The Life of Software Features

An Inspection of Marked Feature Requests in Marlin

van der Hofstad, Aron F.P.

Award date:
2024

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science
Software Engineering and Technology

The Life of Software Features

An Inspection of Marked Feature Requests in Marlin

Master Thesis

Aron van der Hofstad

Supervisor:
Dr.-Ing J. Krüger

Assessment committee:
Dr.-Ing J. Krüger
Dr.ir. L. Cleophas
Dr. rer. nat. C. Dubslaff

Eindhoven, October 2024

Abstract

Software programs consist of various features that define their functional and non-functional properties. Given the widespread nature of software features, it becomes critical to understand their life cycle to ensure success and reliability. By examining the common phases a feature goes through in its life cycle, decisions can be made early on how to prevent potential failures. Additionally, having a clear and concise definition of a feature can significantly improve documentation and feature management in the future. In this thesis, we present an in-depth qualitative case study on 189 labeled software features from the Marlin Firmware repository. Through analyzing the source code and extracting surrounding data from the repository, we identified the life cycle phases of labeled features as well as various edge cases. Furthermore, we identified instances of pull requests that fit our criteria for a feature but were not labeled accordingly. Our findings provide a detailed understanding of the life cycle of labeled software features within the Marlin project, offering guidance for future research on feature life cycles in a more general scope. Additionally, our study sheds light on the accuracy and consistency of maintainers in appropriately labeling features, which is crucial for the effective management and evolution of software projects.

Preface

When I was finishing high school, I hade no real clue what I wanted to do. So, on a whim, I chose computer science at the Technical University of Eindhoven. I did this for four reasons: I was good at math, but not that good at math. I have always liked computers. One of my close friends, Rowin, had already chosen it the year before, and it seemed fun. Lastly, it was close by. This choice, made on a whim, has led me on a 7-year adventure full of both hardships and a lot of fun. If you had asked me when I was young what I wanted to become, I would have told you a car mechanic. But my journey in life has taken me in a vastly different direction.

I want to thank my supervisor, Jacob Krüger, for his support during this thesis, and the rest of my assessment committee for their valuable feedback and for reviewing my work. I also want to thank my friends and family who have supported me throughout all these years. And finally, most importantly, I want to thank my girlfriend, Evelyn. Without her mental support, I would never have been able to finish this journey.

Contents

Contents iv

List of Figures v

List of Tables vi

1 Introduction 1

2 Related Work 3
- 2.1 What is a Software Feature? . . . 3
- 2.2 The Life of a Feature . . . 3
- 2.3 Distributed Version Control . . . 4
- 2.4 Software Feature Location . . . 4
- 2.5 Highly-Configurable Software Systems . . . 5

3 Methodology 6
- 3.1 Research Questions . . . 6
- 3.2 Process of Selecting Repository . . . 7
- 3.3 Tagging Software Features . . . 7
- 3.4 Pull Request Feature Check . . . 11
- 3.5 Feature Tags . . . 12
- 3.6 Automatization . . . 13

4 Results and Discussion 17
- 4.1 Lessons Learned . . . 17
- 4.2 **RQ1:** Feature Life Cycle . . . 30
- 4.3 **RQ2:** Pull Requests Labels . . . 57

5 Threats to Validity 68

6 Conclusion and Future Work 69
- 6.1 Conclusion . . . 69
- 6.2 Future Work . . . 70

Bibliography 71

Appendix 73

A Marlin Label Documentation 74

B Additional Figures 80

C Additional Tables 84

List of Figures

3.1 Flowchart of the case study of Marlin. 8
3.2 Description of pull request #21255. 9
3.3 Pull request information of pull request #21255 10
3.4 Part of the change files of pull request #21255. 10
3.5 Changes made in the `cooler.cpp` file of pull request #21255. 11

4.1 Number of pull requests with a certain label 19
4.2 Number of pull requests in a certain label group 20
4.3 Number of pull requests in a certain group 21
4.4 Number of pull requests submitted per year with and without labels. 22
4.5 Total number of pull requests and number of bug fixes submitted per month 22
4.6 Total number of pull requests and number of New Features submitted per month. . 28
4.7 Average number of commits associated with labeled software features 33
4.8 DFG of pull request #16452. 37
4.9 DFG of pull request #18071. 39
4.10 DFG of pull request #23112. 41
4.11 DFG of pull request #22790. 42
4.12 DFG of all pull requests from the sample set. 43
4.13 DFG of all pull requests from the sample set. 44
4.14 Footprint of all pull request from the sample set. 45
4.15 DFG of pull requests with at most 9 commits associated with them. 46
4.16 Software Features by introduction date by number of commits associated with it. . 50

B.1 Pull request closed by an admin. 81
B.2 Proposed change shut down by the maintainer. 82
B.3 A simple change quickly accepted by the maintainer without the need for a label. . 83

List of Tables

4.1 Table showing the difference between GitHub and API 20
4.2 Percentage of total pull requests covered by `PR: New Feature` label. 25
4.3 Percentage of merged pull requests covered by the `PR: New Feature` label. 27
4.4 Labels used in addition for pull requests. 28
4.5 Labels used for the merged pull request with the `PR: New Feature` label. 31
4.6 Commit title with commit count. 32
4.7 Commit tags together with number of commits with that tag. 34
4.8 Pull requests together with number of commits with that tag. 35
4.9 Pull requests together with number of commit tags. 38
4.10 Commit cycle lengths in pull requests count. 45
4.11 Type of commit before `BugFix` tag, count and percentage. 47
4.12 Labels together with number of different tags for commits. 48
4.13 Labels together with number of different tags for commits. 49
4.14 Commits Count with Feature, Lines, Comments and Files. 50
4.15 Commits per Day with Feature, Lines, Comments and Files. 51
4.16 Time Between Creation and Merge Features. 53
4.17 Label and associated Pull Requests, Time and Comments. 55
4.18 All pull requests investigated and a comment about them. 61
4.19 Pull requests with the `C: LCD & Controllers` label that could be a feature. 62
4.20 Pull requests with the `C: Motion` label that could be a feature. 63
4.21 Pull requests with the `C: Peripherals` label that could be a feature. 64
4.22 Pull requests with the `A: STM32` label that could be a feature. 65
4.23 Pull requests with the `C: Calibration` label that could be a feature. 65
4.24 Pull requests with the `PR: Improvement` label that could be a feature. 66

C.1 Labels together with number of different tags for commits. 85
C.2 All pull request present in this report with their URL. 93
C.3 All pull requests together with number of commit tags. 97
C.4 All pull requests together with lines, comment count, files and commit count. 101

Chapter 1

Introduction

Every software program is composed of various features that define its functional and non-functional properties [2]. Since software features are a fundamental part of every system, it is essential to understand their life cycle—from conception and development to deployment and beyond. Studying the life cycle of software features can yield valuable insights into how features are created, implemented, maintained, and adapted over time. It also provides a deeper understanding of how features succeed, why they become obsolete, and the steps necessary for their continued relevance.

To gain deeper insights into the life cycle of software features, it is crucial to improve our empirical understanding of them. This includes identifying where data about these features can be found and how to process this data into meaningful information. Unfortunately, software features are often poorly documented [4], and as time passes, knowledge about them deteriorates. Their locations within the code base often become unclear and need to be rediscovered [17]. Open-source projects, particularly those hosted on platforms like GitHub[1], GitLab[2], and Bitbucket[3], offer rich sources of data for studying software features. These platforms provide tools such as pull requests, issues, and commits, which can be leveraged to gather and process feature-related data.

This thesis presents a qualitative case study on the life cycle of labeled software features in the Marlin firmware, a widely-used open-source firmware for 3D printers. By exploring Marlin's labeled features, this research aims to shed light on the life cycle of software features in a large, long-running open-source system. Our study offers guidance to developers on making informed decisions while developing and maintaining software features. We perform a thorough analysis of the labeled features in Marlin, tracking them throughout their life cycle by tagging all relevant commits.

In total, we investigated 189 labeled software features in Marlin, processing 2,956 commits and categorizing them with one of 11 custom tags. These commits serve as the foundation for analyzing the life cycle of each software feature. Marlin was chosen due to the availability of previous research on the platform and its complete development history, which is publicly accessible on GitHub. Our approach involved a manual review of each feature, examining its code and the changes introduced through various commits. We also generated statistics from the collected dataset and compared labeled pull requests with our definition of a software feature. Additionally, we analyzed 180 pull requests that were not labeled as features to determine whether they met our criteria for being considered features.

- We present the life cycles of 189 software features, spanning five years of releases.
- We conducted a comprehensive analysis of the Marlin repository to investigate labeled software features and labeling practices.

[1] https://github.com/
[2] https://about.gitlab.com/
[3] https://bitbucket.org/

- We compared pull requests against our definition of a software feature and discussed the results.

The remainder of this report is structured as follows: In chapter 2, we introduce the background and related work on the life cycle of labeled software features. In chapter 3, we outline the research questions, the process of selecting the repository, and the methodology used for gathering data and comparing pull requests. In chapter 4, we present and discuss our findings. In chapter 5, we address the threats to the validity of our research. Finally, in chapter 6, we conclude with a summary of our findings and suggest directions for future research.

Chapter 2

Related Work

As mentioned before, software features are present in every software program in use.In this chapter, we introduce the related work on the topic of software features and the online hosting of projects which contain such software features.

2.1 What is a Software Feature?

To understand the life of a software feature, it is first important to have a clear understanding of what a software feature is. Even though everyone interacts with software features, there is no single, concise answer to the question of what a software feature is. Many developers have their own definitions of what constitutes a feature, and even in academic writing, there is not one universally accepted answer.

One of the early attempts at defining a software feature is given by Kang et al. [19] as "a prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems.". A problem with this definition is that it only focuses on user-visible aspects, so anything that the user cannot see is not considered a software feature such as background calls to keep a database server up to date. Another definition is given by Bosch [5] as "a logical unit of behavior specified by a set of functional and non-functional requirements." This definition is beneficial because it covers both visible and non-user-visible software features, but it is not always the case that there are requirements specified to outline the behavior of a product, such as in open-source projects where only a small percentage of projects make use of software architecture documentation to note down their system [9]. And many more definitions can be found in literature for the definition of a software feature [8][33][18][28][23][15]. To arrive at a comprehensive definition, we will rely on the definition by Apel et al. [2], who has combined many previous definitions into one: "A feature is a characteristic or end-user-visible behavior of a software system. Features are used in product-line engineering to specify and communicate commonalities and differences among the products to stakeholders, and to guide structure, reuse, and variation across all phases of the software life cycle." In this definition, features are seen as both a characteristic of the system and a user-visible behavior, which addresses our concern that not all features are user-visible. It does not rely on the notion of requirements and it also defines the role of features in the context of a software product line.

2.2 The Life of a Feature

The life cycle of a software feature follows the following steps, feature development start, first development, evolution, servicing, phaseout and removal [12][3].

The starting point of a software feature is when either a user makes a request for it or when the developers internally decide that is necessary. This could be to anticipate the demands of the user or because the feature will make testing of the software easier or make the software more

modular to support different systems. Once the need for a feature is known the development of the feature starts. Once the minimum of a feature is developed it will be deployed to be tested out and incrementally added on until the full feature comes into existence. It is important that during this part feedback is collected and processed [11]. Once the feature is successfully deployed the next phase of its life starts, evolution. During the evolution phase the feature is incrementally improved to keep up with the wishes of the users. Over time the features starts to get outdated and most time spent on it is in maintenance to keep it function without improving its functionality. It could even be that the old features will start forming a barrier for the introduction of new features [24]. Then the last stage of the life of a feature is reached removal. At a certain point the cost and effort of maintaining an old feature out ways the usefulness of the feature and the decision has to be made to remove it from the software [27].

2.3 Distributed Version Control

Distributed version control (DVC) is a form of version control used in programming, in which the complete codebase, including its entire history, is mirrored on every developer's computer. The most popular DVC currently on the market is Git[1], an open-source project originally developed by Linus Torvalds for the Linux kernel. According to a survey conducted by Stack Overflow[2], Git has a market share of 93.9%.

Developers often utilize DVC services to host their projects. This allows developers from all over the world to collaborate effectively without the need for or knowledge of how to host their own Git server. Some of the popular DVC platforms include GitHub, GitLab, and Bitbucket. Of these different DVC systems, GitLab and Bitbucket are more centered on enterprise use, while GitHub is widely used by the open-source community as a means of hosting their projects for people to collaborate on[3].

An important feature that such platforms offer is the ability to report issues with the program, making the developer aware of problems or requests for particular features. These platforms also enable developers to publish pull requests, wherein developers can discuss proposed changes before they are merged into the main codebase.

Issues and pull requests provide a wealth of information, indicating which features get accepted into the project and which are rejected. The platforms also allow issues and pull requests to be labeled, better identifying the subject they cover.

2.4 Software Feature Location

To analyze Software Features, we first need information about them. For a typical product, this process would be straightforward: examine the product and its documentation to identify all the features present. However, with software features, this approach is not as simple. As previously discussed, there are many different definitions of what constitutes a software feature, and often, the documentation of software projects is either outdated or nonexistent [6]. Moreover, these documents typically do not track changes over time but focus only on what is present in the latest version of the software. The activity of software feature location is considered one of the most common and expensive activities in software engineering [32].

Due to these challenges, special techniques must be employed to identify software features within projects. There are two main types of software feature location techniques: Dynamic techniques, which collect data while the program is running, and static techniques, which collect data without running the code. Dynamic software feature location techniques offer precise and accurate results about the software program but are limited to its input, making generalization from these data potentially unsafe [20]. In contrast, while static methods can gather safe information when

[1] https://git-scm.com/
[2] https://survey.stackoverflow.co/2022/#section-version-control-version-control-systems
[3] https://survey.stackoverflow.co/2022/#section-version-control-version-control-platforms

used conservatively, they cannot determine properties of the software programs that only become visible during runtime.

A significant drawback of dynamic methods is their high demand for user input, requiring test cases to be prepared for each software program being tested.

Given the importance of Feature Location, extensive research has been conducted to increase the speed and efficiency of feature location through automation. However, studies have shown that manual feature location significantly surpasses automated feature location in terms of precision and F-measure, although automated feature location performs better in recall [30]. Recall measures the ability to retrieve software features out of all possible features, while precision assesses the accuracy of the identification. Thus, while automated feature location may identify a higher number of features, it also misclassifies more non-features as features compared to manual methods.

Given the current limitations of automated software feature location, utilizing such techniques for identifying more software features is beyond the scope of this thesis, as it is not yet automated enough to acquire a large dataset automatically.

2.5 Highly-Configurable Software Systems

Highly-Configurable Software Systems (HCSS) use variability mechanisms to alter configurability options. By altering these options features get enabled or disabled and the variant of the system is changed. So, by use of different configurations options different requirements can be fulfilled [35]. The most widely used way of implementing such variability mechanisms is through the use of `#ifdef` statements in combination with the `C preprocessor` [31]. By using the `#ifdef` statements different features of the system can be enabled or disabled. The system can support different platforms and operating systems. The downside of this system is that it is hard to fully understand the code because of all the obfuscation done by the preprocessor directives, and the C preprocessor removes all the code that is inside the blocks of disabled statements. When code of one feature is removed it has the possibility of breaking another feature. By removing code that both features were depended upon. This can happen because the directives of the different features are not correctly protected [26]. Because of this, a single feature can break the functionality of an entire configuration. And the larger a system grows, the more difficult it becomes to comprehend all the directives, if they are even directly visible. Features are easier to recognize when guarded by these directives, but features are also present in HCSS outside these directives in the core functionality of the program, which cannot be enabled or disabled. Such features can only be found by investigating the code and searching for behaviors that the system can execute.

Chapter 3

Methodology

In the previous chapter we have given our definition what a software feature is, where most open-source projects are hosted and the reasons they are hosted on such platforms and how software features are traditionally looked for in software programs. The goal of this thesis is to delve deeper into the life cycle of a software feature and attempt to extract useful information from the various stages of its life. With the information gathered it could be possible to detect positive patterns in the life cycle or even notice negative patterns and based on those recommendations can be given for what must be considered during the development and maintenance of software features. The life cycle of a software feature will be investigated by doing a qualitative case study on the Marlin Firmware Project[1] hosted on GitHub. We will focus on the life cycle of the features of Marlin and not their exact functionality.

In this thesis the utmost care has been taken to anonymize any personal information that is present in the data, however any pull request or issue that is mentioned in this thesis contains a link directing to the publicly accessible GitHub page of Marlin, on this page personal information of the users involved with are visible unless those users decide to delete their information from GitHub.

3.1 Research Questions

The main goal of this thesis is to investigate the life cycle of marked software features in the Marlin open-source software project. These goals are captured by the following research questions:

- *RQ1: What does the life cycle of a labeled software feature look like?*

  When a software feature is introduced into a project, it starts off with a clean slate. Over time other commits will impact the software feature and it will evolve. What does this evolution look like for labeled software features and what impact does it have on the software feature? Do software features retire or just endlessly develop further until they are unrecognizable from what they started as. And is this life cycle impacted by the properties of the software features at the moment of their introduction? This question is split up into sub-questions:

  - *RQ1.1: What is the life cycle of a labeled software feature?*
  - *RQ1.2: Are extensions of software features themselves also software features?*
  - *RQ1.3: What impact do the properties of a software feature have on its life cycle?*

- *RQ2: Are features accurately labeled by maintainers?*

  As seen in section 2.1 there are many different definitions for software features used by different people. So, are the pull requests that are labeled as a software feature actually

[1] https://marlinfw.org/

software features? Are pull requests labeled consistently or does this change depending on the maintainer? Insights into this could give valuable feedback for the open-source about how issues and pull requests are labeled and processed. This question is split up into two sub-questions:

- *RQ2.1: Do labeled features hold against our definition of a feature?*
- *RQ2.2: Do pull requests not labeled as a feature hold against our definition of a feature?*

We hope that by addressing these questions we will gain a deeper understanding of the life cycle of software features and project maintenance and with that be able to help developers in the future.

3.2 Process of Selecting Repository

For the selection of the repository, a list of possible repositories was made. We chose the top 100 list of most starred repositories as shown by Gitstar Ranking[2] as a starting point. We filtered this list to only contain software programs and not collections of books or other unrelated items. Not all remaining repositories are suitable to analyze. For example, the Facebook React repository technically does have a label for pull requests to label them as a feature request, but this label is never used. So only the repositories in which the features are labeled in the pull requests can be used. Because it is not feasible to apply software feature location methods on such a big dataset of various repositories within the time frame of the thesis.

We chose to focus on the Marlin Firmware GitHub project[3] for a variety of reasons. First and foremost, Marlin makes use of a clear labeling system to label the pull requests, so the pull requests that represent software features are easy to find. Second, the Marlin project is an open-source project in which many different people from professionals to amateurs alike work together not backed by a big company, this offers an interesting dynamic. Third, Marlin is a versatile piece of software that can control a variety of different hardware configurations. What started as only 3D-printing now is also able to control CNC machines and lasers. Marlin can be classified as a HCSS which allows users to customize the program behavior and create different variants of the software by selecting different configuration options that address different requirements [1]. The configuration of different features for different systems is reliant on being guarded by `#ifdef` directives of the preprocessor. This evolution means that a lot of software features must be introduced to achieve this. Fourth, a variety of other studies have already been performed on Marlin, so by this research we widen our knowledge about the Marlin project [22][36][34][21].

3.3 Tagging Software Features

In Figure 3.1 we can see the workflow of extracting the data about a software feature from a repository and creating a dataset. In the following subsections a step-by-step example will be given about how this works in practice.

3.3.1 Analysis of GitHub Page

First the software feature pull request is opened on GitHub. For this example we take pull request #21255[4] of the Marlin project. With as title: Cooler (for Laser) - M143, M193. We first read the description and comments of the pull request, so that we know what the software feature is about and how it is supposed to work. In Figure 3.2 we can see the GitHub page for this pull request. The pull requests in Marlin make use of a standard template. It starts with a description (indicated by (1)) in this description we can read what the idea of the pull request is and what

[2] https://gitstar-ranking.com/
[3] https://github.com/MarlinFirmware/Marlin/
[4] https://github.com/MarlinFirmware/Marlin/pull/21255

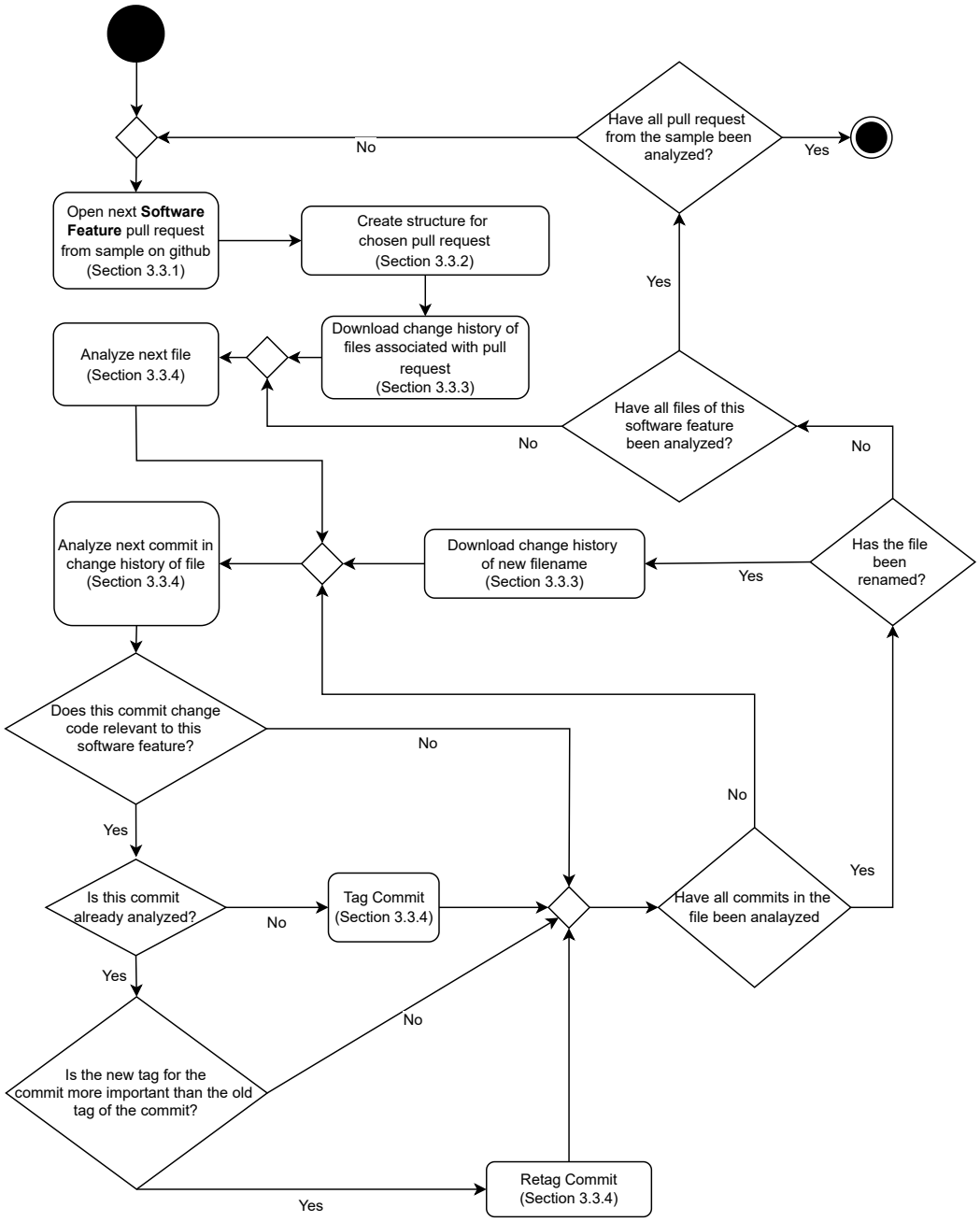

Figure 3.1: Flowchart of the case study of Marlin.

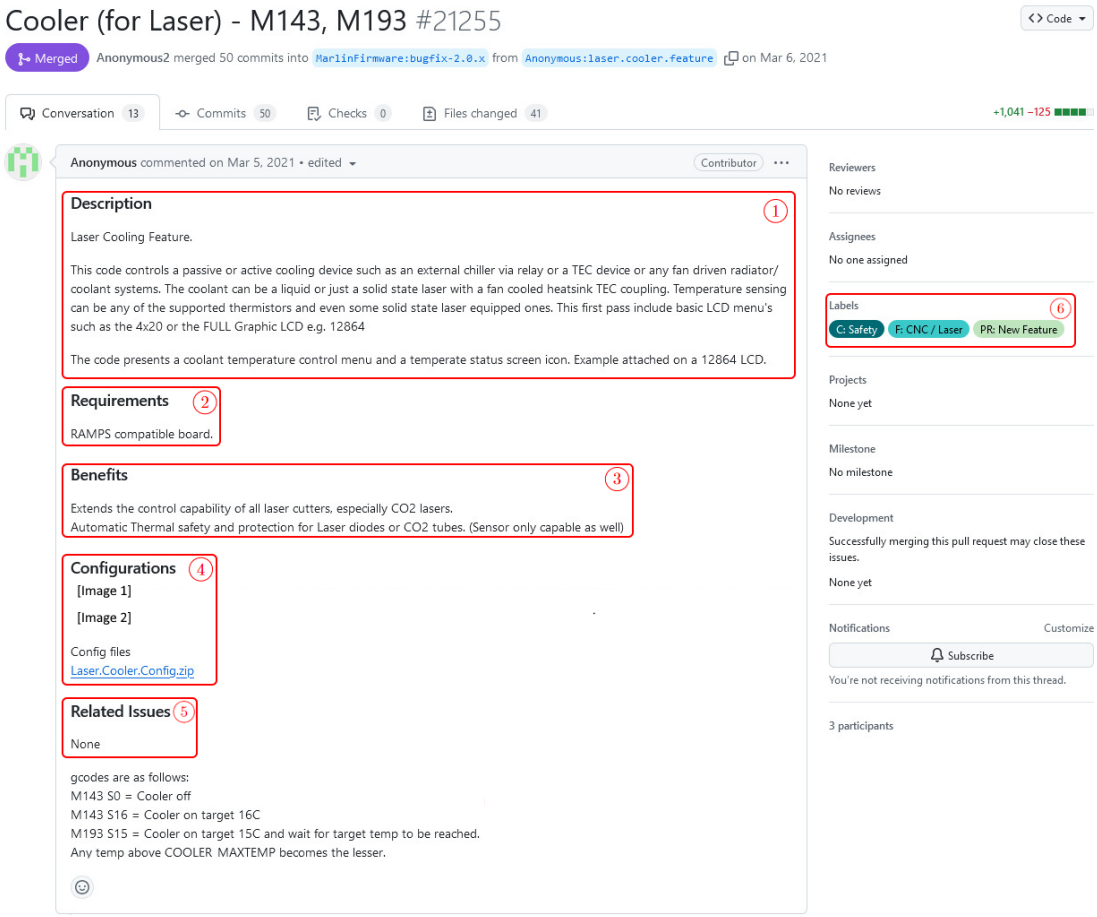

Figure 3.2: Description of pull request #21255 in the Marlin system.

it can do. After this we get the possible requirements (2) that are necessary for the pull request to work. In this case we need a control board for the machine that supports RAMPS, RAMPS stands for RepRap Arduino Mega Pololu Shield, it is a board that contains all the electronics for a complete RepRap printer[5]. After this the benefits of the pull request are listed (3) these make it concrete what the main point of the pull request is about. Then the configuration used is given (4) this can be some pictures and the configuration files for the setup. Lastly related issues are listed (5) these indicate what issue of Marlin this pull request will solve. On the right-hand side the labels are shown (6). The labels indicate what part of Marlin this pull request covers, these labels are added by maintainers of Marlin. They label features, answer questions in the issues and make decisions on what to merge in to the project.

3.3.2 Creation of Folder Structure

Once we understand what the pull request is about, what it does and the possible problems that were discussed in the comments we can create the folder structure for the pull request. When the script is ran to create the folder structure the file containing all the information about the pull request will look like the file shown in Figure 3.3. The file is automatically filled by the script with the correct information. We then must manually check all the files associated with the pull request and see which of them actually are needed for the feature. This is because pull requests

[5] https://reprap.org/wiki/Arduino_Mega_Pololu_Shield

```
Created: 05-03-2021
Merged: 06-03-2021
Pull: #21255
Comments: 13
Files Changed:
Marlin/Configuration.h
....
Marlin/src/pins/pinsDebug_list.h
Marlin/src/pins/sensitive_pins.h

date: name (pull) (tag) (commit) (comment)
06-03-2021: Cooler (for Laser) - M143, M193 (#21255) ($NewFeature) (b95....)
```

Figure 3.3: Pull request information of pull request #21255

.

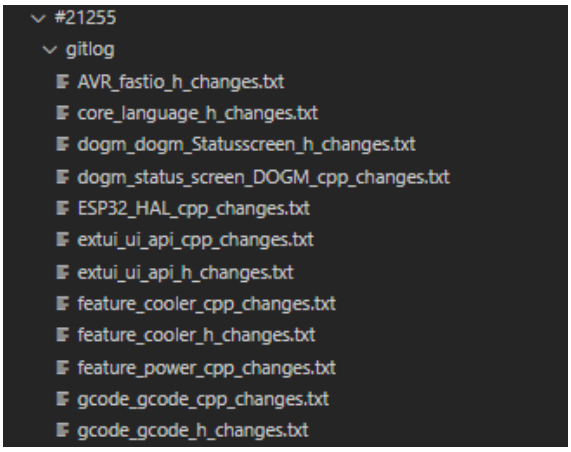

Figure 3.4: Part of the change files of pull request #21255.

can do more than just introduce software features. They could also fix bugs or typos at the same time and those changes are not relevant to the feature. So we have to remove any files not relevant to the software feature.

3.3.3 Change History of Associated Files

After it has been confirmed which files contain changes associated with the software feature. We use a script to download the change history of those files. These change history files contain every single commit that touched the file. With these files we can see which commits touch the code related to our software feature. It is possible that filenames get renamed over time. If this happens the new version of the files also must be obtained. Once this has been done all the change files have been generated and it will look like Figure 3.4. After this the analyzing can begin.

3.3.4 Analysis of Change File

On the GitHub page of the pull request we look at the original changes made to introduce the software feature. We then look for unique keywords in these changes and search the change file for these words to find changes to the code of the software feature. Once such a change has been found the date of the commit and its title are noted down in the dataset made for this pull request. The changes to the code are then closely examined together with the description of the commit, to

```
commit ccdbffbf3f599f0860c643647ce6c40f1eb5a4cd
Author: anonymous
Date:   Mon Mar 29 01:41:56 2021 -0500

    Laser Coolant Flow Meter / Safety Shutdown (#21431)

    Co-authored-by: anonymous2

diff --git a/Marlin/src/feature/cooler.cpp b/Marlin/src/feature/cooler.cpp
index 03640df487..a1f25c5fad 100644
--- a/Marlin/src/feature/cooler.cpp
+++ b/Marlin/src/feature/cooler.cpp
@@ -27,11 +27,21 @@
 #include "cooler.h"
 Cooler cooler;

-uint16_t Cooler::flowrate;        // Flow meter reading in liters, 0 will result in shutdown if equiped
-uint8_t Cooler::mode = 0;         // 0 = CO2 Liquid cooling, 1 = Laser Diode TEC Heatsink Cooling
-uint16_t Cooler::capacity;        // Cooling capacity in watts
-uint16_t Cooler::load;            // Cooling load in watts
-bool Cooler::flowmeter = false;
-bool Cooler::state = false;       // on = true, off = false
+uint8_t Cooler::mode = 0;
+uint16_t Cooler::capacity;
+uint16_t Cooler::load;
+bool Cooler::enabled = false;

+#if ENABLED(LASER_COOLANT_FLOW_METER)
+  bool Cooler::flowmeter = false;
+  millis_t Cooler::flowmeter_next_ms; // = 0
+  volatile uint16_t Cooler::flowpulses;
+  float Cooler::flowrate;
 #endif
+
+#if ENABLED(FLOWMETER_SAFETY)
+  bool Cooler::flowsafety_enabled = true;
+  bool Cooler::fault = false;
+#endif
+
+#endif // HAS_COOLER
```

Figure 3.5: Changes made in the `cooler.cpp` file of pull request #21255.

figure out which tag fits the commit. In Figure 3.5 we can see such a change in a file, here we can see all the changes made in the `Cooler.cpp` file and based on these changes we tag this commit as an `Enhancement` of the cooler for lasers because it expands the functionality of the cooler by giving the ability to keep track of the flow of the water. If we had already encountered the commit for the same pull request we will tag the commit with the more important of the possible tags.

This analysis is done for all the files and once all the files have successfully been handled the next software feature pull request is opened and we start over again. This workflow has been followed to tag 189 pull requests that were labeled with the `PR: New Features` label by the maintainers of Marlin.

3.4 Pull Request Feature Check

To find out if all the pull requests that are labeled by the maintainers of Marlin as a feature are actually features, we need to inspect them. We take a sample of pull requests that are labeled as a software feature and inspect them to determine whether they are features. We do this by reading the description of the pull request and its associated code changes and compare them against our definition of a feature. The results of this are noted down in a file together with a comment describing why it is or is not a feature.

We also check if pull requests that are not labeled as a software feature by the maintainers of Marlin are considered features according to our definition. To do this we use a similar technique as for the labeled software features. We first filter the pull requests to only contain pull requests released in a similar time frame as those of our sample set so that we can compare them. After this we remove any pull request that is not merged into the repository from the dataset. The

pull requests that are open have not had their full chance yet in the Marlin repository and as such are not considered. Pull requests that are rejected do not have a final version of code so it is difficult to determine if it would have been considered a feature if it was properly merged. After this we are not interested in pull requests that are explicitly about bug fixing, because those should not introduce new features. We also remove pull requests that are labeled to indicate that they are used for the cleanup of code. This leaves us with a dataset that contains pull requests that according to the maintainers of Marlin do not contain any features. We then take samples of this dataset to investigate if any of these pull requests could be considered software features according to our definition. We then compare the pull requests just as we did for the once labeled a software feature. We compare them against our definition and write down the results and possible comments describing what has been added to the system by them.

3.5 Feature Tags

We use the following tags when tagging the commits that touch the code of a software feature. These tags are sorted in order of importance. The tag that a commit is given is based on the most important one of the tags it fulfills. So, if a commit modifies some comments and refactors a bit of code, it is tagged as a refactor commit. The tags given to the commit are given with as frame of reference the software feature, this means that the same commit can have different tags for different software features. For one feature, a commit can be an enhancement of functionality of the software feature, but for another software feature, that commit is only a rename of a specific function and does not alter the workings of the software feature at all. These tags are based on the standard tags used in this type of tagging of `Corrective`, `Perfective` and `Adaptive` [14]. We split up the tags further for the following reason it is important for the life cycle of a software feature to make a distinction between the important and unimportant software updates. This insight was gained by a first manual investigation into different software features to see which type of changes happen and how to best tag these changes. We will list the tags one by one and explain what they mean:

NewFeature is given to the commit that introduces the feature to the general codebase. The size of such commits can vary wildly. In the most extreme case you have a new feature that is as short as a single line such as in the pull request #26825[6] that defines two pins on the control board of a 3D-printer to make use of some new functionality or as big as 16,162 lines of the pull request #18071[7] which introduces a new UI library for a specific 3D-printer.

Removal is given to a commit that removes every single line of code associated with that software feature without reimplementing it somewhere else. This signals the end of life of a software feature and its entire removal from the software project. An example of this can be found in the pull request #24229[8] which removes the software feature introduced in pull request #24074[9].

Rework is given to a commit that alters the implementation of a software feature in such a big way in one commit that you can say that the entire feature has been reimplemented from scratch. This happened to the new feature introduced in pull request #16068[10] which was a solution for the laser graphics of the LCD Screen when the bigger rework in pull request #15335[11] was released which updated the laser functionality of Marlin in a big way.

Revert is given to a commit when the commit does exactly that it reverts the changes from a previous commit, seems to happen most often when a big rewrite is merged, but there are still too many bugs present in it.

[6] https://github.com/MarlinFirmware/Marlin/pull/26825
[7] https://github.com/MarlinFirmware/Marlin/pull/18071
[8] https://github.com/MarlinFirmware/Marlin/pull/24229
[9] https://github.com/MarlinFirmware/Marlin/pull/24074
[10] https://github.com/MarlinFirmware/Marlin/pull/16068
[11] https://github.com/MarlinFirmware/Marlin/pull/15335

BugFix is given to a commit when it fixes behavior that exists within a software feature that is unintended. This can be achieved by rewriting larger parts of the code or by simply fixing a spelling mistake.

Enhancement is given to a commit if it increases the functionality of the software feature or broadens its support on the platform. For example pull request #26485[12] changes the code of the software feature introduced in pull request #26328[13] to also work for another platform and by such broadens the support of the feature.

Refactor is given to a commit if it changes the code of the software feature but it does not alter the functionality of the feature. This can be a reordering of the code for layout and readability purposes or the removal of macros by the longer form of the same code.

Cleanup is given to a commit if it removes code of the software feature, but it does not alter the functionality of the feature. This can be because the code has become obsolete because of other changes but was never removed.

Formatting is given to a commit that touches executable code but only slightly changes it by way of adding spacing or removing spacing or similar small changes.

Comment is given to a commit if it only changes a comment written in the code and does not touch any of the executable code.

Whitespace is given to a commit if it only changes whitespace present in the code and does not touch any of the executable code.

We have divided the tags up into two categories. The category of main commits and side commits. The first seven tags belong to the category of main commits while the last four tags belong to the category of side commits. The reason that the side commits are still tagged is to give a complete overview of everything that touches a software feature, but it is important to note that not every commit that modifies a software feature has an impact on the functionality of it.

From this point forward when talking about the word `tag`, we refer to one of the tags as we have just defined here, and the word `label` refers to the labels that the Marlin GitHub project uses for the labeling of pull requests and issues.

3.6 Automatization

To be able to process a lot of data, the workflow for gathering information must be automated as much as possible. Unfortunately, the automatization process of gathering information concerning software features is difficult with many limitations. Key problems involved with the automatization are as follows: Tracking changes to code over time, correctly tagging the commits that touch the code and distinguishing parts of commits that matter.

3.6.1 Tracking Changes to Code

To be able to manage system development over time, developers of software systems make use of version control systems, which keep track of changes over time. However, the version control systems often do not allow to easily retrieve and analyze the changes to the software features as often in a single commit multiple changes are implemented related to a variety of features [16]. Furthermore, software features are often tangled together and scattered over different files. So, there is not one concise software function which can be considered the software feature but many small parts together form the software feature. This makes it difficult to know what software feature has been changed when multiple files and lines are modified in a single commit.

[12] https://github.com/MarlinFirmware/Marlin/pull/26485
[13] https://github.com/MarlinFirmware/Marlin/pull/26328

In the case that software features are reworked, and filenames are changed in the process. Git is only able to follow those file changes if the added content of the file matches enough of the old content, and in the case of bigger reworks this is often not the case. There are other methods with which we can keep better track of file renaming [13]. But this feature as of jet is only applicable to java code. Git only knows when searching from the new file what the old file is called, but it does not know when given the old file what the new file is called. So, the only way to match and old file to a new file is by manually inspecting the code and commit message to see which of the new files is the successor of the old file.

The Git log command can generate the change file created by a commit. But, Git log is not always able to correctly match the changed code parts together. This happens when the code is both changed and reordered in a file. This causes git log to match up the wrong parts of the code and create a bigger difference than actually happened. This also affects the accuracy of the number of additions and deletions because they are artificially increased.

3.6.2 Correct Tagging

To correctly follow the evolution of the software feature over time it is of importance that the changes that happen to the feature are correctly tagged. Before work has been done to train models in correctly tagging commits based on the textual content of these commits and even utilizing the source code changes, with such an approach the authors achieved an accuracy of 76% [25]. But the authors in this case only used three different tags for the tagging of the commits. Further work in this field used natural Language Processing to achieve a higher accuracy of 91% while tagging the commits with five different tags. This is a significant improvement over the machine learning based methods [10]. The problem with both methods is that they only look at description associated with the commit text and the direct code changes. While for our purpose it is important what the commit changes with regard to our desired software feature. If the commit fixes a bug in software feature A and corrects a grammar mistake in software feature B, then for both software features the commit would be marked as a Bug Fix while for one of them it only concerns a comment change. So, to be able to correctly tag any of the commits, the source code changes that are relevant to the software feature must be examined to come to a conclusion, for this a basic insight into the feature and how it works is also mandatory.

3.6.3 Distinguishing Parts of Commits

In an ideal world, a commit only makes a change to fix a single thing or add a single feature, but often it can be seen that a single commit encompasses multiple features or multiple unrelated changes. Because a commit makes changes to more code than just the feature, it is difficult to know for sure what part of a commit is relevant for a software feature. This can only be done by an inspection of the changed code and comparing it to the previous state of the feature and observing if it has changed.

3.6.4 Scripts

As described in the previous subsections, there are many challenges to building a fully automatic system for the processing and tagging of labeled software features in a project. To streamline the process as much as possible, a variety of semi automations were made in the form of scripts. These scripts run based on the user provided input and use Python[14] combined with the GitHub API and Git to automate as much of the process as possible.

Create Structure

We have made a script which takes as input a list of pull requests numbers and outputs a folder hierarchy, in which the structure has been made for every pull request to have its own folder and

[14] https://www.python.org/

within that folder a file is made containing the basic information of the pull request such as: time of creation, time of merge, pull request ID, number of comments, the files changed by the pull request, and the first commit tag of `NewFeature`.

We must check after this step if all the files that are listed as changed for the software feature are actually relevant to the software feature. Because, as mentioned in the previous subsections, it often happens that irrelevant files to the software feature are included in a commit.

Files History

To be able to find out which commits have interacted with the software feature we are gathering data of, we need the change history of all the files that are associated with the software feature in the pull request. For this script to work, a local installation of Git is required as is a local copy of the target repository. The script uses the ability of Git to produce change files based on specified filenames. If the name of one of the files has changed over time git log will return an error code, when this error code is detected a new instance of git log is run with the additional parameter that makes it look for the change history of now non-existing files. It will output what files it could not find. The user then has to find what the new names of these files are or if they were fully removed from the project. This is done by inspecting the last commit associated with the old file.

Pull Requests and Issues

A suite of scripts has also been made to use the GitHub API to pull information about all the pull requests and issues of a repository, and process them to create datasets containing all the issues and pull requests with its labels, creation time and other relevant information.

Importing Data. The name of the repository is given to the script and the script then pulls all the issues or pull requests related to the repository. It is possible to specify a date, the code will only pull information until that date if specified. Once the data has been pulled, it is stored in a file format that can store the native python format. This so-called pickle file is used so that the contents can be opened later in the same form as they have been originally pulled and so data does not have to be collected multiple times. The GitHub API is rate limited for non-enterprise users, so pulling all the pull requests of a project the size of Marlin can take up to 24 hours. To prevent data from being lost, the pickle file is used. The import script also checks for the presence of a CSV file that contains previously processed data, based on whether the ID of the pull request or issue is present in that file it will skip it or get their information. With this, all the items that have already been processed will be skipped and so time is saved while collecting the data. The downside of this method is that if any changes have happened to these items, we will not receive that information.

For pull requests the importer imports the ID, number of comments, its merge state, if it is open or closed, the labels associated with the pull request, all the files changed by the pull request and the GitHub URL of the request. For the issues all the same is gathered besides the merge state and files cause those are not applicable to an issue.

Extracting Data. Once the information has been imported, the data is extracted from the pickle file. This is then all stored in a pandas dataframe which at the end is written to a CSV file for later analysis of the data. These steps are separated so that if something goes wrong not all information would be lost because nothing would have been saved in between

Other Scripts

A variety of other scripts have been written to speed up the workflow of the data gathering process. This includes scripts for the reordering of the commits present in the feature file to a chronological order so that when gathering data, we are not delayed ordering it ourselves. A script was made to check if all the data entered for a commit is correct this includes checking if the tag is written correctly and that the hash of the commit is noted down correctly this is done so that errors will not occur when processing the data that would lead to errors in the later results.

Summary In this chapter we have discussed the research questions and why they are important to be answered. We have also discussed the methodology used for the case study of Marlin. Furthermore, an example was given of how this methodology works in practice.

The dataset acquired through these techniques can be found on GitHub[15] including the scripts necessary to gather all the data for future research.

[15] https://github.com/AHofstad/TheLifeOfSoftwareFeatures

Chapter 4

Results and Discussion

In this chapter we will go over the lessons learned, and we will report and discuss the results of each research question separately.

4.1 Lessons Learned

In this section, we present and discuss our findings that have been found that do not relate to either of our research questions.

4.1.1 Marlin Repository

Results. We finished collecting all the data on the 28 July 2024. The Marlin Repository contained a total of 12,528 pull requests at this point in time. Of these 12,528 pull requests 9,114 have been merged and 3,414 are not merged. Of these 12,528 pull requests, there are 662 labeled with the `PR: New Feature` label. These pull requests are the labeled software features of Marlin. Of these 662 pull requests 189 pull request have been inspected and tagged. These 189 pull requests form our sample for the Marlin repository. This sample has been formed by starting with the newest labeled pull request and working the list down until there was no further time available to inspect more. The labeled software features introduced since 11 June 2019 have been looked at. Of which 189 were merged, 39 have been rejected and 14 are still open.

4.1.2 Marlin Labels

The Marlin GitHub repository has 80 labels that are in use for its issues and its pull requests. The documentation of Marlin itself only provides an explanation of what these labels represent for a number of them. To gain a better understanding of the Marlin system we have defined what every label means and how they are applied to pull requests. All the labels can be found in Appendix A. This information was gathered by analyzing the descriptions of pull requests associated with the labels and by inspecting the code changes of these pull requests to understand what part of the codebase it impacts and what it does.

Marlin separates its labels into different categories marked using a prefix. The following prefixes are in use: `A:`, `Bug:`/`Bug?`, `C:`, `F:`, `K:`, `Needs:`, `PR:`, `S:` and `T:`. There are also 5 labels without a prefix. The Marlin documentation does not define what each prefix means, but by defining all the labels we have come up with some possible meanings:

`A:` stands for `Architecture` because it is used in combination with labels intended to define different microcontrollers.

`Bug` is used for `software bugs`. It is used to label pull requests possibly related to bugs.

`C:` stands for `Code` it is used to refer to general topics in the Marlin codebase.

`F:` stands for `Feature` it is used to refer to specific features of Marlin.

`K:` stands for `Kinematics` it refers to the code about specific motion systems of Marlin.

`Needs:` is used for administrative tasks related to pull requests.

`PR:` stands for `Pull Request` it labels what type of pull request it is.

`S:` It is not know what the `S:` prefix stands for. Some of the labels describe actions concerning the pull request and others describe the state of the pull request.

`T:` stands for `Topic` because the labels cover quite wide topics that are not necessarily related to the code of Marlin.

Marlin clearly separates its labels into distinct categories. The labels have a clear domain of what belongs to them. Most pull requests that get merged get labeled with at least one of the labels containing the prefix `PR:`. Sometimes they are also labeled with an additional label if the pull request falls into the domain of that label. But this does not always occur. For example pull request #26979[1] only has the `PR: Bug Fix` label. The pull request is about fixing a typo made in the `pins_postprocess.h` file. Meanwhile, a similar pull request #22771[2] is in addition of the `PR: Bug Fix` label also given the `C: Board/Pins` label. This shows that the Marlin maintainers in some cases do not apply all the applicable labels to a pull request. And because of that information is lost in the pull request list regarding what part of the project a pull request actually has an impact on.

4.1.3 GitHub UI and API Differences

Results. The GitHub UI[3] reports to us that there are 652 pull requests with the `PR: New Feature` label. Of these pull requests 17 are open and 635 are closed. In Figure 4.1 we can see the number of pull request that are associated with certain labels and how many of them are merged and not merged. We can see in the figure that contains the data we have collected through the GitHub API that there are 662 pull requests. This is 10 more labels than the GitHub UI shows us. This is also the case for other labels. Of the top 10 labels, not a single one has the same number of pull requests on the GitHub UI as it has according to the GitHub API as can be seen in Table 4.1. When taking a deeper look into this we find pull requests with the `S: Superseded` label that is in our collected data but is not present in the GitHub UI. That is the pull request #4811[4]. We can only access this pull request by manually filling in the URL as it is not visible in the Marlin repository. We see the reason for why it is not visible on the GitHub UI. The pull request was made by what is now a ghost account. The GitHub ghost account[5] represents accounts that have been deleted by the original user. This seemingly also deletes their repositories and hides their pull requests and issues from being visible via the GitHub UI directly.

Another thing that happens on the GitHub UI is that some pull requests are in the list of pull requests, but it is impossible to open them. One such example is pull request #16565[6]. This pull request returns a server error on the GitHub UI, but all its data is still able to be collected via the GitHub API.

Discussion. We have seen that the GitHub UI has built in features that hide items created by now deleted accounts. Because of this it is important that for research into repositories the GitHub API is used to collect data. The GitHub API does not hide this information. This indicates that it is important to make use of the GitHub API so that all possible information can be gathered even those not directly visible through the GitHub UI.

[1] https://github.com/MarlinFirmware/Marlin/pull/26979/files
[2] https://github.com/MarlinFirmware/Marlin/pull/22771/files
[3] https://github.com/MarlinFirmware/Marlin/pulls?q=is%3Aopen+is%3Apr+label%3A%22PR%3A+New+Feature%22
[4] https://github.com/MarlinFirmware/Marlin/pull/4811
[5] https://github.com/ghost
[6] https://github.com/MarlinFirmware/Marlin/pull/16565

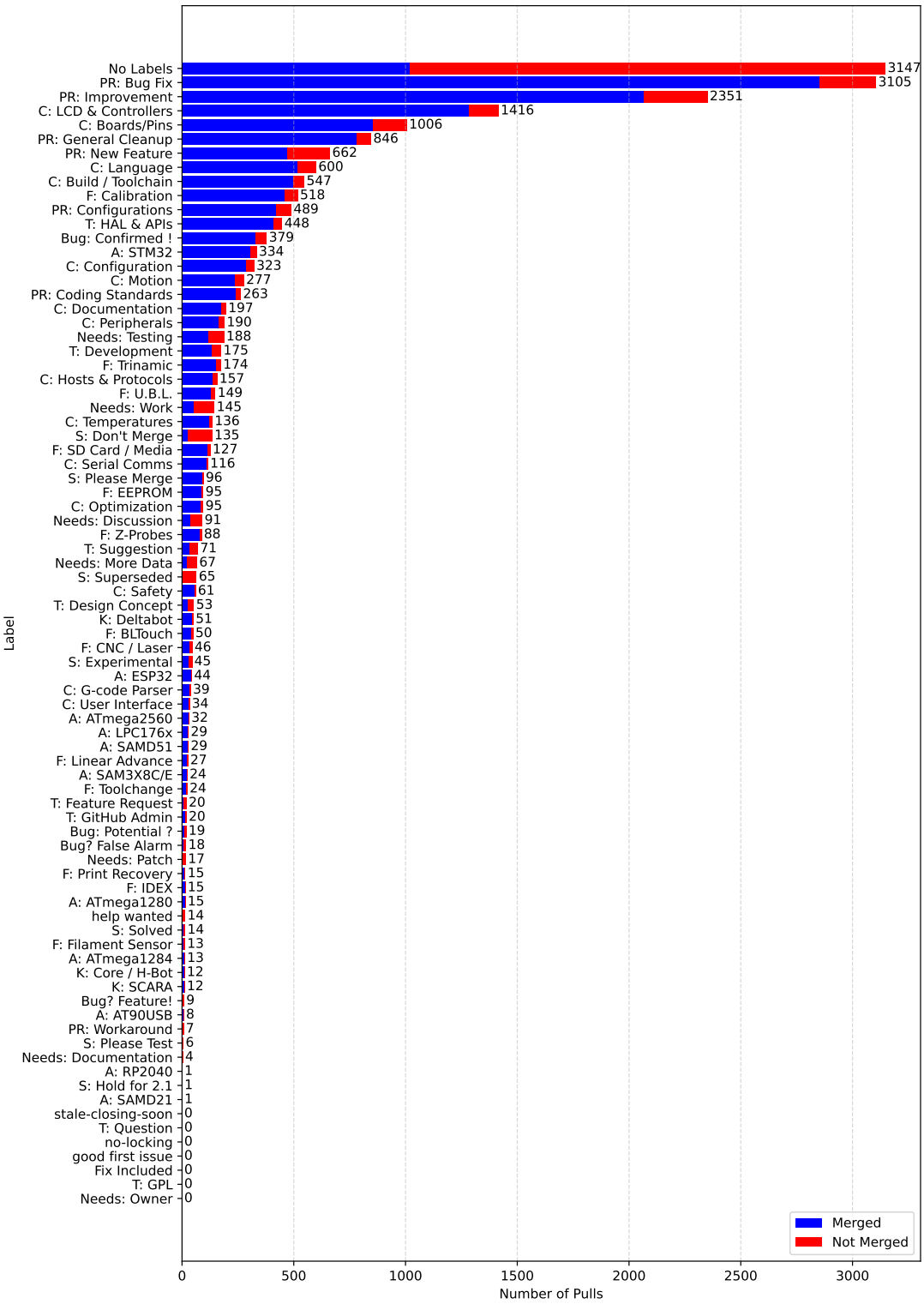

Figure 4.1: Number of pull requests with a certain label. Merged and not merged stacked on top of each other.

Label	GitHub UI	GitHub API	Difference
No Labels	3044	3146	+102
PR: Bug Fix	3056	3105	+49
PR: Improvement	2320	2351	+31
C: LCD & Controllers	1411	1416	+5
C: Board/Pins	1002	1006	+4
PR: General Cleanup	806	846	+40
PR: New Feature	652	662	+10
C: Language	559	600	+41
C: Build / Toolchain	546	547	+1
F: Calibration	511	518	+7

Table 4.1: Table showing the difference between number of pull requests on GitHub Website and through the API.

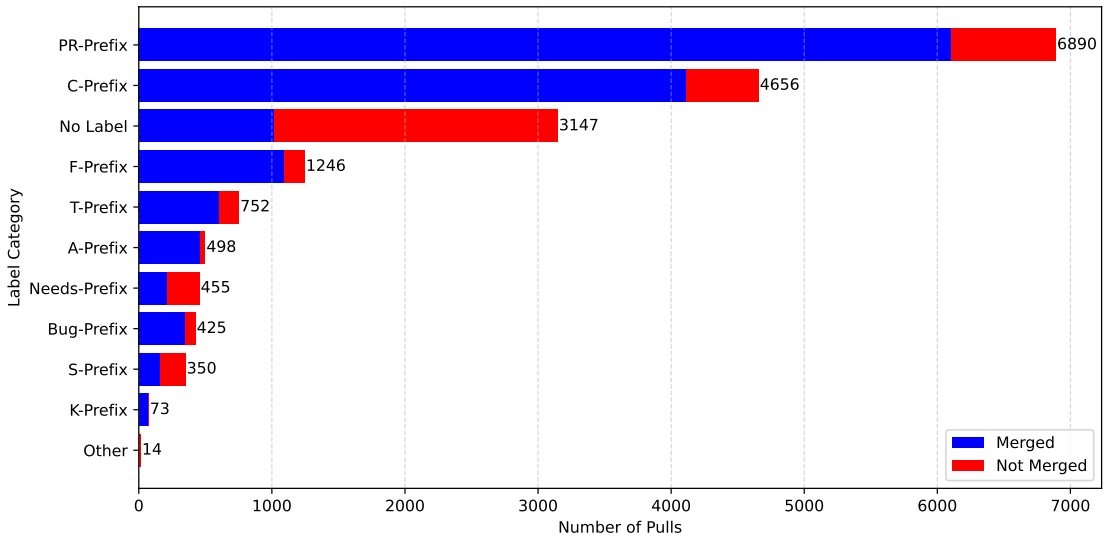

Figure 4.2: Number of pull requests in a certain label group. Merged and not merged stacked on top of each other.

4.1.4 General Pull Requests

Results. We display the distribution of pull requests by the label groups in Figure 4.2. We can see the total number of pull requests and the color blue represent the number of merged pull requests and the color red represents the number of not merged pull requests. Pull requests that are labeled with multiple labels in a prefix group are only counted once. The `PR-Prefix` group represents all the different types of pull requests that there are in the Marlin repository as labeled by the Maintainers of Marlin. This is the biggest group of pull requests present in Marlin. The second-biggest group in Marlin is the `C-Prefix` group. And the third-biggest group is the `No Label` group. When we add the `PR-Prefix` group and the `No Label` group together we get 10,037 pull requests. The total number of pull requests in the repository is 12,528. So, this means that there are 2,491 pull requests that do have a label but are not labeled with any of the labels of the `PR-Prefix` group.

Figure 4.3 we can see the distribution of pull requests by the label groups with pull requests that either have `No Label` or a `PR-Prefix` group label removed. We can observe that the order of the groups has changed. This indicates that for some groups of labels, it is more likely that a

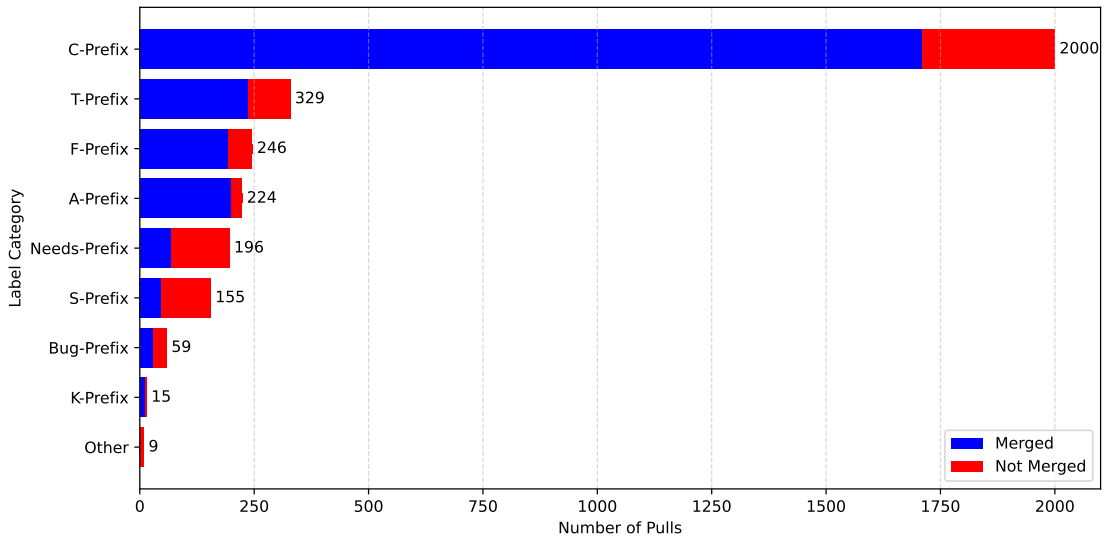

Figure 4.3: Number of pull requests in a certain group. All pull requests containing a `PR-Prefix` label, or no label removed. Merged and not merged stacked on top of each other.

label of the `PR-Prefix` group is not applied than to others. On average 36% (2491/6858) of the pull requests that do have a label are not labeled with a `PR-Prefix` group label. We find that for the pull requests of the `C-Prefix` group 43% (2000/4656) do not have a `PR-Prefix` group label which is in line with the average. There are three outliers: For the `F-Prefix` group only 20% of the pull requests are not labeled with a `PR-Prefix` group label, for the `Bug-Prefix` group this is only 14%, but for the `Other` group which is quite small (only 14 in total) 65% is not labeled with a `PR-Prefix` group label.

In Figure 4.4 we see the number of pull requests with and without a label, the number of them that is merged and the total, per year. In 2015 the Marlin maintainers started labeling a majority of the pull requests. We observe that over time the number of pull requests without a label decreased as a percentage of the total number of pull requests. And the number of pull requests without a label that are merged has also decreased since 2015. If we look at the recent unlabeled pull requests in the Marlin Repository, we find that there are three types of unlabeled pull requests. The first are the pull requests containing no further information and seem to be a mistake (Figure B.1), these are either quickly deleted by the contributor themselves or by the maintainers of Marlin. Secondly we have the pull requests which want to implement actual functionality but the maintainers of Marlin have reasons why it should not be done in such way (Figure B.2). And lastly we have the pull requests that are only a slight change or fixing of a typo, these are usually accepted quite quickly by the Marlin maintainers (Figure B.3).

In Figure 4.1 we see the distribution of pull requests of all the labels. The figure shows us the total number of pull requests with that label, and it shows us how many have been merged and not been merged. In this figure pull requests are counted multiple times if they have multiple labels, so the totals do not add up to the total number of pull requests in the repository. We can see that a few large groups make up most of the pull requests. Because of the large number of specific labels a lot of labels cover only a small number of pull requests.

Besides the pull requests with no label the label that is used most often is the `PR: Bug Fix` label. When we look at the label ”PR: Bug Fix” in Figure 4.5 we can see that the number of bug fixes remains quite steady in ratio to the total number of pull requests over time. During the peaks in 2016 and 2018 of commits to Marlin a notable increase in the number of bug fixes also follows. We observe that the number of bug fixes is on a slow decline, as is the monthly number of pull request submissions in general.

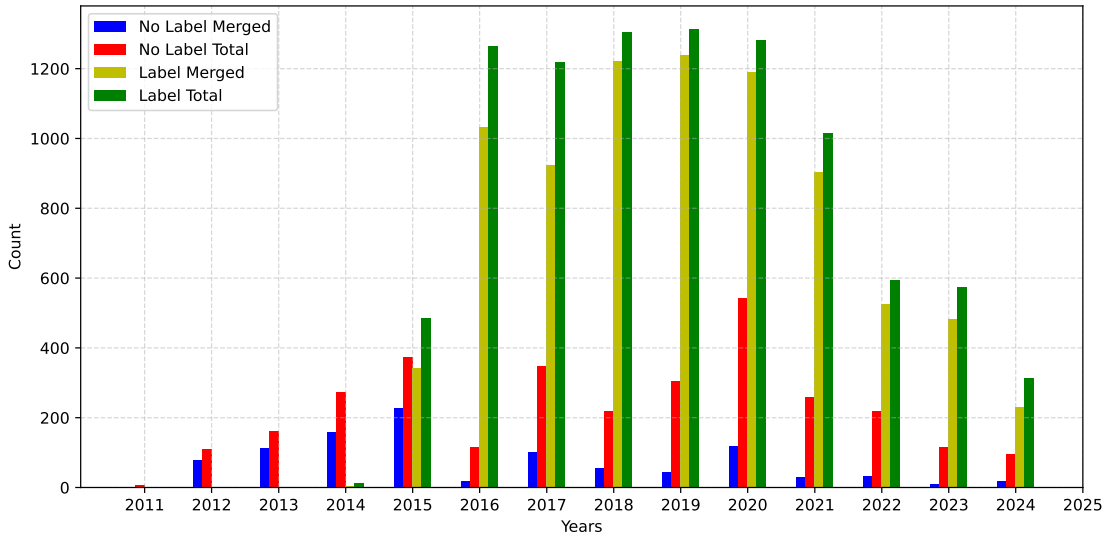

Figure 4.4: Pull requests submitted per year with and without labels, merged and total number.

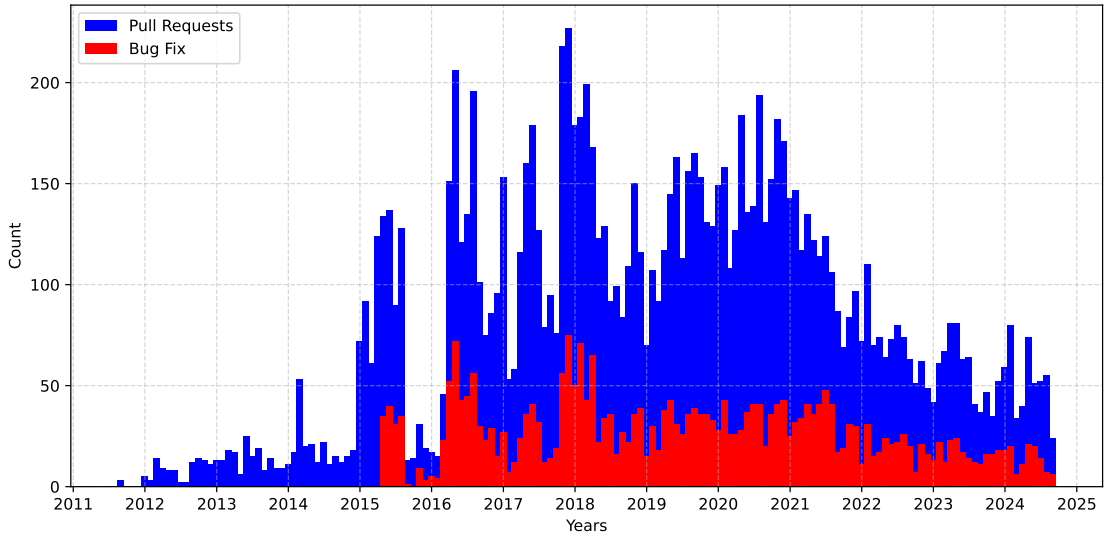

Figure 4.5: Total number of pull requests and number of pull requests with the `PR: Bug Fix` label of submitted per month.

Discussion. Even though the maintainers of Marlin have created labels to indicate what type of pull request a pull request is, in 27% (2491/9381) of the cases they do not apply such a label to a pull request. This is both present in pull requests that get merged as those that are not merged. This could have implications for the accuracy of the dataset as our findings are based on the labels associated with the pull requests.

We conclude that the reason that pull requests without a label are often not merged anymore is that they are either a mistake or the pull requests wants to implement a feature or change that the maintainers do not want. And if they are merged, they are often only a very slight change. So, the reason that pull requests without a label are often not merged is that they are of low quality or not necessary and so likely the effort is not made to label the pull requests before closing them. As there is nothing to label.

By taking a closer look into pull requests without a label in the Marlin repository, a trend can be found. Marlin has become a bigger and more professional piece of open-source software over time. Because of that the need for labels has become necessary to maintain a cohesive and clear organization [7]. And so over time the number of pull requests without a label has decreased.

The reason that the `PR: Bug Fix` label is the most used label in the repository can be explained by the fact that software maintenance is recognized as the most demanding, costly and difficult phase in the software life cycle, with as much as 60-70% of all resources in a software project spent on maintenance [29].

4.1.5 `PR: New Feature`

Results. For this thesis, the most important label in the repository is the "PR: New Feature" label. This label indicates that a new software feature is going to be introduced according to the maintainers of Marlin that have labeled them. Of the 12,528 pull requests that Marlin has, 662 of them have the `PR: New Feature` label that is 5.3% of all pull requests and of the 9425 labeled pull requests that Marlin has it is 7.0%.

Figure 4.6 shows us the pull requests with `PR: New Feature` over time. We can see that the number of new features is quite steady but has been on the decline in number just as all the pull request in Marlin. At the end of 2015 there is a decrease in the number of pull requests being submitted to Marlin. Like mentioned before we can see that before 2015 there are no pull requests labeled with the `PR: New Feature` label. This means that for a period of 4 years we have no information about the possible features that were added to Marlin. So, some of the software features present in Marlin cannot be covered as they have not been labeled by the maintainers of Marlin.

Table 4.2 contains four columns: The first column is the label which the results relate to. The second column is the number of pull requests that are labeled with the `PR: New Feature` label and the label mentioned in the first column. The third column is the number of pull requests in total that are labeled with the label mentioned in the first column. Lastly the fourth column is how much percent the software features cover the total number of pull request with the label mentioned in column one. We know that software features make up only 7.0% of the total number of labeled pull requests. We see however that for a lot of labels the percentage that is labeled with the `PR: New Feature` label is higher than 7%.

Label	PR: New Feature	All Pull Requests	Percentage
PR: New Feature	662	662	100.0
S: Hold for 2.1	1	1	100.0
S: Experimental	16	45	35.56
F: CNC / Laser	16	46	34.78
T: Feature Request	6	20	30.0
help wanted	4	14	28.57
F: Z-Probes	25	88	28.41
Needs: More Data	17	67	25.37

Label	PR: New Feature	All Pull Requests	Percentage
Needs: Documentation	1	4	25.0
S: Superseded	16	65	24.62
Needs: Work	27	145	18.62
Needs: Testing	34	188	18.09
C: Hosts & Protocols	25	157	15.92
F: Filament Sensor	2	13	15.38
C: Safety	9	61	14.75
F: Toolchange	3	24	12.5
F: Calibration	64	518	12.36
C: Peripherals	23	190	12.11
F: SD Card / Media	15	127	11.81
Needs: Patch	2	17	11.76
T: Design Concept	6	53	11.32
C: Temperatures	15	136	11.03
Needs: Discussion	10	91	10.99
C: G-code Parser	4	39	10.26
C: Motion	28	277	10.11
C: User Interface	3	34	8.82
K: Core / H-Bot	1	12	8.33
S: Don't Merge	11	135	8.15
PR: Configurations	38	489	7.77
F: Trinamic	12	174	6.9
F: IDEX	1	15	6.67
C: LCD & Controllers	86	1416	6.07
C: Serial Comms	7	116	6.03
T: Development	10	175	5.71
C: Boards/Pins	56	1006	5.57
S: Please Merge	5	96	5.21
A: STM32	17	334	5.09
F: U.B.L.	7	149	4.7
F: BLTouch	2	50	4.0
PR: Improvement	88	2351	3.74
A: SAMD51	1	29	3.45
T: HAL & APIs	14	448	3.12
T: Suggestion	2	71	2.82
C: Configuration	8	323	2.48
C: Optimization	2	95	2.11
C: Build / Toolchain	11	547	2.01
K: Deltabot	1	51	1.96
PR: Coding Standards	3	263	1.14
Bug: Confirmed !	4	379	1.06
F: EEPROM	1	95	1.05
PR: Bug Fix	16	3105	0.52
C: Documentation	1	197	0.51
PR: General Cleanup	3	846	0.35
C: Language	2	600	0.33
A: ESP32	0	44	0.0
A: ATmega2560	0	32	0.0
A: LPC176x	0	29	0.0
F: Linear Advance	0	27	0.0
A: SAM3X8C/E	0	24	0.0
T: GitHub Admin	0	20	0.0

Label	PR: New Feature	All Pull Requests	Percentage
Bug: Potential ?	0	19	0.0
Bug? False Alarm	0	18	0.0
F: Print Recovery	0	15	0.0
A: ATmega1280	0	15	0.0
S: Solved	0	14	0.0
A: ATmega1284	0	13	0.0
K: SCARA	0	12	0.0
Bug? Feature!	0	9	0.0
A: AT90USB	0	8	0.0
PR: Workaround	0	7	0.0
S: Please Test	0	6	0.0
A: RP2040	0	1	0.0
A: SAMD21	0	1	0.0

Table 4.2: Number of pull requests labeled with label and the `PR: New Feature` label, Number of pull requests in total labeled with label, Percentage of the total that the pull requests with the `PR: New Feature` label form.

For example, for the `S: Experimental` label we have a coverage of 36% and for `F: CNC / Laser` label we have a coverage of 35%. By looking at the pull requests that make up the `S: Experimental` label we notice that a lot of them are about changing already exisiting functionality of Marlin. For example pull request #16864[7] is about doubling the read frequency of the Analog to Digital Converter and pull request #3110[8] is about implementing an alternative implementation of LCD-based manual movement. When looking at the merged pull requests we can see in Table 4.3 that this jumps up to 45% (13 out of 29). Most of the new features of this label are a variation on an already existing feature in a new and unique way.

Label	PR: New Feature	All Pull Requests	Percentage
PR: New Feature	472	472	100.0
S: Hold for 2.1	1	1	100.0
S: Experimental	13	29	44.83
help wanted	2	5	40.0
T: Feature Request	3	8	37.5
Needs: Patch	1	3	33.33
Needs: Documentation	1	3	33.33
F: Z-Probes	22	79	27.85
Needs: More Data	6	24	25.0
Needs: Work	13	53	24.53
S: Don't Merge	6	27	22.22
F: CNC / Laser	7	33	21.21
Needs: Discussion	6	39	15.38
C: Safety	8	56	14.29
C: Hosts & Protocols	19	137	13.87
Needs: Testing	16	117	13.68
F: SD Card / Media	14	113	12.39
F: Toolchange	2	17	11.76

[7] https://github.com/MarlinFirmware/Marlin/pull/16864
[8] https://github.com/MarlinFirmware/Marlin/pull/3110

Label	PR: New Feature	All Pull Requests	Percentage
F: Calibration	52	458	11.35
F: Filament Sensor	1	9	11.11
T: Design Concept	3	28	10.71
C: Peripherals	16	164	9.76
C: Motion	23	238	9.66
C: Temperatures	11	121	9.09
K: Core / H-Bot	1	12	8.33
PR: Configurations	33	420	7.86
C: User Interface	2	30	6.67
F: IDEX	1	15	6.67
C: Serial Comms	7	109	6.42
S: Please Merge	5	90	5.56
C: LCD & Controllers	69	1283	5.38
A: STM32	15	304	4.93
F: BLTouch	2	41	4.88
C: Boards/Pins	41	856	4.79
T: Development	6	133	4.51
F: U.B.L.	5	132	3.79
A: SAMD51	1	29	3.45
F: Trinamic	5	153	3.27
T: HAL & APIs	13	411	3.16
T: Suggestion	1	33	3.03
C: G-code Parser	1	33	3.03
PR: Improvement	60	2067	2.9
C: Optimization	2	85	2.35
K: Deltabot	1	47	2.13
C: Configuration	5	287	1.74
C: Build / Toolchain	8	498	1.61
F: EEPROM	1	90	1.11
Bug: Confirmed !	2	330	0.61
C: Documentation	1	176	0.57
PR: Coding Standards	1	243	0.41
PR: Bug Fix	8	2852	0.28
PR: General Cleanup	2	784	0.26
C: Language	1	518	0.19
A: ESP32	0	43	0.0
A: ATmega2560	0	30	0.0
A: LPC176x	0	27	0.0
A: SAM3X8C/E	0	24	0.0
F: Linear Advance	0	23	0.0
A: ATmega1280	0	14	0.0
T: GitHub Admin	0	14	0.0
F: Print Recovery	0	12	0.0
A: ATmega1284	0	12	0.0
Bug: Potential ?	0	10	0.0
K: SCARA	0	10	0.0
A: AT90USB	0	8	0.0
Bug? False Alarm	0	7	0.0
S: Solved	0	7	0.0
S: Please Test	0	6	0.0
PR: Workaround	0	3	0.0
Bug? Feature!	0	1	0.0

Label	PR: New Feature	All Pull Requests	Percentage
A: SAMD21	0	1	0.0
S: Superseded	0	0	0.0
A: RP2040	0	0	0.0

Table 4.3: Percentage of merged pull requests covered by the `PR: New Feature` label.

The `T: Feature Request` is different from other labels associated with pull requests. This label is intended to be only used for issues and has not been used on a pull requests anymore since 26 September 2017 for pull request #7755[9]. Because the pull request is labeled with `T: Feature Request` we would think that all of them would also be labeled with `PR: New Feature`. We however observe that this is not the case. Of the 20 pull requests that have the T: Feature Request label only 6 are also labeled with `PR: New Feature`. Of the merged pull requests this is 8 and 3. This shows that not everything that is considered a feature request is also labeled as a new feature by the Marlin Maintainers. We can look at the Marlin repository and its issues and find all issues labeled `T: Feature Request`. We can then further refine it by using the filter `linked:pr` which will only show us issues with a directly linked pull request. There are 35 issues with as label `T: Feature Request` with a directly associated pull request. Of those 35 issues only 6 of the associated pull requests have the `PR: New Feature` label most of the rest are either labeled with the `PR: Improvement` label or the `PR: Bug Fix` label. This also holds for feature requests with no associated pull request linked on GitHub. In the comments of these issues a pull request is often mentioned but not officially linked but that pull request is often not labeled as `PR: New Feature`. An example of this is issue #24928[10] and the corresponding pull request of pull request #24951[11]. Issue #24928 asks about the addition of a new flag for a command to disable input shaping and pull request #24951 adds this and more in a pull request labeled with the `C: Motion`, `PR Bug Fix` and `PR: Improvement` labels.

In Table 4.4 we can see the labels of pull requests and how many pull requests of that type there are in the sample set of pull requests with the `PR: New Feature` label. The top ten labels are shown here. A pull request can have multiple labels so a single pull request can be represented multiple times in this table. When looking at the table can can conclude that roughly one-eight of the pull requests with the `PR: New Feature` label also contains the `PR: Improvement` label. Pull requests that have both the `PR: New Feature` label and the `PR: Improvement` label fall roughly into one of two categories. We have pull requests that are seen by the maintainers as both being a new feature and an improvement at the same time for the same piece of code. An example of this is pull request #26652[12] in which a `CONTROLLER_FAN` check is added. This pull request is only about one software feature and is labeled both with the `PR: New Feature` label as the `PR: Improvement` label. The other category is pull requests that are big enough that distinct changes happen in it. An example of this is pull request #17531[13]. This pull request adds both Wi-Fi support to a control board as does it refactor and clarify the SD card logic. So, for the addition of the feature this was labeled with the `PR: New Feature` label and for the clarification this was labeled with the `PR: Improvement` label.

The second most used label for new features is that for LCD screens and Controllers. This label indicates that new software features are introduced to handle the screens and the controllers of the machines that Marlin controls. The Marlin firmware started with the support of basic character-based LCD screens and over time grew to encompass more commonly used low resolution LCD panels and now even supports high resolution screens and touchscreens. Marlin also has a variety of different UI options available depending on the resolution of the screen.

After this comes the label that is used for the calibration of the platform. Software features

[9] https://github.com/MarlinFirmware/Marlin/pull/7755
[10] https://github.com/MarlinFirmware/Marlin/issues/24928
[11] https://github.com/MarlinFirmware/Marlin/pull/24951
[12] https://github.com/MarlinFirmware/Marlin/pull/26652
[13] https://github.com/MarlinFirmware/Marlin/pull/17531

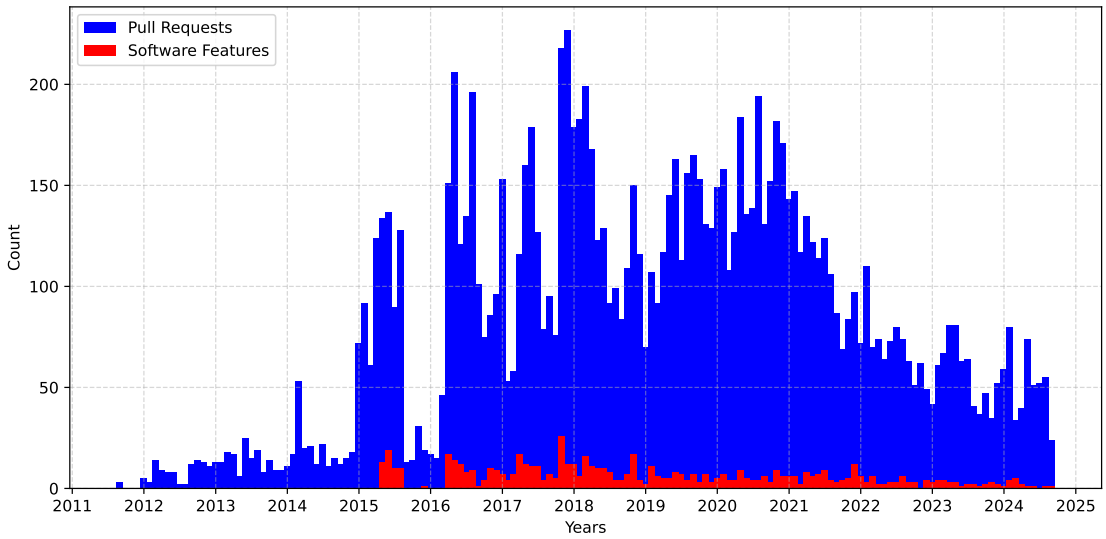

Figure 4.6: Total number of pull requests and number of New Features submitted per month.

Label	number of Pulls	Percentage of Total
PR: Improvement	88	13,29%
C: LCD & Controllers	86	12,99%
F: Calibration	64	9,67%
C: Board/Pins	56	8,46%
PR: Configurations	38	5,74%
Needs: Testing	34	5,14%
C: Motion	28	4,23%
Needs: Work	27	4,08%
C: Hosts & Protocols	25	3,78%
F: Z-Probes	25	3,78%
PR: New Feature	662	100%

Table 4.4: Labels used in addition for pull requests with the `PR: New Feature` label.

that have this label often also have other labels. Such as the `C: LCD & Controllers` label and the `F: Z-Probes` label. This is because the `F: Calibration` label is about the calibration of the machine running Marlin. To properly calibrate the machine a probe is necessary. So, the `F: Z-Probes` label is often involved with calibration. While calibrating the firmware also often returns information to the user by use of the screens, so this is why the `C: LCD & Controllers` label is often involved.

The `C: Board/Pins` label is also often used together with the `PR: New Feature` label. Out of the 56 pull requests with this label 29 of them involve adding support of a new control board to the Marlin firmware. If we look at the pull request with the `C: Board/Pins` label without the `PR: New Feature` label we can still see a lot of pull requests that are about the adding of support for control board and other controllers. For example pull request #24722[14] has both the `C: Boards/Pins` label and the `PR: New Feature` label. But the pull request #27260[15] only has the `C: Board/Pins` label. Both pull requests do the same thing, adding support for a new type of board to Marlin. They both edit the same kind of files to do this, and the one labeled with `PR: New Feature` does not do anything extra over the other pull request.

Discussion. We have been unable to find an explanation for the sudden decrease in the number of pull request being submitted to Marlin at the end of 2015 as can be seen in Figure 4.5. The main maintainer of Marlin changed that year, but that was already in April long before the drop occurred. After this period the pull requests again increased in number.

A possible explanation for the lack of labels before 2015 besides that the maintainers of Marlin did not label anything could be that because the Marlin repository was changed in 2015 from the account of the then owner to a shared Marlin account. It could be that during this transfer the labels were removed from the pull requests for some reason.

A reason for why the number of pull requests for the `F: CNC / Laser` covered by the `PR: New Feature` pull requests is so high could be that the use case of a CNC / Laser device is an edge case that Marlin supports. Because of this the work done for these types of machines only happens when something is broken or when a new feature is being implemented. So because of that the software features occupy a higher portion than average of the number of pull requests. The same type of logic can be applied to a label like `S: Experimental`. The `S: Experimental` is mostly used for experimental new features.

The reason that so many issues with the `T: Feature Request` label are not actually a new feature is likely that Marlin allows users to attach the `T: Feature Request` label themselves to the issues. So, a lot of feature requests are actually bug fixes, improvements or other things according to the maintainers of Marlin.

Because the support for different LCD screens has evolved a lot over the years a lot of new features are required to support all the new properties of such screens such as a higher resolution or touchscreen. Add to this that the support of a specific type of screen is also seen as a new feature and we have a lot of pull requests labeled with the `PR: New Feature` label.

Marlin contains many pull requests that add the support of new control boards for 3D-printers. These pull requests are all very similar and they achieve the same goal. However, some of these pull requests are labeled with the `PR: New Feature` label and some of them are not. This is an example of the inconsistency among the maintainers of Marlin in labeling pull requests as new features.

[14] https://github.com/MarlinFirmware/Marlin/pull/24722
[15] https://github.com/MarlinFirmware/Marlin/pull/27260

4.2 RQ1: The Life Cycle of Labeled Software Features

4.2.1 Marlin Feature Labels

Results. Table 4.5 shows us all the labels in Marlin that have merged pull requests associated with them that are also labeled with the `PR: New Feature` label. For all of these label we show the total number of merged software features, the number of software features that have been analyzed, the percentage of the total the analyzed form, the total number of commits that are associated with pull requests with that label, the average number of commits per pull request with that label and the average time between commits for that label.

Label	Merged New Features	Tagged and Analyzed	Percent of Total	Total Number of Commits	Average Number of Commits	Average Time between Commits
PR: New Feature	472	189	40.0	2956	15.6	167.6
C: LCD & Controllers	69	37	53.6	755	20.4	110.5
PR: Improvement	60	12	20.0	180	15.0	141.4
F: Calibration	52	12	23.1	264	22.0	132.4
C: Boards/Pins	41	9	22.0	203	22.6	89.9
PR: Configurations	33	0	0.0	0	0.0	0.0
C: Motion	23	18	78.3	434	24.1	138.5
F: Z-Probes	22	7	31.8	120	17.1	155.3
C: Hosts & Protocols	19	9	47.4	90	10.0	247.6
C: Peripherals	16	16	100.0	152	9.5	195.9
Needs: Testing	16	8	50.0	92	11.5	188.0
A: STM32	15	13	86.7	295	22.7	94.3
F: SD Card / Media	14	6	42.9	64	10.7	122.2
T: HAL & APIs	13	4	30.8	26	6.5	203.8
S: Experimental	13	3	23.1	48	16.0	162.7
Needs: Work	13	7	53.8	59	8.4	205.1
C: Temperatures	11	11	100.0	124	11.3	181.3
C: Build / Toolchain	8	8	100.0	172	21.5	127.8
C: Safety	8	8	100.0	147	18.4	200.2
PR: Bug Fix	8	1	12.5	13	13.0	89.0
F: CNC / Laser	7	7	100.0	120	17.1	107.7
C: Serial Comms	7	6	85.7	58	9.7	194.2
S: Don't Merge	6	3	50.0	27	9.0	231.3
Needs: More Data	6	1	16.7	7	7.0	264.0
T: Development	6	2	33.3	89	44.5	34.0
Needs: Discussion	6	3	50.0	41	13.7	172.3
S: Please Merge	5	3	60.0	96	32.0	60.3
C: Configuration	5	4	80.0	24	6.0	215.8
F: Trinamic	5	4	80.0	37	9.2	391.8
F: U.B.L.	5	2	40.0	9	4.5	189.5
T: Feature Request	3	0	0.0	0	0.0	0.0
T: Design Concept	3	0	0.0	0	0.0	0.0
F: BLTouch	2	1	50.0	18	18.0	52.0
PR: General Cleanup	2	0	0.0	0	0.0	0.0
Bug: Confirmed !	2	0	0.0	0	0.0	0.0
C: Optimization	2	1	50.0	34	34.0	45.0
C: User Interface	2	2	100.0	79	39.5	20.5
F: Toolchange	2	2	100.0	12	6.0	151.5

Label	Merged New Features	Tagged and Analyzed	Percent of Total	Total Number of Commits	Average Number of Commits	Average Time between Commits
help wanted	2	1	50.0	2	2.0	398.0
T: Suggestion	1	1	100.0	6	6.0	195.0
F: EEPROM	1	0	0.0	0	0.0	0.0
S: Hold for 2.1	1	1	100.0	27	27.0	64.0
F: IDEX	1	0	0.0	0	0.0	0.0
K: Deltabot	1	1	100.0	5	5.0	329.0
C: Language	1	0	0.0	0	0.0	0.0
Needs: Documentation	1	1	100.0	37	37.0	42.0
PR: Coding Standards	1	0	0.0	0	0.0	0.0
C: G-code Parser	1	1	100.0	34	34.0	45.0
A: SAMD51	1	1	100.0	18	18.0	50.0
F: Filament Sensor	1	1	100.0	4	4.0	123.0
C: Documentation	1	0	0.0	0	0.0	0.0
Needs: Patch	1	0	0.0	0	0.0	0.0
K: Core / H-Bot	1	1	100.0	18	18.0	54.0

Table 4.5: Table of the labels used for the merged pull request with the `PR: New Feature` label, with the analyzed pull requests, the total number of commits, the average number of commits and the average time between commits in days.

Table 4.5 shows us that the set of analyzed pull requests sometimes covers all the merged features, and sometimes it covers no merged feature. This is especially prevalent for labels with only one or two associated merged new features. This is caused by the that certain labels have only been in use for a certain amount of time. For example the label `T: Design Concept` has not been used in combination with a pull request with the `PR: New Feature` label since pull request #12895[16] which was merged on 24 January 2019 which is a few months before the date that our sample set starts from. On the other hand, a label like `C: Temperatures` is fully covered by our sample set. This is because the `C: Temperatures` label was not introduced until 15 April 2020 when pull request #17560[17] was introduced. This pull request is the first time that the `C: Temperatures` is used in the Marlin repository. There are three issues that were submitted before this pull request that also have this label. But they were all given this label after the pull request had received it. For the other labels roughly half to a quarter of the software features related to those labels have been covered.

In total 2,957 commits have been investigated while tagging the pull requests. Of those 2,957 commits, 1,635 of them are unique, this is because a commit can have effect on multiple software features at once. In Table 4.6 we can see the 15 commits which interact with the greatest number of pull requests. The biggest example of this is the commit associated with pull request #25908[18]. This pull request removed the two often used macros of `EITHER` and `BOTH`. This causes this commit to impact 62 of the 189 pull requests that we have tagged. This is by far the largest impact any of the commits has that we have examined. The next commit in line only impacts 26 of the pull requests, and that one is also related to macros. Of the top five commits that touch multiple software features, only one of them the dumping of the `BOTH` and `EITHER` macros is associated with a pull request. The other four commits in the top five are all commits that are not associated with any pull request. All these commits have been committed to the Marlin project by the main maintainer of the project.

[16] https://github.com/MarlinFirmware/Marlin/pull/12895
[17] https://github.com/MarlinFirmware/Marlin/pull/17560
[18] https://github.com/MarlinFirmware/Marlin/pull/25908

Commit	Total Number of Commits
02-05-2023: Dump BOTH and EITHER macros (#25908)	62
09-09-2021: Fewer serial macros	26
28-12-2021: Remove extraneous 'inline' hints	22
14-01-2020: Moved configurations to a separate repo	21
19-09-2021: Reduce language file sizes	21
02-06-2023: Remove LOOP macros (#25917)	19
27-04-2023: Optimize PlatformIO source filtering (#25332)	17
11-05-2023: Use 'build_src_filter' (#25810)	16
24-03-2021: More IntelliSense-friendly declarations	15
22-05-2022: Apply F() to more LCD code (#24228)	14
07-02-2024: MARLIN_SMALL_BUILD option (#26775)	13
22-04-2020: Apply TERN to compact code (#17619)	13
14-06-2023: Group STM32G0 pins BUILD option (#26775)	12
23-04-2023: Split Changes.h from SanityCheck.h (#25732)	12
07-09-2021: Standardize G-code reporting	11

Table 4.6: Commit title with number of times commit is present in the sample dataset.

If we take a look at the commits in Table 4.5, we can see that the software features labeled with the `C: LCD & Controllers` label has the highest number of commits. This is inline with the fact that the number of software features related to the label is also the highest of our sample set. This is also the case for the other labels with a high number of associated software features. There are some exceptions, however both `T: Development` and `C: User Interface` have a high number of commits while only having two software features associated with them. We observe in the table that the average number of commits associated with a new software feature is 15.6. Some labels have a much larger number of average commits associated with them, but those labels have only one, two or three pull requests associated with them. Figure 4.7 shows us the average number of commits, but only for labels that have more than four software features associated with them. Here we see that of our very long list of labels only 18 labels have more than four pull requests associated with them. With this smaller set we observe that the extremes on both ends have been reduced significantly with as highest the `C: Motion` label with an average of 24 roughly 1.5 times the average for software features and on the low end we have the `Needs: Work` label with an average of eight commits so roughly half of the average for a software feature.

The last column of Table 4.5 shows the Average time between commits represented in Days. For a average new feature the average time between its commits its 168 days. This number has a relation with the average number of commits of a label. In general when the average number of commits goes up the average time between commits goes down. The label with the lower average time between commits is the `C: User Interface` label. This label only contains two merged software features: One which was merged late 2021 and another that was merged early 2023. Both of these relatively new features have a high number of associated commits. Which has given the label of `C: User Interface` an average time between commits of 20.5. The label with the highest average time between commits is the `help wanted` label. We have only tagged one of its software features. This software feature is one of the two features that was removed from the project and because of this it only has 2 associated commits. Because of this the Average time between commits has gotten up to 398 days.

Discussion. For the average number of commits it is important to know when the pull requests were released. Older pull requests will have had more time to evolve in the Marlin system. This could skew the averages of certain labels up or down depending if those label contain a lot of new or old pull requests.

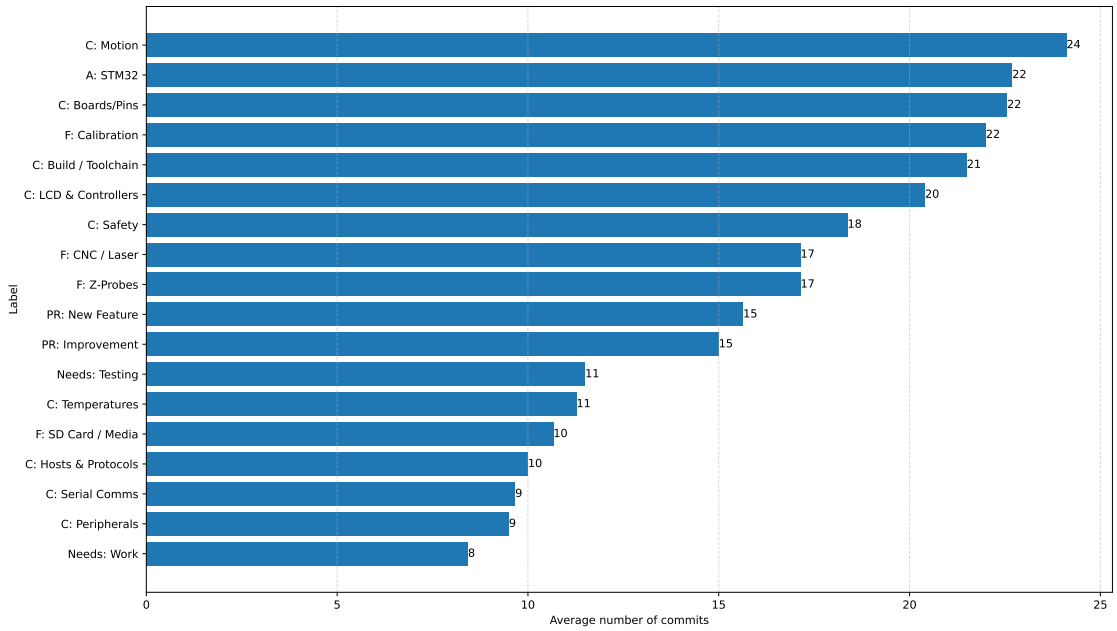

Figure 4.7: Average number of commits associated with manually labeled software features with a certain label with more than 5 occurrences in the dataset.

We see that the very high and very low average numbers are associated with labels that contain only some commits. This indicates the possibility for outliers in datasets with only a few data points. It is important to note that the numbers released for labels with a small sample size might be less reliable than those with a higher number of samples.

While examining the pull requests associated with the `C: User Interface` label we noticed that label is relatively new only first being used at the end of 2020. Because of this many other new features that are related to the UI are labeled with the `C: LCD & Controllers` label. But even now that the `C: User Interface` label has been introduced do the maintainers of Marlin still label changes to the UI with the `C: LCD & Controllers` label. This is another example of the maintainers of Marlin not being consistent with their application of labels.

4.2.2 Feature Commit Tags

Table 4.7 shows us all the tags that we have used to tag the commits, with next to the tags the number of commits associated with it, the number of unique commits associated with it and the number of commits that contain the `NewFeature` tag and at least one other tag. We have ordered it in the order of importance of the tag as described in section 3.5. The biggest category is that of the `Refactor` tag with 1,779 of the 2,956 commits being tagged with it. We can see that commits with the `Refactor` tag are most likely to affect multiple different pull requests at once. As shown by that the unique commit count is roughly half of its total commit count. Most of the commits of this type could be considered as `adaptive maintenance`.

The second-biggest category is that of the tag `Enhancements`. These commits extend the functionality of the original software feature. Most of the commits of this type can be considered `perfective maintenance`.

Thirdly, we have the category of the tag `BugFix`. These commits often happen quite quickly after another commit related to the software feature because that other commit broke something of the feature. It is also possible that a change not directly related to the software feature breaks some of the functionality of the software feature. For those instances, it is not possible to easily discover what the time between the functionality breaking change and the bug fix was. Commits

Tag	Number of Commits	Unique Number of Commits	Number of Commits With Both a `NewFeature` Tag and at Least One Other Tag
`NewFeature`	189	189	79
`Removal`	2	2	0
`Rework`	18	18	4
`Revert`	19	10	1
`BugFix`	353	321	2
`Enhancement`	367	316	39
`Refactor`	1779	902	89
`Cleanup`	51	44	0
`Comment`	101	86	12
`Formatting`	63	43	2
`Whitespace`	14	9	0

Table 4.7: Commit tags together with number of commits with that tag, unique number of commits with that tag and number of commits tagged with `NewFeature` and other tag.

of this type can be considered `corrective maintenance`.

We can see in the Table 4.7 that out of the 189 commits tagged as `NewFeature` 79 of them have at least one other tag associated with them. The commit with the highest number of extra tags is commit e5b651f[19] it is tagged with 1 `NewFeature`, 5 `Enhancement`, 3 `Refactor` and 1 `Comment` tag. This commit is part of the pull request #23112[20] which extends the support of the linear axes from 6 to 9 for Marlin.

Table 4.8 shows us different pull requests that are new features and the other tags it has been tagged with. All the pull requests impact other features by making it so that they are refactored and over half of them impact other features by enhancing them. The most common label associated with these pull requests is the `C: LCD & Controller` label with 4 of the 15 labeled with that. This is also the label that occurs the most in our sample set besides the `PR: New Feature` label.

Discussion. The fact that the `Refactor` tag is the most common tag can be explained by the fact that a lot of changes that happen to a software feature happen to either optimize the code, change the name of a function that has been updated or move code around for readability.

That the commit e5b651f impacts so many different software features is most likely because it is a very large commit with 4540 lines of additions to the code base of Marlin. A lot of features in Marlin depend on the motion system and it being expanded impacts them. Five other software features have gotten extra functionality because of it, three other features have been refactored to work with the new system and for one software feature a comment has been updated. This is a perfect example of how a change to one system can impact many more.

RQ1.2: We have seen that it is possible for software features to have their feature enhanced by the introduction of new software features. This shows that interconnection of different software features together form greater systems than their separate parts. And that systems evolve by adding new software features on top of older ones.

4.2.3 Removal of a Software Feature

Results. The most impactful commit tag besides `NewFeature` is the `Removal` tag. This tag indicates that an entire software feature gets removed from the system. In the sample of software features that we have tagged this has happened only twice. The first time was when the commit

[19] https://github.com/MarlinFirmware/Marlin/commit/e5b651f407fcb743e2d00c45b0d361fb98230efb
[20] https://github.com/MarlinFirmware/Marlin/pull/23112

Title	NewFeature	Rework	Enhancement	Refactor	SideCommit	Total Number
Support for up to 9 axes (linear, rotary) (#23112)	1	0	5	3	1	10
Config INI, dump options (#24528)	1	1	0	3	3	8
Switching extruder/nozzle without servo (e.g., Dondolo) (#24553)	1	0	0	5	0	6
MKS Robin Nano v3 + TFT_LVGL_UI + WiFi module (#22109)	1	0	1	4	0	6
MSC Support for STM32 + SDIO boards ->SKR 2 (#22354)	1	1	0	3	1	6
Polar Kinematics (#25214)	1	0	0	5	0	6
Compact RGB565 TFT boot images (#26011)	1	0	0	4	0	5
More flexible redundant temp sensor (#22085)	1	0	2	2	0	5
Add TEMP_SENSOR_BOARD (#22279)	1	0	1	2	1	5
TEMP_SENSOR_SOC (#25642)	1	0	1	3	0	5
MarlinUI Endstop Test Screen, and more (#25667)	1	0	0	2	1	4
LCD_BACKLIGHT_TIMEOUT for Neopixel LCD (#25438)	1	0	1	1	1	4
MAX Thermocouples for Heated Bed (#26441)	1	0	1	1	1	4
SMUFF (MMU2 clone) support (#19912)	1	0	0	3	0	4
Multi Volume Support - LVGL Media Select Screen (#21344)	1	0	2	1	0	4

Table 4.8: Pull requests together with number of commits with that tag.

associated with pull request #24229[21] which removed a just introduced feature. The second time was when the commit associated with pull request #24427[22] which removes the support for a series of stepper drivers. There is one more commit that drops support of a platform in Marlin in the commits that we have tagged. This commit, however, was not the end of one of our tagged software features, only a partial removal of some of its functionality. This is the commit associated with pull request #20153[23].

Pull request #24229 removes the software feature introduced in pull request #24074[24]. The software feature was merged on 23 May 2022 at 14:19:10 and was removed on 23 May 2022 at 19:39:50. The feature was only present in Marlin for a little over five hours. The feature was the introduction of a simple prompt for uploading the firmware to the SD card to improve the user experience. The pull request was merged too early as it was not working correctly for Macs as can be read in the comments of the pull request. Five hours after the merge a new pull request was opened to remove the feature again. There has not been a new pull request since to introduce this feature again in a working fashion.

Pull request #24427 removes the software feature introduced in pull request #16452[25]. Pull request #16452 introduced a software feature to add support for a variety of stepper drivers of the `STMicro L64XX` family. This new feature was in reality already a reimplementation of an older pull request. This pull request #13498[26] was the initial basis of supporting a specific driver of the L64XX family. This was then rewritten into a new pull request that had support for the entire family of stepper drivers. The pull request got merged on 14 January 2020 and the commit that removed it was merged on 14 July 2022. This means that the feature was present in Marlin for two and a half years before being removed. The reason that it was removed seems to be that the type of drivers was not widely used in the Marlin ecosystem and there was a better alternative

[21] https://github.com/MarlinFirmware/Marlin/pull/24229
[22] https://github.com/MarlinFirmware/Marlin/pull/24427
[23] https://github.com/MarlinFirmware/Marlin/pull/20153
[24] https://github.com/MarlinFirmware/Marlin/pull/24074
[25] https://github.com/MarlinFirmware/Marlin/pull/16452
[26] https://github.com/MarlinFirmware/Marlin/pull/13498

available.

Figure 4.8 shows a directly follows graph (DFG) of the software feature introduced in pull request #16452. A DFG is a type of graph in which all the connections between different activities are shown in the form of directed edges between nodes. The graph is created by going chronological through the order of commit tags that impact a software feature. In it, we can observe the relation between the tags and how many times each of the tags is present. We notice that the net is centered around the `Refactor` tag. The `Refactor` tag has 28 of the 50 commits associated with it. This number is in line with the average proportion of `Refactor` tags to other for all the software features in the sample set. It is also the only tag that has a connection with all other tags. The fact that the `Refactor` tag has the highest number of commits shows us that for this software feature most of the commits where about `adaptive maintenance` just as is the case for the general software features. The `Enhancement` tag is with 14 out of 50 commits overrepresented in comparison with the average of the sample dataset. This can be explained by that this feature is about the introduction of support of a new type of stepper driver. Because this is a clean state of a feature with a lot of possible directions it allows for the growth of many enhancements unlike features that are more specific in their design. The `BugFix` tag on the other hand is underrepresented in this feature in comparison to the average. Of the bug fixes three were caused by previous enhancements that had not fully changed all the variables and the problem was present in an edge case that was not caught until later. And the last one was caused by a change to a different part of the system that broke something in this feature. The `Revert` that happened for this feature is a commit that is present in more features that deal with hardware. It was a refactor of the entire hardware abstraction layer (HAL) structure as a singleton that was merged but still contained too many problems.

Pull request #20153 removes the software features introduced in pull request #10434[27] and pull request #9150[28]. This pull requests removes the `HAL/STM32_F4_F7`. This is the HAL for the `STM32F4` and `STM32F7` environments. The reason for the removal is that few boards make use of this HAL and other boards of the class `STM32F4` and `STM32F7` are already supported by the generic `STM32` HAL. Pull request #10434 added support for the `STM32F4` HAL on 18 April 2018 and pull request #9150 added support for the `STM32F7` HAL on 15 January 2018. These two HALs were merged in pull request #14566[29] on 19 July 2019. The removal of this feature happened on 15 November 2020. This means that the feature was removed after roughly 3 years of service. Neither pull request #10434 nor pull request#9150 was originally marked with the `PR: New Feature` label.

Discussion. Of the features that have been introduced in the five years that our sample set spans only two features have been removed. One of them because it was already broken when it realized and the other because it was not widely used and it was already obsolete by the time it was originally released. The developer of the code at least thinks that is the case as can be read in his comment[30].

The most likely reason that features are not removed, as is normally the case when hardware becomes outdated and hard to maintain is that Marlin has a different design philosophy. A quote from the Marlin website[31]: "One key to Marlin's usefulness is that it's built around the lightweight Arduino framework, so it runs on a huge number of inexpensive micro-controllers from classic Atmel AVR 8-bit boards all the way up to the latest ARM 32-bit OEM and upgrade boards from companies like BigTreeTech and Makerbase.

Marlin aims to support all possible boards and machine configurations. We want it to be configurable, customizable, extensible, and economical for hobbyists and vendors alike. A minimal Marlin build can be very small (under 64KB), for use on a headless printer with only modest hardware. Features are enabled as-needed to support added components."

We can read that the developers of Marlin try their best to keep sort of all possible platforms

[27] https://github.com/MarlinFirmware/Marlin/pull/10434
[28] https://github.com/MarlinFirmware/Marlin/pull/9150
[29] https://github.com/MarlinFirmware/Marlin/pull/14566
[30] https://github.com/MarlinFirmware/Marlin/pull/13498#issuecomment-502046804
[31] https://marlinfw.org/docs/basics/introduction.html

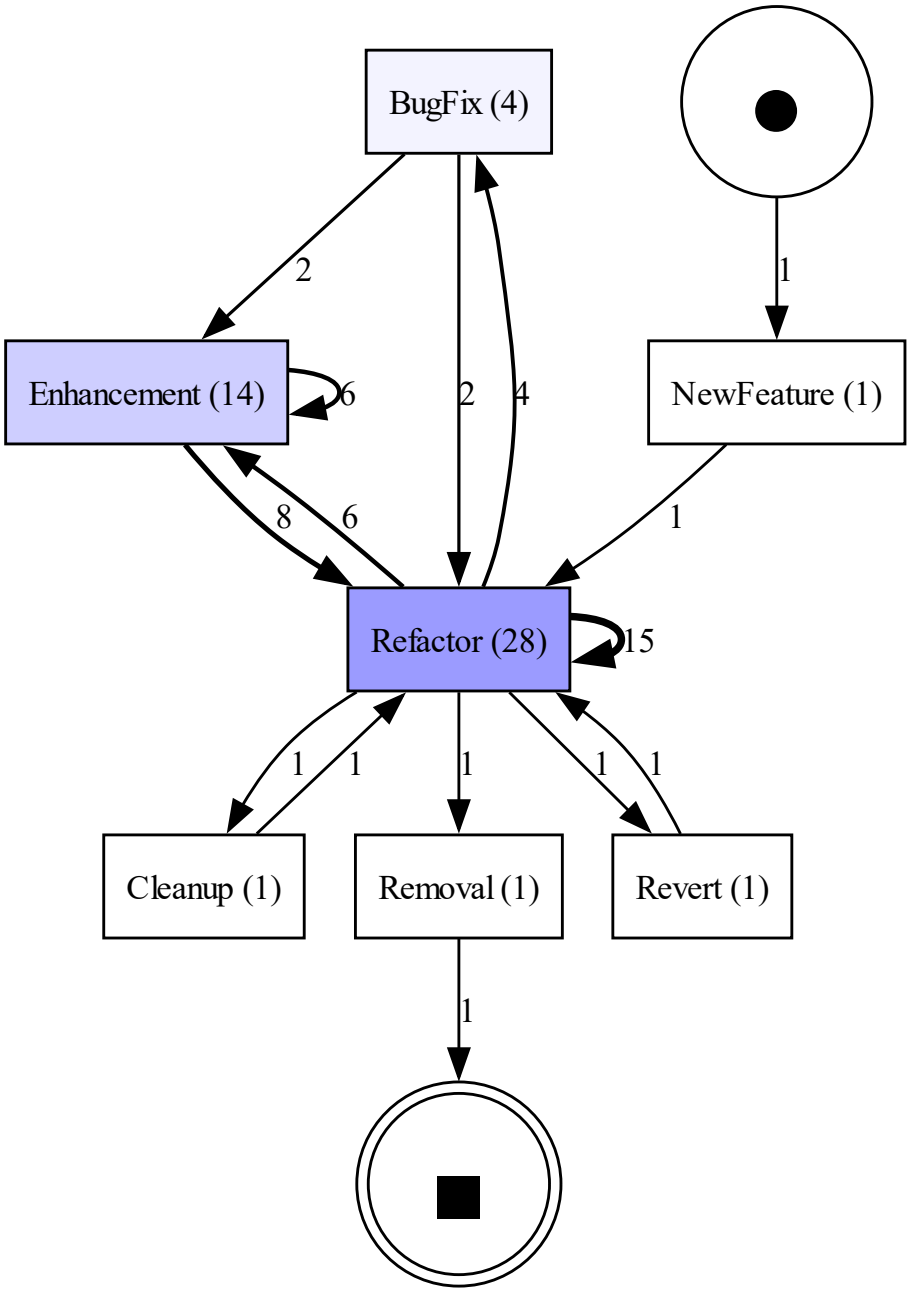

Figure 4.8: DFG of pull request #16452.

Pull	NewFeature	BugFix	Enhancement	Refactor	Revert	Rework	Cleanup	Comment	Formatting	Whitespace	Removal	Total
#18071	1	39	15	119	0	0	3	5	3	1	0	186
#23112	1	11	1	64	0	0	0	1	0	0	0	78
#16293	1	13	19	39	1	2	1	0	0	0	0	76
#16277	1	10	21	39	2	1	2	0	0	0	0	76
#21255	1	3	4	44	1	0	5	6	5	0	0	69
#21503	1	4	4	43	0	0	3	5	3	1	0	64
#22418	1	15	2	29	2	0	2	4	4	3	0	62
#21931	1	4	4	41	0	0	1	2	1	0	0	54
#20384	1	4	8	31	0	0	2	5	1	0	0	52
#15590	1	6	17	27	0	1	0	0	0	0	0	52

Table 4.9: Pull requests together with number of commit tags.

that is configurable for everyone. This can also be seen in several pull requests such as #26775[32] which is a relatively new option that reduces the build size of Marlin by sacrificing some serial feedback. This is done to better support the control boards with a lower amount of flash storage, such as old control boards. A big effort is made to not increase the build size of Marlin or the flash usage for example this comment[33] is about using a technique that will save 37 bytes of data.

4.2.4 Life stages of a Software Feature

Results. In Table 4.9 we can see the 10 pull requests with the highest number of commits associated with them. Next to the pull request number is noted how many of every commit tag is associated with that particular pull request. The average number of commits associated with pull requests is 15.6 so all these pull requests have significantly more associated with them. They have between 3-4 times as many than the average with the pull request with the most commits even having 12 times as many commints.

Example. The pull request #18071[34] has by far the highest number of commits associated with it. The most likely reason for this is that this pull request is the addition of an entire `GUI` library to Marlin. The pull requests added 16,162 lines of code to the project and is spread out over 68 different files. This means that there is a lot of code for other features to be built upon or refactored.

Figure 4.9 shows us the DFG of this pull request. Just like with the DFG of pull request #16452 the center of this DFG is the `Refactor` tag. The `Refactor` tag has an outgoing edge to every other tag besides the `NewFeature` tag and the `Whitespace` tag. The `Refactor` tag has 77 self edges and only the `BugFix` tag also has self edges. 11% (39/353) of all `BugFix` commits in Marlin impact this pull request which is vastly above the average. On average a pull request contains 2 bug fixes. Eventhough this pull request has the highest number of commits associated with it, it has never been impacted by either a revert or rework.

Example. The pull request #23112[35] has the second-highest number of commits associated with it. The reason for the many commits is that it impacts the support of linear axes in Marlin. Because of that this feature interacts with a lot of the movement system and so many changes made to the movement system much change something about this feature. There is only one

[32] https://github.com/MarlinFirmware/Marlin/pull/26775/files
[33] https://github.com/MarlinFirmware/Marlin/pull/25781#issuecomment-1547043417
[34] https://github.com/MarlinFirmware/Marlin/pull/18071
[35] https://github.com/MarlinFirmware/Marlin/pull/23112

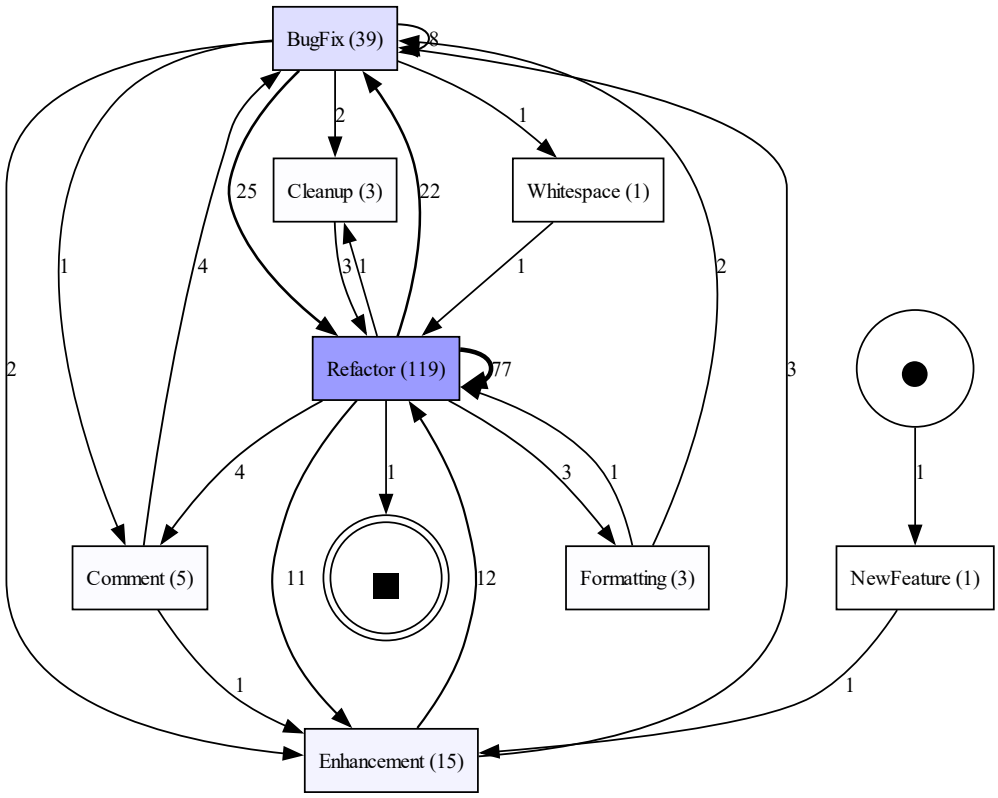

Figure 4.9: DFG of pull request #18071.

`Enhancement` commit that touches this pull request. That is the commit f5daefb[36] which was committed only 2 days after the merger of the feature. It adds the 9 linear axes support to an extra platform.

In Figure 4.10 we can see the DFG of this pull request. Just like the other DFGs this one is centered around the `Refactor` tag. The noticeable thing is that this pull request contains only a `Comment` tag commit from the side commit category of commits.

Example. The pull request #22790[37] is an average pull request. With an average number of total commits associated with it and an average number of `Refactor` commits associated with it. This pull request adds the functionality to Marlin to act as a polargraph. This is now one of the many niche features that Marlin supports. Another niche feature is its ability to function as a laser engraver or a CNC machine.

Figure 4.11 shows us the DFG of this pull request. Just like in the other DFGs here the `Refactor` tag is also central to the DFG. But unlike the other DFGs the `Refactor` tag has only 2 outgoing connections to other tags. In this case the `Refactor` tag has more incoming connections than it has outgoing.

In Figure 4.12 we can see the DFG of all the pull requests from the sample set together with all the side commit tags grouped together. And in Figure 4.13 we can see the DFG of all the pull requests from the sample set but the edges between the main tags have been removed. This has been done for the readability of the figures. In Figure 4.14 we can see the footprint of all the pull requests. In it is shown how all the different tags interact with each other. A `||` sign indicates that there exists both a direct connection from that tag to that other tag and back. The `>` sign indicates that only from the tag on the left side there is a direct connection to the tag on the top side. The `<` sign indicates that only from the tag on the top side there is a direct connection to the tag on the left side. And the `#` sign indicates that there is no direct connection between the two tags. In it we can see that the tags that are most common have the most type of connections with other tags. In the figure we can see that especially for the `Refactor` tag a lot of self-edges happen. Not all commits contain self-edges. Only the commits with a high enough occurrence contain self-edges. The tags that only have a few occurrences do not have connections with every of the tags. The higher the number of occurrences of a tag the more likely it is to have connections to all the tags.

We display how often the tags happen in a row for the pull requests in Table 4.10. All the side commit tags have been gathered together in a single tag. It is visible that for the `Refactor` tag there are many cycles with a length longer than 5 while none of the other tags have this.

In Figure 4.15 we can see the DFG of pull requests with at most 9 commits associated with them. For this DFG the tags have been renamed. A number has been added after the tag to represent which commit it was in the life cycle of the feature. It starts with 0 at the `NewFeature` tag and then becomes 1, 2, 3, etc... This DFG covers 98 of the 189 features in our sample set. In it we can follow the routes that features take through their life. If we at every step follow the most walked route we get the following path: `NewFeature_0`, `Refactor_1`, `Refactor_2`, `Refactor_3`, `Refactor_4`, `Refactor_5`, `Refactor_6`, `Refactor_7`, `END`. Our sample set contains in total 18 features which only contain the `NewFeature` tags, 49 features which only contain the `NewFeature` and `Refactor` tag and 61 features that only contain the `NewFeature`, `Refactor` and `Side Commit` tag. So a third of our sample set is features that are introduced and since their introduction no functional changes have happened to them. A DFG containing 188 pull requests with a depth of 77 is available together with the dataset on the GitHub repository[38]

There are two tags that represent an event that is directly caused by another event and those are the `BugFix` tag and the `Revert` tag. In Table 4.11 we can see the tags of commits that happened before a `BugFix` tag. The `NewFeature` and `BugFix` tags are slightly overrepresented in comparison to how much they occur in the sample set (10.7% vs 6.4% and 16.9% vs 12%) and the `Enhancement` tag is slightly underrepresented in comparison to the sample set (50.2% vs 60%).

[36] https://github.com/MarlinFirmware/Marlin/commit/f5daefb09d1fdf3fc931e2ce84a28d4af1ba2bea
[37] https://github.com/MarlinFirmware/Marlin/pull/22790
[38] https://github.com/AHofstad/TheLifeOfSoftwareFeatures/blob/main/images/DFGSize100.pdf

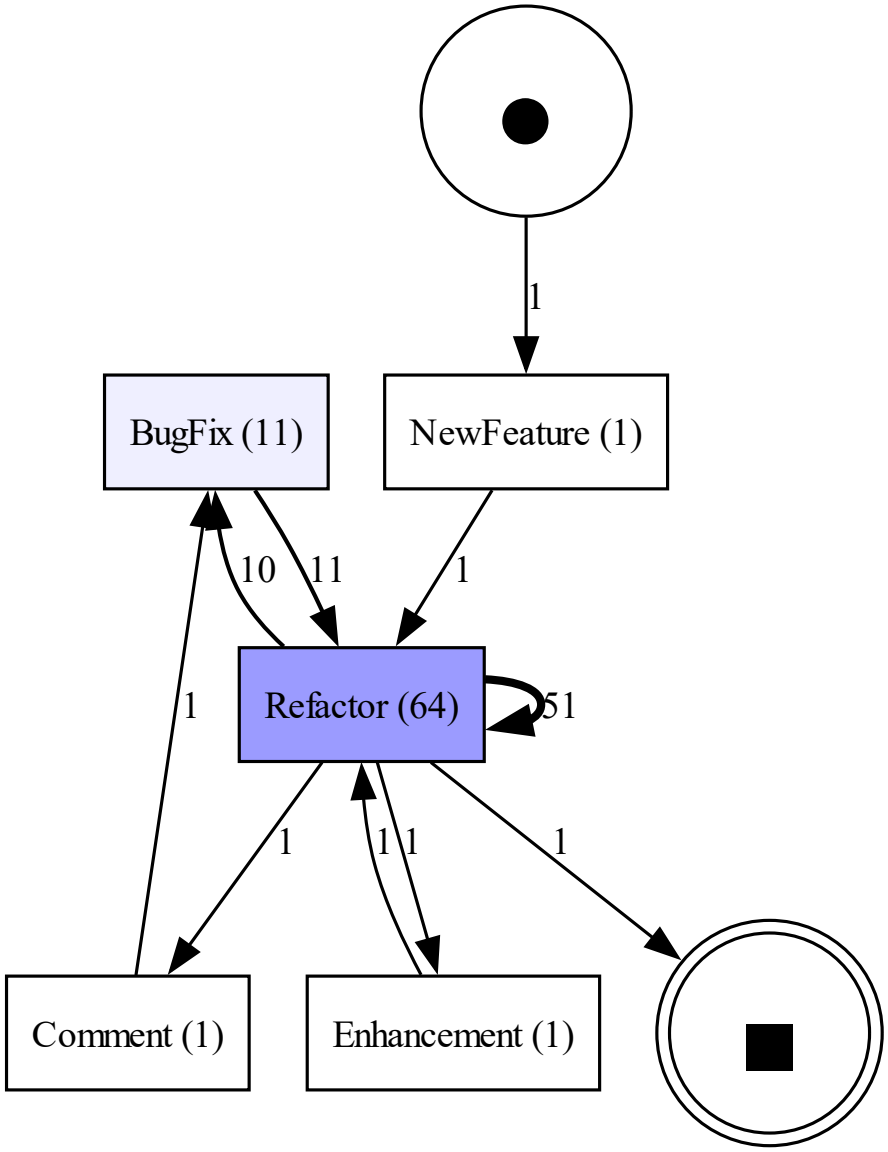

Figure 4.10: DFG of pull request #23112.

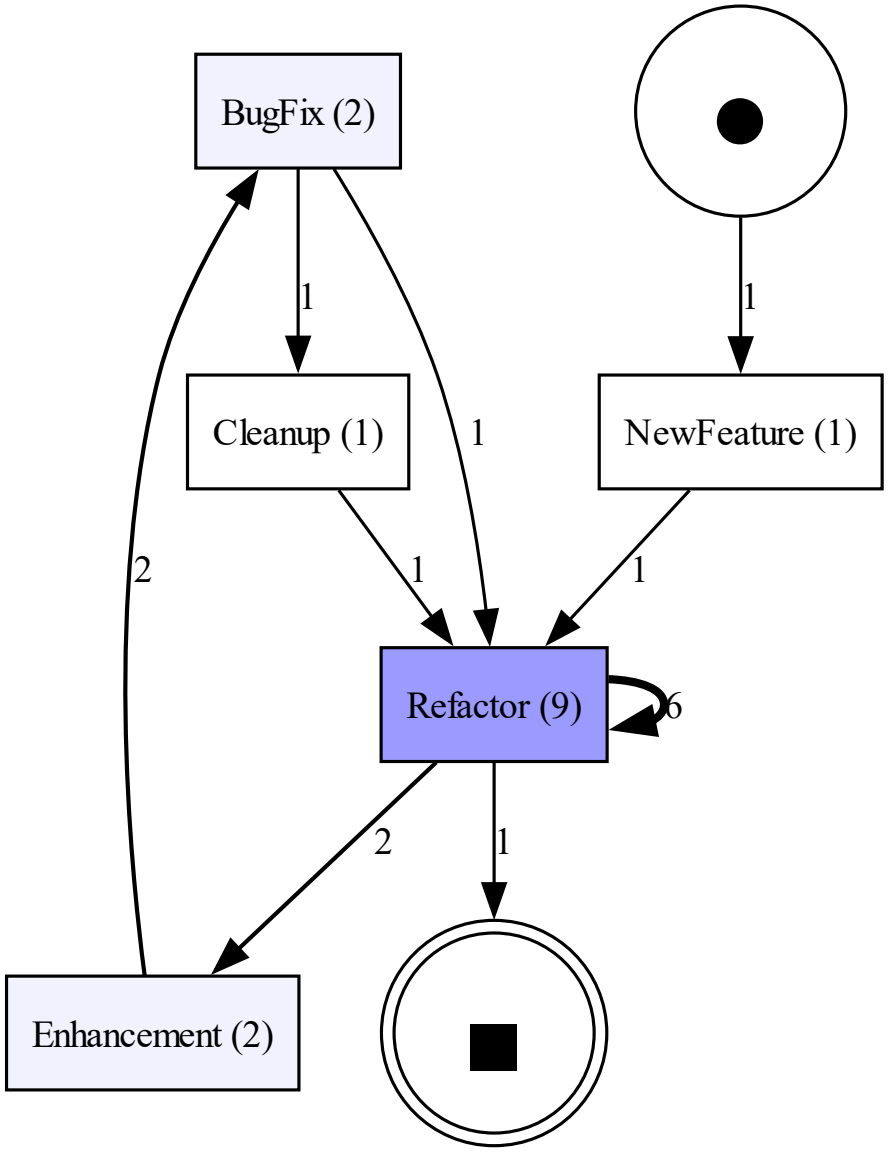

Figure 4.11: DFG of pull request #22790.

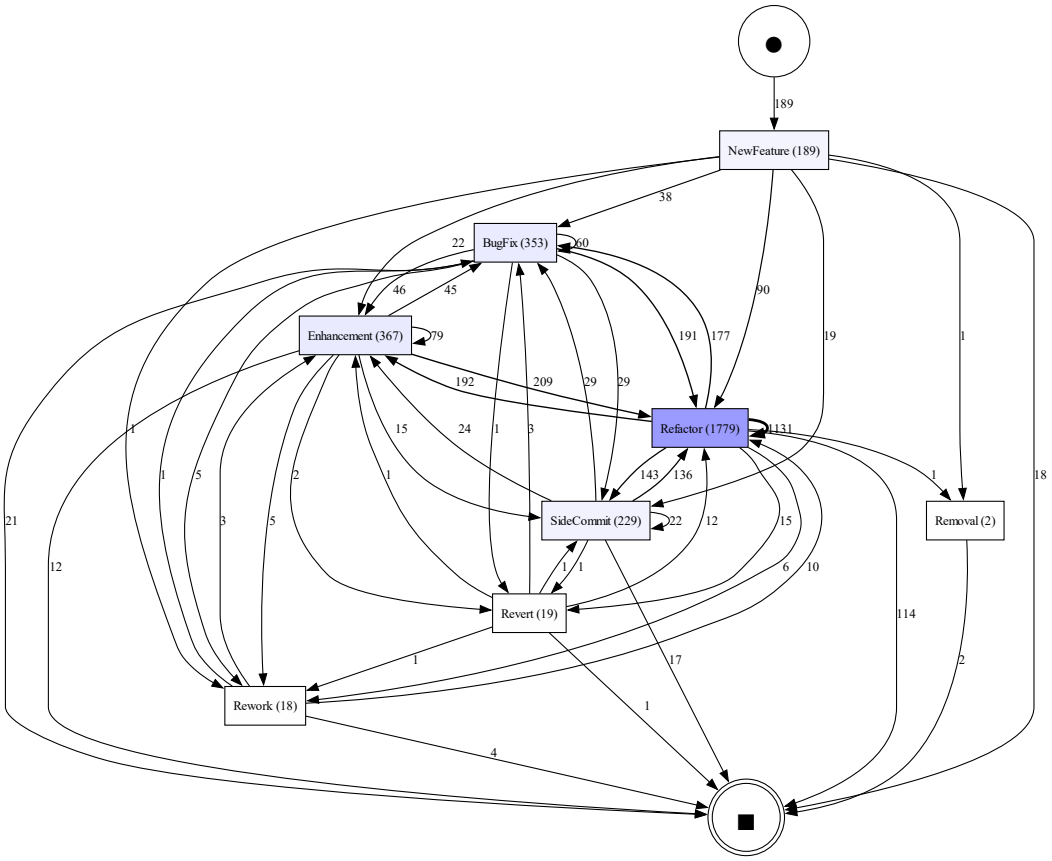

Figure 4.12: DFG of all pull request from the sample set with all the side commit tags grouped together.

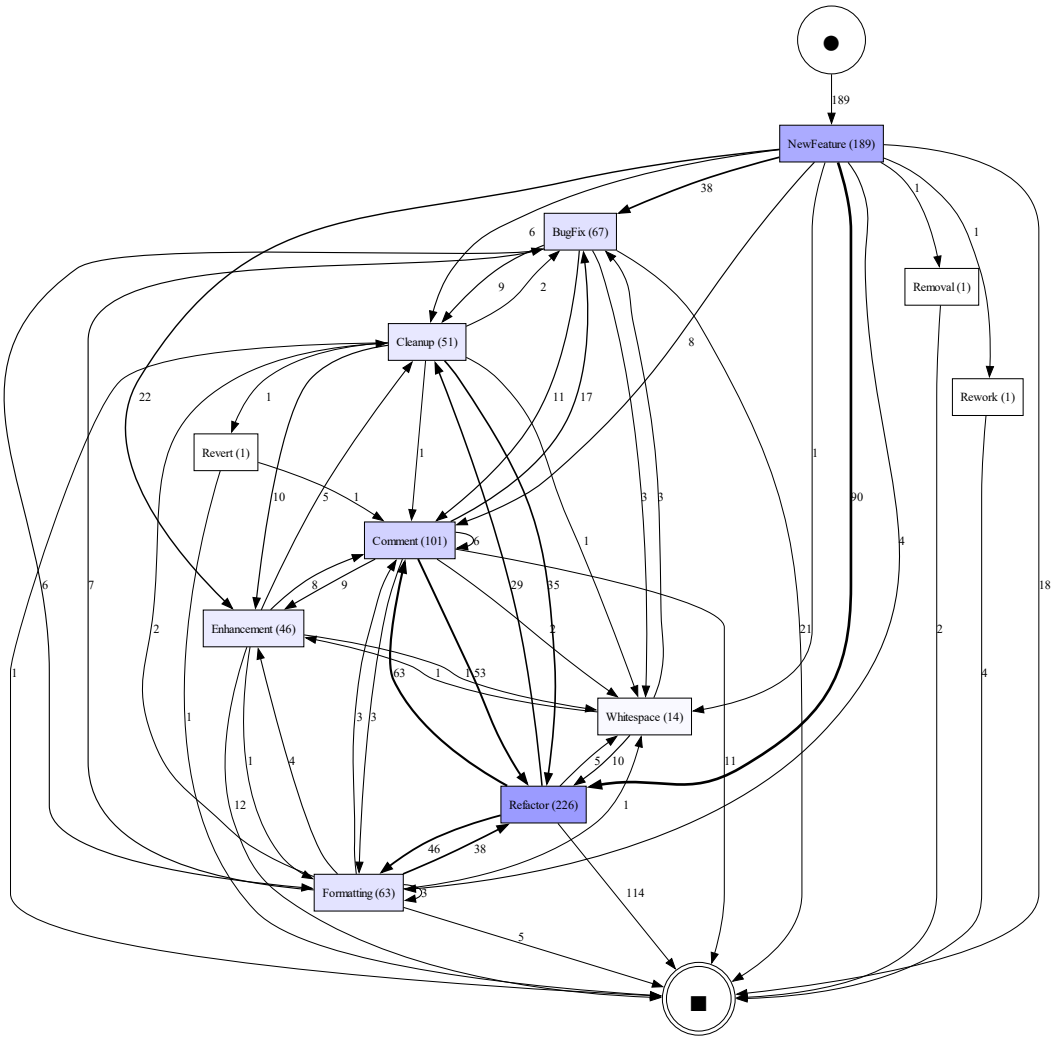

Figure 4.13: DFG of all pull request from the sample set with the edges between the main tags removed.

	BugFix	Cleanup	Comment	Enhancement	Formatting	NewFeature	Refactor	Removal	Revert	Rework	Whitespace
BugFix	‖	‖	‖	‖	‖	<	‖	#	‖	‖	‖
Cleanup	‖	#	>	‖	<	<	‖	#	>	#	>
Comment	‖	<	‖	‖	‖	<	‖	#	<	#	>
Enhancement	‖	‖	‖	‖	‖	<	‖	#	‖	‖	‖
Formatting	‖	>	‖	‖	‖	<	‖	#	#	#	>
NewFeature	>	>	>	>	>	#	>	>	#	>	>
Refactor	‖	‖	‖	‖	‖	<	‖	>	‖	‖	‖
Removal	#	#	#	#	#	<	<	#	#	#	#
Revert	‖	<	>	‖	#	#	‖	#	#	>	#
Rework	‖	#	#	‖	#	<	‖	#	<	#	#
Whitespace	‖	<	<	‖	<	<	‖	#	#	#	#

Figure 4.14: Footprint of all pull request from the sample set.

Label	Count
BugFix-Cycle-1	243
BugFix-Cycles-2	42
BugFix-Cycles-3	6
BugFix-Cycles-4	2
Enhancement-Cycle-1	231
Enhancement-Cycles-2	38
Enhancement-Cycles-3	16
Enhancement-Cycles-4	3
NewFeature	189
Refactor-Cycle-1	253
Refactor-Cycles-2	150
Refactor-Cycles-3	85
Refactor-Cycles-4	53
Refactor-Cycles-5	38
Refactor-Cycles-6-10	59
Refactor-Cycles-10+	10
Removal-Cycle-1	2
Revert-Cycle-1	19
Rework-Cycle-1	18
SideCommit-Cycle-1	190
SideCommit-Cycles-2	15
SideCommit-Cycles-4	1
SideCommit-Cycles-5	1

Table 4.10: Commit cycle lengths in pull requests count.

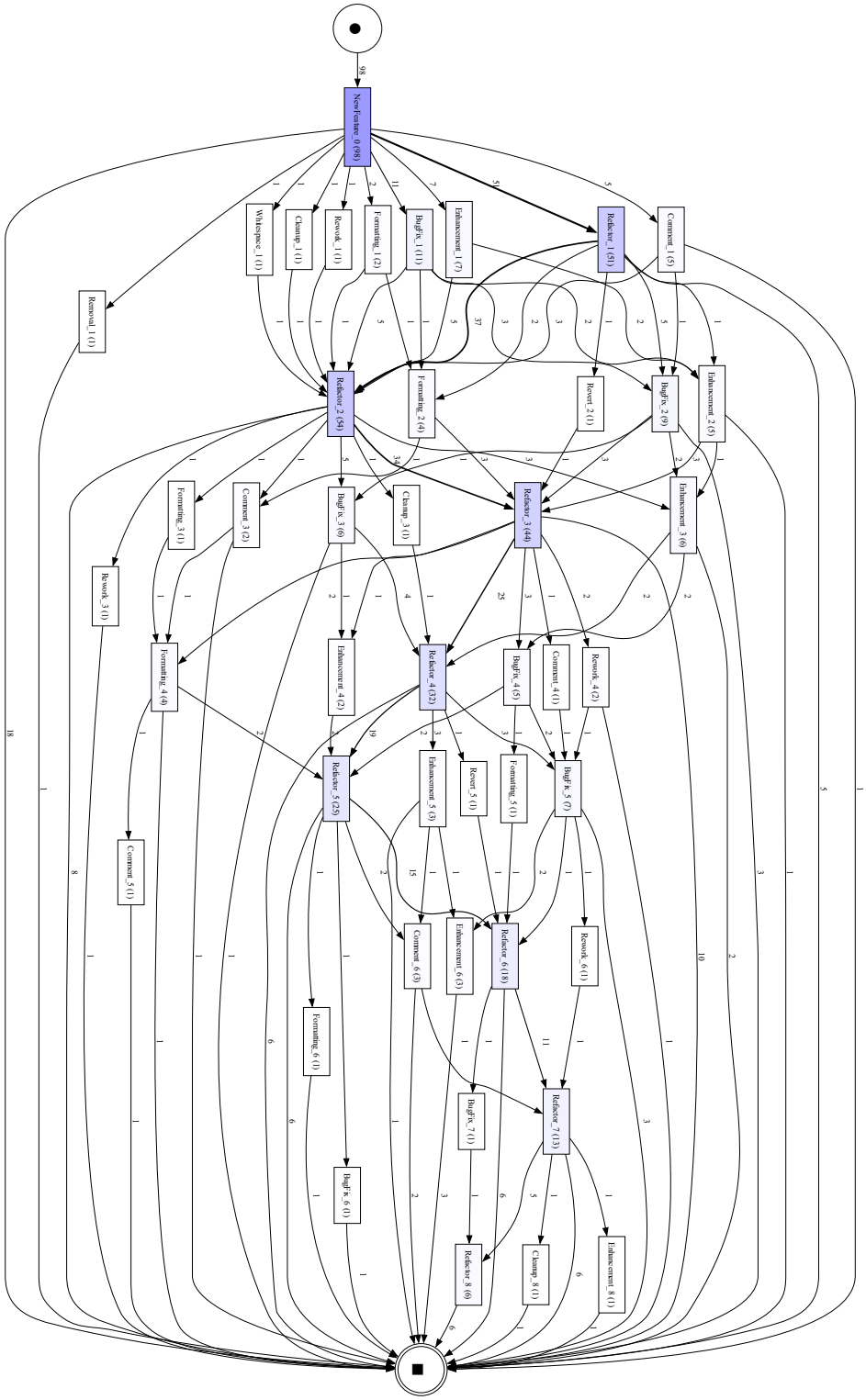

Figure 4.15: DFG of pull requests with at most 9 commits associated with them.

Tag	Count	Percentage
NewFeature	38	10.7%
Removal	0	0%
Rework	1	0.3%
Revert	3	0.9%
BugFix	60	16.9%
Enhancement	45	12.7%
Refactor	177	50.2%
Cleanup	2	0.6%
Formatting	7	2.0%
Comment	17	4.8%
Whitespace	3	0.9%

Table 4.11: Type of commit before `BugFix` tag, count and percentage.

Discussion. While changes do build on top of other changes, it is important to note that changes themselves do not predict future changes. Just because a commit has happened to a feature does not imply that in the future more commits will happen to a feature. Not a single tag is an indicator of what the tag of the possible next commit will be.

The large number of self-edges and cycles of the `Refactor` tag is explained by that it is the most common tag. If we assume that no tag is an indicator of the next tag than with the data that we have found, it is most likely that any tag will be followed by a `Refactor` tag. This is based on the number of commits that are considered refactors. Based on this it is more likely for the `Refactor` tag to achieve higher self-cycles than other tags.

Unlike what was expected beforehand, the life of software features is constantly evolving, being touched by different kinds of changes across its entire life, but there are no visible periods that could be described as static. It only seldom happens that a feature gets reworked or fully removed.

Figure 4.15 shows us that the life cycle of software features for a big part revolves around changes that have no functional impact on the features. It really highlights how many refactors happen in features with the `Refactor` tag being the most traveled edge on every layer of the DFG.

Following the data presented in Table 4.11 it, seems that after the introduction of a `NewFeature` or after the fixing of a bug it is more likely for a `BugFix` to appear. However, it is important to note that not all bug fixes are caused by its preceding commit. Many times, it happens that a problem goes unnoticed for a long time before eventually being fixed. For a precise answer we would have to investigate which commit caused the change that broke the feature and caused the bug fix to happen.

RQ1.1: The life of a labeled software feature consists mainly of refactoring of its code. Only seldom does the feature get an entire rework at once. It is more common for the code to change over time through the endless iterations of refactoring. It is extremely rare for a feature to be removed all together. Explicit small changes are often not done by themselves but done while the feature is being refactored anyway. Bug fixes and enhancements happen sporadically.

4.2.5 Labels and Commit Tags

Results. In Table 4.12 we can see label groups and the different number of tags associated with them. All the labels besides the ones with the `PR-Prefix` are labeled as a group. After the `PR: New Feature` label, the most common group of labels is that for the `C-Prefix` group. The `C-Prefix` group contains all the labels that indicate what segment of the code a pull request impacts. There is one other group with a relatively high number of new features associated to it the `F-Prefix`

Label	NewFeature	BugFix	Enhancement	Refactor	Revert	Rework	Cleanup	Comment	Formatting	Whitespace	Removal	Total
PR: New Feature	189	353	367	1779	19	18	51	101	63	14	2	2956
PR: Bug Fix	1	1	1	8	0	0	0	0	2	0	0	13
PR: Improvement	12	15	27	117	1	0	3	1	3	0	1	180
A-Prefix	14	27	31	193	2	0	6	24	13	3	0	313
C-Prefix	130	262	259	1432	14	7	42	85	61	11	3	2306
F-Prefix	42	92	94	367	6	5	14	19	8	1	0	648
K-Prefix	2	4	0	16	0	0	0	0	1	0	0	23
Needs-Prefix	7	9	7	32	0	1	2	0	0	0	1	59
S-Prefix	10	19	35	107	1	5	7	8	4	2	0	198
T-Prefix	7	22	9	60	3	1	2	7	7	3	0	121
Other	1	0	0	0	0	0	0	0	0	0	1	2

Table 4.12: Labels together with number of different tags for commits.

group. The `F-Prefix` group contains labels that indicate what feature of Marlin a pull requests impacts according to the maintainers. In this way, the group is similar to that of the `C-Prefix` in that it points to specific areas of the code. It can be seen that labels that indicate features of Marlin or code areas are more likely to be used in conjunction to the `PR: New Feature` label than other labels. Both the `C-Prefix` and `F-Prefix` groups have a proportional number of tags associated with them for how many new features they contain. But both the `PR: Improvement` label and `Needs-Prefix` group have a below average number of Bug Fixes associated with them. They also have little to no side commits associated with them. The other groups have only some new features associated with them in our sample set.

Table 4.13 shows a more detailed overview of the labels and the different number of tags associated with them. Only the labels with more than five new features are shown. Because for labels with fewer features than that the data has too few points to give an accurate result for the labels. For completeness, a full list containing all the labels can be found in Table C.1. When we compare this table to Figure 4.1, we can see that `C: LCD & Controllers` is in the top for both. But after that things change, both `C: Motion`, `C: Peripherals` and `A: STM32` are overrepresented compared to how many pull requests they have in total for the Marlin repository. While things such as `C: Build / Toolchain` is underrepresented and `C: Language` is not even present. Here we can see in detail that the `Needs: Work` label has on average a lot less bug fixes, enhancements and refactors. The `Needs: Testing` label also has on average a bit less bug fixes, enhancements and refactors but not by such a big amount as the `Needs: Work` label. The `C: Serial Comms` also has on average a lot less bug fixes, enhancements and refactors. Meanwhile both `C: LCD & Controllers` and `C: Motion` have an above average number of refactors associated with them.

Discussion. Because the `C-Prefix` group and `F-Prefix` group both point to areas of code. It often makes sense that tags of the `F-Prefix` group are found together with tags of the `C-Prefix` group to which that feature belongs. The problem arises in that the maintainers of Marlin often only apply either one of the labels on pull requests.

4.2.6 Properties of Software Features

Commit Count

Results. Table 4.14 displays the relation between commit count and the average lines changed, average comment count, and on average over how many files the original introduced software

Label	NewFeature	BugFix	Enhancement	Refactor	Revert	Rework	Cleanup	Comment	Formatting	Whitespace	Removal	Total
PR: New Feature	189	353	367	1779	19	18	51	101	63	14	2	2956
C: LCD & Controllers	37	95	85	486	2	3	8	25	13	1	0	755
C: Motion	18	51	69	273	4	2	7	5	2	2	1	434
C: Peripherals	16	13	14	96	1	0	2	5	5	0	0	152
A: STM32	13	24	31	182	2	0	6	23	12	2	0	295
F: Calibration	12	39	49	147	4	1	6	6	0	0	0	264
PR: Improvement	12	15	27	117	1	0	3	1	3	0	1	180
C: Temperatures	11	18	9	73	2	0	2	6	3	0	0	124
C: Boards/Pins	9	13	19	125	1	0	5	17	11	2	1	203
C: Hosts & Protocols	9	12	12	53	0	1	2	0	1	0	0	90
C: Build / Toolchain	8	28	13	88	2	0	4	11	13	4	1	172
C: Safety	8	9	12	96	2	1	5	6	8	0	0	147
Needs: Testing	8	14	10	51	0	1	1	5	1	1	0	92
Needs: Work	7	9	7	32	0	1	2	0	0	0	1	59
F: Z-Probes	7	16	22	69	1	2	1	2	0	0	0	120
F: CNC / Laser	7	7	6	80	1	0	6	6	6	1	0	120
F: SD Card / Media	6	15	8	31	0	0	0	2	2	0	0	64
C: Serial Comms	6	7	5	29	0	0	2	6	2	1	0	58

Table 4.13: Labels together with number of different tags for commits.

feature is spread. We can see that a majority (98 out of 189) of the software features contain between 0–9 commits associated with them. Especially with the software features that only have 1–4 commits associated with them, it is visible that on average they impact only some lines, have only some comments and often only change some files. As the commit counts increases, the associated features have an increase in all three of these categories. These numbers become stable for a long while until we reach software features with a very high number of associated commits. At that point, the number of lines changed by the original software feature increases 3-6-fold over the previous features and both the comments and files stay high.

Figure 4.16 shows the number of commits associated with software features over time by the merge date of the software feature. It can be seen that there are outliers with a very high number of commits associated with them but a trend can be seen that the closer we reach to the present the less commits are associated with software features. There are 20 commits that impact more than 10 software features. 15 of those 20 commits have been made before `2023-06-31` which means that they cannot impact the newest software features.

Discussion. The reason that more recent software features have less commits associated with them than the older software features can most likely be found in the fact how many commits have happened after the introduction of the feature. For more recent features a smaller number of commits have happened since their introduction so there are less commits that can possibly impact those features. Marlin has also started receiving less pull requests since 2021, with in 2024 the number of pull requests dropping to half that of 2021 standards. This shows that purely using the commit count is not a good indicator of finding a relation between software features and their properties. We need an indicator that takes the difference in time out of the factor.

Commit Count	Feature Count	Lines Changed	Comment Count	Files Changed
1-4	52	60	5	4
5-9	46	283	7	9
10-14	23	156	13	10
15-19	24	531	8	11
20-24	7	488	10	10
25-29	10	930	12	12
30-34	5	583	12	17
35-39	8	562	39	11
40-44	2	380	124	8
45-49	1	187	11	7
50-59	4	2382	24	21
60+	7	3592	29	46

Table 4.14: Commits Count with Feature Count, Average Number of Lines Changed, Average Comment Count and Average Number of Files Changed.

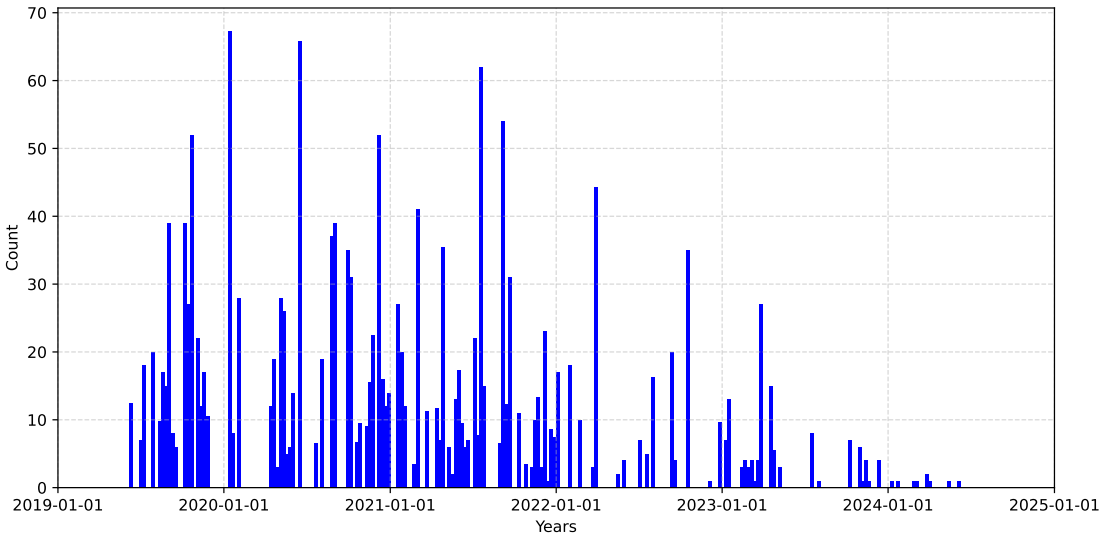

Figure 4.16: Software Features by introduction date by number of commits associated with it averaged per week.

Commits per Day	Feature Count	Lines Changed	Comment Count	Files Changed
0.0000-0.00249	15	27	1	2
0.0025-0.00499	38	142	5	4
0.0050-0.00749	30	70	5	6
0.0075-0.09999	21	252	12	7
0.0100-0.01249	19	442	9	11
0.0125-0.01499	8	283	9	12
0.0150-0.01749	10	413	14	14
0.0175-0.01999	9	428	12	13
0.0200-0.02249	7	1369	19	26
0.0225-0.02499	8	244	14	12
0.0250-0.02749	3	371	4	9
0.0275-0.03000	4	643	15	11
0.0300+	17	2421	44	28

Table 4.15: Commits per Day with Feature Count, Average Number of Lines Changed, Average Comment Count and Average Number of Files Changed.

Commits per Day

An alternative to purely using the commits is to take the date of merger into account. There are two possible ways to do this. One is the active time of a feature, which is the time between merger and its last commit and the other is the time between merger and when the data was collected. The problem with taking the active time of a feature is that it skews the data a lot. A feature that is introduced on day 1 and receives a bug fix on day 20 has an active time of 20 days. A feature that is introduced on day 1 and gets a bug fix on day 20 and a refactor on day 600 has an active time of 600 days. While both features were introduced at the same time and have been in the system for the same amount of time, one has a value of 0.10 commits per day and the other has a value of 0.005 commits per day. For this reason, taking the total time in the system gives a more accurate view, with which a feature that has been in the system for 600 days with 3 commits can be better compared with a feature that has been in the system for only 200 days with 1 commit.

Results. Table 4.15 displays the relation between commits per day and the average code size, average comment count and on average over how many files the original introduced software feature is spread. The difference with Table 4.14 is that the commit count has been replaced by a measurement that takes into account the time a feature has been in the system. The table shows that most of the features hover around the bottom of the commits per day values. However, there are certain features that have a significantly higher number of commits associated with them. There are just as many features that on average get more than 0.03 commits per day as there are features that receive less than 0.0025 commits per day. We can see that especially for the lower end of the table that the lines changed, comment count and files changed are significantly lower than for the middle half of the table often by a factor of 2 to 3 with the lowest commits per day even being a factor of 5 to 10 smaller. At the extreme high end, the number of lines changed is more than twice as high compared to the other parts of the table and the same applies for the comment count. The number of files changed is higher than all others but only slightly.

Discussion. It can be seen that the number of commits per day associated with a software feature over its life has a relation to how big the original feature is. The amount of discussion surrounding a feature also seems to have an impact. This can be explained by the fact that the software features with a lot of discussion are often in more impactful areas of the project, so more commits will touch them over time while changes occur.

We can see that there is a connection between lines changed, comment count and the number of files changed. It seems that when the size of the feature increases, it will on average impact more files and more discussion will be had about it. It is also possible that when more discussion is held about a feature that its size increases and because of that the number of files impacted

increases.

Merger Time

Results. Table 4.16 shows the relation between the time it took for the pull request of the software feature to be merged and certain aspects of the features. More than half of the pull requests are closed within 10 days (102/189), and of those closed within this period, are third are even closed on the same day or the next day.

For these pull requests that are closed within zero to one days we see that often these are just small changes in regard to the number of lines used. Such as pull request #16050[39] which allows users to set the time for PSU power-on delay or pull request #22165[40] which increased the digit for the manual move coordinates if the user has a large bed to print on. We can see that sometimes these changes are bigger than average and still merge fast. These features have one of the following characteristics. Most of the refactoring for the feature is done by original contributor of the feature instead of by the maintainers of Marlin as usually happens. This is the case in pull request #21255[41] where the original contributor provides most of the refactoring in the pull request, or the pull request is almost fully ready on the branch of the contributor and only requires approval to be merged. This is the case for pull request #22418[42] in which only small adjustments have to be made for it to be merged.

There are also examples of features not being merged immediately because of the limited resources the maintainers have. The maintainers of Marlin can only work on so many pull requests at once, and because of this, sometimes pull requests are not attended to for a while. This is mentioned in a comment[43] left on a pull request of a contributor by the maintainers of Marlin. Some examples of this are pull request #23184[44] and pull requests #26825[45]. The first pull request was already mostly done and just required the checking of the maintainer. This happened 11 days after the original submission and was then merged. The second pull request was submitted and already fully done, no files have been edited after the day it was submitted. And it only merged after another 40 days.

At the bottom of Table 4.16 there are two rows. One row contains the information about pull requests with a time difference between creation and merge of 0–9 days and the second row the information about pull requests with a time difference greater than 9 days. These categories contain roughly the same number of features (54% versus 46%). If it takes longer for a feature to be merged that the average number of lines changed, comment count and files changed all increases by a large amount. The number of comments also goes up for software features with a longer time. Features with less lines on average get merged faster than features with more lines changed.

The two biggest outliers with comments are pull request #24797[46] with 232 comments and #23751[47] with 197 comments. Pull request #24797 is about a new feature called `Input Shaping`, which is an advanced technique using a different print algorithm to create better print quality. The high number of comments are caused by many different people trying out the software feature to increase their print quality and sharing their feedback. A substantial portion of the conversation also took place after the software feature was merged into Marlin. Only 71 of the 232 comments associated with it happened before the merger of the pull request. There were 40 participants in this conversation. Then pull request #24797 is about a new way of controlling the temperature of the 3D printer nozzle. Instead of the old PID system that must be manually tweaked by the user, this new system can automatically adapt to changes. For this software feature, most of the

[39] https://github.com/MarlinFirmware/Marlin/pull/16050
[40] https://github.com/MarlinFirmware/Marlin/pull/22165
[41] https://github.com/MarlinFirmware/Marlin/pull/21255
[42] https://github.com/MarlinFirmware/Marlin/pull/22418
[43] https://github.com/MarlinFirmware/Marlin/pull/27324#issuecomment-2282956815
[44] https://github.com/MarlinFirmware/Marlin/pull/23184
[45] https://github.com/MarlinFirmware/Marlin/pull/26825
[46] https://github.com/MarlinFirmware/Marlin/pull/24797
[47] https://github.com/MarlinFirmware/Marlin/pull/23751

Time Difference	Feature Count	Lines Changed	Comment count	Files Changed
0	15	240	1	7
1	20	160	3	6
2	22	190	6	7
3	16	345	13	9
4	7	136	9	6
5	4	446	8	11
6	4	20	5	6
7	7	281	7	9
8	3	1051	2	7
9	4	266	9	11
10-14	11	1267	21	23
15-19	7	451	10	11
20-24	8	180	11	13
25-29	10	1882	36	14
30-34	14	592	10	18
35-39	3	583	13	10
40-44	6	1285	38	7
45-49	1	563	18	15
50-54	2	795	23	10
55-59	1	41	31	17
60-79	7	265	13	11
80-99	2	143	13	10
100-119	2	2554	16	19
120-139	1	227	51	4
140-159	2	1947	29	55
180-199	1	21	1	3
200-219	1	205	52	16
220-239	1	79	22	11
240-260	2	218	29	4
260+	5	57	5	10
0-9	102	250	6	7
9+	87	1442	36	27

Table 4.16: Time Between Creation and Merge Features. with Feature Count, Average Number of Lines Changed, Average Comment Count, Average Number of Files Changed and Average Number of Commits.

comments are placed before the feature is merged and most messages are written by two people. The author of the software feature and a different person helping to test the feature. Even though this pull request has almost just as many comments as #24797 there are only 13 participants in the conversation.

The average time for a pull request of our sample to be merged after being created is 34 days and the median is seven days. Our sample contains four pull requests that took roughly 10 times as long and one pull request that took 20 times as long. Pull requests #17462[48], #17239[49], #24215[50] and #18177[51] all took 10 times as long and pull request #21427[52] took 20 times as long. We can see that the number of lines changed and the comment count for these pulls is quite low. It seems that these are smaller features that had low priority for the maintainers and so they were only touched sometimes.

Table 4.17 shows us the average time between creation of the pull request and merging. This is based on all merged pull requests in Marlin. In it we can see that the `PR: New Feature` label is four times as high as that for all merged pull requests the only labels with a higher time is that of `Needs: Labels` and `Other`. The Average number of comments is also high for the `PR: New Feature` label. Being roughly 3 times as high as the average but again we can see that the `Needs: Labels` are even higher. We can see in this table that the `PR: New Feature` label is one of the more demanding labels in the system with both a high average time and comment count. The `PR: New Feature` also has pull request with the second-highest time between creation and merger. The only pull request with a longer time is pull request #20147[53]. This pull request was eventually merged into Marlin after a long time of inactivity for wider testing by the community. It contains the labels of `PR: Improvement`, `Needs: Work`, `Needs: Testing` and `F: Filament Sensor`. The shortest average time is attributed to the `PR: Workaround` label with an average time of 0.33 days. This label group only contains three merged pull requests. These workaround were introduced four, three and two year ago. The shortest average time after this is the `Bug-Prefix` label group. Containing 348 merged features with an average time of only 1.36 days. This group also contains the lowest max time of all the bigger groups. With a pull request that was only open for 60 days.

Discussion. That bigger pull requests take more time to be merged can be explained by that features with more code need more time to properly be tested and reviewed. All the code must be reviewed against the coding standards and possible bugs take longer to find with more code.

The reason for conversations taking place in a pull request after merging can be found in the context of the software feature. For the case of pull request #24797 is that the feature that was introduced required a large amount of manual fine tuning to be successful. So, a lot of users were discussing optimal settings for their printers and sharing their experiences.

The big difference between the average and the median can be explained by having a lower bound of 0 days before merging but no upper bound on the maximum time before merging. This means that the pull requests with a very high time between merging have a bigger impact on the average than the pull requests with 0 days.

For the pull requests that took a lot longer than average to merge it appears that the maintainers only have a limited amount time to spent on working on pull requests. So if the priority of the pull request is low enough this can cause it be endless stuck until eventually the maintainer takes time to finish it.

The higher amount of time for the `Needs-Prefix` labels can be explained by that the `Needs:` labels only get applied when more effort is required. This means that more discussion must take place on how to solve the issue and this takes time. This also explains the higher number of comments associated with the label group.

[48] https://github.com/MarlinFirmware/Marlin/pull/17462
[49] https://github.com/MarlinFirmware/Marlin/pull/17239
[50] https://github.com/MarlinFirmware/Marlin/pull/24215
[51] https://github.com/MarlinFirmware/Marlin/pull/18177
[52] https://github.com/MarlinFirmware/Marlin/pull/21427
[53] https://github.com/MarlinFirmware/Marlin/pull/20147

Label	Merged Pull Request Count	Average Time Between Creation and Merge	Max Time Between Creation and Merge	Average Comment Count
All Merged	9114	5.85	900	4.09
PR: New Feature	472	23.82	715	11.4
PR: Bug Fix	2852	3.27	422	3.13
PR: Coding Standards	243	4.63	369	1.9
PR: Configurations	420	3.03	267	1.77
PR: General Cleanup	784	1.45	123	1.92
PR: Improvement	2067	6.58	900	5.7
PR: Workaround	3	0.33	1	9.0
The Rest	8642	4.87	900	3.69
A-Prefix	492	6.32	253	7.97
Bug-Prefix	348	1.36	60	2.29
C-Prefix	4591	7.11	595	4.32
F-Prefix	1175	10.48	900	5.42
K-Prefix	69	5.22	60	4.45
Needs-Prefix	239	33.16	900	19.83
S-Prefix	160	15.67	337	15.98
T-Prefix	627	10.15	595	7.58
Other	5	26.4	75	8.8
No Label	1019	3.86	288	2.59

Table 4.17: Labels and the associated Merged Pull Request Count, the average Time Between Creation and Merge and the Average Comment Count with Time Between Creation and Merge Features.

RQ1.3: The size of a feature, its centralization and interaction all serve as a rough indicator of how its life will continue in the future. If the feature has for all these metrics a very low value, the feature receives only very few commits that interact with it over its life cycle. While if all these metrics have an extremely high value the feature receives a lot of commits that interact with it over its life cycle. The properties do not show any impact on the middle group of features that are in neither extreme side. For the merge time of a feature it is more clear that smaller features are merged faster than larger features.

4.3 RQ2: Correctness of Pull Requests Labeling by Maintainers

In this section, we present and discuss our findings for **RQ2.1** and **RQ2.2**. We compare pull requests against our definition of what a feature request is as defined in section 2.1.

4.3.1 RQ2.1: Accuracy of labeled features

Results: In section 2.1 we have defined what we consider a software feature for this work. To see if Marlin maintainers are consistent with their labeling of the features all the 189 software features of the sample set have been compared against our definition of a software feature. In Table 4.18 we can see all the pull requests of the sample set together with a comment about what functionality was added to Marlin. All the 189 pull requests labeled with the `PR: New Feature` label are considered features according to our definition.

In the table we can see that a variety of different types of features have been added. Some are about the additions of configurable settings to control the behavior of Marlin others are about the addition of commands to execute specific behavior. Some are large and complicated, and others are small and simple. All of them are either a characteristic or an end-user-visible behavior of the software system.

Examples: Pull request #14251[54] is the first pull request we have analyzed. It is a simple feature that has end-user-visible behavior of the software system. The pull request introduces the feature to configure the firmware to leave the heaters on after the print has been aborted. A bigger pull request that added a feature was #18071[55]. This pull requests added the `LVLGL GUI` library for the `MKS Robin Nano` to Marlin. Which means that it added end-user-visible behavior to the system. Another UI related pull requests that added a feature was #20940[56]. This pull request added the `more` menu on the user interface in which the user can store up to 7 custom commands. So, we can see that there are features of different sizes and functionality.

Discussion: Looking at all the labeled new features a limitation of the investigation system can be found, very large software pull requests run into the problem that not a single but multiple software features are introduced. Such as in pull request #18071 where it could be argued that the individual menus and actions of the UI are all separate features. Because for example a menu that allows the user to enable a power monitoring system in the UI is end-user-visible behavior and as such could be seen as a feature according to our definition. A good example of this pull request #20940 which also touches the code for the UI system of Marlin but has one dedicated function that is being added. Both pull requests are labeled as a feature but one of them adds a single dedicated feature and the other adds an entire system of multiple features.

> **RQ2.1:** All the pull requests of our sample set hold to our definition of a software feature. This means that all the labeled features of Marlin that we have investigated are indeed features.

Pull	Comment
14251	Added configurable setting to leave heaters on
14265	Added ability to turn of backlights after timeout
14309	Added ability for pin debugging on STM32
14595	Added SPI touchscreen driver for printers
14619	Added ability to clean nozzle with limited axis
14648	Added support for analog joystick

[54] https://github.com/MarlinFirmware/Marlin/pull/14251
[55] https://github.com/MarlinFirmware/Marlin/pull/18071
[56] https://github.com/MarlinFirmware/Marlin/pull/20940

Pull	Comment
14757	Added M575 command to change baud rate
14924	Added M16 command to check for expected printer
14953	Added ability to run scrip at startup
14956	Added configurable setting to control specific fans
14961	Added Animated Marlin boot screen
14991	Added configurable setting for turbo back button
15081	Added support for SPI daisy chain support
15195	Added ability to calculate bed alignment for triple-Z printers
15209	Added M997 command support to STM32F1
15245	Added configurable settings for touch buttons
15497	Added ability to show estimated remaining time
15549	Added M73 command to support remaining time output of slicer
15585	Added feedforward to fan scaling options for better control
15590	Added M486 command to cancel individual objects
15690	Added stepper current to sensorless homing for better support
15739	Added input variables to M220 command
15930	Added support for temperature sensor
16026	Added E input variable to M114 command to get stepper position
16050	Added configurable setting for PSU power-on delay
16068	Added ability to show power percentage of Laser and Spindle
16277	Added support for quad Z stepper
16293	Added support for z-probe with thermistor
16362	Added hotend protection after timeout
16452	Added support for L64XX stepper driver
16554	Added configurable G12 defaults per tool
16641	Added M672 command to reset sensitivity of smart effector
16741	Added M360 command to report capabilities of printer
16756	Added ability to use onboard SD card for STM32duino
16897	Added bed tramming assistant
17017	Added ability to limit feedrate based on volumetric extrusion
17222	Added support for SD cards on STM32
17239	Added support for tool sensor to check if tool is still present
17248	Added ability to change tool automatically during print
17437	Added ability to monitor power and display current reading
17462	Added ability to instantly stop movement
17523	Added support for MMU2S features
17531	Added support for WIFI on SKR PRo 1.1
17536	Added support to use raw values of digipot
17560	Added configurable setting to control AUTOTEMP
17788	Added ability for G12 command to check endstop parameters
17853	Added ability for direct stepping through the G6 command
17884	Added configurable setting for SD card read only mode
18039	Added configurable setting for thermocouple sensor errors
18071	Added UI library for MKS Robin Nano
18177	Added ability to add custom user menu items
18316	Added configurable setting handling small segments
18389	Added configurable pin macro for commands
18399	Added ability to password protect printer
18735	Added ability to ignore controller fan for z axis
18737	Added support for pre and post scrip to command G425
18866	Added z probe offset calibration menu
18906	Added configurable Z stepper position after deactivate

Pull	Comment
18972	Added G34 command to align Z steppers automatically
19139	Added support for TFT controller to LPC1768
19225	Added support for debug codes
19235	Added support for MARKFORGED kinematics
19330	Added support of realtime reporting to commands
19674	Added D576 command to report buffer status
19801	Added ethernet support to Teensy 4.1
19828	Added configurable TMC interpolation per driver
19837	Added support to run commands during PSU POWER action
19901	Added configurable setting for sound on menu items
19912	Added support for SMUFF feeder
19971	Added support for hobby servos
20084	Added M808 command to repeat last command
20087	Added support for UTF in long filenames
20099	Added configurable setting to mount SD on boot
20151	Added ability to go to file browser on media insert
20241	Added ability to use probe to level corners
20383	Added configurable minimum temperatures for nozzle and bed
20384	Added support for TFT controller to STM32
20455	Added support for white LED channel on PCA9632
20549	Added support for progress estimation to FTDI EVE UI
20571	Added support for USB drive to specific boards
20711	Added support for control board
20755	Added ability to run commands from LVGL UI
20802	Added compression for serial data
20835	Added configurable setting to wait for hotend to heat up between probes
20940	Added more menu item in LVGL interface for custom items
20956	Added support for shared media on STM32
21005	Added FWK kinematics for TPARA
21255	Added M143 and M193 command to cool lasers
21344	Added support for multi volume in LVGL UI
21387	Added Hilber space probing sequence
21403	Added support for extruder with dual stepper drivers
21427	Added ability to probe by use of a servo
21431	Added support for coolant flowmeter monitoring
21503	Added support for control board
21534	Added menu for non standard settings in DWIN
21668	Added M10 and M11 command for air evacuation of laser
21753	Added air assist menu for laser cutter
21835	Added laser power consumption monitoring
21888	Added configurable setting to sync redundant fan to FAN0
21931	Added DGUS LCD UI Library
21949	Added configurable setting for independent baud rates
21962	Added support for multiple neopixels to be background LEDs
21999	Added STM32 support for printer
22040	Added STM32 support for printer
22052	Added STM32 support for printer
22075	Added configurable settings for I J and K axis
22085	Added support for more generic redundant temperature sensor
22109	Added WIFI control support to MKS Robin Nano v3
22165	Added support for larger bed sizes to manual control
22191	Added confirmation for power off command

Pull	Comment
22279	Added support for TEMP_SENSOR_BOARD on control board
22354	Added support for Mass Storage Class on STM32
22391	Added laser support to TFT screen
22418	Added HAL platform for simulator
22478	Added M256 command to set LCD brightness
22617	Added support for screen sleep after timeout
22669	Added support for beltprinter
22715	Added support for control board
22760	Added M282 command to detach servo
22790	Added support for polargraph machines
22844	Added mesh viewer to DWIN UI
22880	Added support for control board
22897	Added support for control board
22908	Added support for external shutdown action
22916	Added configurable setting for BL Touch operating mode
22941	Added support of long filenames to M115 command
23080	Added support of printer and control board
23086	Added support for fans that need tacho
23101	Added support for probe
23112	Added support for up to 9 axes
23130	Added support for screen controller
23163	Added support for MARKFORGED YX kinematics
23172	Added ability to halt nozzle before halting for a temperature error
23184	Added M3426 command to communicate with MCP3426 over i2c
23192	Added support for electromagnetic probe
23238	Added support for X-Axis twist compensation
23345	added configurable hold multiplier per stepper
23396	Added support of conditional delays to M81
23400	Added M919 command to set TMC stepper chopper times
23462	Added ability to firmware update over USB cable
23658	Added support for control boards
23751	Added command M306 for predictive temperature control
23768	Added backlight timeout to screen
23944	Added configurable setting for freeze pin state
23992	Added M255 command to allow display sleep on DOGM screen
24074	Added prompt for firmware upload destination
24189	Added watchdog timer for laser safety
24215	Added configurable setting to only deactivate servo after stow
24420	Added support for klicky probe
24461	Added support to run command after endstop trigger
24521	Added support for screen adapter to board
24528	Added support to parse config.ini to apply settings to firmware
24553	Added ability to switch extruder without use of servo
24554	Added support to bed level with BDsensor
24624	Added support of screen for printer
24722	Added support of control board
24797	Added M593 command for input shaping
24995	Added support to run multiple controller fans at once
25093	Added ability to store autotemp to eeprom settings
25143	Added DGUS UI for older screens
25150	Added safety check to see if runout sensor is correctly defined
25214	Added support for Polar kinematics

Pull	Comment
25232	Added ability to setup MPC Autotune in ProUI
25256	Added ability to avoid bed clips while probing
25268	Added support to display heater power on screen
25394	Added M493 command to apply fixed-time motion
25438	Added support for LCD timeout for neopixel
25548	Added ability for one extruder to enable multiple runout sensors
25642	Added support for SOC temperature sensor
25667	Added ability to live monitor endstop and z probe on ui
25738	Added ability to report live position of tool
25781	Added ability to one click print a model
26011	Added support for lower resolution marlin logos
26127	Added M592 command to enable nonlinear extrusion
26142	Added ability to M300 command to enable or disable sounds
26267	Added configurable settings for probe offset edit limits
26328	Added support for serial reading to circular buffer
26341	Added ability to M190 command to slowly cool down over time
26344	Added configurable settings for move distances
26401	Added ability to G27 command to raise the Z axis and park the XY axis
26441	Added support for heated bed
26652	Added ability for controller fan to function during PID autotune
26696	Added support to recover heated chamber after power loss
26751	Added support for redundant psu control with external device monitoring
26793	Added support of custom bootscreen to old screens
26806	Added reporting of machine kinematics to M115 command
26825	Added support of UART 5 to control board
26892	Added support for 4 axis cnc configuration to board
27072	Added support for linear fan kickstart

Table 4.18: All pull requests investigated and a comment about them.

4.3.2 RQ2.2: Pull Requests not Labeled as Features

To see if all pull requests that could be considered a feature according to our definition were labeled as such, we have taken the six most common labels associated with the `PR: New Features` labeled pull requests that were present in our sample set and inspected 30 pull request associated with them. The six most common labels are `C: LCD & Controllers`, `C: Motion`, `C: Peripherals`, `A: STM32`, `C: Calibration` and `PR: Improvement`. This is based on the ranking as seen in Table 4.13. We follow the method as described in section 3.4. The labels of `PR: New Feature`, `PR: Bug Fix` and `PR: General Cleanup` have been filtered out. We did this for the following reasons: The `PR: New Feature` label has already been covered and known to be a feature, so we only want to investigate pull request that are not labeled as a feature. The `PR: Bug Fix` label is specifically used for pull requests that solve confirmed bug reports so they are not relevant to new features. Lastly the `PR: General Cleanup` has been filtered out because the label indicates that the pull request is only about cleaning up the code. Subsequently, the dataset is filtered by date. Such that no pull request that were created before our sample set starts are included and no pull results are included that start after our sample set ends. This is to make sure the data is comparable to what we have obtained in our sample set. From this dataset we then filter on pull requests that contain the desired label and select 30 at random for an inspection.

Classified as Feature	Comment
15060	Added Back button for touchscreen
15498	Added support for screen and control board
16317	Added extra functionality to DGUS Screen
16792	Added extra controls to ExtUI
18326	Added support for M73 command to mks_ui
18655	Added support for specific board ui
19384	Added ability to change Z offset while printing
19465	Added extra controls Touch UI
19878	Added support for control board
20280	Added support for screen and control board
24610	Added information to MKS UI
25073	Added support for additional fonts
26596	Added support for screen
Not Classified as Feature: 14546, 14883, 15462, 15593, 15714, 17281, 17298, 18830, 20427, 21148, 22288, 23124, 23502, 24278, 26539, 27154, 27275	

Table 4.19: Pull requests with the `C: LCD & Controllers` label that could be additionally labeled as a feature according to our definition of a feature.

`C: LCD & Controllers`

Results. In Table 4.19 we see the pull requests that we would classify as features for the `C: LCD & Controllers` label and the list of the other investigated labels. The 30 pull requests that we have investigated have been sampled from a set of 472 pull requests with the `C: LCD & Controllers` label. We would consider 13 of the 30 pull requests features according to our definition. 6 of the 13 pull requests are labeled with the `PR: Improvement` label.

Examples. Pull request #18326[57] introduces the feature for the `MKS UI` of using the `M73` command to display the remaining time on the screen. This can be categorized as end-user-visible behavior as now the Marlin system shows the user the remaining time. Pull request #26596[58] introduces the `I3DBEE TECH Beez Mini 12864` screen to Marlin. It is now a characteristic of the system that it supports this specific piece of screen. These are two examples of pull requests that we would also consider a feature that are not labeled as such. Pull request #15498[59] is about cleaning up the `LCDPRINT` function and removing nonfunctional code. This we do not consider a feature because it neither changes a characteristic nor an end-user-visible behavior of the system.

`C: Motion`

Results. In Table 4.20 we see the pull requests that we would classify as features for the `C: Motion` label and the list of the other investigated labels. The 30 pull requests that we have investigated have been sampled from a set of 89 pull requests with the `C: Motion` label. We would consider 10 of the 30 pull requests features according to our definition. 6 of the 10 pull requests are labeled with the `PR: Improvement` label.

Examples. Pull request #18736[60] introduces the ability for the user to define an explicit sequence for the nozzle wipe. This is end-user-visible behavior as now Marlin will perform a specific sequence defined by the user. Pull request #24684[61] adds support of the linear advance movement system to ESP32 boards. This is also an end-user-visible behavior because without this pull request these boards would not be able to make use of the linear advance feature. Pull

[57] https://github.com/MarlinFirmware/Marlin/pull/18326
[58] https://github.com/MarlinFirmware/Marlin/pull/26596
[59] https://github.com/MarlinFirmware/Marlin/pull/15498
[60] https://github.com/MarlinFirmware/Marlin/pull/18736
[61] https://github.com/MarlinFirmware/Marlin/pull/24684

Classified as Feature	Comment
15374	Added support for more mixing steppers
16551	Added ability for arc segment radius to be scaled
17697	Added ability to park nozzle with only one axis
18736	Added ability to define nozzle wipe sequence
18870	Added configurable setting for x axis for feature
20113	Added configurable Home_Z_First setting
20218	Added configurable software stepper enable setting
20678	Added ability for each Z axis stepper to move to a different direction
23158	Added ability to configure park move options
24684	Added linear advance to ESP32 boards
Not Classified as Feature: 15475, 15481, 16533, 17817, 17945, 18112, 18342, 18738, 19135, 20444, 21612, 22504, 22824, 24366, 25791, 26027, 26720, 26770, 27031, 27292	

Table 4.20: Pull requests with the `C: Motion` label that could be additionally labeled as a feature according to our definition of a feature.

request #18342[62] is about fixing the `BABYSTEP_XY` on the `CoreXY` platform. The `BABYSTEP_XY` was broken because of duplicate code being present in two files. We do not consider this pull request a feature, as it only fixes the program to introduce the behavior that should already be present in the system. This pull request was not labeled with the `PR: Bug Fix` label.

`C: Peripherals`

Results. In Table 4.21 we see the pull requests that we would classify as features for the `C: Peripherals` label and the list of the other investigated labels. The 30 pull requests that we have investigated have been sampled from a set of 81 pull requests with the `C: Peripherals` label. We would consider 16 of the 30 pull requests features according to our definition. 6 of the 16 pull requests are labeled with the `PR: Improvement` label.

Examples. Pull request #14667[63] introduces the ability to control two separate strips of neopixel LEDs at the same time. This is both a characteristic of the system and an end-user-visible behavior of the system. Pull request #26163[64] adds the ability to re-initialize the on power supply on. This fixes the problem that the screen ends up corrupt if not re-initialized. This is an end-user-visible behavior of the system. Pull request #21811[65] fixes that Marlin is unable to be built with `CASE_LIGHT_USE_RGB_LED` enabled. This pull request fixes a build error of Marlin and is not considered a feature according to our definition. This pull request was not labeled with the `PR: Bug Fix` label.

`A: STM32`

Results. In Table 4.22 we see the pull requests that we would classify as features for the `A: STM32` label and the list of the other investigated labels. The 30 pull requests that we have investigated have been sampled from a set of 174 pull requests with the `A: STM32` label. We would consider 14 of the 30 pull requests features according to our definition. 4 of the 14 pull requests are labeled with the `PR: Improvement` label.

Examples. Pull request #14539[66] allows users to configure pins for stepper drives in their own configuration. Before these pins where hard-coded in the Marlin source files. It is now a

[62] https://github.com/MarlinFirmware/Marlin/pull/18342
[63] https://github.com/MarlinFirmware/Marlin/pull/14667
[64] https://github.com/MarlinFirmware/Marlin/pull/26163
[65] https://github.com/MarlinFirmware/Marlin/pull/21811
[66] https://github.com/MarlinFirmware/Marlin/pull/14539

Classified as Feature	Comment
14667	Added support for second neopixel light strip
16539	Added support to send pulses for photos
16731	Added support for thermistor
16960	Expands the capabilities of servos
17806	Added support for thermistor
20123	Added support for expansion unit
20262	Added support for cold extrusion
20410	Added support for percentage viewing of statistics
21680	Added support for digipot stepper controller for board
23066	Added ability for M150 command to control individual led strip
23322	Added ability to drive more than 128 leds on a neopixel
23986	Added support for more than 3 axes to command M350 and M114
24130	Added address flag to M3426 command
25163	Added support for PCA9632 for the RGB_STARTUP_TEST
25303	Added menu item to control pen for polargraph kinematics
26163	Added ability to re-intialize AMX7219 on power supply on
Not Classified as Feature: 15463, 17285, 17814, 18504, 18808, 19436, 19856, 21212, 21811, 21855, 22304, 22557, 22682, 22789	

Table 4.21: Pull requests with the `C: Peripherals` label that could be additionally labeled as a feature according to our definition of a feature.

characteristic of the system that these pins are user configurable. Pull request #24760[67] adds support for the `Creality V5.2.1` control board to the firmware. It is now a characteristic of the system that this board is supported. Pull request #22537[68] defines the `#ifdef HAL_STM32` this a refactor of the code by replacing the individual definition of the three boards by one statement. Nothing in the software will react differently to it. Therefore, it does not meet the criteria for it to be considered a feature.

`C: Calibration`

Results. In Table 4.23 we see the pull requests that we would classify as features for the `C: Calibration` label and the list of the other investigated labels. The 30 pull requests that we have investigated have been sampled from a set of 110 pull requests with the `C: Calibration` label. We would consider 12 of the 30 pull requests features according to our definition. 4 of the 12 pull requests are labeled with the `PR: Improvement` label.

Examples. Pull request #15376 introduces the ability for command `M290` to report over serial whats it current situation is without a screen attached to the printer. This is end-user-visible behavior and so we consider it a feature. Pull request #23033[69] expands the probe temperature compensation feature to function with more probes and makes it user configurable. The different probes supported for this feature a now a characteristic of the system. Pull request #22657[70] improves the usability of the tramming wizard. This is done by clearing up the text of the steps and making values less ambiguous. This does not change the behavior of the system nor change any of its characteristics. Consequently, it is not regarded as a feature.

[67] https://github.com/MarlinFirmware/Marlin/pull/24760
[68] https://github.com/MarlinFirmware/Marlin/pull/22537
[69] https://github.com/MarlinFirmware/Marlin/pull/23033
[70] https://github.com/MarlinFirmware/Marlin/pull/22657

Classified as Feature	Comment
14407	Added support for control board
14539	Made TMC_SW_* pins configurable
14639	Added support for screen
15631	Added support for control board
15931	Added support for screen
16207	Added support for printer
18145	Added sanity checks for serial_stats
18347	Made Robin Nano pins configurable
19978	Added ability to get clock rates from framework
23464	Added DMA interrupt method
24760	Added support for control board
25387	Added support for control board
25636	Added support for control board
25796	Added support for control board
Not Classified as Feature: 14388, 14807, 14810, 15645, 16217, 16404, 16538, 16562, 17056, 17865, 17937, 19905, 19963, 20544, 21590, 22537	

Table 4.22: Pull requests with the `A: STM32` label that could be additionally labeled as a feature according to our definition of a feature.

Classified as Feature	Comment
14456	Added menu option to allow small movement on Z axis
14533	Added extra argument to command G34 for extra functionality
15376	Added M290 Added reporting
15640	Added ability to speed up homing after X and Y are homed
19682	Added ability to report z deviation for tramming
20733	Added 3 point bed leveling
21161	Added support of BLTOUHC_HS_MODE to LEVEL_CORNERS_USE_PROBE
21612	Added support for backlash compensation for COREXY kinematics
22707	Added additional feature to G33 Command
23033	Added probe temperature compensation
23489	Added ability to change Z offset in LVGL UI
25371	Added support for dual endstops on X and Y
Not Classified as Feature: 14380, 14391, 14882, 15525, 16872, 17195, 17273, 18695, 20537, 21237, 21899, 22657, 22975, 23050, 23704, 23987, 26255, 26762	

Table 4.23: Pull requests with the `C: Calibration` label that could be additionally labeled as a feature according to our definition of a feature.

Classified as Feature	Comment
14535	Added ability to turn led lights on after print is done
15394	Added support for the M997 Command on new control board
15650	Added support for redundant sensor when single nozzle
16466	Added support to invert analog joystick values
16599	Added support for Serial 2 on new control board
17741	Added support to save more axis homing thresholds to EEPROM
17908	Added support for TMCStepper to specific printer
18576	Added debugging for probe method
18952	Added Marlin Color UI to STM32F1
19349	Added new menus in Touch UI for BLTouch controls
20159	Added support for dummy thermistors without pin definition
22425	Added ability to hide files on SD card
23764	Added probe temperature compensation to more commands
24401	Added ability to change bed dimensions at run time for polargraph
Not Classified as Feature: 15475, 15481, 16533, 17817, 17945, 18112, 18342, 18738, 19135, 20444, 21612, 22504, 22824, 24366, 25791, 26027, 26720, 26770, 27031, 27292	

Table 4.24: Pull requests with the `PR: Improvement` label that could be additionally labeled as a feature according to our definition of a feature.

`PR: Improvement`

Results. In Table 4.24 we see the pull requests that we would classify as features for the `C: Improvement` label and the list of the other investigated labels. The 30 pull requests that we have investigated have been sampled from a set of 765 pull requests with the `C: Improvement` label. We would consider 14 of the 30 pull requests features according to our definition.

Examples. Pull request #15394[71] adds support for the M997 command on the STM32 platform. Marlin is now able to flash firmware to those boards. This is a change in the end-user-visible behavior and so we consider it a feature. Pull request #23764[72] adds support for probe temperature compensation in all commands that it would be beneficial. This changes the end-user-visible behavior when these commands are used. Therefore, we consider this a feature. Pull request #22504[73] fixes the problem that after tool changes an unexpected extruder move would happen at the end of the tool change. This is a bug fix that solves behavior that was not intentional. This pull request was not labeled with the `PR: Bug Fix` label.

Discussion. We have established that, among the 180 pull requests we investigated, 79 can be classified as a feature according to our definition. We have only investigated the most popular labels that are associated with the `PR: New Feature` label. Of the 5 non-`PR` labels that we looked into we consider 65 of the pull requests to be a feature. Of these 65 pull requests, 26 had the `PR: Improvement` label. This means that if a pull request has a `PR: Improvements` label that it is a good indicator that it could also be considered a feature. The reason for this can very come from a difference between the definition of a feature that is used by the maintainers of Marlin and that is used by us. Another reason may be that people often consider a feature to be only something that implements entirely new functionality, rather than when on existing software component has it functionality expanded. We can conclude that for at least the popular categories as much as 40% of the pull requests might not be labeled as a feature even though we would consider it one. This means that the data set of features is quite incomplete if only the labeled pull requests are taken as a basis.

Something we can also notice is that the Marlin maintainers are not consistent with their labeling of pull requests. In the `A: STM32` label category we have pull requests that add control

[71] https://github.com/MarlinFirmware/Marlin/pull/15394
[72] https://github.com/MarlinFirmware/Marlin/pull/23764
[73] https://github.com/MarlinFirmware/Marlin/pull/22504

boards to Marlin that are not labeled with the `PR: New Feature` label. While we have examples of pull request such as pull request #20711[74] being labeled with the `PR: New Feature` label. This is not consistent. Another thing we have noticed is that of the 180 inspected pull requests we have 31 pull requests that are not labeled with the `PR: Bug Fix` label but we would consider a bug fix. Pull request #21811 is an example of this. Where the pull request fixed a build error in Marlin but was not labeled as a bug fix.

RQ2.2: After examining the pull requests we can say that pull requests not labeled as a feature can still be a feature according to our definition. We have concluded that up to 44% (79/180) of the pull requests in our sample would be considered a feature according to our definition.

[74] https://github.com/MarlinFirmware/Marlin/pull/20711

Chapter 5

Threats to Validity

There are a number of threats that can affect the validity of this study. They are outlined in this section.

Interpretation. A threat to the validity is that the results completely depend on our interpretation of the pull requests. We tag the commits with the utmost care, but if the description of the commit/pull request is not informative or if the code of the change is not clear enough, then there exists the possibility that we would apply the wrong tag and so have lowered the reliability of our dataset. For example, for some commits that change a significant amount of code at once, it is difficult to see if the code only gets moved around and renamed or if it changes the actual functionality of the feature.

Incompleteness. Another threat to the validity is that our results could be incomplete if not all the commits that touch a software feature are found. Especially for the larger software features that contain a lot of code and a lot of changes, it is not easy to keep track of what code is still part of the feature and as such should be considered for interaction by code changes. This would mean that we have missed the information that was present in the files.

Open-Source A possible threat is that the Marlin project is an open-source collaborative work, in which amateurs and professional a like work together. This leads to a different style of development than in the more structured environment of a company, where there is one person in charge that decides the direction of the project. For an open-source project like Marlin, the project only evolves in the direction that the volunteers are interested in. Because of these differences it is possible that our results will not apply for non-open-source projects.

Marlin Hardware Project. A threat to validity is that Marlin might be a relatively unique style of project. As mentioned earlier in thesis, the Marlin project has it as goal to support the same hardware on which it originally ran when it was released in 2011. While building in new features for more modern systems, other software projects try to support as many platforms and hardware as possible, but often not as one of its stated goals. This could have impact on how Marlin threats the removal of software features. The program experiences minimal software feature removal following its initial introduction. Support for hardware is discontinued only when it is demonstrated that a particular platform is rarely used. These factors suggest that the Marlin project may not be directly comparable to other software projects.

Programming Language. A threat to validity is that we have only looked at one project written in the `C++` programming language that works with external hardware. Therefore, it is possible that the results we have found may not applicable to other programming languages.

Timeframe. Lastly a threat to validity could be that we have only looked at the labeled features of Marlin that have been introduced since June 2019. This leaves a big gap of 4 years in which Marlin was also developed and possibly some of the more fundamental features were introduced. This could impact our conclusions with regard to the life cycle of features and how their properties impact their life cycle.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we performed an in-depth qualitative case study of the labeled software features of the Marlin Firmware. We have investigated 189 pull requests and tagged 2956 commits. We reported on the most common steps in the life cycle of Software features. These are the following: The refactoring of a feature, followed by bug fixing of a feature and finally the enhancement of a feature. One third of features never have their functionality changed after their creation, they only have their code changed in non-functional ways. Features are also affected by minor changes, but this seldom happens by itself and is often combined into other commits. Our findings show that previous changes to features are not an indication of what feature changes will happen. We have also noted a downside of the practice of following labeled software. Unlike with a manual software location, the maintainers of the system have the task of labeling features. And this is often done at the size of a pull request. So it is possible that multiple software features are labeled together because they are brought in one pull request. We have also reported on the impact that properties of the software features have on its later life. We have shown that there is a relation between the size of the feature, its centralization and discussion during implementation and how it will be impacted in its later life. Features with a very low number of lines changed, centralized in few files and with not a lot of discussion often have a low number of commits per day associated to them, and features with very large number of lines changed, decentralized over many files and with a lot of discussion often have a higher number of commits per day associated to them. The size together with the interaction of the community on the pull request also plays a role in the time it takes for a feature to be merged into the project. It is not clear if the size influences the interaction of the community and the time before merging, or if the interaction of the community affects the size and time before merging.

Another aspect that was investigated was the action of labeling of pull requests by maintainers of the project. We shown that all the pull requests that were present in our sample set hold to our definition of a feature and so we consider them correctly labeled. But we have also analyzed pull requests related to the labels with the highest number of software features. We have taken six labels and investigated 30 pull requests for each of them. We have discovered that for our samples consisting of pull requests not labeled as a new feature, bug fix or a cleanup up to 40% of them we would also consider a software feature. We also found pull requests that were not labeled as a feature similar to pull requests that were labeled as a feature by the maintainers of Marlin, and this is still happening, so it cannot be that the definition of a feature has changed for the maintainers over time. This indicates that either the definition of a feature is not the same for all the maintainers of Marlin or another explanation is that it is a hard and subjective matter to label a pull request as a software feature or not. To properly determine what constitutes a feature, it is essential to have a clearly defined definition against which to test your code. In the context of an open-source project, however, such priority may often be lacking. To enhance the accuracy of

labeling, it is crucial for a project to have well-defined labels. In the case of Marlin, only 16 labels are clearly defined, while the remaining 64 labels are indicated solely by their names, without further clarification of their purpose.

6.2 Future Work

Regarding future work related to this research, the results can be extended by investigating the life cycle of labeled features for other `C++` projects that are not hardware related, other hardware related projects written in a different programming language, the possible difference between an open-source project and closed-source project and possibly older projects to see if the life cycle has changed over time. Another avenue of research could be the comparison of features found through the labeling of pull requests and features found by more traditional software feature location methods. Developers themselves could be involved and asked for their definition of features after which the labeled features will be compared against their definition.

Methods to ease the research of topics like this could also be developed. Two major problems are in the way for the automatization of the problem, these are the tagging of the commits, and following changes made to the code of the features. For the automatization of tagging an avenue of approach could be the use of natural language processing for the tagging. For this to work correctly a large dataset would have to be gathered. And the problem of that commits have to be labeled from the point of view of the software feature it is impacting remains. Software would have to be developed to automate the following of changes made to the code of features. This software would have to be able to follow all the changes made to the feature through every iteration of the code. Some of the current problems associated with this is that files are not able to be followed forwards only backwards so changes to files are difficult to figure out and big changes to software features cause diff algorithms to lose track of the feature within the files. We have to follow the features forwards through time because it is not possible to know unless we have very detailed documentation what code belongs to a feature that was introduced in the past.

Bibliography

[1] Iago Abal, Jean Melo, Ştefan Stănciulescu, Claus Brabrand, Márcio Ribeiro, and Andrzej Wąsowski. Variability Bugs in Highly Configurable Systems: A Qualitative Analysis. *ACM Transactions on Software Engineering and Methodology*, 2018. doi: 10.1145/3149119. 7

[2] Sven Apel, Don Batory, Christian Kstner, and Gunter Saake. *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer, 2013. 1, 3

[3] K.H. Bennett, V.T. Rajlich, and N. Wilde. Software Evolution and the Staged Model of the Software Lifecycle. In Marvin V. Zelkowitz, editor, *Advances in Computers*, volume 56, pages 1–54. Elsevier, 2002. doi: https://doi.org/10.1016/S0065-2458(02)80003-1. 3

[4] Thorsten Berger, Daniela Lettner, Julia Rubin, Paul Grünbacher, Adeline Silva, Martin Becker, Marsha Chechik, and Krzysztof Czarnecki. What is a Feature? A Qualitative Study of Features in Industrial Software Product Lines. In *International Conference on Software Product Line (SPLC)*, SPLC '15, page 16–25. ACM, 2015. ISBN 9781450336130. doi: 10.1145/2791060.2791108. 1

[5] Jan Bosch. *Design and Use of Software Architectures: Adopting and Evolving a Product-line Approach*. ACM, 2000. 3

[6] Satish C J and Anand Mahendran. Software Documentation Management Issues and Practices: A Survey. *Indian Journal of Science and Technology*, 9, 2016. doi: 10.17485/ijst/2016/v9i20/86869. 4

[7] Jordi Cabot, Javier Luis Cánovas Izquierdo, Valerio Cosentino, and Belén Rolandi. Exploring the use of labels to categorize issues in Open-Source Software projects. In *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 550–554, 2015. doi: 10.1109/SANER.2015.7081875. 23

[8] Andreas Classen, Patrick Heymans, and Pierre Yves Schobbens. What's in a Feature: A Requirements Engineering Perspective. In *Fundamental Approaches to Software Engineering (FASE)*, pages 16–30. Springer, 03 2008. doi: 10.1007/978-3-540-78743-3_2. 3

[9] Wei Ding, Peng Liang, Antony Tang, Hans Van Vliet, and Mojtaba Shahin. How Do Open Source Communities Document Software Architecture: An Exploratory Survey. In *International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 136–145, 2014. doi: 10.1109/ICECCS.2014.26. 3

[10] Geanderson Esteves dos Santos and Eduardo Figueiredo. Commit Classification using Natural Language Processing: Experiments over Labeled Datasets. In *Conferencia Iberoamericana de Software Engineering (CIbSE)*, 2020. 14

[11] Aleksander Fabijan, Helena Holmström Olsson, and Jan Bosch. Customer Feedback and Data Collection Techniques in Software R&D: A Literature Review. In *International Conference on Software Business (ICSOB)*, pages 139–153, Cham, 2015. Springer International Publishing. doi: 10.1007/978-3-319-19593-3_12. 4

[12] Aleksander Fabijan, Helena Holmström Olsson, and Jan Bosch. Time to Say 'Good Bye': Feature Lifecycle. In *Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 9–16, 2016. doi: 10.1109/SEAA.2016.59. 3

[13] Akira Fujimoto, Yoshiki Higo, and Shinji Kusumoto. Towards Accurate File Tracking Based on AST Differences. In *Asia-Pacific Software Engineering Conference (APSEC)*, pages 553–558, 2021. doi: 10.1109/APSEC53868.2021.00067. 14

[14] Sirine Gharbi, Mohamed Wiem Mkaouer, Ilyes Jenhani, and Montassar Ben Messaoud. On the classification of software change messages using multi-label active learning. In *ACM/SIGAPP Symposium on Applied Computing (SAC)*, SAC '19, page 1760–1767. ACM, 2019. doi: 10.1145/3297280.3297452. 12

[15] M.L. Griss. Software reuse architecture, process, and organization for business success. In *Israeli Conference on Computer Systems and Software Engineering*, pages 86–89, 1997. doi: 10.1109/ICCSSE.1997.599879. 3

[16] Steffen Herbold, Alexander Trautsch, Benjamin Ledel, Alireza Aghamohammadi, Taher Ahmed Ghaleb, Kuljit Kaur Chahal, Tim Bossenmaier, Bhaveet Nagaria, Philip Makedonski, Matin Nili Ahmadabadi, Kristóf Szabados, Helge Spieker, Matej Madeja, Nathaniel Hoy, Valentina Lenarduzzi, Shangwen Wang, Gema Rodríguez-Pérez, Ricardo Colomo Palacios, Roberto Verdecchia, Paramvir Singh, Yihao Qin, Debasish Chakroborti, Willard Davis, Vijay Walunj, Hongjun Wu, Diego Marcilio, Omar Alam, Abdullah Aldaeej, Idan Amit, Burak Turhan, Simon Eismann, Anna-Katharina Wickert, Ivano Malavolta, Matús Sulír, Fatemeh H. Fard, Austin Z. Henley, Stratos Kourtzanidis, Eray Tuzun, Christoph Treude, Simin Maleki Shamasbi, Ivan Pashchenko, Marvin Wyrich, James Davis, Alexander Serebrenik, Ella Albrecht, Ethem Utku Aktas, Daniel Strüber, and Johannes Erbel. Large-Scale Manual Validation of Bug Fixing Commits: A Fine-grained Analysis of Tangling. *Computing Research Repository*, abs/2011.06244, 2020. doi: 10.48550/arXiv.2011.06244. 13

[17] Wenbin Ji, Thorsten Berger, Michal Antkiewicz, and Krzysztof Czarnecki. Maintaining feature traceability with embedded annotations. In *International Conference on Software Product Line (SPLC)*, SPLC '15, page 61–70. ACM, 2015. doi: 10.1145/2791060.2791107. 1

[18] K.C. Kang, Jaejoon Lee, and P. Donohoe. Feature-oriented product line engineering. *IEEE Software*, 19(4):58–65, 2002. doi: 10.1109/MS.2002.1020288. 3

[19] Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-021, Carnegie-Mellon University Software Engineering Institute, 1990. 3

[20] Rainer Koschke and Jochen Quante. On Dynamic feature Location. In *International Conference on Automated Software Engineering (ASE)*, page 86–95, 2005. doi: 10.1145/1101908.1101923. 4

[21] Jacob Krüger, Wanzi Gu, Hui Shen, Mukelabai Mukelabai, Regina Hebig, and Thorsten Berger. Towards a Better Understanding of Software Features and Their Characteristics: A Case Study of Marlin. In *International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)*. ACM, 2018. doi: 10.1145/3168365.3168371. 7

[22] Jacob Krüger, Mukelabai Mukelabai, Wanzi Gu, Hui Shen, Regina Hebig, and Thorsten Berger. Where is My Feature and What is it About? A Case Study on Recovering Feature Facets. *Journal of Systems and Software*, 152:239–253, 2019. doi: 10.1016/j.jss.2019.01.057. 7

[23] Jaejoon Lee and Dirk Muthig. Feature-oriented variability management in product line engineering. *Communications of the ACM*, 49(12):55–59, 2006. doi: 10.1145/1183236.1183266. 3

[24] Franz Lehner. Software life cycle management based on a phase distinction method. *Microprocessing and Microprogramming*, 32(1):603–608, 1991. doi: https://doi.org/10.1016/0165-6074(91)90409-M. Euromicro symposium on microprocessing and microprogramming. 4

[25] Stanislav Levin and Amiram Yehudai. Boosting Automatic Commit Classification Into Maintenance Activities By Utilizing Source Code Changes. In *Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE)*, PROMISE, page 97–106. ACM, 2017. doi: 10.1145/3127005.3127016. 14

[26] Jorg Liebig, Sven Apel, Christian Lengauer, Christian Kästner, and Michael Schulze. An Analysis of the Variability in Forty Preprocessor-Based Software Product Lines. In *International Conference on Software Engineering (ICSE)*, volume 1, pages 105–114, 2010. doi: 10.1145/1806799.1806819. 5

[27] David Lorge Parnas. Software Aging. In *International Conference on Software Engineering (ICSE)*, ICSE '94, page 279–287, Washington, DC, USA, 1994. IEEE. 4

[28] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005. 3

[29] Roger Pressman. *Software Engineering: A Practitioner's Approach*. McGraw Hill, 7 edition, 2009. 23

[30] Francisca Pérez, Jorge Echeverría, Raúl Lapeña Martí, and Carlos Cetina. Comparing Manual and Automated Feature Location in Conceptual Models: A Controlled Experiment. *Information and Software Technology*, 125:106337, 2020. doi: 10.1016/j.infsof.2020.106337. 5

[31] Rodrigo Queiroz, Leonardo Passos, Marco Tulio Valente, Claus Hunsen, Sven Apel, and Krzysztof Czarnecki. The Shape of Feature Code: An Analysis of Twenty C-Preprocessor-Based Systems. *Soft Systems Methodology*, 16(1):77–96, 2017. doi: 10.1007/s10270-015-0483-z. 5

[32] Julia Rubin and Marsha Chechik. *A Survey of Feature Location Techniques*, pages 29–58. Springer, 2013. doi: 10.1007/978-3-642-36654-3_2. 4

[33] Juha Savolainen and Juha Kuusela. Violatility analysis framework for product lines. *ACM Sigsoft Software Engineering Notes*, 26(3):133–141, 2001. ISSN 0163-5948. doi: 10.1145/379377.375277. 3

[34] Ştefan Stănciulescu, Sandro Schulze, and Andrzej Wąsowski. Forked and integrated variants in an open-source firmware project. In *International Conference on Software Maintenance and Evolution (ICSME)*, pages 151–160, 2015. doi: 10.1109/ICSM.2015.7332461. 7

[35] Mikael Svahnberg, Jilles van Gurp, Gurp And, and Jan Bosch. A Taxonomy of Variability Realization Techniques. *Software: Practice and Experience*, 35, 2005. doi: 10.1002/spe.652. 5

[36] Sören Viegener. Empirical evaluation of feature trace recording on the edit history of Marlin. Bachelor's thesis, Universität Ulm, 2021. 7

Appendix A

Marlin Label Documentation

List of all the definition we have made for the labels used in the Marlin repository:

A: AT90USB [1] This label indicates that the pull request covers code for the functionality of the AT90USB Family AVR platform.

A: ATmega1280 [2] This label indicates that the pull request covers code for the functionality of the ATmega1280 AVR platform.

A: ATmega1284 [3] This label indicates that the pull request covers code for the functionality of the ATmega1284 AVR platform.

A: ATmega2560 [4] This label indicates that the pull request covers code for the functionality of the ATmega2560 AVR platform.

A: ESP32 [5] This label indicates that the pull request covers code for the functionality of the ESP32 platform

A: LPC176x This label indicates that the pull request covers code for the functionality of the LPC176X Family of ARM processors from NXP.

A: RP2040 [6] This label indicates that the pull request covers code for the functionality of the ATmega2560 AVR platform

A: SAM3X8C/E [7] This label indicates that the pull request covers code for the functionality of the ATSAM3X8C or ATSAM3X8E ARM microchip from Atmel.

A: SAMD21 [8] This label indicates that the pull request covers code for the functionality of the ATmega2560 AVR platform. Only a single pull request is labeled with this label.

A: SAMD51 [9] This label indicates that the pull request covers code for the functionality of the ATSAMD51N19A

A: STM32 [10] This label indicates that the pull request covers code for the functionality of the STM32 family of microcontrollers.

[1] https://www.microchip.com/en-us/product/at90usb162
[2] https://www.microchip.com/en-us/product/atmega1280
[3] https://www.microchip.com/en-us/product/atmega1284
[4] https://www.microchip.com/en-us/product/atmega2560
[5] https://www.espressif.com/en/products/socs/esp32
[6] https://www.raspberrypi.com/products/rp2040/
[7] https://www.microchip.com/en-us/product/atsam3x8c
[8] https://www.microchip.com/en-us/product/atsamd21g18
[9] https://www.microchip.com/en-us/product/atsamd51n19a
[10] https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html

These are all the labels that start with the prefix of `A:`. This `A:` stands for `Architecture` referring to the different architectures of the chips. The pull requests associated with these labels are most often bugs that have been introduced when general code was changed, and the microcontroller functionality breaks because of it.

Bug: Confirmed ! This label indicates that it is indeed confirmed that a bug in the software caused the problem.

Bug: Potential ? This label indicates that it is possible that a bug caused the problem. The label is removed once it has been confirmed that it is a bug or stays if it turns out it was not a bug. This label should not be used for pull requests but for a small number of requests it was used mostly in 2016.

Bug? False Alarm This label indicates that what was thought to be a bug that causes the problem was user error or a malfunction of a device.

Bug? Feature! This label indicates that behavior that was thought to be a bug turned out to be intentional behavior.

These are all the labels that start with the prefix of `Bug`. This `Bug` stands for a software bug. The labels seem to not be in use anymore for the labeling of pull requests since that the last pull request to be labeled #26404[11] is almost a year old.

C: Boards/Pins This label indicates that the pull request has to do with a specific board or the datapins of those boards.

C: Build / Toolchain This label indicates that the pull request has to do with the process of building the Marlin firmware and the tools involved in this.

C: Configuration This label indicates that the pull request is related to the Configurations of Marlin. Difference between this label and `PR: Configurations` is not clear.

C: Documentation This label indicates that the pull request is involved with the comments in code files, the warnings that the system gives the user or documentation files of the project.

C: G-code Parser [12] This label indicates that the pull request is involved with the parsing of G-code, G-code are the commands that Marlin executes to perform specific tasks. This label has not been used since February 2020.

C: Hosts & Protocols This label indicates that the pull request has to do with the connection between the host device and Marlin and the protocols involved in the data transfer between the two.

C: Language This label indicates that the pull request has to do with the language files of Marlin such as it many translations.

C: LCD & Controllers This label indicates that the pull request has to do with the driving of screens and their controllers.

C: Motion This label indicates that the pull request has to do with the movement of the nozzle for printing and other purposes.

C: Optimization This label indicates that the pull request has to do with optimizing parts of the codebase to be more efficient or less space consuming.

C: Peripherals This label indicates that the pull request has to do with peripherals that are not directly related to the printing process such as fans and lighting.

[11] https://github.com/MarlinFirmware/Marlin/pull/26404
[12] https://marlinfw.org/meta/gcode/

C: Safety This label indicates that the pull request has to do with making sure that the device operates safely and if something goes wrong that it is properly handled.

C: Serial Comms This label indicates that the pull request has to do with the serial communication between Marlin and the host

C: Temperatures This label indicates that the pull request has to do with the heated bed of the 3D printer or the hotend

C: User Interface This label indicates that the pull request has to do with the user interface of the Marlin machine. Such as how menus are organized or what certain button actions do.

These are all the labels that start with the prefix of `C:`. This `C:` could refer to different things, one possible explanation is `Code` and that the labels refer to the area of the codebase that is about the topic in the name, another explanation might be that it stands for `Component` that the different labels talk about components of the Marlin system.

F: BLTouch This label indicates that the pull request has to do with the BLTouch sensor that is used for the leveling of the bed.

F: Calibration This label indicates that the pull request has to do with all the configuration methods that Marlin offers to calibrate the device to perform as well as possible. Such as extending the range of certain values when sensor to measure them become better or improving the algorithm for the probing sequence of the bed leveling procedure.

F: CNC / Laser This label indicates that the pull request has to do with the alternative function of the Marlin firmware of being used as a CNC machine or a Laser machine.

F: EEPROM [13] This label indicates that the pull request has to do with EEPROM a part of the read-only memory where Marlin stores the settings for the machines.

F: Filament Sensor This label indicates that the pull request has to do with the Filament Sensor which detects if something is wrong with the filament being used for the printing such as it running out.

F: IDEX This label indicates that the pull request has to do with a machine that supports dual independent extruders.

F: Linear Advance [14] This label indicates that the pull request has to do with the linear advance which is a method of controlling the speed of the extruder such that the print quality increases.

F: Print Recovery This label indicates that the pull request has to do with the ability of Marlin to recover from an error such as a power loss of the system. This is done by recovery files of where Marlin is in the progress of printing.

F: SD Card / Media This label indicates that the pull request has to do with the handling of SD cards and other storage media. Most print files are stored on SD cards that get inserted into the machine.

F: Toolchange This label indicates that the pull request has to do with the ability of Marlin to swap out extruders and other tools automatically.

F: Trinamic This label indicates that the pull request has to do with Trinamic drivers which are stepper motors for 3D printers that are more precise that normal stepper motors.

[13] https://marlinfw.org/docs/features/eeprom.html
[14] https://marlinfw.org/docs/features/lin_advance.html

F: U.B.L. This label indicates that the pull request has to do with Unified Bed Leveling (U.B.L.) which is the superset of all other bed leveling systems in Marlin.

F: Z-Probes This label indicates that the pull request has to do with the Z probe, also known as the bed probe which enables the probing of the bed to figure out the exact distance to the bed surface.

These are all the labels that start with the prefix of `F:`. This `F:` refers to `Feature` because all the labels highlight a specific feature/capability of Marlin. The labels seem to not fully cover all the features that Marlin claims to have because on the webpage[15] of Marlin we can find more documentation about features that do not have labels in the GitHub Repository.

K: Core / H-Bot This label indicates that the pull request has to do with Core or H-bot type of kinematics of the extruder.

K: Deltabot This label indicates that the pull request has to do with Delta kinematics of the extruder.

K: SCARA This label indicates that the pull request has to do with SCARA kinematics.

These are all the labels that start with the prefix of `K:`. This `K:` stands for `Kinematics` which refers to the motion system used to move the head of the printer around. The only label missing here is for the Cartesian system that Marlin also supports.

Needs: Discussion This label indicates that the pull request needs more discussion before development is able to continue.

Needs: Documentation This label indicates that the pull request needs more comments in the code file to explain how it works and what the settings should be set to. The label is often removed once merged.

Needs: More Data This label indicates that the pull request needs more information before development is able to continue.

Needs: Owner This label is currently not in use. It is only used when a pull request needs someone to be in charge of development.

Needs: Patch This label indicates that the pull request needs work to be able to be merged. Same purpose as the `Needs: Work` label. It seems to be that the `Needs: Patch` label is intended for issues and the `Needs: Work` label is intended for pull requests.

Needs: Testing This label indicates that the pull request has to be tested with different machines to see if it works. The difference between this label and `S: Please Test` is not clear.

Needs: Work This label indicates that the pull request needs work to be able to be merged. Same purpose as `Needs: Patch`.

These are all the labels that start with the prefix of `Needs:`. The `Needs:` label in Marlin is used for administrative tasks of communicating information to the contributors about the commits.

PR: Bug Fix This label indicates that the pull request is a Bug Fix

PR: Coding Standards This label indicates that the pull request is related to the Coding Standards of Marlin

PR: Configurations This label indicates that the pull request is related to the Configurations of Marlin. Difference between this label and `C: Configuration` is not clear.

[15] https://marlinfw.org/meta/features/

PR: General Cleanup This label indicates that the pull request is to clean up the code and make it more readable or remove unnecessary parts.

PR: Improvement This label indicates that the pull request is an improvement to a function of Marlin.

PR: New Feature This label indicates that the pull request implements a new feature in the Marlin system.

PR: Workaround This label indicates that the pull request is a temporary solution and should be properly solved in the future.

These are all the labels that start with the prefix of `PR:`. The `PR:` stands for `Pull Request` and is about what type of pull request it is.

S: Don't Merge This label indicates that the pull request is a work in process and needs more work or discussion before it can be merged. The label is often removed once merged.

S: Experimental This label indicates that the pull request is an experimental nontraditional way of solving a problem.

S: Hold for 2.1 This label indicates that the pull request is not to be merged until version 2.1 was released. Only one pull request left with this label.

S: Please Merge This label indicates that the pull request is done, and the maintainer should merge it into the codebase.

S: Please Test This label indicates that the pull request has to be tested with different machines to see if it works. Difference between this label and `Needs: Testing` is not clear.

S: Solved This label indicates that the pull request has solved the problem. This label is not used anymore and before it was not often used as a label.

S: Superseded This label indicates that the pull request has been superseded by another pull request and for that reason is not necessary anymore.

These are all the labels that start with the prefix of `S:`. The `S:` we have not found a satisfying answer of what it could stand for. Some of the labels describe actions concerning the pull request and others describe the state of the pull request. There is no conclusive answer to what binds these labels together.

T: Design Concept This label indicates that the pull request is a concept for a new way to solve a problem or a new way to implement an already existing feature.

T: Development This label indicates that the pull request is about the makefiles, PlatformIO and Python scripts of the project. The files that are necessary for developing Marlin and not running it.

T: Feature Request This label indicates that the pull request is a feature request. Last used in October 2017 and now only present as a label used for issues.

T: GitHub Admin This label indicates that the pull request is about Github administrative actions

T: GPL This label is not used for pull requests only for issues.

T: HAL & APIs [16] This label indicates that the pull request is about the Hardware Abstraction Layer of Marlin

[16] https://marlinfw.org/docs/development/hal.html

T: Question This label is not used for pull requests only for issues.

T: Suggestion This label indicates that the pull request is a suggestion to do some functionality differently.

These are all the labels that start with the prefix of `T:`. The `T:` could stand for `Topic` because the labels cover quite wide topics that are not necessarily related to the code of Marlin such as a feature or a specific component of the system.

stale-closing-soon This label is not used for pull requests only for issues.

no-locking This label is currently not in use but seems to indicate that a pull request should not be locked by the Lock-bot.

Fix Included This label is not used for pull requests only for issues.

good first issue This label is not used for pull requests only for issues.

help wanted This label indicates that the pull request needs help from other people that might be more knowledgeable in the topic. The label is often removed once merged.

These are all the labels without any prefix. Most of these labels are not used for pull requests besides the `help wanted` label but that label would fit better in the `Needs:` category as it is like the behavior of those labels.

Appendix B

Additional Figures

This Appendix will contain figures that provide extra context to the work and the numbers shown within it.

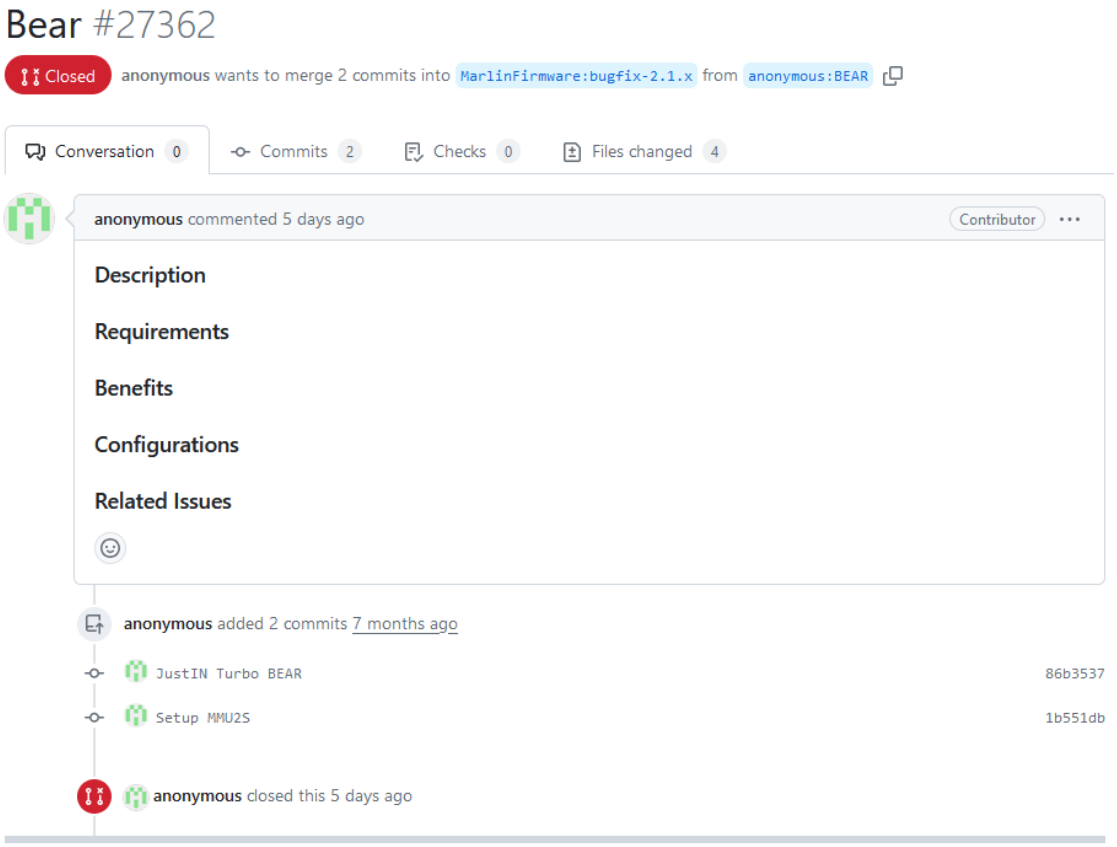

Figure B.1: A pull request which does not make clear what it is about and is for that reason closed by an admin.

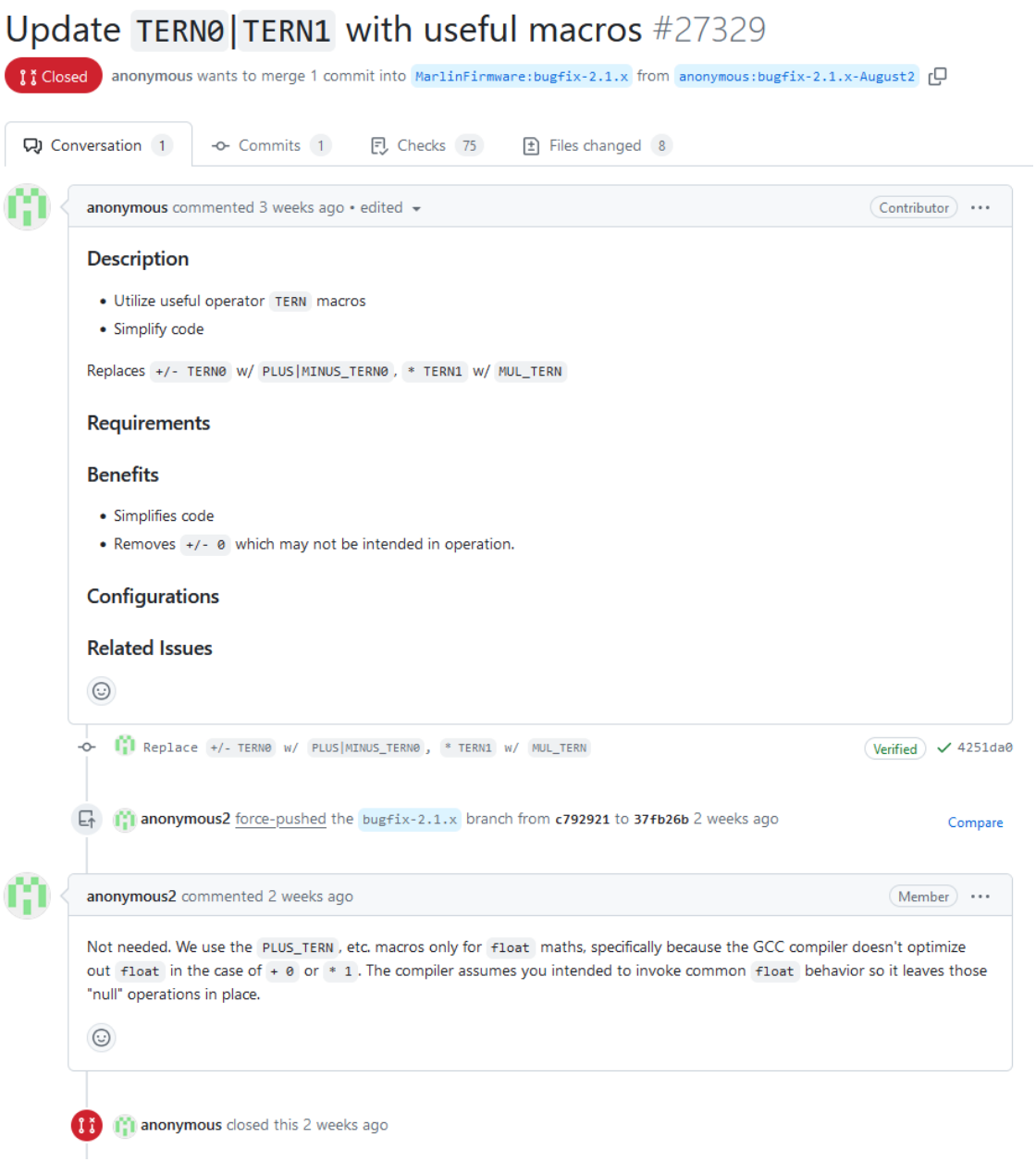

Figure B.2: The change proposed by the contributor is shut down by the maintainer of the project with a reason.

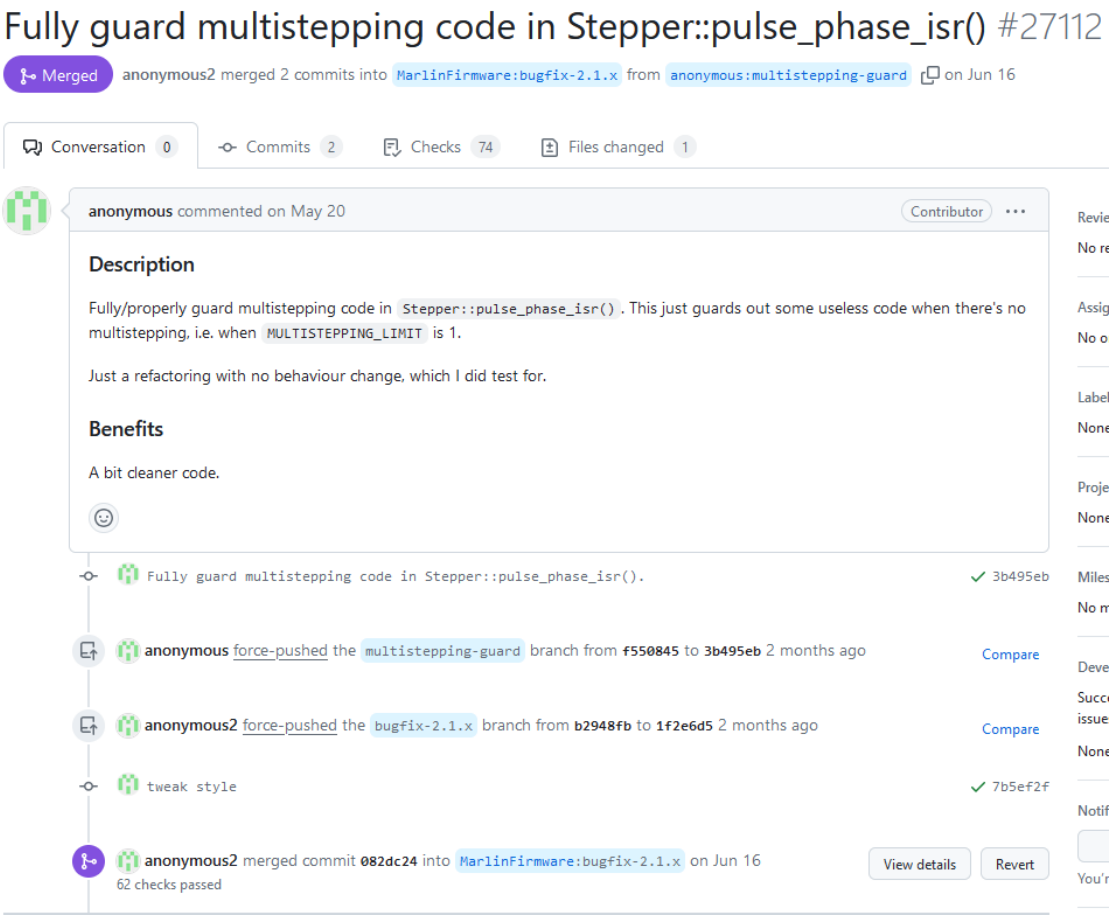

Figure B.3: A simple change quickly accepted by the maintainer without the need for a label.

Appendix C

Additional Tables

This Appendix will contain tables that provide extra context to the work and the numbers shown within it.

Label	NewFeature	BugFix	Enhancement	Refactor	Revert	Rework	Cleanup	Comment	Formatting	Whitespace	Removal	Total
PR: New Feature	189	353	367	1779	19	18	51	101	63	14	2	2956
C: LCD & Controllers	37	95	85	486	2	3	8	25	13	1	0	755
C: Motion	18	51	69	273	4	2	7	5	2	2	1	434
C: Peripherals	16	13	14	96	1	0	2	5	5	0	0	152
A: STM32	13	24	31	182	2	0	6	23	12	2	0	295
F: Calibration	12	39	49	147	4	1	6	6	0	0	0	264
PR: Improvement	12	15	27	117	1	0	3	1	3	0	1	180
C: Temperatures	11	18	9	73	2	0	2	6	3	0	0	124
C: Boards/Pins	9	13	19	125	1	0	5	17	11	2	1	203
C: Hosts & Protocols	9	12	12	53	0	1	2	0	1	0	0	90
C: Build / Toolchain	8	28	13	88	2	0	4	11	13	4	1	172
C: Safety	8	9	12	96	2	1	5	6	8	0	0	147
Needs: Testing	8	14	10	51	0	1	1	5	1	1	0	92
Needs: Work	7	9	7	32	0	1	2	0	0	0	1	59
F: Z-Probes	7	16	22	69	1	2	1	2	0	0	0	120
F: CNC / Laser	7	7	6	80	1	0	6	6	6	1	0	120
F: SD Card / Media	6	15	8	31	0	0	0	2	2	0	0	64
C: Serial Comms	6	7	5	29	0	0	2	6	2	1	0	58
F: Trinamic	4	7	7	16	0	1	0	2	0	0	0	37
C: Configuration	4	3	1	14	0	0	0	2	0	0	0	24
T: HAL & APIs	4	2	2	12	0	1	0	2	3	0	0	26
S: Please Merge	3	7	8	60	1	0	3	8	4	2	0	96
Needs: Discussion	3	8	7	22	0	1	0	0	0	0	0	41
S: Don't Merge	3	2	7	11	0	3	1	0	0	0	0	27
S: Experimental	3	6	9	27	0	2	1	0	0	0	0	48
T: Development	2	18	6	46	3	0	2	5	4	3	0	89
F: Toolchange	2	1	1	6	0	1	1	0	0	0	0	12
C: User Interface	2	7	8	53	0	0	3	2	3	1	0	79
F: U.B.L.	2	2	0	4	0	0	0	1	0	0	0	9
PR: Bug Fix	1	1	1	8	0	0	0	0	2	0	0	13
F: BLTouch	1	4	1	12	0	0	0	0	0	0	0	18
K: Core / H-Bot	1	3	0	13	0	0	0	0	1	0	0	18
help wanted	1	0	0	0	0	0	0	0	0	0	1	2
A: SAMD51	1	3	0	11	0	0	0	1	1	1	0	18
Needs: Documentation	1	4	3	29	0	0	0	0	0	0	0	37
C: G-code Parser	1	3	6	23	0	0	1	0	0	0	0	34
C: Optimization	1	3	6	23	0	0	1	0	0	0	0	34
T: Suggestion	1	2	1	2	0	0	0	0	0	0	0	6
K: Deltabot	1	1	0	3	0	0	0	0	0	0	0	5
S: Hold for 2.1	1	4	11	9	0	0	2	0	0	0	0	27
Needs: More Data	1	1	2	2	0	1	0	0	0	0	0	7
F: Filament Sensor	1	1	0	2	0	0	0	0	0	0	0	4

Table C.1: Labels together with number of different tags for commits.

Pull request	URL
3110	https://github.com/MarlinFirmware/Marlin/pull/3110
4811	https://github.com/MarlinFirmware/Marlin/pull/4811
9150	https://github.com/MarlinFirmware/Marlin/pull/9150
10434	https://github.com/MarlinFirmware/Marlin/pull/10434
12895	https://github.com/MarlinFirmware/Marlin/pull/12895
13498	https://github.com/MarlinFirmware/Marlin/pull/13498
14251	https://github.com/MarlinFirmware/Marlin/pull/14251
14265	https://github.com/MarlinFirmware/Marlin/pull/14265
14309	https://github.com/MarlinFirmware/Marlin/pull/14309
14313	https://github.com/MarlinFirmware/Marlin/pull/14313
14380	https://github.com/MarlinFirmware/Marlin/pull/14380
14388	https://github.com/MarlinFirmware/Marlin/pull/14388
14391	https://github.com/MarlinFirmware/Marlin/pull/14391
14407	https://github.com/MarlinFirmware/Marlin/pull/14407
14456	https://github.com/MarlinFirmware/Marlin/pull/14456
14533	https://github.com/MarlinFirmware/Marlin/pull/14533
14535	https://github.com/MarlinFirmware/Marlin/pull/14535
14539	https://github.com/MarlinFirmware/Marlin/pull/14539
14546	https://github.com/MarlinFirmware/Marlin/pull/14546
14566	https://github.com/MarlinFirmware/Marlin/pull/14566
14595	https://github.com/MarlinFirmware/Marlin/pull/14595
14619	https://github.com/MarlinFirmware/Marlin/pull/14619
14639	https://github.com/MarlinFirmware/Marlin/pull/14639
14648	https://github.com/MarlinFirmware/Marlin/pull/14648
14667	https://github.com/MarlinFirmware/Marlin/pull/14667
14730	https://github.com/MarlinFirmware/Marlin/pull/14730
14757	https://github.com/MarlinFirmware/Marlin/pull/14757
14807	https://github.com/MarlinFirmware/Marlin/pull/14807
14810	https://github.com/MarlinFirmware/Marlin/pull/14810
14882	https://github.com/MarlinFirmware/Marlin/pull/14882
14883	https://github.com/MarlinFirmware/Marlin/pull/14883
14924	https://github.com/MarlinFirmware/Marlin/pull/14924
14953	https://github.com/MarlinFirmware/Marlin/pull/14953
14956	https://github.com/MarlinFirmware/Marlin/pull/14956
14961	https://github.com/MarlinFirmware/Marlin/pull/14961
14991	https://github.com/MarlinFirmware/Marlin/pull/14991
15060	https://github.com/MarlinFirmware/Marlin/pull/15060
15069	https://github.com/MarlinFirmware/Marlin/pull/15069
15081	https://github.com/MarlinFirmware/Marlin/pull/15081
15195	https://github.com/MarlinFirmware/Marlin/pull/15195
15209	https://github.com/MarlinFirmware/Marlin/pull/15209
15245	https://github.com/MarlinFirmware/Marlin/pull/15245
15252	https://github.com/MarlinFirmware/Marlin/pull/15252
15285	https://github.com/MarlinFirmware/Marlin/pull/15285
15333	https://github.com/MarlinFirmware/Marlin/pull/15333
15335	https://github.com/MarlinFirmware/Marlin/pull/15335
15374	https://github.com/MarlinFirmware/Marlin/pull/15374
15376	https://github.com/MarlinFirmware/Marlin/pull/15376
15394	https://github.com/MarlinFirmware/Marlin/pull/15394
15462	https://github.com/MarlinFirmware/Marlin/pull/15462
15463	https://github.com/MarlinFirmware/Marlin/pull/15463
15475	https://github.com/MarlinFirmware/Marlin/pull/15475

Pull request	URL
15481	https://github.com/MarlinFirmware/Marlin/pull/15481
15497	https://github.com/MarlinFirmware/Marlin/pull/15497
15498	https://github.com/MarlinFirmware/Marlin/pull/15498
15525	https://github.com/MarlinFirmware/Marlin/pull/15525
15549	https://github.com/MarlinFirmware/Marlin/pull/15549
15585	https://github.com/MarlinFirmware/Marlin/pull/15585
15590	https://github.com/MarlinFirmware/Marlin/pull/15590
15593	https://github.com/MarlinFirmware/Marlin/pull/15593
15631	https://github.com/MarlinFirmware/Marlin/pull/15631
15640	https://github.com/MarlinFirmware/Marlin/pull/15640
15645	https://github.com/MarlinFirmware/Marlin/pull/15645
15650	https://github.com/MarlinFirmware/Marlin/pull/15650
15690	https://github.com/MarlinFirmware/Marlin/pull/15690
15714	https://github.com/MarlinFirmware/Marlin/pull/15714
15739	https://github.com/MarlinFirmware/Marlin/pull/15739
15796	https://github.com/MarlinFirmware/Marlin/pull/15796
15930	https://github.com/MarlinFirmware/Marlin/pull/15930
15931	https://github.com/MarlinFirmware/Marlin/pull/15931
16026	https://github.com/MarlinFirmware/Marlin/pull/16026
16050	https://github.com/MarlinFirmware/Marlin/pull/16050
16068	https://github.com/MarlinFirmware/Marlin/pull/16068
16207	https://github.com/MarlinFirmware/Marlin/pull/16207
16217	https://github.com/MarlinFirmware/Marlin/pull/16217
16277	https://github.com/MarlinFirmware/Marlin/pull/16277
16293	https://github.com/MarlinFirmware/Marlin/pull/16293
16317	https://github.com/MarlinFirmware/Marlin/pull/16317
16362	https://github.com/MarlinFirmware/Marlin/pull/16362
16404	https://github.com/MarlinFirmware/Marlin/pull/16404
16452	https://github.com/MarlinFirmware/Marlin/pull/16452
16466	https://github.com/MarlinFirmware/Marlin/pull/16466
16533	https://github.com/MarlinFirmware/Marlin/pull/16533
16538	https://github.com/MarlinFirmware/Marlin/pull/16538
16539	https://github.com/MarlinFirmware/Marlin/pull/16539
16551	https://github.com/MarlinFirmware/Marlin/pull/16551
16554	https://github.com/MarlinFirmware/Marlin/pull/16554
16562	https://github.com/MarlinFirmware/Marlin/pull/16562
16599	https://github.com/MarlinFirmware/Marlin/pull/16599
16641	https://github.com/MarlinFirmware/Marlin/pull/16641
16731	https://github.com/MarlinFirmware/Marlin/pull/16731
16741	https://github.com/MarlinFirmware/Marlin/pull/16741
16756	https://github.com/MarlinFirmware/Marlin/pull/16756
16792	https://github.com/MarlinFirmware/Marlin/pull/16792
16864	https://github.com/MarlinFirmware/Marlin/pull/16864
16872	https://github.com/MarlinFirmware/Marlin/pull/16872
16897	https://github.com/MarlinFirmware/Marlin/pull/16897
16960	https://github.com/MarlinFirmware/Marlin/pull/16960
17017	https://github.com/MarlinFirmware/Marlin/pull/17017
17056	https://github.com/MarlinFirmware/Marlin/pull/17056
17195	https://github.com/MarlinFirmware/Marlin/pull/17195
17222	https://github.com/MarlinFirmware/Marlin/pull/17222
17239	https://github.com/MarlinFirmware/Marlin/pull/17239
17248	https://github.com/MarlinFirmware/Marlin/pull/17248

Pull request	URL
17273	https://github.com/MarlinFirmware/Marlin/pull/17273
17281	https://github.com/MarlinFirmware/Marlin/pull/17281
17285	https://github.com/MarlinFirmware/Marlin/pull/17285
17298	https://github.com/MarlinFirmware/Marlin/pull/17298
17437	https://github.com/MarlinFirmware/Marlin/pull/17437
17462	https://github.com/MarlinFirmware/Marlin/pull/17462
17523	https://github.com/MarlinFirmware/Marlin/pull/17523
17531	https://github.com/MarlinFirmware/Marlin/pull/17531
17536	https://github.com/MarlinFirmware/Marlin/pull/17536
17560	https://github.com/MarlinFirmware/Marlin/pull/17560
17697	https://github.com/MarlinFirmware/Marlin/pull/17697
17741	https://github.com/MarlinFirmware/Marlin/pull/17741
17788	https://github.com/MarlinFirmware/Marlin/pull/17788
17806	https://github.com/MarlinFirmware/Marlin/pull/17806
17814	https://github.com/MarlinFirmware/Marlin/pull/17814
17817	https://github.com/MarlinFirmware/Marlin/pull/17817
17818	https://github.com/MarlinFirmware/Marlin/pull/17818
17853	https://github.com/MarlinFirmware/Marlin/pull/17853
17865	https://github.com/MarlinFirmware/Marlin/pull/17865
17884	https://github.com/MarlinFirmware/Marlin/pull/17884
17908	https://github.com/MarlinFirmware/Marlin/pull/17908
17937	https://github.com/MarlinFirmware/Marlin/pull/17937
17945	https://github.com/MarlinFirmware/Marlin/pull/17945
18039	https://github.com/MarlinFirmware/Marlin/pull/18039
18071	https://github.com/MarlinFirmware/Marlin/pull/18071
18112	https://github.com/MarlinFirmware/Marlin/pull/18112
18145	https://github.com/MarlinFirmware/Marlin/pull/18145
18177	https://github.com/MarlinFirmware/Marlin/pull/18177
18316	https://github.com/MarlinFirmware/Marlin/pull/18316
18326	https://github.com/MarlinFirmware/Marlin/pull/18326
18342	https://github.com/MarlinFirmware/Marlin/pull/18342
18347	https://github.com/MarlinFirmware/Marlin/pull/18347
18389	https://github.com/MarlinFirmware/Marlin/pull/18389
18399	https://github.com/MarlinFirmware/Marlin/pull/18399
18504	https://github.com/MarlinFirmware/Marlin/pull/18504
18576	https://github.com/MarlinFirmware/Marlin/pull/18576
18655	https://github.com/MarlinFirmware/Marlin/pull/18655
18680	https://github.com/MarlinFirmware/Marlin/pull/18680
18695	https://github.com/MarlinFirmware/Marlin/pull/18695
18735	https://github.com/MarlinFirmware/Marlin/pull/18735
18736	https://github.com/MarlinFirmware/Marlin/pull/18736
18737	https://github.com/MarlinFirmware/Marlin/pull/18737
18738	https://github.com/MarlinFirmware/Marlin/pull/18738
18808	https://github.com/MarlinFirmware/Marlin/pull/18808
18830	https://github.com/MarlinFirmware/Marlin/pull/18830
18866	https://github.com/MarlinFirmware/Marlin/pull/18866
18870	https://github.com/MarlinFirmware/Marlin/pull/18870
18906	https://github.com/MarlinFirmware/Marlin/pull/18906
18912	https://github.com/MarlinFirmware/Marlin/pull/18912
18952	https://github.com/MarlinFirmware/Marlin/pull/18952
18972	https://github.com/MarlinFirmware/Marlin/pull/18972
19135	https://github.com/MarlinFirmware/Marlin/pull/19135

Pull request	URL
19139	https://github.com/MarlinFirmware/Marlin/pull/19139
19225	https://github.com/MarlinFirmware/Marlin/pull/19225
19235	https://github.com/MarlinFirmware/Marlin/pull/19235
19330	https://github.com/MarlinFirmware/Marlin/pull/19330
19349	https://github.com/MarlinFirmware/Marlin/pull/19349
19384	https://github.com/MarlinFirmware/Marlin/pull/19384
19436	https://github.com/MarlinFirmware/Marlin/pull/19436
19465	https://github.com/MarlinFirmware/Marlin/pull/19465
19674	https://github.com/MarlinFirmware/Marlin/pull/19674
19682	https://github.com/MarlinFirmware/Marlin/pull/19682
19801	https://github.com/MarlinFirmware/Marlin/pull/19801
19817	https://github.com/MarlinFirmware/Marlin/pull/19817
19828	https://github.com/MarlinFirmware/Marlin/pull/19828
19837	https://github.com/MarlinFirmware/Marlin/pull/19837
19856	https://github.com/MarlinFirmware/Marlin/pull/19856
19878	https://github.com/MarlinFirmware/Marlin/pull/19878
19901	https://github.com/MarlinFirmware/Marlin/pull/19901
19905	https://github.com/MarlinFirmware/Marlin/pull/19905
19912	https://github.com/MarlinFirmware/Marlin/pull/19912
19963	https://github.com/MarlinFirmware/Marlin/pull/19963
19971	https://github.com/MarlinFirmware/Marlin/pull/19971
19978	https://github.com/MarlinFirmware/Marlin/pull/19978
20079	https://github.com/MarlinFirmware/Marlin/pull/20079
20084	https://github.com/MarlinFirmware/Marlin/pull/20084
20087	https://github.com/MarlinFirmware/Marlin/pull/20087
20099	https://github.com/MarlinFirmware/Marlin/pull/20099
20113	https://github.com/MarlinFirmware/Marlin/pull/20113
20123	https://github.com/MarlinFirmware/Marlin/pull/20123
20147	https://github.com/MarlinFirmware/Marlin/pull/20147
20148	https://github.com/MarlinFirmware/Marlin/pull/20148
20151	https://github.com/MarlinFirmware/Marlin/pull/20151
20153	https://github.com/MarlinFirmware/Marlin/pull/20153
20159	https://github.com/MarlinFirmware/Marlin/pull/20159
20218	https://github.com/MarlinFirmware/Marlin/pull/20218
20241	https://github.com/MarlinFirmware/Marlin/pull/20241
20262	https://github.com/MarlinFirmware/Marlin/pull/20262
20280	https://github.com/MarlinFirmware/Marlin/pull/20280
20383	https://github.com/MarlinFirmware/Marlin/pull/20383
20384	https://github.com/MarlinFirmware/Marlin/pull/20384
20410	https://github.com/MarlinFirmware/Marlin/pull/20410
20427	https://github.com/MarlinFirmware/Marlin/pull/20427
20444	https://github.com/MarlinFirmware/Marlin/pull/20444
20455	https://github.com/MarlinFirmware/Marlin/pull/20455
20537	https://github.com/MarlinFirmware/Marlin/pull/20537
20544	https://github.com/MarlinFirmware/Marlin/pull/20544
20549	https://github.com/MarlinFirmware/Marlin/pull/20549
20571	https://github.com/MarlinFirmware/Marlin/pull/20571
20678	https://github.com/MarlinFirmware/Marlin/pull/20678
20711	https://github.com/MarlinFirmware/Marlin/pull/20711
20733	https://github.com/MarlinFirmware/Marlin/pull/20733
20755	https://github.com/MarlinFirmware/Marlin/pull/20755
20802	https://github.com/MarlinFirmware/Marlin/pull/20802

Pull request	URL
20835	https://github.com/MarlinFirmware/Marlin/pull/20835
20940	https://github.com/MarlinFirmware/Marlin/pull/20940
20956	https://github.com/MarlinFirmware/Marlin/pull/20956
21005	https://github.com/MarlinFirmware/Marlin/pull/21005
21148	https://github.com/MarlinFirmware/Marlin/pull/21148
21161	https://github.com/MarlinFirmware/Marlin/pull/21161
21212	https://github.com/MarlinFirmware/Marlin/pull/21212
21237	https://github.com/MarlinFirmware/Marlin/pull/21237
21255	https://github.com/MarlinFirmware/Marlin/pull/21255
21344	https://github.com/MarlinFirmware/Marlin/pull/21344
21387	https://github.com/MarlinFirmware/Marlin/pull/21387
21403	https://github.com/MarlinFirmware/Marlin/pull/21403
21427	https://github.com/MarlinFirmware/Marlin/pull/21427
21431	https://github.com/MarlinFirmware/Marlin/pull/21431
21503	https://github.com/MarlinFirmware/Marlin/pull/21503
21534	https://github.com/MarlinFirmware/Marlin/pull/21534
21590	https://github.com/MarlinFirmware/Marlin/pull/21590
21612	https://github.com/MarlinFirmware/Marlin/pull/21612
21668	https://github.com/MarlinFirmware/Marlin/pull/21668
21680	https://github.com/MarlinFirmware/Marlin/pull/21680
21753	https://github.com/MarlinFirmware/Marlin/pull/21753
21811	https://github.com/MarlinFirmware/Marlin/pull/21811
21835	https://github.com/MarlinFirmware/Marlin/pull/21835
21855	https://github.com/MarlinFirmware/Marlin/pull/21855
21888	https://github.com/MarlinFirmware/Marlin/pull/21888
21899	https://github.com/MarlinFirmware/Marlin/pull/21899
21931	https://github.com/MarlinFirmware/Marlin/pull/21931
21949	https://github.com/MarlinFirmware/Marlin/pull/21949
21962	https://github.com/MarlinFirmware/Marlin/pull/21962
21999	https://github.com/MarlinFirmware/Marlin/pull/21999
22040	https://github.com/MarlinFirmware/Marlin/pull/22040
22052	https://github.com/MarlinFirmware/Marlin/pull/22052
22075	https://github.com/MarlinFirmware/Marlin/pull/22075
22085	https://github.com/MarlinFirmware/Marlin/pull/22085
22109	https://github.com/MarlinFirmware/Marlin/pull/22109
22165	https://github.com/MarlinFirmware/Marlin/pull/22165
22191	https://github.com/MarlinFirmware/Marlin/pull/22191
22279	https://github.com/MarlinFirmware/Marlin/pull/22279
22288	https://github.com/MarlinFirmware/Marlin/pull/22288
22304	https://github.com/MarlinFirmware/Marlin/pull/22304
22354	https://github.com/MarlinFirmware/Marlin/pull/22354
22391	https://github.com/MarlinFirmware/Marlin/pull/22391
22418	https://github.com/MarlinFirmware/Marlin/pull/22418
22425	https://github.com/MarlinFirmware/Marlin/pull/22425
22478	https://github.com/MarlinFirmware/Marlin/pull/22478
22504	https://github.com/MarlinFirmware/Marlin/pull/22504
22537	https://github.com/MarlinFirmware/Marlin/pull/22537
22557	https://github.com/MarlinFirmware/Marlin/pull/22557
22617	https://github.com/MarlinFirmware/Marlin/pull/22617
22657	https://github.com/MarlinFirmware/Marlin/pull/22657
22669	https://github.com/MarlinFirmware/Marlin/pull/22669
22682	https://github.com/MarlinFirmware/Marlin/pull/22682

Pull request	URL
22707	https://github.com/MarlinFirmware/Marlin/pull/22707
22715	https://github.com/MarlinFirmware/Marlin/pull/22715
22760	https://github.com/MarlinFirmware/Marlin/pull/22760
22771	https://github.com/MarlinFirmware/Marlin/pull/22771
22789	https://github.com/MarlinFirmware/Marlin/pull/22789
22790	https://github.com/MarlinFirmware/Marlin/pull/22790
22824	https://github.com/MarlinFirmware/Marlin/pull/22824
22844	https://github.com/MarlinFirmware/Marlin/pull/22844
22880	https://github.com/MarlinFirmware/Marlin/pull/22880
22897	https://github.com/MarlinFirmware/Marlin/pull/22897
22908	https://github.com/MarlinFirmware/Marlin/pull/22908
22916	https://github.com/MarlinFirmware/Marlin/pull/22916
22941	https://github.com/MarlinFirmware/Marlin/pull/22941
22975	https://github.com/MarlinFirmware/Marlin/pull/22975
23033	https://github.com/MarlinFirmware/Marlin/pull/23033
23050	https://github.com/MarlinFirmware/Marlin/pull/23050
23066	https://github.com/MarlinFirmware/Marlin/pull/23066
23080	https://github.com/MarlinFirmware/Marlin/pull/23080
23086	https://github.com/MarlinFirmware/Marlin/pull/23086
23101	https://github.com/MarlinFirmware/Marlin/pull/23101
23112	https://github.com/MarlinFirmware/Marlin/pull/23112
23124	https://github.com/MarlinFirmware/Marlin/pull/23124
23130	https://github.com/MarlinFirmware/Marlin/pull/23130
23158	https://github.com/MarlinFirmware/Marlin/pull/23158
23163	https://github.com/MarlinFirmware/Marlin/pull/23163
23172	https://github.com/MarlinFirmware/Marlin/pull/23172
23184	https://github.com/MarlinFirmware/Marlin/pull/23184
23192	https://github.com/MarlinFirmware/Marlin/pull/23192
23238	https://github.com/MarlinFirmware/Marlin/pull/23238
23322	https://github.com/MarlinFirmware/Marlin/pull/23322
23345	https://github.com/MarlinFirmware/Marlin/pull/23345
23396	https://github.com/MarlinFirmware/Marlin/pull/23396
23400	https://github.com/MarlinFirmware/Marlin/pull/23400
23462	https://github.com/MarlinFirmware/Marlin/pull/23462
23464	https://github.com/MarlinFirmware/Marlin/pull/23464
23489	https://github.com/MarlinFirmware/Marlin/pull/23489
23502	https://github.com/MarlinFirmware/Marlin/pull/23502
23658	https://github.com/MarlinFirmware/Marlin/pull/23658
23704	https://github.com/MarlinFirmware/Marlin/pull/23704
23751	https://github.com/MarlinFirmware/Marlin/pull/23751
23764	https://github.com/MarlinFirmware/Marlin/pull/23764
23768	https://github.com/MarlinFirmware/Marlin/pull/23768
23944	https://github.com/MarlinFirmware/Marlin/pull/23944
23986	https://github.com/MarlinFirmware/Marlin/pull/23986
23987	https://github.com/MarlinFirmware/Marlin/pull/23987
23992	https://github.com/MarlinFirmware/Marlin/pull/23992
24074	https://github.com/MarlinFirmware/Marlin/pull/24074
24130	https://github.com/MarlinFirmware/Marlin/pull/24130
24189	https://github.com/MarlinFirmware/Marlin/pull/24189
24215	https://github.com/MarlinFirmware/Marlin/pull/24215
24229	https://github.com/MarlinFirmware/Marlin/pull/24229
24278	https://github.com/MarlinFirmware/Marlin/pull/24278

Pull request	URL
24366	https://github.com/MarlinFirmware/Marlin/pull/24366
24401	https://github.com/MarlinFirmware/Marlin/pull/24401
24420	https://github.com/MarlinFirmware/Marlin/pull/24420
24427	https://github.com/MarlinFirmware/Marlin/pull/24427
24461	https://github.com/MarlinFirmware/Marlin/pull/24461
24521	https://github.com/MarlinFirmware/Marlin/pull/24521
24528	https://github.com/MarlinFirmware/Marlin/pull/24528
24553	https://github.com/MarlinFirmware/Marlin/pull/24553
24554	https://github.com/MarlinFirmware/Marlin/pull/24554
24610	https://github.com/MarlinFirmware/Marlin/pull/24610
24624	https://github.com/MarlinFirmware/Marlin/pull/24624
24684	https://github.com/MarlinFirmware/Marlin/pull/24684
24722	https://github.com/MarlinFirmware/Marlin/pull/24722
24760	https://github.com/MarlinFirmware/Marlin/pull/24760
24797	https://github.com/MarlinFirmware/Marlin/pull/24797
24928	https://github.com/MarlinFirmware/Marlin/pull/24928
24951	https://github.com/MarlinFirmware/Marlin/pull/24951
24953	https://github.com/MarlinFirmware/Marlin/pull/24953
24995	https://github.com/MarlinFirmware/Marlin/pull/24995
25073	https://github.com/MarlinFirmware/Marlin/pull/25073
25093	https://github.com/MarlinFirmware/Marlin/pull/25093
25143	https://github.com/MarlinFirmware/Marlin/pull/25143
25150	https://github.com/MarlinFirmware/Marlin/pull/25150
25163	https://github.com/MarlinFirmware/Marlin/pull/25163
25214	https://github.com/MarlinFirmware/Marlin/pull/25214
25232	https://github.com/MarlinFirmware/Marlin/pull/25232
25256	https://github.com/MarlinFirmware/Marlin/pull/25256
25268	https://github.com/MarlinFirmware/Marlin/pull/25268
25303	https://github.com/MarlinFirmware/Marlin/pull/25303
25371	https://github.com/MarlinFirmware/Marlin/pull/25371
25387	https://github.com/MarlinFirmware/Marlin/pull/25387
25394	https://github.com/MarlinFirmware/Marlin/pull/25394
25438	https://github.com/MarlinFirmware/Marlin/pull/25438
25548	https://github.com/MarlinFirmware/Marlin/pull/25548
25636	https://github.com/MarlinFirmware/Marlin/pull/25636
25642	https://github.com/MarlinFirmware/Marlin/pull/25642
25667	https://github.com/MarlinFirmware/Marlin/pull/25667
25738	https://github.com/MarlinFirmware/Marlin/pull/25738
25781	https://github.com/MarlinFirmware/Marlin/pull/25781
25791	https://github.com/MarlinFirmware/Marlin/pull/25791
25796	https://github.com/MarlinFirmware/Marlin/pull/25796
25908	https://github.com/MarlinFirmware/Marlin/pull/25908
26011	https://github.com/MarlinFirmware/Marlin/pull/26011
26027	https://github.com/MarlinFirmware/Marlin/pull/26027
26127	https://github.com/MarlinFirmware/Marlin/pull/26127
26142	https://github.com/MarlinFirmware/Marlin/pull/26142
26163	https://github.com/MarlinFirmware/Marlin/pull/26163
26255	https://github.com/MarlinFirmware/Marlin/pull/26255
26267	https://github.com/MarlinFirmware/Marlin/pull/26267
26328	https://github.com/MarlinFirmware/Marlin/pull/26328
26341	https://github.com/MarlinFirmware/Marlin/pull/26341
26344	https://github.com/MarlinFirmware/Marlin/pull/26344

Pull request	URL
26401	https://github.com/MarlinFirmware/Marlin/pull/26401
26441	https://github.com/MarlinFirmware/Marlin/pull/26441
26485	https://github.com/MarlinFirmware/Marlin/pull/26485
26501	https://github.com/MarlinFirmware/Marlin/pull/26501
26539	https://github.com/MarlinFirmware/Marlin/pull/26539
26596	https://github.com/MarlinFirmware/Marlin/pull/26596
26652	https://github.com/MarlinFirmware/Marlin/pull/26652
26696	https://github.com/MarlinFirmware/Marlin/pull/26696
26720	https://github.com/MarlinFirmware/Marlin/pull/26720
26751	https://github.com/MarlinFirmware/Marlin/pull/26751
26762	https://github.com/MarlinFirmware/Marlin/pull/26762
26770	https://github.com/MarlinFirmware/Marlin/pull/26770
26775	https://github.com/MarlinFirmware/Marlin/pull/26775
26793	https://github.com/MarlinFirmware/Marlin/pull/26793
26806	https://github.com/MarlinFirmware/Marlin/pull/26806
26825	https://github.com/MarlinFirmware/Marlin/pull/26825
26892	https://github.com/MarlinFirmware/Marlin/pull/26892
26979	https://github.com/MarlinFirmware/Marlin/pull/26979
27031	https://github.com/MarlinFirmware/Marlin/pull/27031
27072	https://github.com/MarlinFirmware/Marlin/pull/27072
27154	https://github.com/MarlinFirmware/Marlin/pull/27154
27260	https://github.com/MarlinFirmware/Marlin/pull/27260
27275	https://github.com/MarlinFirmware/Marlin/pull/27275
27292	https://github.com/MarlinFirmware/Marlin/pull/27292

Table C.2: All pull request present in this report with their URL.

Pull	NewFeature	BugFix	Enhancement	Refactor	Revert	Rework	Cleanup	Comment	Formatting	Whitespace	Removal	Total
18071	1	39	15	119	0	0	3	5	3	1	0	186
23112	1	11	1	64	0	0	0	1	0	0	0	78
16293	1	13	19	39	1	2	1	0	0	0	0	76
16277	1	10	21	39	2	1	2	0	0	0	0	76
21255	1	3	4	44	1	0	5	6	5	0	0	69
21503	1	4	4	43	0	0	3	5	3	1	0	64
22418	1	15	2	29	2	0	2	4	4	3	0	62
21931	1	4	4	41	0	0	1	2	1	0	0	54
20384	1	4	8	31	0	0	2	5	1	0	0	52
15590	1	6	17	27	0	1	0	0	0	0	0	52
16452	1	4	14	28	1	0	1	0	0	0	1	50
21999	1	2	7	33	0	0	1	2	2	0	0	48
16897	1	9	11	19	1	0	1	0	0	0	0	42
23751	1	8	3	25	0	0	1	2	1	0	0	41
15497	1	9	13	15	0	1	0	0	0	0	0	39
14648	1	2	11	23	1	0	1	0	0	0	0	39
19235	1	2	5	29	0	0	0	0	2	0	0	39
17248	1	4	3	29	0	0	0	0	0	0	0	37
19139	1	2	0	32	0	0	1	1	0	0	0	37
24797	1	6	6	18	0	0	0	3	1	0	0	35
18972	1	4	4	23	0	0	1	1	0	1	0	35
18866	1	3	4	25	0	0	0	2	0	0	0	35
17853	1	3	6	23	0	0	1	0	0	0	0	34
17437	1	3	5	22	1	0	1	0	0	0	0	33
20241	1	2	2	23	0	0	1	2	0	0	0	31
22844	1	5	1	21	0	0	1	1	1	0	0	31
24528	1	5	2	15	0	0	1	3	2	1	0	30
20711	1	1	0	19	0	0	1	6	1	0	0	29
14595	1	1	15	11	0	1	0	0	0	0	0	29
18399	1	3	3	15	0	0	0	6	0	0	0	28
16741	1	4	4	17	0	0	2	0	0	0	0	28
16756	1	5	4	18	0	0	0	0	0	0	0	28
15195	1	4	11	9	0	0	2	0	0	0	0	27
19225	1	3	4	17	1	0	0	1	0	0	0	27
25394	1	3	1	19	0	0	0	1	0	2	0	27
20802	1	3	3	13	0	0	1	4	1	1	0	27
25143	1	3	4	12	0	0	2	0	2	1	0	25
23238	1	6	0	16	0	0	0	0	0	0	0	23
22279	1	2	2	15	1	0	0	0	1	0	0	22
15690	1	4	7	9	0	1	0	0	0	0	0	22
20549	1	2	2	14	0	0	0	1	0	0	0	20
14757	1	2	5	9	1	1	1	0	0	0	0	20
16050	1	1	5	12	0	1	0	0	0	0	0	20
24722	1	1	1	13	0	0	0	3	1	0	0	20
15585	1	3	6	9	0	0	0	0	0	0	0	19
21431	1	2	2	14	0	0	0	0	0	0	0	19
17536	1	0	4	13	0	0	0	0	0	0	0	18
23658	1	3	0	11	0	0	0	1	1	1	0	18

Pull	NewFeature	BugFix	Enhancement	Refactor	Revert	Rework	Cleanup	Comment	Formatting	Whitespace	Removal	Total
14953	1	0	7	9	0	0	1	0	0	0	0	18
22916	1	4	1	12	0	0	0	0	0	0	0	18
23163	1	3	0	13	0	0	0	0	1	0	0	18
14265	1	3	0	13	0	0	0	0	1	0	0	18
24554	1	2	2	12	0	0	0	1	0	0	0	18
20956	1	2	3	9	1	0	0	2	0	0	0	18
16362	1	6	2	9	0	0	0	0	0	0	0	18
23462	1	5	2	6	0	0	0	0	3	0	0	17
19912	1	1	1	12	0	0	0	2	0	0	0	17
14991	1	2	2	12	0	0	0	0	0	0	0	17
22617	1	2	2	10	1	0	0	1	0	0	0	17
15930	1	3	3	7	1	2	0	0	0	0	0	17
22085	1	1	1	12	0	0	0	0	1	0	0	16
20383	1	1	1	9	1	0	1	2	0	0	0	16
15081	1	1	5	7	0	1	0	0	0	0	0	15
25667	1	2	0	12	0	0	0	0	0	0	0	15
22790	1	2	2	9	0	0	1	0	0	0	0	15
21344	1	2	1	8	0	0	0	2	1	0	0	15
17523	1	1	3	8	0	0	1	1	0	0	0	15
22478	1	1	0	13	0	0	0	0	0	0	0	15
23992	1	2	1	8	0	0	0	1	1	0	0	14
20571	1	1	1	8	0	0	0	1	1	1	0	14
20151	1	2	3	8	0	0	0	0	0	0	0	14
23086	1	2	0	10	0	0	0	0	1	0	0	14
20084	1	2	2	8	0	0	0	0	1	0	0	14
19801	1	0	0	10	0	0	1	1	1	0	0	14
17017	1	0	1	12	0	0	0	0	0	0	0	14
21962	1	1	1	8	0	0	0	0	2	0	0	13
25232	1	1	3	8	0	0	0	0	0	0	0	13
21005	1	1	1	10	0	0	0	0	0	0	0	13
15549	1	2	4	4	0	1	0	0	0	0	0	12
17531	1	0	3	7	0	0	1	0	0	0	0	12
19330	1	1	2	7	0	0	0	0	0	0	0	11
23400	1	3	0	5	0	0	0	2	0	0	0	11
20755	1	1	0	6	0	0	1	1	1	0	0	11
22897	1	0	0	9	0	0	0	0	1	0	0	11
16554	1	2	3	5	0	0	0	0	0	0	0	11
20087	1	5	1	4	0	0	0	0	0	0	0	11
21668	1	0	0	8	0	0	1	0	0	0	0	10
23130	1	0	0	7	0	0	0	2	0	0	0	10
23768	1	2	0	6	0	0	0	0	1	0	0	10
19971	1	1	0	7	0	0	0	1	0	0	0	10
18906	1	2	1	5	0	0	0	1	0	0	0	10
25642	1	1	1	5	0	0	0	1	0	0	0	9
21835	1	1	0	7	0	0	0	0	0	0	0	9
22165	1	1	0	7	0	0	0	0	0	0	0	9
22109	1	1	0	4	0	0	0	1	2	0	0	9
17239	1	1	1	4	0	1	1	0	0	0	0	9

Pull	NewFeature	BugFix	Enhancement	Refactor	Revert	Rework	Cleanup	Comment	Formatting	Whitespace	Removal	Total
18177	1	0	0	8	0	0	0	0	0	0	0	9
19901	1	0	0	8	0	0	0	0	0	0	0	9
14924	1	0	1	6	0	0	1	0	0	0	0	9
14961	1	2	2	3	0	0	0	0	0	0	0	8
15209	1	1	1	4	1	0	0	0	0	0	0	8
23080	1	0	0	7	0	0	0	0	0	0	0	8
18737	1	0	0	7	0	0	0	0	0	0	0	8
22191	1	0	0	7	0	0	0	0	0	0	0	8
26011	1	0	1	5	0	0	0	0	1	0	0	8
24420	1	0	0	6	0	0	0	0	0	0	0	7
23192	1	0	0	6	0	0	0	0	0	0	0	7
22391	1	0	0	5	0	0	0	0	1	0	0	7
26127	1	3	1	2	0	0	0	0	0	0	0	7
14619	1	0	4	2	0	0	0	0	0	0	0	7
14309	1	1	2	2	0	1	0	0	0	0	0	7
22669	1	0	0	6	0	0	0	0	0	0	0	7
14251	1	0	0	4	1	0	0	1	0	0	0	7
22354	1	0	0	5	0	0	0	0	1	0	0	7
22040	1	1	0	3	0	0	0	1	1	0	0	7
20099	1	2	1	3	0	0	0	0	0	0	0	7
21753	1	1	0	3	0	0	0	0	1	1	0	7
25214	1	0	0	6	0	0	0	0	0	0	0	7
20940	1	2	0	3	0	0	0	0	0	0	0	6
19674	1	2	0	3	0	0	0	0	0	0	0	6
15245	1	0	0	5	0	0	0	0	0	0	0	6
16068	1	0	1	3	0	1	0	0	0	0	0	6
26267	1	0	0	5	0	0	0	0	0	0	0	6
21387	1	2	0	2	0	0	0	1	0	0	0	6
17884	1	1	1	3	0	0	0	0	0	0	0	6
21403	1	1	0	4	0	0	0	0	0	0	0	6
17462	1	2	1	2	0	0	0	0	0	0	0	6
22715	1	0	0	0	0	0	0	2	3	0	0	6
17560	1	1	1	3	0	0	0	0	0	0	0	6
24521	1	0	0	4	0	0	0	0	0	0	0	5
22760	1	0	0	4	0	0	0	0	0	0	0	5
17222	1	0	0	2	0	1	1	0	0	0	0	5
23184	1	0	1	3	0	0	0	0	0	0	0	5
18735	1	0	0	3	0	0	0	1	0	0	0	5
18039	1	2	0	1	0	0	0	1	0	0	0	5
22052	1	0	0	2	0	0	0	1	1	0	0	5
16641	1	1	0	3	0	0	0	0	0	0	0	5
25548	1	1	0	2	0	0	0	0	0	0	0	4
26328	1	2	1	0	0	0	0	0	0	0	0	4
24189	1	0	0	3	0	0	0	0	0	0	0	4
26344	1	0	1	2	0	0	0	0	0	0	0	4
25438	1	0	1	2	0	0	0	0	0	0	0	4
24624	1	0	0	3	0	0	0	0	0	0	0	4
21534	1	0	0	2	0	0	0	0	1	0	0	4

Pull	NewFeature	BugFix	Enhancement	Refactor	Revert	Rework	Cleanup	Comment	Formatting	Whitespace	Removal	Total
20455	1	0	0	3	0	0	0	0	0	0	0	4
23396	1	2	0	1	0	0	0	0	0	0	0	4
14956	1	0	0	2	0	1	0	0	0	0	0	4
15739	1	0	0	3	0	0	0	0	0	0	0	4
16026	1	0	0	3	0	0	0	0	0	0	0	4
18389	1	1	1	1	0	0	0	0	0	0	0	4
22880	1	0	0	2	0	0	0	0	1	0	0	4
21427	1	0	1	1	0	0	0	1	0	0	0	4
21888	1	0	0	2	0	0	0	0	0	0	0	3
17788	1	0	2	0	0	0	0	0	0	0	0	3
25781	1	2	0	0	0	0	0	0	0	0	0	3
19828	1	0	0	2	0	0	0	0	0	0	0	3
19837	1	0	0	2	0	0	0	0	0	0	0	3
20835	1	0	0	2	0	0	0	0	0	0	0	3
25256	1	0	0	2	0	0	0	0	0	0	0	3
23944	1	1	0	1	0	0	0	0	0	0	0	3
24553	1	0	0	2	0	0	0	0	0	0	0	3
22908	1	0	0	2	0	0	0	0	0	0	0	3
23101	1	0	0	2	0	0	0	0	0	0	0	3
22075	1	1	0	1	0	0	0	0	0	0	0	3
21949	1	0	0	0	0	0	0	1	0	0	0	2
18316	1	0	0	1	0	0	0	0	0	0	0	2
25738	1	0	0	1	0	0	0	0	0	0	0	2
25150	1	0	0	1	0	0	0	0	0	0	0	2
24074	1	0	0	0	0	0	0	0	0	0	1	2
26892	1	0	0	1	0	0	0	0	0	0	0	2
25093	1	0	0	1	0	0	0	0	0	0	0	2
26806	1	0	0	0	0	0	0	0	0	0	0	1
26401	1	0	0	0	0	0	0	0	0	0	0	1
26825	1	0	0	0	0	0	0	0	0	0	0	1
26793	1	0	0	0	0	0	0	0	0	0	0	1
26751	1	0	0	0	0	0	0	0	0	0	0	1
26696	1	0	0	0	0	0	0	0	0	0	0	1
26652	1	0	0	0	0	0	0	0	0	0	0	1
26441	1	0	0	0	0	0	0	0	0	0	0	1
26142	1	0	0	0	0	0	0	0	0	0	0	1
24215	1	0	0	0	0	0	0	0	0	0	0	1
26341	1	0	0	0	0	0	0	0	0	0	0	1
23345	1	0	0	0	0	0	0	0	0	0	0	1
24461	1	0	0	0	0	0	0	0	0	0	0	1
23172	1	0	0	0	0	0	0	0	0	0	0	1
22941	1	0	0	0	0	0	0	0	0	0	0	1
25268	1	0	0	0	0	0	0	0	0	0	0	1
24995	1	0	0	0	0	0	0	0	0	0	0	1
27072	1	0	0	0	0	0	0	0	0	0	0	1

Table C.3: All pull requests together with number of commit tags.

Pull	Lines Changed	Comment Count	Files Changed	Commit Count
14251	244	4	3	7
14265	65	3	6	18
14309	1022	4	15	7
14595	1472	24	17	29
14619	361	3	5	7
14648	1540	38	16	39
14757	382	15	10	20
14924	805	6	7	9
14953	712	6	9	18
14956	106	2	7	4
14961	507	3	5	8
14991	319	6	5	17
15081	1592	11	5	15
15195	1500	26	7	27
15209	5	6	2	8
15245	379	9	3	6
15497	644	10	15	39
15549	338	14	7	12
15585	5355	23	8	19
15590	910	8	21	52
15690	908	6	3	22
15739	13	1	2	4
15930	173	8	3	17
16026	4	2	1	4
16050	137	1	6	20
16068	185	0	3	6
16277	783	38	51	76
16293	975	36	15	76
16362	155	10	7	18
16452	3239	45	30	50
16554	6	0	4	11
16641	122	5	6	5
16741	223	9	15	28
16756	25	20	15	28
16897	227	51	4	42
17017	151	8	9	14
17222	17	6	2	5
17239	146	3	6	9
17248	424	13	16	37
17437	624	2	32	33
17462	46	12	7	6
17523	97	30	5	15
17531	103	50	7	12
17536	5	4	12	18
17560	19	2	2	6
17788	7	0	4	3
17853	795	24	16	34
17884	68	2	6	6
18039	27	8	2	5
18071	16152	20	68	186
18177	73	7	34	9
18316	6	0	3	2

Pull	Lines Changed	Comment Count	Files Changed	Commit Count
18389	283	21	3	4
18399	563	18	15	28
18735	1	0	3	5
18737	27	0	4	8
18866	161	13	5	35
18906	8	7	5	10
18972	235	3	8	35
19139	813	0	11	37
19225	230	11	20	27
19235	66	9	9	39
19330	205	52	16	11
19674	153	37	6	6
19801	467	7	13	14
19828	80	8	4	3
19837	15	0	2	3
19901	61	0	10	9
19912	6	15	27	17
19971	30	4	6	10
20084	233	17	13	14
20087	24	3	5	11
20099	9	4	3	7
20151	9	26	9	14
20241	142	29	10	31
20383	71	11	7	16
20384	421	21	6	52
20455	14	5	2	4
20549	71	0	7	20
20571	267	19	12	14
20711	196	2	3	29
20755	220	24	26	11
20802	409	5	6	27
20835	21	4	5	3
20940	41	19	3	6
20956	166	0	5	18
21005	168	1	12	13
21255	916	13	38	69
21344	267	4	27	15
21387	59	12	3	6
21403	50	8	5	6
21427	16	6	3	4
21431	339	0	18	19
21503	751	51	11	64
21534	359	10	3	4
21668	95	4	9	10
21753	56	6	7	7
21835	280	2	12	9
21888	27	5	5	3
21931	4958	24	30	54
21949	8	0	3	2
21962	15	11	12	13
21999	187	11	7	48
22040	31	2	7	7

Pull	Lines Changed	Comment Count	Files Changed	Commit Count
22052	11	5	3	5
22075	212	2	1	3
22085	319	16	8	16
22109	621	15	11	9
22165	5	2	2	9
22191	26	0	5	8
22279	221	16	16	22
22354	-5	7	6	7
22391	128	1	2	7
22418	1831	1	36	62
22478	101	0	10	15
22617	151	8	17	17
22669	26	0	12	7
22715	80	3	3	6
22760	62	14	10	5
22790	177	5	21	15
22844	127	0	10	31
22880	20	0	8	4
22897	116	0	6	11
22908	17	10	12	3
22916	73	10	14	18
22941	3	1	1	1
23080	320	16	10	8
23086	548	25	21	14
23101	36	5	7	3
23112	3739	48	103	78
23130	7	13	10	10
23163	16	0	13	18
23172	6	1	1	1
23184	229	0	9	5
23192	44	0	6	7
23238	394	31	19	23
23345	120	0	3	1
23396	88	11	8	4
23400	307	7	10	11
23462	746	24	8	17
23658	1298	1	4	18
23751	534	197	13	41
23768	79	1	9	10
23944	-3	14	7	3
23992	139	1	13	14
24074	59	11	2	2
24189	64	2	12	4
24215	8	0	2	1
24420	96	10	7	7
24461	7	5	2	1
24521	54	2	1	5
24528	1229	7	17	30
24553	79	22	11	3
24554	401	3	23	18
24624	54	0	1	4
24722	1308	1	12	20

Pull	Lines Changed	Comment Count	Files Changed	Commit Count
24797	615	232	14	35
24995	11	3	2	1
25093	62	2	7	2
25143	2960	1	10	25
25150	9	1	1	2
25214	301	0	32	7
25232	53	7	13	13
25256	55	12	3	3
25268	21	1	3	1
25394	1722	9	14	27
25438	27	1	5	4
25548	174	1	4	4
25642	486	21	20	9
25667	161	0	21	15
25738	5	5	2	2
25781	156	3	10	3
26011	5536	26	81	8
26127	182	20	11	7
26142	11	0	1	1
26267	41	31	17	6
26328	580	44	11	4
26341	42	2	3	1
26344	-156	10	40	4
26401	18	0	2	1
26441	187	6	8	1
26652	26	4	2	1
26696	15	1	2	1
26751	47	2	8	1
26793	64	17	5	1
26806	14	1	1	1
26825	1	0	1	1
26892	46	6	5	2
27072	10	12	4	1

Table C.4: All pull requests together with lines changed, comment count, files changed and commit count.