear link between commit type distribution and project management outcomes can be challenging due to the complex nature of software development.

Research plan and individual research tasks.

- Identify projects that have publicly documented their transition to Conventional Commits.
- Collect commit data from these projects before and after the transition point.

- Analyze changes in commit patterns using descriptive statistics and hypothesis testing.

WP4 Thesis Writing and Results Integration.

- Literature Review Integration: incorporate relevant findings from WP1, WP2, and WP3 into the thesis draft.
- Results Interpretation: Analyze and interpret the results of WP2 and WP3, highlighting key findings and their implications.
- Thesis Structure Development: Create a clear outline for the thesis, including introduction, literature review, methodology, results, discussion, and conclusion.
- Drafting and Revision: Write the thesis draft, incorporating feedback
- Finalization: Polish the thesis, ensuring proper formatting and referencing.

2.4 Data handling

All results from the thesis experiments will be publicly available, including all data and all code. In addition, the thesis source code will be made available through a public repository (e.g. github.com), such that additional researchers can use it and contribute to the project. Each published experiment will be made replicable through included automated scripts.

3 Bibliography

References

[1] Minna Pikkarainen, J. Haikara, O. Salo, Pekka Abrahamsson, and J. Still. The impact of agile practices on communication in software development. *Empirical Software Engineering*, 13:303–337, 06 2008.

[2] Conventional commits - a specification for commit messages, 2024.

[3] E. Burton Swanson. The dimensions of maintenance. In *Proceedings of the 2nd International Conference on Software Engineering*, ICSE '76, page 492–497, Washington, DC, USA, 1976. IEEE Computer Society Press.

[4] Jian Yi David Lee and Hai Leong Chieu. Co-training for commit classification. In *WNUT*, 2021.

[5] Mockus and Votta. Identifying reasons for software changes using historic databases. In *Proceedings 2000 International Conference on Software Maintenance*, pages 120–130, 2000.

[6] Yingchen Tian, Yuxia Zhang, Klaas-Jan Stol, Lin Jiang, and Hui Liu. What makes a good commit message? In *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, page 2389–2401, New York, NY, USA, 2022. Association for Computing Machinery.

[7] GitHub. Git automatic semantic versioning. `https://github.com/marketplace/actions/git-automatic-semantic-versioning`, 2024.

[8] Sofia Reis, Rui Abreu, Hakan Erdogmus, and Corina Păsăreanu. Secom: Towards a convention for security commit messages. In *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, pages 764–765, 2022.

[9] Nazatul Nurlisa Zolkifli, Amir Ngah, and Aziz Deraman. Version control system: A review. *Procedia Computer Science*, 135:408–415, 2018. The 3rd International Conference on Computer Science and Computational Intelligence (ICCSCI 2018) : Empowering Smart Technology in Digital Era for a Better Life.

[10] Angular Team. Contributing, 2024.

[11] JSHint Team. Contributing, 2024.

[12] Ember.js Team. Contributing: Commit tagging, 2024.

[13] Robert Dyer, Hoan Nguyen, Hridesh Rajan, and Nguyen Tien. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. pages 422–431, 05 2013.

[14] Iris Ma and Cristina Lopes. Improving the quality of commit messages in students' projects. 04 2023.

[15] Midnight. Pull requests for 2024, 2024.