CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

A Case Study of Feature Location in an Open Source Embedded System

Master's thesis in Software Engineering and Technology

Wanzi Gu

Hui Shen

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2018

Master's thesis 2018

A Case Study of Feature Location in an Open Source Embedded System

How to identify and locate features with source of information from Github, system artifacts and source code.

WANZI GU
HUI SHEN

CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Department of Computer Science and Engineering
Division of Software Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2018

A Case Study of Feature Location in an Open Source Embedded System
WANZI GU
HUI SHEN

Supervisors: Thorsten Berger and Regina Hebig, Computer Science and Engineering
Examiner: Robert Feldt, Computer Science and Engineering

Master's Thesis 2018
Department of Computer Science and Engineering
Division of Software Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden

Gothenburg, Sweden 2018

Case Study on Feature Location in an Open Sourced Embedded System
WANZI GU
HUI SHEN
Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg

Abstract

In this master thesis, we conducted a case study to look for effective methodologies of feature location in software maintenance. We started with domain study and analysis on the target subject to obtain first hand knowledge and background information about the system. Later we came up with two methodologies that could be systematically applied to the chosen subject to identify and locate features. The methodologies were named as Release Log and Source Code methodology since these were the artifacts where the feature are identified from. The methods were applied to the system, and we found totally 44 features including both mandatory and optional features. Our methods could also be applied to other systems with similar development process. Finally we also measured the characteristics of the features found using some metrics, the results also showed positive correlations between several pairs of measurement metrics.

Keywords: Feature Location Techniques, Manual Feature Location, Marlin, 3D printing, Software Maintenance, Software Product Lines.

Acknowledgements

We want to specially thank our supervisors Thorsten and Regina who had provided us with support and guidance along the way. And we also thank them for directing us to the right path of this thesis.

Wanzi Gu and Hui Shen, Gothenburg, 06 2017

Contents

List of Figures

List of Tables

1

Introduction

As modern technology develops, software has evolved from simple applications to large and complex systems. There are simple mobile alarm Apps and calculators, and there are also intelligent software such as infotainment system in cars, and self-driving cars. Gradually, "software feature" became a term that is widely used in the industry and academic world to describe functionalities. From an user's perspective, it is usually used to talk about the functionalities of the software or how it differs from similar products. From the developer's perspective, especially for those who work with maintenance tasks, it is more relevant to know the location of a certain feature or functionality to be able to perform maintenance tasks. Maintenance tasks usually consists of adding, removing, or modifying code that is related to a feature. To effectively locate the target feature, and to reduce effort in maintenance assignments, it becomes demanding to discover appropriate methods. This process of locating different functionalities in a system is usually referred as feature location [13].

1.1 Feature Location in Industrial Practices

Feature location in software, is the process of identifying an implemented user-observable feature from the source code entities, such as methods or files [4][13]. It is also a fundamental step in software maintenance tasks such as debugging, understanding, and reuse. However, feature location is also necessary in companies that have adopted the Software Product Line (SPL) approach. There are many software companies that have grown in their business, and have developed more than one product. For the companies where products could share a common code base, in order to effectively manage those products, to reduce production cost, increase asset reuse, and decrease time-to-market, more and more companies adopted the software product line approach to extract the common code base and develop adequate variability mechanism [17].

In today's industry, not many companies would invest in changing directly into SPL approach, as the cost of transforming the architecture and the processes requires to support SPL approach is very high. A transitional approach called "forking" has been adopted in the industry, and the reusable assets are managed through an integrated platform [2]. An integrated platform is a project where reusable assets are kept at one place and untouchable by developers of other projects. For the forking approach, in order to create a variant of a product, the developers would

fork the project from the base project and make changes directly in the forked project, leaving the base project untouched. There are many forms of forking such as completely ad-hoc or using feature models or configurations. Tactics are suggested [2] for the developers to manage the disadvantages of the different forms of forking. For companies that require either direct or transitional approaches, the assets have to be reorganized and features need to be identified and annotated in the source code according to the system's feature model. Therefore effective Feature Location Techniques(FLT) are also important for companies that are adopting SPL approach.

1.2 Case Study Subject

There is a wide adopted software product line approach named *clone and own* [2] which is by forking the original product to create a new variability. Marlin is a embedded system of a 3D printer firmware which has been heavily forked since its creation in 2011 [16]. There could be variables of Marlin from the large mount of forks, therefore it is a good example to investigate Marlin and research on feature location for SPL. Another reason is that, in the software industry, there are many embedded system within automotive industry or aerospace industry etc. A survey study conducted by Berger et. al. [3] reported 42 responses that came from different countries and application domains, and among those application domains, there are many embedded systems that have either started transition into SPL approach or had already evolved into a mature SPL. Therefore choosing Marlin as the subject to study would provide a good case study that could indicate the effective methodologies in feature location in embedded system domain.

1.3 Currently Existing FLTs

There are several popular techniques that are used for feature location today. To name a couple, scenario-based probabilistic ranking of events and information-retrieval-based technique that uses latent semantic indexing [12]. More feature location techniques can be found in the survey conducted by Julia Rubin and Marsha Chechik (2013) [15]. These methods had been applied by researchers to evaluate their advantages and disadvantages on different software systems. It would be interesting to apply those techniques and test out their effectiveness in feature location in our thesis, however in the background study we describe in chapter 2, although the techniques are capable of finding features, there are still disadvantages with each technique. The accuracy of finding large amount of complete features is not high. In common sense, one could expect certain amount of codes only belong to one feature. These features we can call them complete features. There are some features may not be called complete feature since they overlap with others features. And the results were not generalized for different systems. Therefore we will investigate why the accuracy is low, and plan to do so through studying Marlin and locate features manually. We perform manual feature locations after thorough domain analysis on the system and its development ecosystem since those steps prepare for manual feature location. The processes and reflections that we gain provide good insights

on how do the developers locate features without the assistance of tools, and how can they achieve high location accuracy with the absence of tools. This is a major contribution of this thesis, and the results may provide guidance to developers of embedded software to follow similar approaches as ours to manually locate features during maintenance tasks or migration projects from traditional development to SPL development.

1.4 Problem Statement

In the industry, most people that are involved in developing the software application will not be the same people who maintain the system. The time it takes to maintain the software is usually much longer than the time it takes to develop the application. Thus good techniques for effective feature location are very important today. New developers would not need to start from scratch to learn about the system and the FLTs could help them to locate directly the feature that they need to work with. There are many techniques for feature location; however, these techniques are rather unique in terms of their input requirements, method used to locate the feature in the source code, and the presentation of the results. There is also very little evidence on high accuracy percentage of those techniques. On the other hand, software systems are very different in many perspectives, thus choosing the right FLT for a maintenance project can be time consuming, and the results are not optimized. For companies that have adopted the SPL approach, the assets were reorganized into features for the creation of feature models and management of variability mechanism. Therefore during the transition from traditional development approach to SPL approach, it is also required to locate features and reorganize them into SPLs.

For most of the studies conducted in FLTs, there is rarely any dataset being created to record the located features. If we can obtain a dataset with correctly located features, it could serve as ground truth. A good dataset recorded and annotated for a system, it is beneficial for evaluating the effectiveness of FLT more thoroughly in the same system, and it could also be used as basis to improve the FLTs. An example of feature annotation of three projects was done by Ji et al. [8]. Since the dataset from the example is not large enough to have better understanding of feature, we will create more dataset from this case study, and we focus our study on how to obtain the dataset. The results serve as inspirations for companies or researchers to create their own dataset or to locate features. We use Marlin as our study subject, and we will retrieve its features manually to summarize our experiences on how to conduct manual feature retrieval. We will also analyze the features located to study the characteristics of features using several metrics.

1.5 Research Goal

One of our main research of this thesis is to understand the processes that are required to locate features. Feature location is still an issue today, and not many

mature FLTs are available or can be applied effectively. Therefore we want to know what does is take for developers to retrieve features even without the assistance of any techniques, and we could summarize the process and elements needed for this action. The result can also used for further comparisons with the underlying mechanisms of other FLTs, or it could be used to improve them. Another main research of this thesis is the study on feature characteristics, we hope it could inform us the different characteristics of the features in Marlin, and whether there is any interesting correlations between the measurements of characteristics and the underlying reasons behind them.

1.6 Research Questions

In order for us to investigate feature location methodology and reach our research goals, we devised three research questions. Below are the research questions we intend to answer:

RQ1: Is there an existing notion of features used in Marlin? This is a preparation question, and we are only interested in finding superficial information on whether the developers use the concept of features in communicating new functionalities, or whether the source code is distinguishable in terms of features with mechanisms such as ifdef or naming conventions.

RQ2: What are the strategies and sources of information for identifying features and their locations manually? After some domain analysis and pre-studies to gain some useful background information regarding Marlin, we retrieve features manually. We summarize the methods that we used and discuss them.

RQ3: What are the characteristics of the found features? Here we statistically summarize and analysis on the features results we found, and we use metrics such as Lines of Feature Code (LoFC), Scattering Degree (SD) and Tangling Degree (TD) [10] to find correlations between those metrics and characteristics of features.

1.7 Main Contributions

After the completion of this thesis, we have several contributions to the research field:

- Introduced two new methods for identifying new features.
- Created a dataset of 44 retrieved features in Marlin with annotations in the source code. These can be found in the repository[1].
- Recorded Fact Sheets for each found feature with information e.g. feature name, description, time consumption, ratios of sources, feature characteristics, etc.

[1] https://github.com/hui8958/Marlin/tree/MarlinFeatureAnnotations

- Developed scripts to automate the process of metric measurement for features' characteristics. The scripts can be found in the repository[2]
- Identified most useful source of information and strategies for identifying and locating features.

1.8 Document Structure

The rest of the thesis is organized into the following structure. Chapter 2 reviews several papers to learn about the current standings on the related researches, and that there is a need for our research. Chapter 3 describes in detail the processes of the various studies that we conducted in order to answer the research questions. The results of the conducted studies are presented in the Chapter 4. Chapter 5 further processes the results produced and analyzes them to extract the significant points. These will be used to answer the research questions. Chapter 6 states the limitations and delimitation of this study. And Chapter 7 concludes the findings of our study and suggests future work.

[2]https://github.com/hui8958/FeatureCharacteristicsMeasurementTool

2

Background

There are many papers research on automatic FLTs, and on the contrary there are very few research on manual FLTs. For the automatic FLTs, there are not only case studies on independent techniques but also combinations of techniques to achieve higher accuracy. However, none of those literature provide adequate result on feature location, and there are only cases studies without gallant affirmations of generalization on other systems. This proves that there is a need for effective methods on feature location, and because of the important role feature location plays for maintenance and software product line, it is essential to start investigating the root problems of low accuracy of the techniques. The research papers on manual FLTs are very few, therefore we want to fill this gap by studying how to manually locate features correctly in this thesis.

Poshyvanyk et al. (2007) conducted a research on studying the effectiveness of three FLTs on three case studies [4]. This paper explained two widely used FLTs in the industry. One is Latent Semantic Index (LSI), and it is a static analysis method that indexes the source code data that contains methods names, class names, or comments etc. Then the user can query the data by typing words in the fashion that are commonly used by developers. Another method is Scenario-Based Probabilistic Ranking, and it is a dynamic analysis method that requires execution of scenarios to find the traces of related methods or functions that are executed. Poshyvanyk et al. (2007) stated that both methods have their advantages and disadvantages, and none of them can provide adequate results if performed separately. This paper had conducted three case studies in Mozilla project's sources codes that were written in C++. The result had shown significant result improvement in the suggested combination of the two methods instead of performing the two methods separately.

Revelle and Poshyvanyk (2009) performed a case study over the effectiveness of different combinations of three main FLTs [14]. The three main techniques are textual, dynamic, and static analysis methods. Textual analysis is the same as static analysis in Poshyvanyk et al.'s (2007) paper. And static analysis makes use of tools to obtain Program Dependency Graphs. By finding one method related to a feature, the other methods can also be traced with the dependency graph. Revelle and Poshyvanyk (y) studied 10 different combinations of the three methods, and concluded that most of the combinations could find 30% of methods that are the actual methods belonged to a feature. They also found that using automatically generated queries for textual analysis works just as well as human formed queries. Lastly, marked traces works better than full traces for dynamic analysis because it

limits the methods to be executed.

Emily Hill, Alberto Bacchelli and other researchers (2013) proposed a rank topology metric to fairly compare FLTs [7]. The comparison has been made based on the likelihood of a developer finding the bug fix locations from a ranked list of results. This topology could be used in our case study for comparing the FLTs that we used to measure their precision, recall and effectiveness. This topology determine the shortest number of hops required to find it in terms of structural topology and the ranked list as metrics. The result set will be the minimum cost of navigating to each fix location. The result has shown that this rank topology has the same relevant results at the exact same ranks with a state-of-the-art IR technique.

Julia Rubin and Marsha Chechik (2013) performed a survey of feature location techniques [15]. In this survey, the authors provide an overview of existing twenty-four feature location techniques for software product line engineering such as: Formal Concept Analysis, Latent Semantic Indexing, Term frequency - inverse document frequency (tf-idf) and Hyper-link Induced Topic Search, etc. The author also described their implementation strategies and exemplify the techniques on a realistic use-case. In the survey they also discussed the properties, strengths and weaknesses of each FLTs. This survey provides guidelines that can be used by us when deciding which feature location technique to choose.

For most of the studies conducted in FLTs, there is no datasets being created to record the features either by the developers or by maintenance personnel. An example of feature annotation added for three projects were done by Ji et al. [2]. With a good dataset created to record and annotate features of a system, it would be beneficial for evaluating the effectiveness of FLT more thoroughly in an entire system, and it could also be used as basis to improve the FLTs. We would conduct studies to manually locate features for Marlin, and use them as ground truth for evaluating different FLTs for future work. One more study conducted by Krüger et al. (2017) [9] is about identifying and mapping features from cloned system. Their a step-wise process to locate and map feature were tested in a case study, and the results were compared to a reference study based on the same case. As a conclusion, there appear to be variations on the results. This also suggests that different methodologies produce different results.

Wang et al. (2013) [18] researched on manual FLTs by performing an exploratory study on feature location process. This study is done by giving six feature location tasks in unfamiliar systems to developers and study how they solve them. They found that the process for manual locating a feature can be divided into three levels: phase, pattern and action, and these levels can help improving manual locating features for junior developers.

3
Methodology

In order to answer the research questions proposed in the Introduction chapter, we planned on performing several activities that would help us to answer the questions. In our research proposal, we planned on conducting a pre-study around Marlin's ecosystem to gain some insight on the development process, and we also planned on studying pull requests that are labelled with "New Feature" on GitHub. These two planned activities evolved into five activities eventually. The first activity that we conducted was domain analysis, here we studied 3D printers from domain perspective. Then we studied the ecosystem of Marlin in its development processes and its key players. These two activities makes of the preparation stage, and it would allow us to gather useful information such as domain entities i.e. hardware components and development flow. Those information would improve our understandings on the system in a higher perspective. It also sets the foundation for us to retrieve features manually. Thus the third activity is manual feature location. This activity provided us results on strategies and source of information that are needed for manual feature location. The fourth activity is the analysis of the previous result from the third activity, and we would use several metrics to measure the characteristics of the features found. For the last activity, we were able to construct a Feature Model with all the information we have after the domain analysis and features that we have retrieved. We conducted all five activities in sequential order, and the result provided in each activity would help us in answering the research question, and it would also provide us useful information for other activities in order to answer other research questions.

3.1 Domain Analysis

For the first activity of this thesis, we aimed at obtaining domain knowledge about 3D printing through domain analysis. The way we approached this is through printer construction. We could learn about what parts does a 3D printer have in order to function, and what mechanisms there are in order to drive a 3D printer. Two 3D printers were purchased for this purpose. The two printers are of different types, and there are a number of different types of printers and printer Firmwares in the 3D printing community. This two printers are of rather common types, one is a Delta printer while the other one is a Cartesian printer, and Marlin firmware supports both types of printers. With domain analysis through working with the printers, it would also allow us to observe a 3D printer in real life and see its entire process during printing. The two printers were bought in parts, and there are instructions

on assembling, we follow them most of the times, but it is also required us to be creative at uncleared points. We constructed both printers, and installed Marlin on the Cartesian printer to test out different optional features. These knowledge would prepare and help us for the in depth study in Marlin's source code later on.

3.1.1 Delta 3D printer construction

We started with constructing the simpler and smaller Delta printer. It has fewer parts and cables to be assembled in comparison to the Cartesian printer. It has a cylinder shaped frame, and the Hotend is held by three plastic pieces above it, see figure 3.1. The plastic pieces are attached to three axes that are part of the frame and movement mechanism, and their movements are controlled by Motors. There are three motors that drive those plastic pieces, and they control the three dimensional movements of the Hotend. In order for the material to be extruded and feed to the Hotend for melting, an Extruder is required, and it sits on top of the printer frame, and the Filament is feed through a plastic pipe to the hotend below to be melted. The motors sits together with the extruder and the mother Board on the top of the printer frame. For Marlin firmware, it supports up to four multiple extruders so far, and it is allows the users to insert four filaments in different colors to each extruder so that the designed object can be printed with different colors. Another crucial mechanism for 3d printing is the Endstops, and they are positional sensors that are located at the top end of the axes in Delta printers. When the plastic pieces move upwards pulling the Hotend, and while the pieces reaches the top of the axes and hits the endstops, it would tell the printer firmware that it has reached the end of the axes, and it is the home position where it could stop all movement and enter rest mode.

It was relatively less cumbersome to assemble and install this printer, however we had issues with the printing bed. This printer has a metal piece as printing bed. There is no sticky medium to allow the newly squeezed hot materials to stick on the bed. We made several trials with different options such as putting papers, tapes or glues etc. And finally it worked quite well with a combination of glue and paper on the bed, and it proved to have good level of stickiness. Other than that, this printer did not have Marlin installed as default firmware, and instead it had Repetier which is also a widely used 3D printing firmware. We downloaded Repetier host to direct the printing from our laptop. We also downloaded some 3D models made available by other hobbyist to print them out. The printed object is quite good, although there are problems with scales, and it usually becomes larger than it was designed to be.

3.1.2 Cartesian 3D printer construction

After the construction of the first printer, and everything worked out well, we moved on to the construction of the Cartesian printer. This printer is bigger and more complex than the Delta printer. It turned out to be a great help for us to have assembled the Delta printer first, and we had some basic understandings on the major hardware components and frame parts from there. The major challenge with

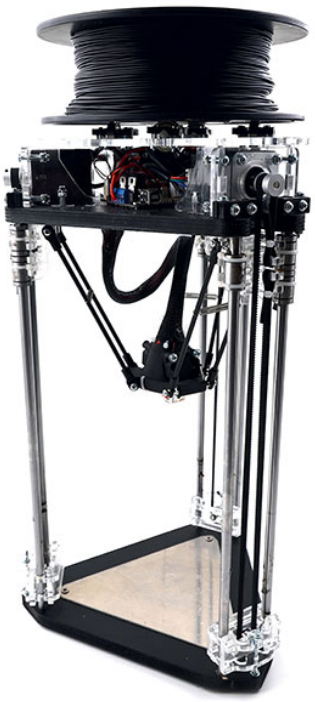


Figure 3.1: Delta 3D Printer

this printer is that, adds to the complexity of hardware and cables, the instruction on electrical connections are very blurry, and some places do not correspond with the parts that we have. There is no clear instructions anywhere. So we eventually had several pictures from finished printers we found on Internet and our unclear instructions, and we tested our way out step by step by connecting all cables to places where we think it should be and plug in the power supply to run the printer.

Instead of a cylinder shape like the Delta printer, this printer is of rectangular shape, and the frame that holds everything together sits in the middle of the Y Axis, see figure 3.2. Y axis lies parallel to the table, and there is a Bed attached to it, so instead of moving the extruder to find its position in Y axis, the bed is moved along the Y axis. The bed on this printer is soldered to power input, so it can be heated up to 60 degrees, and there is a thermistor attached to it to control the temperature to not be overheated. This is an advantage over the other printer, as the newly squeezed material sticks directly to the Heated Bed without the necessity of any sticky mediums. The z axis is driven by two motors on each side of the Frame, and two spiral metal cords are attached to the motors, so as the motors turn, the spiral cords turns as well to produce z axis movements, and they brings up or down the x axis that is attached to the spiral cords.

This setup results in very precise movements along the z axis, and the scale of printed objects are exactly the size as it is designed in the 3D model. Unlike the Delta printer where the printed object can have very imprecise scales. Although we encountered problems such as motors' and endstops' positive and negative cables could wrongly connected or connected to the wrong pins on the board. However

we managed to get everything right in the end, and the printer were able to print beautiful objects. The Chinese good fortune cat displayed on the cover page is one of the objects printed, and the printing quality is quite fair. After the construction, we installed the newest release version of Marlin, and we were able to configure it test out different optional features of this release.

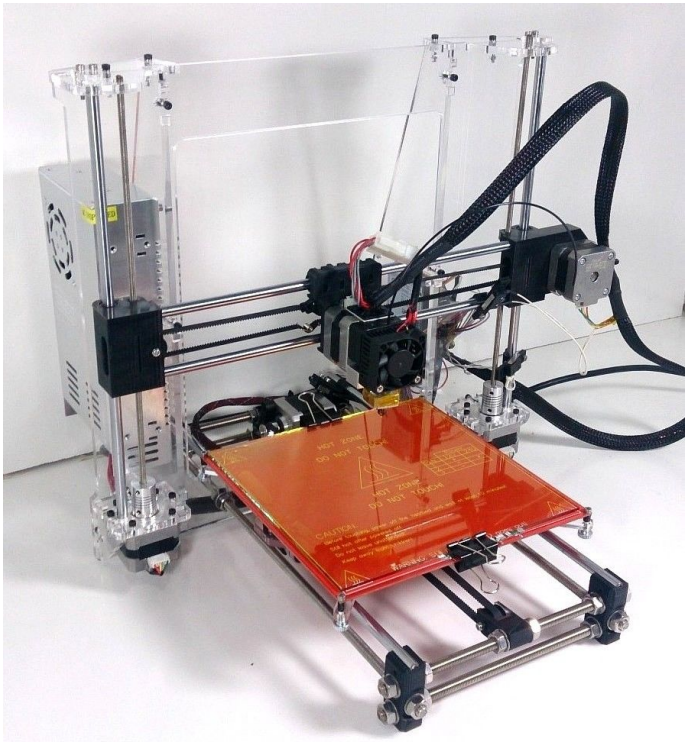


Figure 3.2: Cartesian 3D Printer

3.2 Marlin Ecosystem Pre-study

The first activity helped us to answer partially to the first research question, for details of the answer please see chapter 5. It is also beneficial to conduct a study on Marlin's ecosystem. A pre-study prior to manual retrieval could give us more insight on how features are formed and handled during development, and it would also set the stage for us to learn how to manually retrieve features. Since Marlin's firmware development is hosted on the open source development platform GitHub, its entire development process is publicly visible. The development of the firmware relies on the collaboration of developers that either belong to MarlinFirmware organization, or they are willing to make contributions to Marlin's source code. We would firstly study on the development and release history of the system, and we would secondly study how do the developers collaborate with each other to make contributions, and lastly we would like to know what are the usual development flow of a feature or a bug fix.

For learning about the development history, we used sources such as the Release log provided by the developers, and Wiki (both are available on Github) to learn

about the development of Marlin and its evolution. For results on the leading developers of Marlin, we used GitHub's contribution list and our own metric to analyze the developers' pull request statistics. For studying about Marlin's forks, we used previous study result conducted by other researchers on forks [16] and GitHub's API to retrieve forks using our self defined search criteria. Lastly, we went through the issue trackers and pull requests as well as other documentations to learn about the organization's development culture and process i.e. how are issues raised, and how they are handled and contributed into the mainline. The outcome of the pre-study will be information regarding Marlin's ecosystem with the description of the development history, important forks and contributors, as well as their development cultures and process.

3.3 Feature Location

Manual feature location is a complicated and cumbersome task, it is always a challenge or software maintenance and transformation towards software product line. With earlier activity preparations and Marlin's development tracing possibilities through GitHub, we would be able to retrieve features through development traces such as Pull Requests and Commits. The development of a software usually breaks down into small features and tasks. When developer complete a task or feature, he or she usually create a pull request which contain all the source code with the feature information, therefore it is a good way to trace the code and the modifications belong to a feature. After every iteration of the development process, there will be a release containing all the pull requests and the code modification. In Marlin, we found that there are many releases, and the release log has a very good traceability of the pull requests and code changes, and we choose to study the source code of the latest release candidate RC8. This release contains much more features comparing to earlier releases, and there are about 200 files in this release. During the pre-study, we found that for each release, the new feature developments, enhancements, and bug fixes are mentioned and categorized by the developers into lists of new features, bugs fixes, and enhancements etc 4.2.1. This is a very clever attempt by the Marlin developers to make it easier for users or other developers to learn about the changes made to the new release. The latest Marlin releases such as RC7 and RC8 also links the pull requests made to those changes. This made it easier for us to retrieve the new features developed for the releases. So we started the manual retrieval from those release documentations. After identifying and locating 31 optional features with release log, we moved on to retrieve mandatory features directly through source code which is the most difficult task of this thesis. The reason to retrieve features from source code is for the projects that are not developed with GitHub or other version control system, the available source of information are limited to software artifacts and source code. There are also other possible alternative methods that we could use, but most automatic FLTs are difficult to implement, and the accuracy rates are generally low. There are neither any ground truth that we can use to compare the results found by the automatic FLTs.

3.3.1 Feature Identification and Location through Release Log

We took the release log recorded by the developers from GitHub, and there are different types of development such as new feature, bug fix, quality improvement. For an example of the different types of development, please see our development history result 4.2.1 in pre-study. We extracted only the developments regarding new features. For the extracted list, see appendix A. Feature are firstly identified from this list. We commenced with the new features implemented from the last Marlin release because it contains a lot of new features, and the codes related to them are not changed so much for bug fixes. After the identification of a feature, the extracted list also contains the links to related pull requests, and we were able to find the commits related to the pull request as well. Usually there are more than one commits for a feature, or there are two small features committed together. Therefore we had to look into the source code to finally locate the features from the commits that contains the actual feature development. Sometimes the commit does not only have feature development, there are also other small improvements or bug fixes. Therefore it is very important to not presume that the pull requests or commits contain solely the development of a feature, and one should always have a look into the source code and extract the code segments that are related to the proposed feature implementation.

3.3.2 Feature Identification and Location through Source Code

We found 31 optional features through the Release Log method, and all of those feature are optional. However, there are 20 features that are completely wrapped with ifdef conditionals. Marlin could perform its usual functionalities without those features. Whereas Marlin's mandatory features were implemented much earlier in the project, and those features can only be adjusted but not disabled in the configuration file. Marlin's creation back in 2011 was through merging of two other 3D printer firmware projects: Sprinter and grbl. It was not a feature-by-feature development in the beginning, and it contained sometimes with large amount of source code and features in the very first commits followed by certain fine tuning and small fixes. Those mandatory features had also evolved through years of development and enhancement, and they look quite different as they were in the beginning of the project. So the identification and location of those features are almost impossible to be performed through release log and pull requests. The option that we could see was that the mandatory features have to be retrieved directly through source code. And there will be limited source of information that also requires to be scrutinized since the source code could be difficult for comprehension without domain knowledge and adequate programming skills. This type of feature retrieval is motivated by systems that are not developed on platforms such as Github, and there are also rather limited source of information to start with. The experience and results found could be helpful for people working with those system to locate features for maintenance or software product line migration.

The only available source of information that are useful for this type of retrieval in Marlin are g-code documentation, code comments and source code. This task would have been very difficult if the previous activities were not performed in advance. As it requires extensive domain knowledge on the hardware components as well as printer frame parts. And it also requires good understandings on the overall architecture of the system. Therefore obtaining useful domain knowledge and architecture overview are important strategies for this type of feature retrieval. Feature identification becomes possible after understanding both regarding to the domain knowledge and the source code. A feature was named after its domain purpose such as "Move_To_Destination" and "Command_Handling", and the source codes are also programmed in a way that these features could be identified. Therefore feature identification is performed through a combined understanding from domain knowledge and source code comprehension. Sometimes variable naming that are rather close to the names we gave to the features such as "command_queue" and "Destination[xyz]", and we used them as inspirations to name the features. The optional features we found with release log and pull request provided us insights on some of the features, and it gets easier for us to understand the rest of the source code. We start feature location by studying the most important and central logic of the system, Marlin_Main.cpp file. This file contains more than 10 671 LoC. After that, the rest of the system files that contain minor classes and macro files etc. will also be studied to retrieve more features or parts of the features discovered in Marlin_Main.cpp.

We read through the file and retrieve features by annotating them (see example in figure 3.3). The annotation makes it easier for us to mark the feature, and it also makes it easier for the next activity of analyzing features' characteristics. The results produced from this activity are fact sheets that records the features found and how we found them. All features found are annotated to with syntax of //&begin[feature_name] and //&end[feature_name] [1]. These annotations are dataset that could be used for improving machine learning algorithms and other techniques on feature location. The annotated feature dataset could be found on our GitHub repository. We also created fact sheets and source code annotations for the mandatory features found. Together with the optional feature results produced from release log, the second research question will be answered.

```
463  //&begin[PRINTCOUNTER]
464  // Print Job Timer
465  #if ENABLED(PRINTCOUNTER)
466    PrintCounter print_job_timer = PrintCounter();
467  #else
468    Stopwatch print_job_timer = Stopwatch();
469  #endif
470  //&end[PRINTCOUNTER]
```

Figure 3.3: Annotated source code In Marlin_Main.cpp

3.4 Feature Characteristics

For the third research question, we wanted to study what kinds of characteristics do the retrieved features resemble. We used the source code annotations from the last step, and we collected statistical information regarding the features. Four metrics were defined for this purpose. The four metrics and their definitions are listed below:

LoFC Short for Line of Feature Code. This metric defines how many lines of source code there are for each feature. There are cases where some LoFC belong to different features, meaning that there are overlaps among features.

SD short for Scattering Degree. This metric defines how many locations does each feature spread in the source code. This means every source code blocks of variable declaration and/or method definition located in different places of the system will be counted.

TD short for Tangling Degree. This metric measures how many other feature constants (ifdef conditional statements) appear in a feature which is simpler to be automated for measurement. As tangling degree increases, it could impair comprehension capability [10].

All measurement results are recorded as part of the fact sheets of the features. During the measurement, we started by manually measuring each metric to help us to learn the measurement process. Later we wrote scripts to automate the process, and the results are double checked against the manual measurement results to ensure that the scripts can produce accurate results. The scripts could produce good results for most of the except for Tangling Degree. Due to unclear definition of the feature macros, it escalates the difficulty of obtaining accurate measurement results of the tangling degrees. This results in a rather large deviation from the true results of the tangling degree. It requires manual examination to eliminate the ifdef macros that are not feature expressions. However, in case of large amount of features, the metric measurement process has to be automated. Therefore, we are aware of the problem of this measurement result. The measurement results will be centralized for further analysis and data interpretation in Result 4 and Result Analysis 5 chapters.

3.5 Marlin Feature Model

With the domain knowledge obtained from domain analysis and all the features that we have retrieved and, we built a feature model of Marlin with all the information we have. This is an incremental process where we gradually add and refine the model until the latest feature analysis were completed for the thesis. For companies that are migrating from traditional development approaches into software product lines, it is a fundamental step to create feature models for the system so that product variants can be derived by selecting features from the feature model. After domain analysis, we had certain basic understandings on the hardware components of Marlin, so the hardware components identified would be added to the feature model first. After

manual retrieval, features found other than those we found earlier would also be added to the feature model at appropriate places, and dependencies are handled if there is any. It should be noted that the feature model that we are building is not complete since not all features are retrieved due to time constraints and thesis scope, it should be a continuous process to continuously find more features and add to the feature model.

4

Results

In this chapter, we present our results collected in sequential order by using the methodologies mentioned in the last chapter. For domain analysis, some findings on the study of 3D printing are presented. Then the results regarding the development of Marlin and its ecosystem are listed. The results produced from these two steps had prepared and equipped us with knowledge and information to carry out the task of feature identification and location. Some raw data of fact sheets are presented first, and then statistical summaries were make across all data fact sheets for data interpretation. Feature characteristics are also extracted from the raw data and summarized for interpretation. Lastly the final feature model created is presented and described.

4.1 Domain Analysis

The process of constructing and installing the two printers provided us with domain information regarding 3D printing. Those information are basic functioning parts (See Glossary), key hardware components (See Glossary), and the printer input structure. Those 3D printers takes G codes as input. And in order to obtain G-codes, 3D models are drawn first in .stl (Standard Triangle Language) format, and it specifies the surface geometry of 3D objects [21]. Those files are then processed by slicer programs where it converts the model into G-codes. G-code [19] is a language designed by MIT for people to instruct and control the movements of the machines. 3D firmwares such as Marlin would read the G-codes either through input file or through software connected to the printer, and they would perform movements as they are defined for the RepRap firmware.

After the installation of the newest version of Marlin on the Cartesian printer, we tested out several features, and there is a configuration file where the user can enable and disable a feature through removing the comment lines of a #define macro. For example, there is an optional feature of Marlin named "SWITCHING_EXTRUDER", and since Marlin supports more than one extruder, and some extruders are connected together with a switching servo, and the extruders can be switch from one to another one to print in different colors. There is a macro in the user configuration file defined as "//#define SWITCHING_EXTRUDER". If the user wants to enable this feature, he/she would only need to remove the comment line "//". And the source code that implement this feature are surrounded with conditional inclusions such as "#ifdef". So if the feature is enabled by an user,

the code that are surrounded with the conditional inclusions will be included after compilation, and if the feature is not enabled, those codes will be pruned after compilation.

4.2 Pre-study

For the pre-study, we used available source of information provided by GitHub such as Wiki, release log, commits, Pull Requests, issues, and contributor etc. We read through those sources and extracted relevant information as needed. As described in the Method chapter 3, the results we are interested in having are development history, important forks, key developers, and development process. The following subsections will present these results collected for Marlin's ecosystem.

4.2.1 Development History

For the history of development of releases, we looked at the release log of Marlin (see Appendix A). We found that Marlin had some early stable release versions with rather basic functionalities, those stable versions work pretty well in producing 3D printings, and they are less prone to errors. For the last two years, the developers have been working towards making a new release of version 1.1.0. Since this is a rather major release than the stable versions, the developers have made 8 release candidates so far. Each release candidate contain plenty of new features, quality improvements, and bug fixes etc. For a complete release history their corresponding information, see table 4.1. For a graphical view over the release history and commit sequences related to them see figure 4.1. There were almost 7000 commits for the latest release candidate (RC8) while there were less than 2000 commits for the last stable release. RC8 was released at the end of 2016 which is also the latest release at the time of this thesis study, this version supports much more G-codes, different types of hardware, and configuration possibilities. However it is more prone to errors, and the developers would fix the bugs found after release and push the changes to a branch named RCBugFix. We installed RC8 on our Cartesian printer, and we tested out several new features by enabling them in the configuration files, and it was a very fun experience to watch how the features come out alive during printing. And they worked out in the way they were proposed from the issue trackers, discussions, or G-code documentations.

As stated earlier that improvements made for the stable releases are much less in comparison to the pre-releases of version 1.1.0. Here we list an example of how much development were made for a major release and a minor release. See below:

Recorded improvements for stable release 1.0.1, there were only 9 documented significant changes for stable release 1.0.1:

- Progress bar for character-based LCD displays.
- SD Card folder diving up to 10 levels deep.
- Added support for Melzi electronics.
- Fixed issues with Babystepping.

Table 4.1: Release history of Marlin. The latest RC branch recorded at the end of March 2017 had 6790 commits. The commit difference comparisons with other releases were made based on this record. RC branch contains the latest RC releases. There could also be some commits after a release in the branch just to fix minor issues, therefore there are 3 commits even after RC8. Different from RCBugFix branch where it is used for bug fix and new development, RC branch is only used for releasing new versions. In this table, all Pre-releases are the preparation of the major release 1.1.0. Stable releases are earlier releases prior to 1.1.0. These versions are very small in size, but they are functional and less prone to errors and bugs.

Release History List				
Date	Type	Version	Name	Difference from the latest RC branch (Number of commits)
23 Dec 2016	Pre-release	1.1.0-RC 8	Woozy Wookiee	3
01 Dec 2016	Latest stable release	1.0.2-2	N/A	N/A
21 Aug 2016	Pre-release	1.1.0-RC 7	Jittery Jedi	939
29 Apr 2016	Pre-release	1.1.0-RC 6	Trembling Tusken	2129
13 Apr 2016	Pre-release	1.1.0-RC 5	Sapient Saber	2449
25 Mar 2016	Pre-release	1.1.0-RC 4	Earnest Ewok	2576
01 Dec 2015	Pre-release	1.1.0-RC 3	Gunshy Gungan	2914
03 Oct 2015	Pre-release	1.1.0-RC 2	Ten Ton Tauntaun	2975
19 Sep 2015	Pre-release	1.1.0-RC 1	Wacky Wampa	4956
30 May 2015	Previous stable release	1.0.2-1	N/A	N/A
08 Jan 2015	Previous stable release	1.0.2	N/A	N/A
28 Dec 2014	Previous stable release	1.0.1	N/A	N/A
20 Nov 2011	Previous stable release	1.0.0-beta	N/A	N/A

- Fixed issue with out-of-order command acknowledgement.
- Split up languages.h into separate files.
- Added names for board numbers and boards.h file.
- Support for Toshiba stepper drivers.
- M0 / M1 message string support.

Recorded improvement for pre-release RC8, only summaries on different types of development are listed here since the change details are too much and irrelevant to

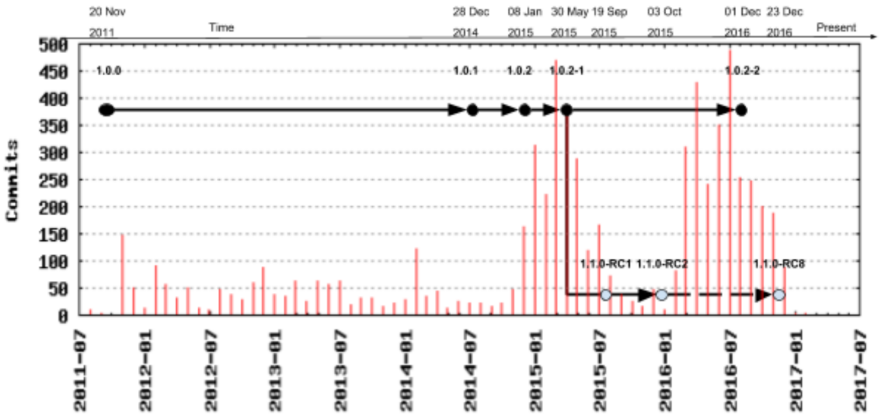


Figure 4.1: Release timeline and commits statistic

be displayed here:

- 19 new features.
- 5 code clean up and documentation.
- 9 Improvements on planner and stepper.
- 32 Improvements on performance and stability.
- 13 Improvements for configuration.
- 25 Homing and Bed Leveling.
- 3 Mesh (manual) Bed Leveling.
- 17 LCD controllers.
- 15 improvements on languages.
- 12 improvements for developers.

4.2.2 Marlin's Forks

Then we studied the forks of Marlin, Marlin has 3941 forks (taken at 9th April 2017). The heavy fork creation in an open source development platform can be seen as integrated platform[16]. It is seen as a transitional approach in adopting SPL. The study conducted by Stanciulescu et al. [16] in 2014 reported that there were 1588 forks in the project since its creation. The number of forks had almost doubled in two and half years of time. At the time of their study, there were 700 active forks. It was reported from their study that 75% of the forks were used for configuring a new version of the Firmwares. 43% of the forks were used for developing new features, and 32% were used for bug fixing. Among the 43% of the forks that served for developing new features, some of them were used by the mainline coding contributors to develop features and integrate into the mainline while the others are not integrated into the mainline due to it is not approved or not intentionally developed for the mainline. There are forks even contain advanced and interesting features that they developed for their own printers. Our pre-study confirms Stanciulescu et al.'s finding in terms that 43% of the 700 active forks are used for developing new features. As stated in Marlin's contribution guidelines, if anyone who wishes to make any contribution, he/she must fork the project and develop in their own fork. There have been 282 developers that have contributed

to Marlin (taken at 9th April 2017), and their forks are used either for developing new features or bug fix. We also found an interesting repository that had modified Marlin quite a lot, and it is called MarlinKimbra, and it is developed upon request for Italian RepRap community. This gives an indication that there are different variants of Marlin among the forks.

We used GitHub's API to list a number of forks according to some search criteria. This is one approach for us to try to find out what Marlin's forks look like. Table 4.2 is retrieved based on whether there are key words such as "printer, 3d, marlin" in the repository name and description, and the repository has to be forked from other repositories, and they are written in language c. Some of those forks are developer forks and there are also others whom have made some changes of their own and not integrated into the mainline. However we did not focus our study on those forks.

Table 4.2: Displays the top most ranked forks with the search criteria we used.

List of important forks			
ID	Full Name	Last Date Pushed	Score
77801444	SkyNet3D/Marlin	2017-03-03	13.38
48942734	TinyBoy3D/Marlin	2016-01-04	12.8
4228806	kikailabs/Marlin	2015-10-21	11.73
79190327	dot-bob/Marlin	2016-12-19	8.65
70604234	gcormier/Marlin	2017-03-03	7.04
73470079	Pipshag/Marlin	2017-03-03	7.04
35349235	thingslab/Marlin	2015-05-09	6.52
83622346	Roxy-3D/Marlin	2017-03-02	6.29
54937948	computergeek1507/Marlin	2016-01-29	5.75

4.2.3 Key Developers

For the study of the key developers of Marlin, we used the contribution rankings provided by GitHub. It shows that there are 283 contributors (recorded at the end of March 2017) that have contributed to Marlin since the start of the project. And among those contributors, 100 of them made contributions to the RC (release candidate) branch. Many contributors that does not appear in the commits in RC may have contributed in other branches such as RCBugFix. In order to identify the most important developers, we came up with three metrics for measurement : "who made the most commits" 4.3, "who made the most additions" 4.4, and "who made the most deletions" 4.5. These three metrics would give some indications to whom have the most interactions with the RC branch in mainline. Tables below to lists the top most contributors to RC branch in each metric. We measured totally 18 developers. And concluding from the tables 4.3, 4.4 and 4.5, there are 7 contributors that appeared on all three tables, they are thinkyhead, AnHardt, ErikZalm, jbrazio, daid, Wackerbarth, boelle. Therefore it could be evident that they are the key developers for Marlin. Here are some information about them.

Table 4.3: Top contributors who have made more than 50 commits to Marlin's RC branch

Rank	Contributor	Commits
1	thinkyhead	2007
2	AnHardt	333
3	ErikZalm	203
4	jbrazio	154
5	esenapaj	142
6	daid	112
7	Wackerbarth	95
8	boelle	91
9	bkubicek	61

Table 4.4: Top contributors who made more than 30 000 additions

Rank	Contributor	Commits
1	Wackerbarth	207023
2	boelle	152836
3	thinkyhead	145784
4	STB3	117521
5	MarikStohn	111277
6	ErikZalm	51217
7	daid	36225
8	domonoky	35531
9	AnHardt	32985

Table 4.5: Top contributors whom made more than 15 000 deletions

Rank	Contributor	Commits
1	jbrazio	526936
2	Wackerbarth	152368
3	boelle	150980
4	thinkyhead	118826
5	daid	37377
6	ErikZalm	29315
7	bkubicek	24205
8	whosawhatsis	18904
9	AnHardt	15540

Thinkyhead had made the most commits into Marlin. He is also placed at 3rd and 4th place for addition and deletion. ThinkyHead's name is Scott Lahteine, and he is a programmer who devoted a lot of his time in creating applications for games, music, and web etc. He got interested in Marlin as he was looking for projects to learn about how to program for electronics in 2013. Thinkyhead's contributions had been the most active since the middle of 2014. And he is undoubtedly the most important developer for Marlin, he developes new features, makes quality

improvements, fixes bugs etc. And he also review other's source code in case of someone is not so confident about their code. He would give suggestions or make improvements on the source code, then he runs Travis CI build tests to test out the code for errors. Lastly, he would also assist in merging the newly developed code into RCBugFix branch. Furthermore, Thinkyhead is a member of the MarlinFirmware Organization.

AnHardt is from Germany, and he also makes constant contributions to the RC branch since late 2014. AnHardt has a repository named MarlinKimbra as stated earlier which makes a different version of Marlin in support of requests from Italy RepRap community.

ErikZalm' real name is Erik van der Zalm, he and boelle (the last key developer according to our metrics) were the two developers who initialized Marlin back in 2011. Erik was most active in the beginning of the project all the way until 2015 where the project was moved to FirmwareMarlin organization. He has another 3D printing project ongoing named ultimaker [22]. Ultimaker is also an organization on GitHub, and this organization builds advanced and high quality 3D printers, and they are available to be purchased.

Jbrazio real name Joao Brazio is located in Portugal. He has two repositories that concerns Marlin. One is forked from Marlin, and the other is from MarlinDocumentation where system documentations such as Marlin's G-code functionality are kept. He makes contributions to both. He is a member of the MarlinFirmware organization. He's contributions to Marlin started since the end of 2015.

Daid belongs to Ultimaker organization. He works on various projects in C++, java, javascripts, C#, PHP etc. His contributions to Marlin lies mainly from the beginning of the project until the end of 2013. There is no contributions from him after that, and his fork is inactive since 2013 dec. However this fork tells about how marlin was until the end of 2013.

Wackerbarth (Richard) is from Texas, he made a lot of contributions to Marlin in 2015, and he had made the most contributions to Marlin in terms of addition, and he is placed at the second place in terms of deletion. His contributions were noticeable from the release of stable version 1.0.2 all the way until the release of RC3, where his contributions were stopped, and his development branches were inactive since then.

Boelle does not have any fork for marlin, and he could have deleted as he took part in initializing and contributing to the project. He opens a lot of issues for Marlin, and also provides comments and reviews for discussions. His contributions lies mainly from the end of 2014 until the end of 2016. He was responsible for releasing the stable versions.

Some recent key players are not mentioned above like (Sebastianv650, esenapaj, Blue-Marlin) since they have joined recently. Although they are very active recently in Marlin, their contributions to Marlin as a whole is not weighted as much as the

above mentioned 7. However, all of those top developers are quite active in terms of making contributions to the source code and involvement in issue discussions. We also made pull request analysis over the entire project to see which developers make the most pull requests. Since pull request are directly related to the changes made to source code, it is also interesting to find out whether those people are the same as what we found using GitHub's contributor list. As can be seen from figure 4.2, "thinkyhead" has made 33.9% of the total pull requests, followed by "essenapaj" and "AnHardt", with approximately 6% each. There are 54% of pull requests are from "jbrazuo", "Kaibob2" and the rest of developers.

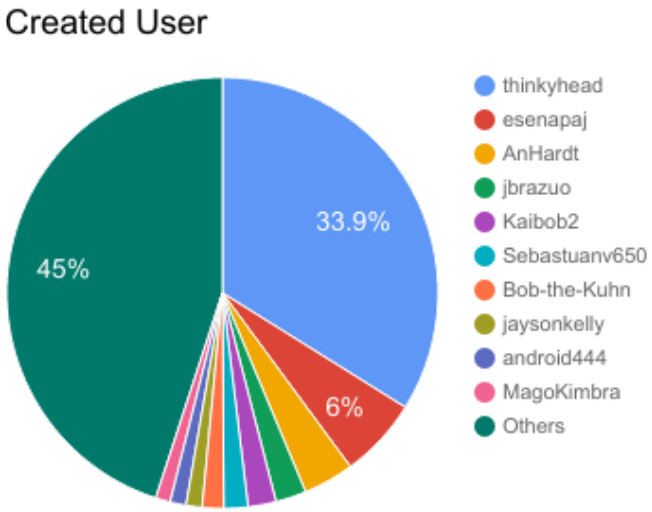


Figure 4.2: Percentage of pull requests created by the developers

4.2.4 Development Culture and Process

Marlin's development culture is open sourced, and anybody who are willing to contribute, give suggestions, raise issues or has opinions are encouraged to do so. Marlin-Firmware Organization has 5 members, and they are Philip Schell, Jochen Groppe, Joao Brazio, Panayiotis Savva, and Scott Lahteine. Besides them, anyone outside of the organization may also contribute, and they are tagged as Contributors on discussions if they made commits to any branches. In order for any new developers to join and contribute, he/she has to have his/her own fork. In general, there's a lot of active discussions on Issues and Pull Requests in Marlin. There are discussions about new features, coding standards, solutions, and suggestion on improvement etc. People or organizations with new ideas for Marlin would propose it on the issue tracker. Those ideas could be either from themselves or from G-codes defined by RepRap Wiki [19]. Any suggestions for quality improvement or bug fixes are posted on the issue tracker as well. Anyone who are interested in resolving the issue

proposed could assign themselves on the task. However there could be competitions sometimes. For example, some developer assigns him/herself for the task, and he/she announces it on the issue tracker, nonetheless another developer also sets forward in developing their own solution. This action leads to conflict, yet they find a way to resolve the issue. They would compare the solutions and make the best out of them.

A typical development flow in Marlin would usually start with an issue being raised by users, developers or organizations. After the issue is proposed, someone would either go straight up into development or there are unclear points or technical problems that lead to lengthy debate and discussions. With the issues cleared up, the task is self assigned by anyone who is interested, and he/she becomes responsible for the development. Task assignment in Marlin are usually self assigned. In order to make contributions to source code, the developer needs to fork the project into his own repositories, and this is demanded on the Wiki page [5] if one wants to become a developer for Marlin. Thus development usually occur in the developer's own fork. After the new code finished, the developer should create a pull request which contains one or multiple commits from the original repository. Then some key developers such as thinkyhead would help out to review the code, for bugs or bad contribution formats. When the key developers thinks the source code is fine, he would create another branch in mainline with the name of the development i.e. "Support_G20_Feature" to merge the changes into RCBugFix branch. Sometimes the new code will be tested with Travis CI builds to avoid certain errors. The code would also be tested by people who are willing to help out and test the code on their own machines. They install the changes onto their printers and run the code. In case of errors, they would be reported for later improvements. When RCBugFix's source code has good enough new features and bug fixes, the new code would be moved to the RC branch and get released.

Pull request is a specific feature provided by GitHub for the developers to discuss and test out the changes made before integrating the new code into the mainline. Those pull requests are usually made for different purposes. For the development of Marlin, each pull request had been labeled and divided into different categories. To name a few, there are "PR:Bugfix", "PR:Coding Standards", "PR:Configurations", and "PR:New Feature". The pull requests labeled with "PR:Bugfix" are used for fixing bugs. The pull requests labeled with "PR:Coding Standards" are used for unifying coding standards. The pull requests labeled with "PR:Configurations" are used for adding or changing the configuration files. The pull requests labeled with "PR:New Feature" are used for new feature development. The bar graph in figure 4.3 shows the lifetimes between the creations and endings of the pull requests.

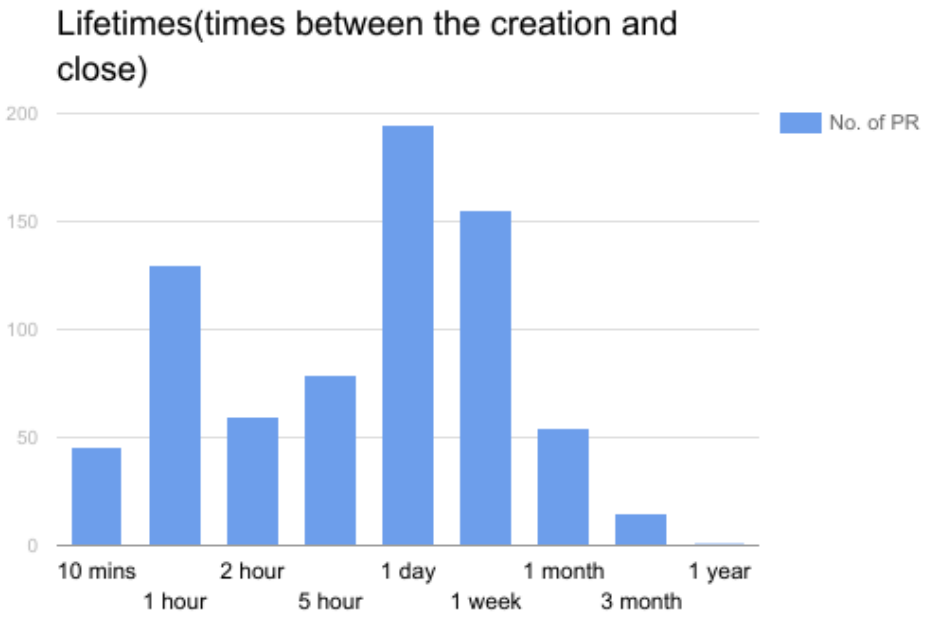


Figure 4.3: The lifetimes between the creations and endings of the pull requests.

4.3 Feature Identification and Location through Release Log

By applying the method of using release log, pull requests, and commits as described in Method chapter 3.3.1, we found totally 26 features. We identified and retrieved 4 to 5 features each day, and this process lasted for 5 days. It took short amount of time to identify a feature from release log, usually only one minute. However, the time it requires to locate the feature by browsing through varies pull requests and commits takes longer time. And depending on the size and complexity of a feature, it would take either less than half an hour or more than one hour to locate the feature. If a feature is small in size, ex. *Park Nozzle*, and there is only one pull request with a couple of commits linked to this feature, the time it takes could be 20 minutes. If a feature is large in size, and it had taken many pull requests to refine the source code, the time takes to locate all feature was more than an hour. The location process also requires reading and understanding the source code to extract the feature related code from the rest. Sometimes there are other changes such as code style improvement and bug fixing in the same commit, then these need to be separated from the feature codes. Therefore due to those issues, the overall time it takes to identify and locate a feature varies depending on feature size, complexity and involvement with other changes.

For each feature found, we used fact sheets to record them. Each fact sheet contains several key points as followed:

Feature Name The name of the feature. It is usually identified through the description in the release log or taken from its feature definition from macros. The feature names are usually named to be easily understandable from domain perspective.

Feature Name in Annotation The name of the feature in the source code annotations such as //&begin[PRINTCOUNTER]. Sometimes the names can be slightly different from the name we gave for the Feature Name. These names are usually named same as the macro definition of the feature.

Source of Information lists the source of information used to identify and locate the feature.

Strategy used describes the strategy used for identifying and locating the feature.

Feature Release Version States in which version was this feature released. This information was useful for us to know if the changes in the commits do not correspond with our annotation source code of RC8. For those cases, we would know that the feature was implemented in a much earlier version. And it requires more time to find the right feature locations.

Time takes to identify and locate the feature Records the time took for finding the feature.

Time takes to understand the implementation Records the time took to comprehend the feature. As feature size and complexity goes up, this number also goes up.

Feature Description Describes the functionality of the feature, and what it is used for. There could be videos and graphs in case if it is needed to help understand the feature.

Feature Information and Statistics Pull Request number, name, date merged are recorded, and some statistical information directly shown from GitHub such as number of commits, files, lines added, and lines deleted. These statistical numbers are not accurate as we only had taken them directly from GitHub, and they do not represent the real number of files and commits after the features are located.

Feature characteristics Feature's characteristics can be evaluated using the four metrics defined in method Chapter 3.4, here we only record the value for each metric per feature.

Ratios of Sources (%) The percentage of every sources.

Pull Request Links Lists the links to all directly related pull requests to the feature. These pull requests either are taken from the release log or other referenced pull requests within other found pull requests.

Historical Relevant Data Records information such as when was the issue raised, who raised it, links to the issue tracker, and other related pull requests such as bug fixing if there is any. All these information helped us to understand the whole development process of each feature, and it also help us to correctly locate the feature.

All features found through Release Log method are recorded with most of the listed points. Sometimes, there is no other relevant historical data to record, and thus it is omitted. After the recording of the fact sheets, we also annotated the source code

using annotation marks mentioned in Method chapter 3.3.2. Here we only present one of the optional features we found, and that is *PRINTCOUNTER* (see figure 4.5) and its annotated source code, see figure 4.4. This feature is not completely wrapped by ifdef since some codes (e.g. one line of code of menu display) are not as important, and they will not effect other source codes' usual functionalities. The other 25 optional features can be found in our data repository[1] due to large amount. All annotated source code can be found in our GitHub repository. Each feature can be found by searching for the name stated in "Feature Name in Annotation" with annotation mark such as //&begin[PRINTCOUNTER].

```
5348  inline void gcode_M76() { print_job_timer.pause(); }
5349  //&begin[PRINTCOUNTER]
5350  /**
5351   * M77: Stop print timer
5352   */
5353  inline void gcode_M77() { print_job_timer.stop(); }
5354
5355  #if ENABLED(PRINTCOUNTER)
5356    /**
5357     * M78: Show print statistics
5358     */
5359    inline void gcode_M78() {
5360      // "M78 S78" will reset the statistics
5361      if (code_seen('S') && code_value_int() == 78)
5362        print_job_timer.initStats();
5363      else
5364        print_job_timer.showStats();
5365    }
5366  #endif
5367  //&end[PRINTCOUNTER]
```

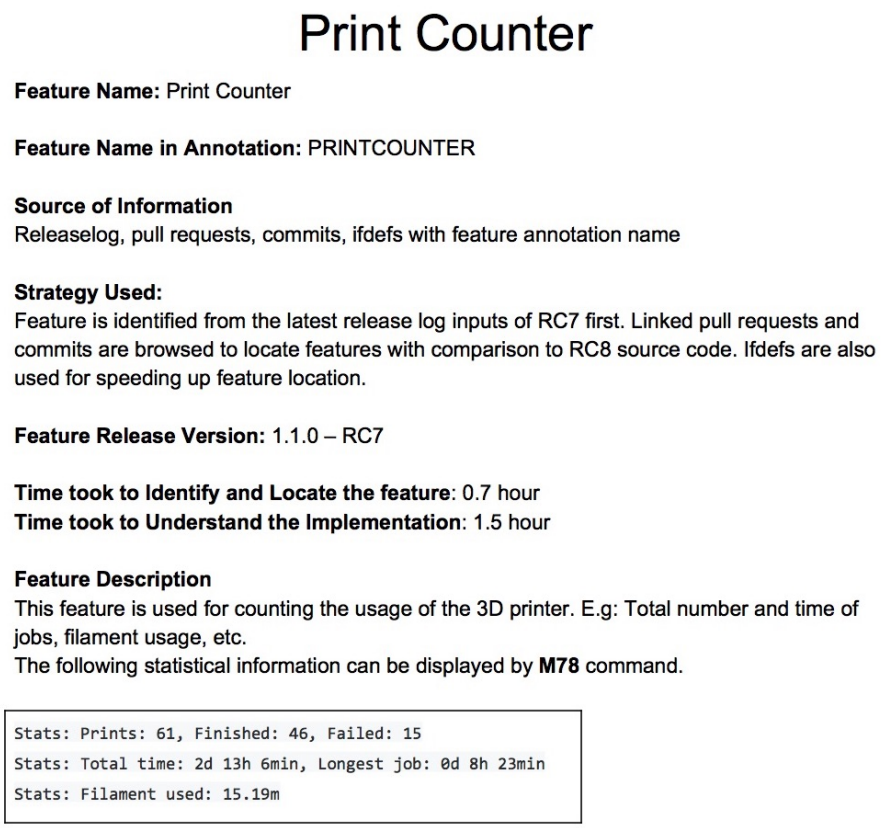
Figure 4.4: Annotated code for PRINTCOUNTER in Marlin_main.cpp

To sum up the optional features found, and to observe for any significance of the data , we created the table B.1 to examine the data across. During the process of retrieving the features through Github, we also collected lists on all pull requests and commits' names and identification numbers for each feature, see table 4.6 and 4.7 for example. These data can also be used in the future to find quicker ways to retrieving pull requests and commits using machine learning algorithms.

Table 4.6: Example List of Feature Pull Request

PR No.	Title
#3676	Advance extrusion algorithm – LIN_ADVANCE
#4035	Patch LIN_ADVANCE to use code_value_float
#4040	Follow-up the PR #3676(Advance extrusion algorithm – LIN_ADVANCE)
#4126	Patch LIN_ADVANCE timing issue

[1]https://github.com/hui8958/Marlin/tree/MarlinFeatureAnnotations/FeatureDocuments/FeatureFactSheets



Print Counter

Feature Name: Print Counter

Feature Name in Annotation: PRINTCOUNTER

Source of Information
Releaselog, pull requests, commits, ifdefs with feature annotation name

Strategy Used:
Feature is identified from the latest release log inputs of RC7 first. Linked pull requests and commits are browsed to locate features with comparison to RC8 source code. Ifdefs are also used for speeding up feature location.

Feature Release Version: 1.1.0 – RC7

Time took to Identify and Locate the feature: 0.7 hour
Time took to Understand the Implementation: 1.5 hour

Feature Description
This feature is used for counting the usage of the 3D printer. E.g: Total number and time of jobs, filament usage, etc.
The following statistical information can be displayed by **M78** command.

```
Stats: Prints: 61, Finished: 46, Failed: 15
Stats: Total time: 2d 13h 6min, Longest job: 0d 8h 23min
Stats: Filament used: 15.19m
```

Figure 4.5: Example of feature fact sheet

Table 4.7: Example List of Feature Commits

Commit No.	Content
6d62a4ffc8010ca56f5f438a9da96e781ee65099	Patch LIN_ADVANCE for style and forward-compatibility
fb8e880734bb099b80b031ee2b876e628a50135e	Add LIN_ADVANCE
506d78b2f89bb8e50b9e54fbe51266653299abba	Run the advance_isr faster instead of doing multiple e-steps per interrupt
aad9c0ed8d6cb61c701aae4cd94da8d9619c4dd9	Apply updated ISR timing code
45b701d38c02892b5d1f233c22142aa578df7c3d	Travis test for LIN_ADVANCE
2b340f5acb605c9c54c880067742f6a34dec7136	patch LIN_ADVANCE to use code_value_float
741cda0e476823f07c88153e30eb536568327083	Follow-up the PR #3676(Advance extrusion algorithm – LIN_ADVANCE)
0c5192b288ea66c66938f36b002538df0f95853a	Patch LIN_ADVANCE timing issue Also the extruder stepper ISR has to keep an eye on step_loops count.

4.4 Feature Identification and Location through Source Code

Through release log, we could only identify and trace optional features. For the mandatory features and features that are not recorded in release log, it requires other methods. Mandatory features were developed very early on, and there were many large amount of source code from two other projects. It made the task of retrieving mandatory features from release log or pull requests very difficult. We have to use completely different source of information and strategy to retrieve those mandatory features. Mandatory features do not usually have ifdefs surrounded them since they are the most basic functionalities that have to be there in order for

a system to work. And in Marlin, those features are not named in any way. So in order to identify those features, it has to be started from understanding the domain and the source code as base, and one has to be creative in defining and separating the right feature from the rest of the source code. Starting from the main and most important source file Marlin_main.cpp, we identified 13 mandatory features, some of the features are not completely located. The rest of those incomplete features as well as other features were located in the rest of the files. For those features identified and located, we also created fact sheets to record information about them.

For each feature found, we also used fact sheets to record them. Each fact sheet contains several key points as followed:

Feature Name states the name of the feature. The source code annotations for the feature are the same with the feature name with the only exception of having underscores instead of spaces.

Source of Information lists the source of information used to identify and locate the feature.

Strategy used describes the strategy used for identifying and locating the feature.

Time takes to identify and locate the feature Records the time took for finding the feature.

Time takes to understand the implementation Records the time it takes to comprehend the feature. As feature size and complexity goes up, this number also goes up.

Feature Description Describes the functionality of the feature, and what it is used for.

Feature characteristic Statistics Feature's characteristics can be evaluated using the four metrics defined in method Chapter 3.4, here we only record the value for each metric per feature.

Ratios of Sources (%) The percentage of every sources.

All 18 mandatory features are recorded with most of the listed points. After the recording of the fact sheets, we also annotated the source code using annotation marks mentioned in Method chapter 3.3.2. Here we only present one of the mandatory features we found Endstop (see figure 4.6) and its annotated source code (see figure 4.7). The other 17 mandatory features can be found in our data repository[2] due to large amount. Each feature can be found by searching for the name stated in "Feature Name" with annotation mark such as //&begin[Endstop].

4.5 Feature Characteristics and Feature Model

The fact sheets collected are raw data collected per feature, and they are useful for understanding single feature's location process and its characteristics. Nonetheless, we are interested in generalizing the feature location methodology and characteristics of features to be used for feature location in other systems. And to be able to achieve

[2]https://github.com/hui8958/Marlin/tree/MarlinFeatureAnnotations/FeatureDocuments/FeatureFactSheets

Endstop

Feature Name: Endstop

Feature Name in Annotation: Endstop

Source of Information
Domain knowledge, Code comment, Source Code, G-code documentation

Strategy Used:
Through 3D printer construction, we learned that there are endstop hardwares for controlling the minimum and maximum points of axes. Then we started from the largest and most central logic file MarlinMain.cpp, and browsing through the code comment and source code for whether the comments, variables and method names contain keywords such as “endstop”. Sometimes, both method comments and variables within a method contain a lot of endstops, however the method name is called ”gcode M120” which has nothing to do with endstop. According to G-code documentation, there are certain g-codes that are directly related to settings for the endstops.

Feature Description: This feature is mandatory for Marlin, and it is responsible for recording end points or reference points of all axes. The reference points are then reported and stored in Marlin after it is triggered, and those reference points are used for calculating relative distance to be traveled on each axis. The endstops also work as an indicator for terminating motion on an axis if the endstop is hit on the same axis.

Time spent: 15 Hours to go through 10 000 lines of code and handled feature retrieval for different features.

Feature Characteristics:

Feature Name	LoFC	NoFL	TD	SD	Completely Wrapped by ifdef
Board	198	14	17	0	No

Figure 4.6: Endstop fact sheet

```
1823  //&begin[Endstop]
1824  static void setup_for_endstop_or_probe_move() {
1825    #if ENABLED(DEBUG_LEVELING_FEATURE)
1826      if (DEBUGGING(LEVELING)) DEBUG_POS("setup_for_endstop_or_probe_move", current_position);
1827    #endif
1828    saved_feedrate_mm_s = feedrate_mm_s;
1829    saved_feedrate_percentage = feedrate_percentage;
1830    feedrate_percentage = 100;
1831    refresh_cmd_timeout();
1832  }
1833
1834  static void clean_up_after_endstop_or_probe_move() {
1835    #if ENABLED(DEBUG_LEVELING_FEATURE)
1836      if (DEBUGGING(LEVELING)) DEBUG_POS("clean_up_after_endstop_or_probe_move", current_position);
1837    #endif
1838    feedrate_mm_s = saved_feedrate_mm_s;
1839    feedrate_percentage = saved_feedrate_percentage;
1840    refresh_cmd_timeout();
1841  }
1842  //&end[Endstop]
```

Figure 4.7: Annotated code of Endstop in Marlin_main.cpp

that goal, we centralized the data into one place and performed analysis on them. See the table B.1 in Appendix B with the reorganized data. Metric measurement results for feature characteristics can be found in the last three columns. And there are totally 44 features, among them there are 31 optional features and 13

mandatory features. The latest version of the feature model that we built can be seen in Appendix C.

5

Result Analysis and Discussions

In this chapter we conclude and discuss the results presented from Result 4 chapter to answer the research question. We discuss the methods that we have used, and how they are useful and relevant for other activities. We also listed all source of information used during feature retrieval using the two approaches, and the data are analyzed to see how many features were found with each source of information. We also summarized the entire process of feature identification and location through an activity diagram. Finally, the results on feature characteristics were analyzed to see the characteristics of Marlin's features.

5.1 Domain Analysis Reflection

With the results obtained from constructing the two printers, it lightens the load for us to understand the various concepts and logic within the source codes. For example, there is a file named as endstops.cpp, and after installing the printer, we know directly what it is and their functionality during printing. We had a rough idea on what that class could possibly contain, and it stands out clearly for us that this file belongs to part of the Endstop feature we found later. As we understand after domain analysis about the inputs of the printers and how different parts and hardware connections collaborates together to print, we were able to obtain a high level comprehension on the system, and it makes it easier to follow the program logic. The discovery of configuration file where users can configure features through enabling and disabling macros gives an indication that there is a notion of feature in Marlin. And there are a lot of optional features presented within the user configuration files, and this allows the users to select the features that they are willing to have for their own printers.

5.2 Pre-study

In pre-study, the development history result showed that there were more development effort for version 1.1.0 than version 1.0.2. This is due to the fact that 1.1.0 is a major release, and the developers invested a lot of effort in developing new functionalities, improving existing functionality, and improving source code qualities. Earlier stable releases were mainly focused on releasing a working software. This is also lead us in choosing the latest RC as our source of study for feature location later on. The study of forks gives an indication that there are marlin variants such as Marlin developed for Italian RepRap Community. Our study was not extensive

due to the fact that we wanted to focus our study on feature location, therefore we did not have any conclusive result over the forks of Marlin. It would be interesting to have an overview of the forks of Marlin similar to Stanciulescu et al.'s [16] study.

The two key developer study provided slightly different results on whom makes the most contribution. However the pull request analysis 4.2 confirms the study by contribution Table 4.3, 4.4 and 4.5 with overlapping results such as Thinkyhead and AnHardt as the top most developer and contributor to Marlin. The result on pull request life span 4.3 shows very clear that most of the pull requests are closed between 1 day and 1 week, and these numbers are followed by those pull request which closed in 10 minutes. There are only a few of pull requests' lifetimes are more than 1 month. we can conclude that the hosts of Marlin response quickly and they have high efficiency on processing each pull request.

5.2.1 Insights and Reflections

The pre-study gave us insights on the ecosystem around Marlin, we had much better understanding on how do the developers collaborate and how do they make contributions to the system. During our investigation on the development process, we also looked at the Pull Requests and the committed codes that are linked to them. So we had a closer to into some of the commits on feature development. We read the comments for the commit as well as the codes to see how much code there is for a feature, and how they are implemented. This prepared for our next step on manual feature retrieval. We were able to plan our approach to manually retrieve features. As we already found that there is a notion of "feature" in marlin concerning feature selection in configuration file and their implementation within the source code, "feature" is also named during issue proposal and pull requests. As the result found on different categories of pull request, that there is a category named "new feature". Whenever an issue is raised concerning a new feature, the developers would assign a tag "New Feature" to it to for better organization. The same tag is also assigned to the pull request when the feature development is completed. This way, one can use the tag to filter out only the issues and pull requests related to new features. It is good for organization of the system, and this also makes it easy for us to find features and the commits that are linked to the new feature development. We also found that Marlin uses conditional statements to define optional features in the source code.

5.3 Feature Identification and Location through Release Log

For the features found through this method, some implemented new features such as added G-codes for new printer movements, and some implemented alternative methods for existing features. The reason that we only found optional features with this method was that, we used the latest release at the time, and the available release log recorded with traceable new feature implementations only for the latest

releases. The developers focus on system improvements and enhancements for those releases i.e. adding newly requested features for a more customized and applicable firmware for various types of machines and hardware. The mandatory features and some early on developed optional features were either untraceable from the release log or had too many new changes added on top so that they became difficult to be located through pull requests and commits. Therefore we used another approach to retrieve those features.

Among the located optional features, some were complex and difficult to implement, it took more than one pull request and by different developers to ensure the codes were functional. Sometimes, the developers would push small code fragments into the RCBugFix branch. And other developers would start on discussions to resolve controversial issues. The rest of the development could be taken over by anyone that can provide adequate feature implementation. In those type of developments, more than one pull requests were required to complete the same feature development, and the pull requests were all referenced to the first pull request of the feature to be traceable. The time required for locating this type of feature among all the pull request would increase as one need to view all pull requests to allocate the feature from unrelated source code. As a conclusion of feature identification and location through release log approach, we had a systematic process to identify and locate features because of the traceable references from release log to pull requests and commits. Although this was a relative simple process to find features in comparison to manual allocation through source code approach, it still demanded much effort to prune source codes that were unrelated to the feature such as small fixes.

5.3.1 Approach Generalization

For systems that are open sourced and hosted on development platforms such as GitHub, a similar approach as our release log method could be applied to retrieve features. If there is a release log like Marlin where all related pull requests to features are documented and referenced, then one can follow the exact approach as we did, and as mentioned earlier, some more effort should be spent on finding the codes that are implemented only for the feature. If there is no release log to document the development of features, one can turn to the pull request list to find features. Marlin's pull request have "New Feature" tagged to the pull request if it is a feature development, and if the system also have a similar management approach to their development, one can use those tags to sort out the pull requests that are only related to the development of new features, and one can continue feature retrieval from there. Assuming that even such managerial approach of managing the pull requests is not available in the system, one has to conduct feature retrieval with more effort. Each pull request needs to be scanned, and the discussions and comments need to be read to understand whether the development is regarding a feature. This is the approach to manually retrieve features with minimum amount of help provided by the developers as there is no pull request management and release documentation. Granted that the developers of the system are very helpful and provide good pull request management and release documentation, it will reduce the amount of effort required for finding features, and it lightens the load for a new

developer to find the feature that they want to work on.

For manual identification with development tracking on GitHub, keyword could also be used in identifying whether a commit messages or a pull request is related to feature implementation. Some examples of those keywords could be “Add”, “Support”, ”implement”, “Feature” and “Extend”. If a commit message or pull request contains those words, it could be considered as feature related. Most of the developers would create a new branch in their own fork to continue feature development. It is a common behavior that the developers name the branch with the feature name. For example, if a developer develops a new feature called Case Light, he/she would name the branch to be case_light_feature. To summarize, if a commit message or a pull request contains keyword such as “Add”, “Support”, ”implement”, “Feature” and “Extend”, and it does not contains any keywords such as ”corrected”, “fix”, “bugfix”, “bug”, “fixed” and “replace”, it has a high probability that it is feature related.

5.4 Feature Identification and Location through Source Code

Without development traceability, finding of features becomes a cumbersome and difficult task as it takes time and effort to understand a system from point zero. Therefore with all the studies we have performed early on have reduced the level of difficulty gradually for the application of this approach. With the first step of domain analysis, we summarized the main hardware components that we learned during construction. The figure 5.1 shows the hardware components of the Sintron 3D printer. This gave us a high level understanding of the system. Looking into the source code of the system, one is required to understand the architecture of the system to be able to know where one can find certain features. And it is even more helpful if you can relate those features back to the hardware components. The main strategy for this approach is to obtain domain knowledge and architectural overview of the system as background knowledge. Then one can start with the core of the system, and gradually move on to smaller and less important classes and files to identify and locate features.

5.4.1 Insights and Reflections

For feature identification and location without development tracking on development platforms such as GitHub, correct and useful keywords identification is crucial. Keywords within comments, variable names (either outside or inside a method), and method names are very essential in determining whether the source code is related to a feature in manual identification. During reverse engineering, the developers would need to read comments, variable names, method names, and source code logic to be able to understand the function of a piece of code. This would help them on determining whether a piece of code belongs to a feature. There are easy and simple cases of identifying a feature such as the Endstop feature. Almost all comments,

Figure 5.1: Cartesian 3D printer Hardware Components

variable names, and method names contain "endstop" keyword. Therefore it was quite straightforward for one to retrieve this feature manually. There are also difficult cases such as the Extruder feature. Extruder feature contains mechanism such as the speed on feeding the filament to the hotend, and the volume of the material. This requires the developers to create different variable names such as feedrate and volume_calculation to assign and store values. Therefore feedrate and volume also need to be included as the keyword for manual identification of Extruder feature.

There are many cases which the keywords are the same as the feature name, such as "command" for Command_Handling feature. Sometimes the developers choose to use shorthand namings like "cmd" instead of "command". They would also use synonyms or a domain related word such as "heater" instead of "hotend". Therefore we emphasis the importance of identifying correct keywords for manual identification since the developers make their own decisions during development, and they would choose names that they think are the most appropriate for a variable or a method. Only using feature names as keyword would be able to identify certain simple features such as Endstop, but it would neglect a lot of features that have high complexity in their functionalities and naming conventions.

A variable or method could be used in more than one feature. Take the example of the macros declared in header files, those declarations can be found in different features, and they might not be declared to be intentionally used only by one feature. Therefore a mechanism of handling those situations are required so that if one feature is optional and one feature is mandatory, the method or variable need to be included

for the mandatory feature. Beside that, it is also very important to check that the methods being called within a feature are also identified to be part of the feature. For example, a method has some codes within, and it also have method calls. However those method calls could to be part of the feature if one can make sure that the method called does not also belong to other features. For some of the methods, only partial of the codes belong to one feature. For instance, the setup() method where it sets up the whole printer, and it initializes different hardware settings. The initialization of a hardware setting is only be one line of code, and that line belongs to the hardware feature.

Marlin is an open source project, thus the code comments are quite descriptive, and a lot of the comments are good enough for one to figure out which feature certain method or variables belong to. If it is not enough, one could read through the code understanding the meaning of the code. Anyways, reading code comment and logic takes time, interest and patience.

Another important source of information is the g code documentation on RepRap wiki, and it is basically the requirement specification of 3D printers. It describes the functionalities of different G-codes, and it also provides examples of the scenarios of the G-codes during running. For example, " G28 Move to Origin" means when G28 is given in the file, the printer should move all axis to home position. The G code documentation helps us to better understand the meaning of the G-codes, and which G code belongs to a mandatory feature and which G-code belongs to optional feature.

5.4.2 Approach Generalization

For systems or partial system without development traceability, one can apply similar approach as ours. The key is to have good background knowledge such as system documentation, domain information, and architectural overview. And this is an incremental procedure where one's understanding of the system accumulates and reforms the more one has studied the system. The identification of right keywords are also crucial. Knowing the right keywords would assist the developers to find the right place to start and allocate features.

5.5 Feature Identification and Location Result Analysis

We have used totally 8 source of information for release log method and source code study method. They were release log, pull request, commit, ifdef expression, domain knowledge, code comment, source code, and g-code documentation. The summary on the data can be found in Appendix B, the source of information percentage are calculated based on the data. By applying the first three source of information i.e. release log, pull request, and commits, 26 feature locations were found out of 44 feature locations, and it makes up 59% of all the located features. Out of the 26 feature locations, all are optional features except one, and that is *Fan*. This feature

is mandatory, and it was developed originally to support a single fan for a single extruder. However due to the requests upon multiple extruders, more fans are added to match up with the multiple extruder request. The development of the multiple fan support was made in RC4, and this development adds the support of more fans on top of the original development. Therefore this feature was found through the release log method. We can conclude that by using this three source of information, a majority of optional features can be located if they are available through release log. There are 18 features that are not found with these three source of information, and most of those are mandatory feature. Only 6 are optional features. These 6 features are *Auto Bed Leveling Bilinear*, *Heated Bed*, *Buzzer*, *Servo Motor*, *Print Job Timer*, and *WatchDog*, and they are are all found through source code method instead of release log method. There is a countable amount of optional features are not found through release log method. Those features are either not recorded in the release log or changes were made too early in the project thus the actual source code differs vastly from the original development. Leaving the solely option to be found through source code method.

Ifdef expression is another source of information. As long as the feature's name is identified alongside of the ifdef statement, similar conditional groups can be identified for the feature. By using this source of information, 30 feature locations were identified, which makes up 68% of all features. Among those features, only one is mandatory, and that is *Arc Movement*. we categorized this feature as mandatory since printers produce round shapes by performing arc movements. Nonetheless, Marlin developers used ifdef macros for this feature for advanced settings to allow the users to configure different types of arc movements Marlin supports. This is the reason that there are ifdef macros around this feature even though it is a mandatory feature. This is another example of flexible usage of the ifdef expressions, and they are not only used for distinguishing features in Marlin. 14 features were not found through ifdef expressions, and only two are optional features. These two features are small in size, and they were not wrapped with ifdef, and those type of optional features are minorities in Marlin.

19 feature locations were found with domain knowledge as the source of information, and out of those there are 6 optional features, which makes up 43% of all features. They match up with the findings earlier that these 6 features are not found through release log, they are either developed much earlier on or they are not found to be recorded in the release log. All mandatory features are found with this method, and the rest 25 feature locations that are not found with this method are all optional features.

18 feature locations were found with code comment and source code study, which makes up 40% of all features. And out of those there are the same 6 optional features as mentioned earlier. All mandatory features are found with this method, and the rest 26 features that are not found with this method are all optional features with the exception of *PMW*. The reason is as explained earlier.

G-code documentation as source of information is project specific, there could also

Table 5.1: Feature Ratios of Sources Result

Feature Name	RL (%)	PR (%)	CM (%)	Ifdef (%)	DK (%)	CC (%)	SC (%)	GD (%)
Allow distinct factors for multiple extruders	50	20	15	15	0	0	0	0
Arc Movement	0	0	0	5	45	25	25	0
Auto Bed Leveling Bilinear	0	0	0	5	15	25	55	0
Auto filament change	50	20	10	20	0	0	0	0
BLTouch Sensor for Homing	50	20	15	15	0	0	0	0
Board	0	0	0	0	40	10	50	0
Buzzer	0	0	0	15	35	5	45	0
Case Light Menu	50	20	15	15	0	0	0	0
Command Input Process	0	0	0	0	20	30	50	0
Emergency Command Parser	50	30	5	15	0	0	0	0
Endstop	0	0	0	0	45	5	30	20
Extended Capabilities Report	50	20	15	15	0	0	0	0
Extruder	0	0	0	0	40	10	50	0
G20 Set units to inches	50	30	10	10	0	0	0	0
Heated Bed	0	0	0	10	50	20	20	0
Hotend	0	0	0	5	50	5	35	5
Input and Output Process	0	0	0	0	50	35	15	0
Linear Advance Extrusion Algorithm	50	20	15	15	0	0	0	0
M108 Cancel Heat Up	50	25	25	0	0	0	0	0
M149 set temperature units	50	15	5	30	0	0	0	0
M155 Auto temp report	50	20	15	15	0	0	0	0
M211 Enable/Disable Software Endstops	50	10	10	30	0	0	0	0
M43 Pin report and debug	50	20	10	20	0	0	0	0
Minimum Stepper Pulse Option	50	25	25	0	0	0	0	0
Mixing Extruders	50	35	10	5	0	0	0	0
Move to Destination	0	0	0	0	0	20	80	0
Move to Home Position	0	0	0	0	15	15	70	0
Nozzle Clean	50	20	10	10	0	0	0	10
Park Nozzle	50	20	10	10	0	0	0	10
Power Supply	0	0	0	0	45	5	50	0
Print Counter	50	10	15	25	0	0	0	0
Print Job Timer	0	0	0	30	20	10	40	0
Servo Motor	0	0	0	5	35	5	45	10
Single Nozzle Multiple Extruders	50	35	10	5	0	0	0	0
Stepper Motor	0	0	0	5	35	5	45	10
Support for an RGB LED light using 3 pins	50	40	5	5	0	0	0	0
Support for COREXY, COREXZ, and COREYZ	50	25	15	10	0	0	0	0
Support for multiple PWM fans	50	30	20	0	0	0	0	0
Support G2/G3 with R parameter	50	20	5	5	0	0	0	20
Switching Extruders	0	0	0	40	10	5	45	0
Temperature Control	0	0	0	0	30	10	30	30
Temperature watch protection for heated bed	0	50	15	5	30	0	0	0
TMC2130 Silent StepStick support	50	20	15	15	0	0	0	0
WatchDog	0	0	0	40	0	40	20	0

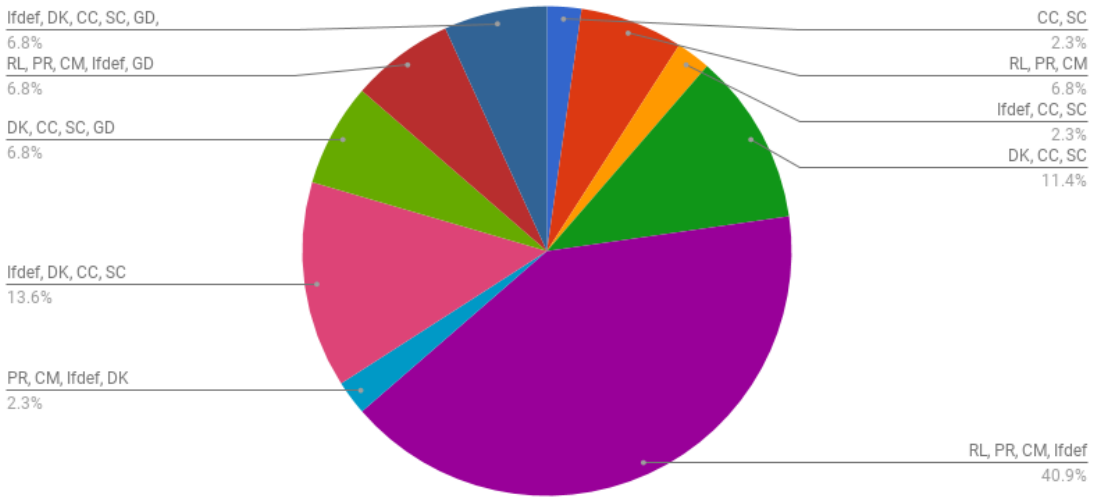
RL: Release Log; **PR**:Pull Request; **CM**: Commit; **Ifdef**: ifdef with feature expression; **DK**: Domain Knowledge; **CC**: Code Comment; **SC**: Source Code; **GD**: G-code Documentation;

be project specific methods depending on the characteristic of the project. There are only 5 features found with this method, which makes up 11% of all features. 4 are mandatory, 1 is optional but was also found with source code study method. This project specific method is not as useful as the rest and they are also applicable to most projects if they use Github and release log.

To sum up the analysis results above, we made a table, see Table 5.2. The second column records the total number of feature found using each source of information, and the third column records the percentage of each source of information in total number of found features. And the bar chart in Figure 5.3 visualizes the source of information percentage over all found features. And We could see that the leaders on the source of information used for finding features are Ifdef, Release Log, Pull Requests and Commits.

The Ratio of Sources of each feature are also summarized into a table, see 5.1. Each column estimates the effort percentage that was used for each source of information. The source of information that helped the most for identifying and locating

mandatory features were Domain Knowledge, Code Comment, and Source Code. The source of information that helped the most for identifying and locating optional features were Release Log, Pull Requests, and Commits. The combinations of multiple sources of information were also recorded based on this table, and we created a Pie Chart to show the combination of sources that were most often used together to retrieve features. And they were the combination of Release Log, Pull Reuqests, Commits, and ifdef. See Figure 5.2.



RL: Release Log; **PR**:Pull Request; **CM**: Commit; **Ifdef**: ifdef with feature expression; **DK**: Domain Knowledge; **CC**: Code Comment; **SC**: Source Code; **GD**: G-code Documentation;

Figure 5.2: Combination of Source Information Percentage.

Table 5.2: Source of Information Summary

	Number of found features	Found feature ratio
Release Log	26	16%
Pull Requests	26	15%
Commits	26	15%
Ifdef	30	18%
Domain Knowledge	19	11%
Code Comment	18	11%
Source Code	18	11%
G-Code	5	3%

The result also shows that 20 features are completely wrapped with ifdefs. They are all optional features, and there are 11 optional features that are not completely wrapped with ifdefs. 70% are completely wrapped with ifdef. This suggests that as long as you know the feature's expression name, around 70% of the optional features can be found thoroughly, and there are also a large number of optional

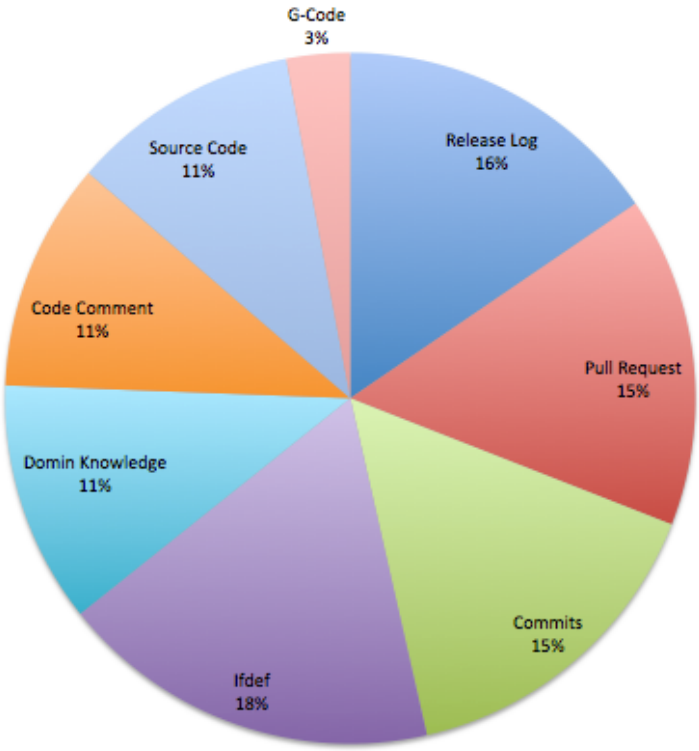


Figure 5.3: Source of Information Percentage.

features found, however need to have more time to find the rest of the features that are not wrapped with ifdef. One example is the PrintCounter feature, all related code implementations are surrounded by Ifdef in MarlinMain with the exceptions in printcounter.cpp, printcounter.h, and language.h. It seems like the whole files are not marked with preprocessor statements, and the implementation in language.h has only one line of code that's just unnecessary to be surround with "ifdef". The output of that line of code could be left blank, thus there is no harm done even if that line of code is not prune after compilation. We drew an activity diagram to conclude the process that we used for identifying features in Marlin. See figure 5.4.

As we recorded in our fact sheets, we summed up the total time used for locating features found using the release log method. And the result was that two persons together used around 9 hours to locate 26 features, and optional features were a majority among the found features. We also used roughly 25 hours by one person to locate 18 features using source code study method, and mandatory features were a majority among the found features. This resulted in an average of feature location of 0.34 hours for the release log method and 1.39 hours for the source code study method. This means that the development tracings technology provided by GitHub had greatly reduced the time and effort needed to identify and locate features.

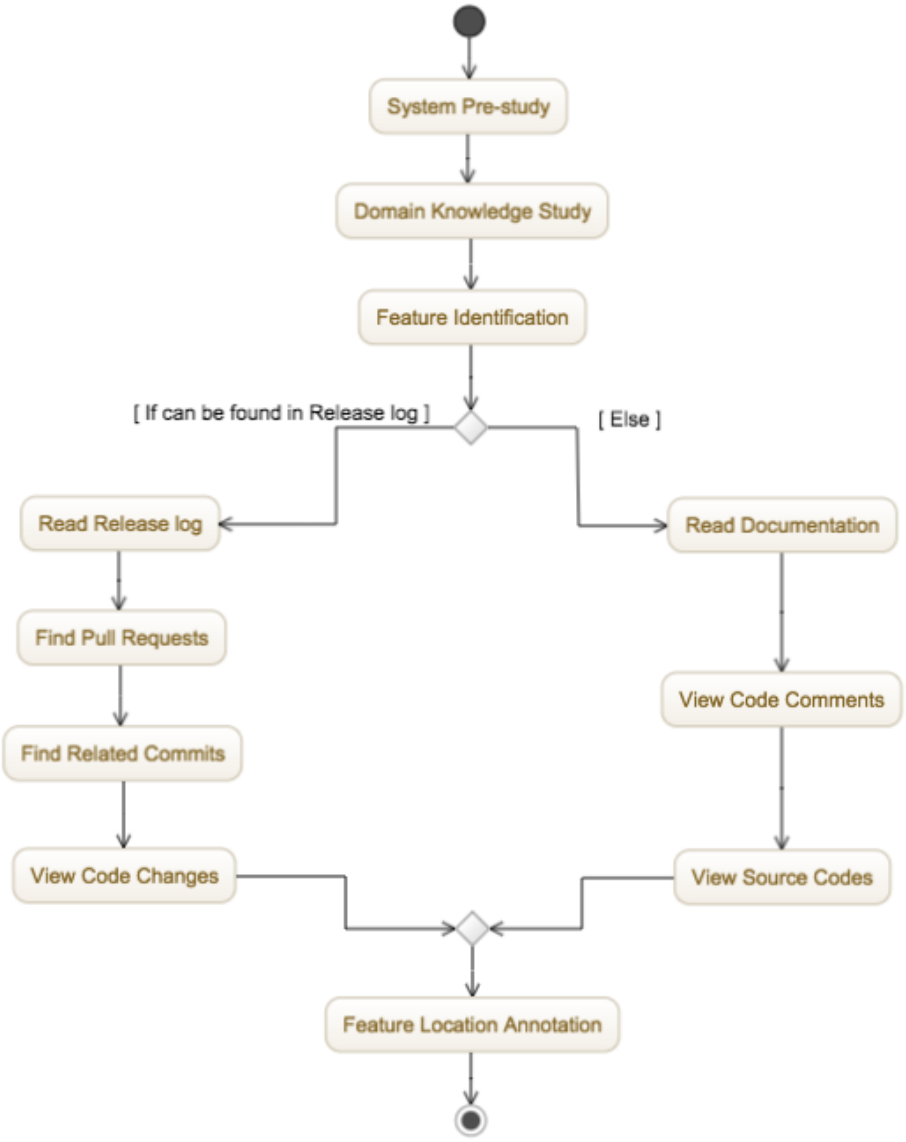


Figure 5.4: Activity Diagram for Feature Identification and Location.

5.6 Feature Characteristics Analysis

For the metrics measured for the feature characteristics, we used R to perform analysis on the data. Firstly, we used the summary() function to get some general statistical results. We found that the averages of LoFC, SD, and TD are 196, 10, and 7. The smallest feature has a size of 4 LoFC whereas the largest feature has a size of 823 LoFC. The feature with the lowest SD value is 1 where as the feature with the highest SD value is 72. The feature with the lowest TD value is 0 where as the highest has a TD value of 46. For more general statistical results and all R functions we that used for data analysis see Appendix D. After the general statistical test, we used the function density() to calculate the probability density of the data, and plotted it with histogram as shown in Figure 5.5. The density() function calculates the probability density of the dataset. The explanation of probability density is, if

there is a random feature from our dataset, then its probability of falling under the 0-5 TD interval instead of other TD intervals is 0.06 (6%). The Probability Density axis of the graph indicates the probability of any random variable falls under the different intervals of the histogram. We used the line() function to plot the model of data distribution based on density for LoFC, SD, and TD. This model helps us to see the data distribution of the dataset. The data distribution are all left skewed, and it indicates that the data is not normally distributed. The data that we have currently collected are very limited, and we need to collect more data in the future to achieve better results that are closer to population mean. The histogram figure also shows that the majorities of the features have 0-400 LoFC, 0-20 SD, and 0-40 TD.

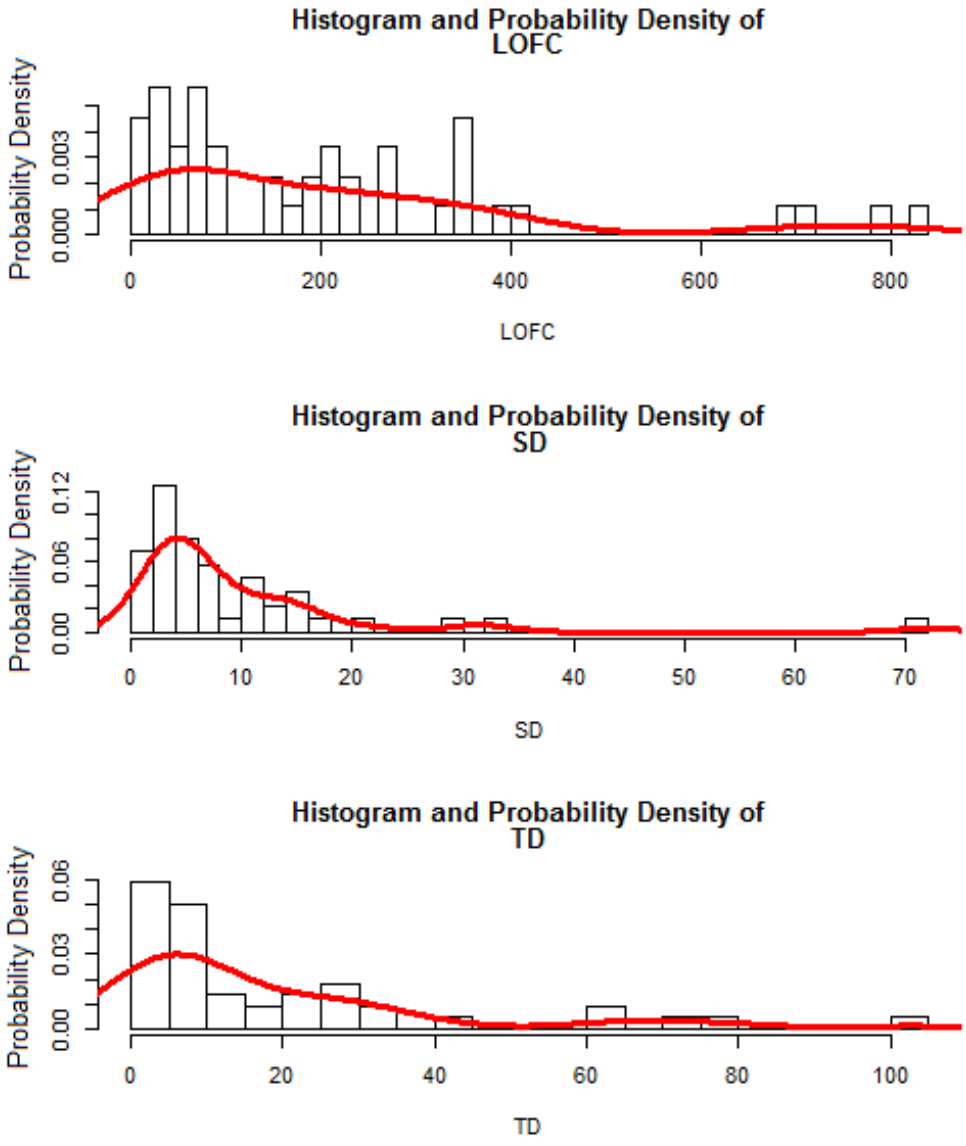


Figure 5.5: Histogram and Probability Density plot for LoFC, SD and TD.

For further analysis, we wanted to see if there is any correlation between any columns of the data. So we used the correlation function cor() to calculate the Pearson's coefficients of the data. Because we used different feature locating methods to identify and locate features, the feature identification and comprehension time data for reviewing source code method were not recorded per feature, therefore we used *"pairwise.complete.obs"* parameter in cor() function to pairwise remove the missing data of Feature Identification Time and Feature Comprehension Time in order to obtain correlation of feature identification time and feature comprehension time with the rest of the data [11]. For the rest of the correlation test we used complete dataset to obtain results. Table 5.3 displays the coefficient matrix of Feature Identification Time, Feature Comprehension Time, LoFC, SD, and TD. Pearson's Coefficient lies between -1 and 1 [20]. The values between -1 and 0 indicates negative correlation whereas values between 0 and 1 indicates a positive correlation. According to Evans' guide [6], the strength of the correlation can be categorized into 5 ranges: very week (.00-.19), weak (.20-.39), moderate (.40-.59), strong (.60-.79) and very strong (.80-1.0). Based on this categorization, most of the coefficient values are very weak and weak as the absolute values are closer to 0 rather than 1. According to the correlation test result in table 5.3, the very strong correlation with value that is larger than 0.8 is LoFC and TD, the strong correlations with values between 0.6 and 0.8 are Feature Identification Time and Feature Comprehension Time, Feature Comprehension Time and LoFC, Feature Comprehension Time and TD. There is also a moderate correlation that is worth noticing, and it is the correlation between Feature Comprehension Time and SD. We continued the analysis by creating linear models for the all these correlations using the lm() function. After obtaining linear models for the correlations, we used abline() function to visualize the models, see Figure 5.6, 5.7, 5.8, 5.9 and 5.10. In these figures we only used available data from the Release Log method since there are missing data from the Source Code method for Feature Identification Time and Feature Comprehension Time. However, the Feature Identification Time and Feature Comprehension Time are all estimated based on personal performance, thus some estimations might appear to be the same. Those models could be used to predict future values. However the linear models are simple models, and the model that can provide the good prediction is the one with the strongest correlation 0.85 for LoFC and TD. And the values provides by this model deviates the least from the actual values.

We could see some interesting phenomenon from the correlation results and linear models. The very strong correlation between LoFC and TD suggests that the larger the feature size, the larger TD value would be, meaning there are more features that are tangled within larger features. The three strong correlations also tell us some interesting relations among Feature Identification Time, Feature Comprehension Time, LoFC, TD, and SD. The correlation between Feature Identification Time and Feature Comprehension time indicates the longer time it takes to identify and locate a feature, the longer time it takes to comprehend it. This correlation could mean that features that are harder to locate, the more effort it is required to understand it. The correlation between Feature Comprehension Time and LoFC simply tells us that the larger the feature, the longer time it takes to understand the feature. The correlation

between Feature Comprehension Time and TD tells us that the more tangling there is with other features, the longer time it takes to understand it. This means that the higher complexity a feature has with high TD, the more effort it is required to understand it. The moderate correlation between Feature Comprehension Time and SD indicates that the more locations a feature is scattered, the longer time it takes to comprehend the feature. A feature that is widely scattered would effect feature identification time as one need to look for more places to locate the feature, but it should not have direct effect on comprehension time unless there is an indication on the increased complexity of the code such as high TD value. As this is a moderate correlation, this relationship could be merely a coincident.

Table 5.3: Correlation Test Result

	FeatureIdentificationTime	FeatureComprehensionTime	LoFC	SD	TD
FeatureIdentificationTime	1.000000000	0.6267697	0.09751381	-0.001107099	-0.06730504
FeatureComprehensionTime	0.626769726	1.0000000	0.62389647	0.402473097	0.60503966
LoFC	0.097513811	0.6238965	1.00000000	0.345745100	0.84588717
SD	-0.001107099	0.4024731	0.34574510	1.000000000	0.08384359
TD	-0.067305040	0.6050397	0.84588717	0.083843588	1.00000000

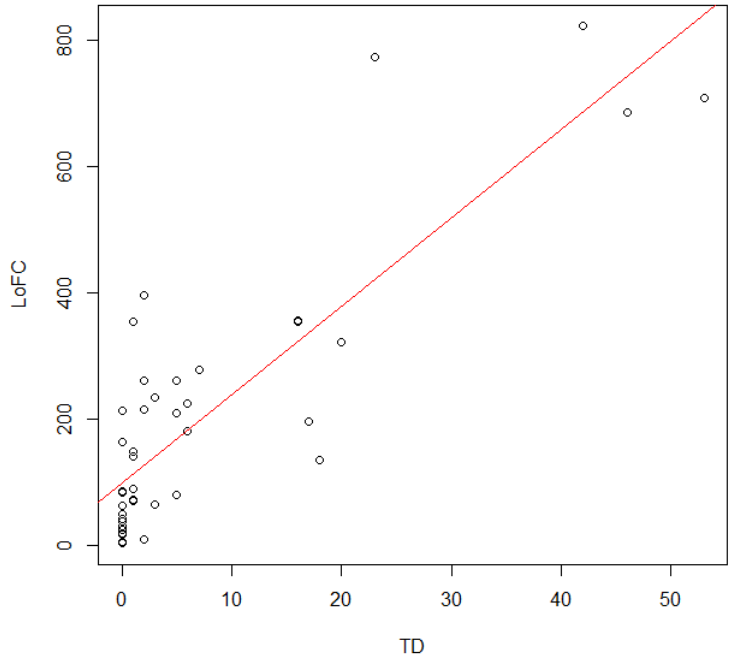


Figure 5.6: Linear correlation plot for LoFC vs TD (0.84588717).

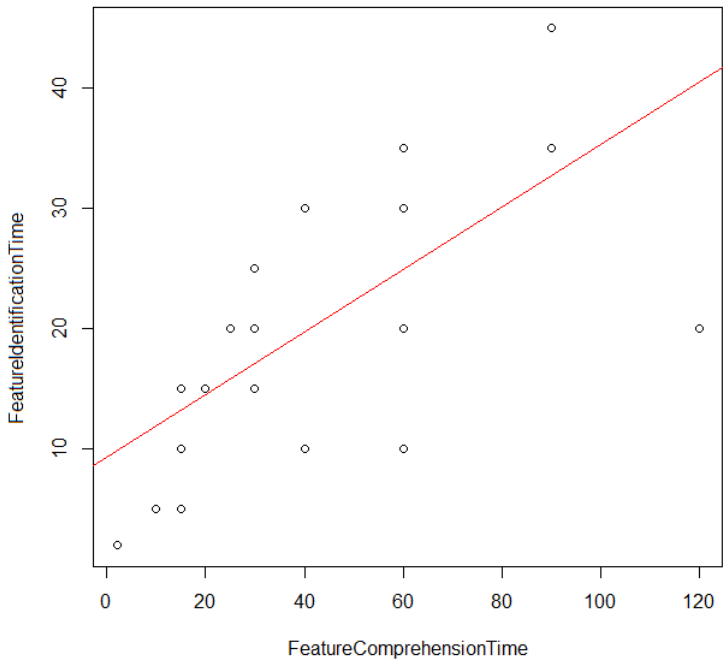


Figure 5.7: Linear correlation plot for FeatureIdentificationTime vs FeatureComprehensionTime (0.626769726).

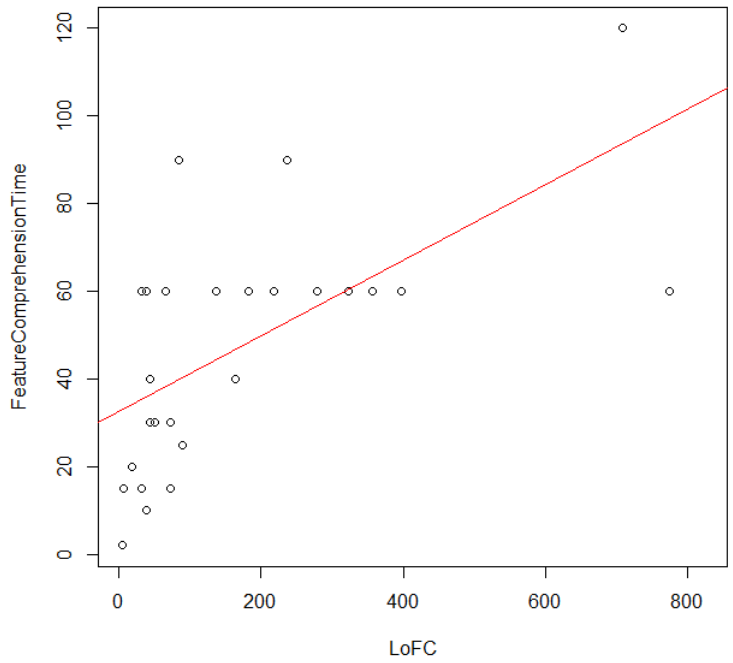


Figure 5.8: Linear correlation plot for FeatureComprehensionTime vs LoFC (0.6238965).

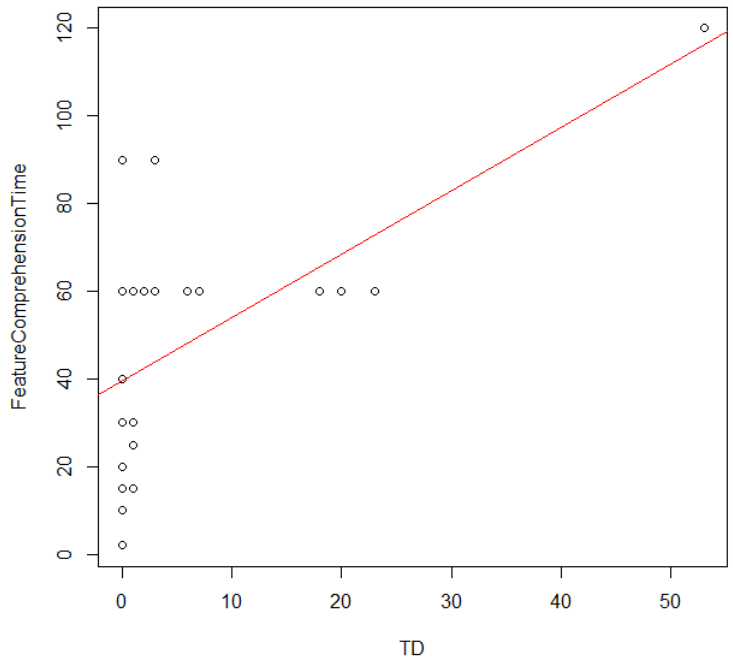


Figure 5.9: Linear correlation plot for FeatureComprehensionTime vs TD (0.60503966).

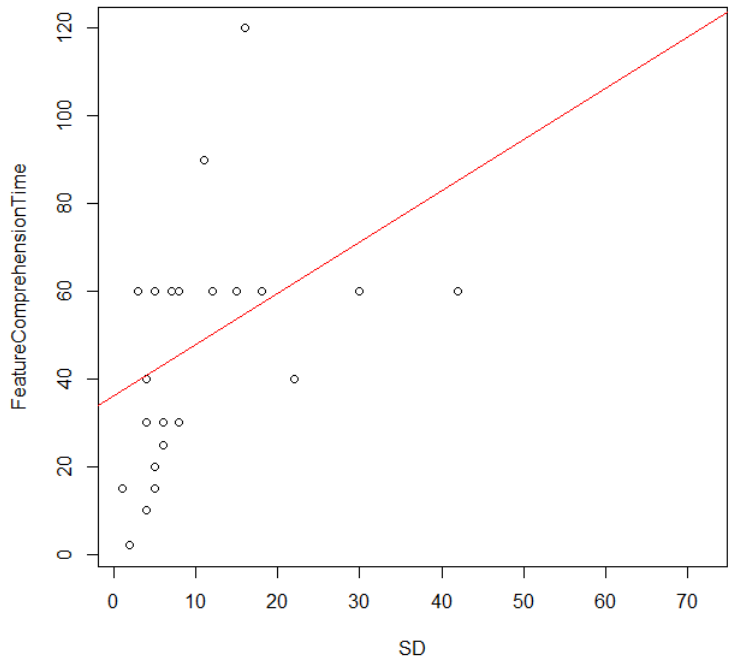


Figure 5.10: Linear correlation plot for FeatureComprehensionTime vs SD (0.4024731).

6

Threats to Validity

For this thesis work, there are several threats to internal and external validity. The first is that, the features found through source code reading needs validation for confirmation to avoid human errors. Our study was based entirely on the available information we had access to and our own program comprehension skills, and unless the results are confirmed with testings or by other means, there are chances that the parts of the features were not correctly located. For example, in Release Log method, the developers could have missed linking some related pull requests and used improper commit message description. Another possibility is during feature location with the Source Code method, due to the lack of knowledge about the entire system, we could miss out or over identified some parts of the feature. This could lead to reduced measurement accuracy, and we could both over and under estimate the measurement of SD, LoFC and even TD result. However, we think the possibilities for this type of human error to occur is minuscule because we have studied the source code and our knowledge base is adequate enough for locating features.

Another internal threat to validity is the features' location and comprehension time. Due to the developers' performance on source code comprehension could differ largely from one and another, we could only measure the result based on our limited resources. Therefore the result of time measurement for feature identification and feature comprehension were subjective for this case study. It is possible to obtain rather different measurement result if the experiment is repeated by other people with different programming capability backgrounds. And to be able to achieve a more objective result, we need to perform more experiment by different people on similar cases studies in the future. Another threat to the time measurement is that there is a learning effect between the methods. The accuracy of the manual locating feature time is reduced due to the reason that we already had a good overview of the system during the process of the first method. The time for pre-study and domain analysis are not included into the time for feature locating. The time measurement for feature locating will be much higher if these set up times are included.

One more internal threat is the result of TD measurements. TDs are expressed by different types conditional statements in Marlin, however not all types of conditional statements are used for wrapping optional features. Therefore if we include all types of conditional statements, the TD measurement will be too high, and if we do not include all conditional statement, we would still be having more TDs while missing some TDs from the excluded types of conditional statement. Therefore we only

tried to include the ones that could give us results that are closest to the actual TDs found manually. And the results are better for some features while worse for others.

For external validity, we consider our case study method and result would withhold to a certain degree with other similar studies on different software systems especially embedded systems. Software systems differ from one and other in many contexts such as the means of development culture or rules of feature implementation. Some systems that were developed in other languages might not use ifdefs at all for developing new features. There are even the chances that some systems do not use development tracing technologies for development and maintenance. These are all possible threats that would deviate our results and findings from studies on those systems. Nevertheless the Release Log methods could still be applied to systems that have similar development tracings as Marlin. And the Source Code method could be applied to other types of systems, and its main source of information required are not as specific as the Release Log method, as domain knowledge, system documentations, source code, and code comments are available in many systems.

7

Conclusion and Future Work

In this case study, we studied and explored an open sourced embedded system Marlin. And we wanted to find out what are the most useful source of information and strategies that can be used to retrieve features from software systems. With the consecutive accumulation of domain and system knowledge throughout the study, we successfully retrieved 44 features, both mandatory and optional using two different approaches. And we named them as Release Log method and Source Code method. The two approaches that we have followed could be used and applied by other systems to handle maintenance tasks and software product line migrations since both types of work requires feature allocation as a primary step. Besides the discovery of the two approaches, we also recorded all the features retrieved during the feature location process including the metrics measurements on the features' characteristics in Marlin. We will summarize the major findings of this case study and answer our research questions.

Besides the domain knowledge we learned during domain analysis, the pre-study process also helped us to answer the first research question. We found out that the developers use tags such as "New Feature" to indicate whether a pull request is related to new feature development. And Marlin's configuration files also uses conditional statements to define optional features in the source code. Therefore Marlin has a notion of feature in the system. After we identified and located 44 features that includes both mandatory and optional features, we summarized that there were 8 sources of information were used for finding features. Among those, Ifdef, Release Log, Pull Requests and Commits are the source of information that could be used to find most features. We also found that optional features that are wrapped with ifdef conditional statements, 70% of the features are completely wrapped by ifdef. This means that by searching the feature's name defined in macros in the source code, the majorities of the optional features could be completely located. Thus conditional statement are quite powerful for locating optional features in Marlin. The average time used for retrieving features by the two methods differs much, and it requires 4 times more effort to identify and locate a feature for Source Code method than Release Log methods. Therefore development tracing technologies provides greate assistance to the developers to locate features in less time. For the feature characteristics, the majorities have 0-400 LoFC, 0-20 SD, and 0-40 TD, however this result is far from the actual result of the population as our data is not normally distributed. And we need much more data both from Marlin and other systems to achieve better results. The last finding is that there is a positive correlation between two sets of the data, and they are LoFC and TD. Pearson's coefficient indicates a

quite strong correlation, and it could mean that complexity of the features raise as their size increase. Therefore in order to maintain simplicity for system comprehension, we think it is better to take in the consideration of having a reasonable LoFC per feature during development.

Glossary

Axis There are usually three axes to control the movements of the extruder or bed so that the printed object could be printed in three dimensions. In Delta typed printers, the axis are named as abc where as Cartesian typed printers named as xyz in Marin. 11

Bed All printers have a bed where the printing object locates.. 11

Board The board where all hardware components are connected. It has a microprocessor to store data and process them. It also has many pins where the hardware components can be connected as designed. 10

commits Adding changes to the local repository. 13

Endstops Used for the firmware to locate the lowest and/or highest points of an axis. This is used for the firmware to calculate relative distance to travel. 10

Extruder Hardware component that controls the feeding rate of the materials, there are usually two metal gears that pressurize the material and pushes the material downwards. 10

Filament Printing material for 3D printers. 10

Firmwares In electronic systems and computing, firmware is a type of software that provides control, monitoring and data manipulation of engineered products and systems. 9, 22

FLT Feature Location technique. 2

Frame The frame to hold together all the components and stabilizes the axis. 11

Heated Bed The bed can be heated up through electrical wires. 11

Hotend It has temperature sensors to tell what temperature it is. It also has a nozzle with a small opening where the heated materials come out. 10

Motors It is controlled by the stepper . Connects with the axes with rubber belt to control the movement of the extruder or bed. 10

Pull Requests Pull requests are a feature that makes it easier for developers to collaboration. They provide a user-friendly web interface for discussing proposed changes before integrating them into the official project. 13, 20, 36

SPL Software product line. 1

Bibliography

[1] Berima Andam et al. “FLOrIDA: Feature LOcatIon DAshboard for extracting and visualizing feature traces”. In: *Proceedings of the Eleventh International Workshop on Variability Modelling of Software-intensive Systems*. ACM. 2017, pp. 100–107.

[2] Michal Antkiewicz et al. “Flexible product line engineering with a virtual platform”. In: *Companion Proceedings of the 36th International Conference on Software Engineering*. ACM. 2014, pp. 532–535.

[3] Thorsten Berger et al. “A survey of variability modeling in industrial practice”. In: *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*. ACM. 2013, p. 7.

[4] Ted J Biggerstaff, Bharat G Mitbander, and Dallas Webster. “The concept assignment problem in program understanding”. In: *Software Engineering, 1993. Proceedings., 15th International Conference on*. IEEE. 1993, pp. 482–498.

[5] thinkyhead Bob-the-Kuhn. *Contributing Code with Pull Requests*. July 2017. URL: `http://marlinfw.org/docs/development/getting_started_pull_requests.html`.

[6] Jonathon St BT Evans and David E Over. *Rationality and reasoning*. Psychology Press, 2013.

[7] Emily Hill et al. “Which Feature Location Technique is Better?” In: *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*. IEEE. 2013, pp. 408–411.

[8] Wenbin Ji et al. “Maintaining feature traceability with embedded annotations”. In: *Proceedings of the 19th International Conference on Software Product Line*. ACM. 2015, pp. 61–70.

[9] Jacob Krüger et al. “Finding Lost Features in Cloned Systems”. In: *Proceedings of the 21st International Systems and Software Product Line Conference-Volume B*. ACM. 2017, pp. 65–72.

[10] Jorg Liebig et al. “An analysis of the variability in forty preprocessor-based software product lines”. In: *Software Engineering, 2010 ACM/IEEE 32nd International Conference on*. Vol. 1. IEEE. 2010, pp. 105–114.

[11] Dipak V Patil and RS Bichkar. “Multiple imputation of missing data with genetic algorithm based techniques”. In: *IJCA Special Issue on" Evolutionary Computation for Optimization Techniques* (2010), pp. 529–543.

[12] Denys Poshyvanyk et al. “Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval”. In: *IEEE Transactions on Software Engineering* 33.6 (2007).

[13] Václav Rajlich and Norman Wilde. “The role of concepts in program comprehension”. In: *Program Comprehension, 2002. Proceedings. 10th International Workshop on*. IEEE. 2002, pp. 271–278.

[14] Meghan Revelle and Denys Poshyvanyk. “An exploratory study on assessing feature location techniques”. In: *Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on*. IEEE. 2009, pp. 218–222.

[15] Julia Rubin and Marsha Chechik. “A survey of feature location techniques”. In: *Domain Engineering*. Springer, 2013, pp. 29–58.

[16] Ştefan Stănciulescu, Sandro Schulze, and Andrzej Wąsowski. “Forked and integrated variants in an open-source firmware project”. In: *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*. IEEE. 2015, pp. 151–160.

[17] Frank J Van der Linden, Klaus Schmid, and Eelco Rommes. *Software product lines in action: the best industrial practice in product line engineering*. Springer Science & Business Media, 2007.

[18] Jinshui Wang et al. “How developers perform feature location tasks: a human-centric and process-oriented exploratory study”. In: *Journal of Software: Evolution and Process* 25.11 (2013), pp. 1193–1224.

[19] Wikipedia. *G-code — Wikipedia, The Free Encyclopedia*. [Online; accessed 8-May-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=G-code&oldid=777161528.

[20] Wikipedia. *Pearson correlation coefficient — Wikipedia, The Free Encyclopedia*. [Online; accessed 26-September-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Pearson_correlation_coefficient&oldid=802449654.

[21] Wikipedia. *STL (file format) — Wikipedia, The Free Encyclopedia*. [Online; accessed 8-May-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=STL_(file_format)&oldid=778911217.

[22] Wikipedia. *Ultimaker — Wikipedia, The Free Encyclopedia*. [Online; accessed 8-May-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Ultimaker&oldid=777269244.

A

Appendix Release log

Update for 1.1.0 - RC8
https://github.com/MarlinFirmware/Marlin/issues/5077

New / Updated Features

- #4595, #4606, #4608, #4620, #4625, #4718 : Improved i2c messaging
- ~~#4667 : Add M211 to Enable/Disable Software Endstops~~
- ~~#4722 : Add MINIMUM_STEPPER_PULSE option~~
- #4832, #5088, #5094 : Enable M0/M1 with M108 (EMERGENCY_PARSER)
- #4833 : Remove SCARA axis_scaling
- ~~#4840 : Add support for G2/G3 with R parameter~~
- #4900 : Add G38.2 / G38.3 commands for CNC-style probing
- ~~#4955, #4974, #5118, #5132 : PINS_DEBUGGING and M43: Read pin states~~
- #5082 : Only trigger MAXTEMP error during heating
- ~~#5133 : Add M355 to turn the case light on/off and set brightness~~
- #5109 : Save configured hotend offsets to EEPROM
- ~~#5179 : Support for Trinamic TMC2130 SilentStepStick drivers~~
- ~~#5188 : Add M155 auto-report temperature (AUTO_REPORT_TEMPERATURES)~~
- ~~#5189 : Add M115 capabilities protocol (EXTENDED_CAPABILITIES_REPORT)~~
- #5238 : Support for AUTOTEMP options in M104 (not just M109)
- #5184, #5252 : Support endstops interrupts to improve performance
- ~~#5255 : Case Light menu item (MENU_ITEM_CASE_LIGHT)~~
- ~~#5330 : Support for a 3-pin RGB LED (RGB_LED)~~
- ~~#5371 : Each E stepper can have different steps/mm, acceleration, max feedrate~~

Update for 1.1.0 - RC7
https://github.com/MarlinFirmware/Marlin/issues/4237

New / Updated Features

- ~~#3611 : Add M108 command to cancel M109, M190, and M303~~
- ~~#3625, #3813, #3819, #4298 : Add Print Job Timer and statistics (PRINTCOUNTER)~~
- ~~#3653, #4106 : Add temperature watch for the heated bed (WATCH_BED_TEMP_PERIOD)~~
- ~~#3662 : New Filament Change feature (FILAMENT_CHANGE_FEATURE)~~
- ~~#3676, #4035, #4040, #4126 : New advance extrusion algorithm (LIN_ADVANCE)~~
- #3720 : Use a positive flag for Host Keepalive (HOST_KEEPALIVE)
- ~~#3789 : Add M999 S1 to restart without flushing the buffer~~
- ~~#3806 : Add CoreYZ support, fix CoreXY, CoreXZ (COREYZ)~~
- ~~#3808, #3895 : SINGLENOZZLE basic multi-extruder support~~
- ~~#3985 : Support for inches, Fahrenheit, Kelvin (INCH_MODE_SUPPORT, TEMPERATURE_UNITS_SUPPORT)~~
- ~~#4054, #4354 : Add NOZZLE_CLEAN_FEATURE~~
- ~~#4163, #4339 : Add MIXING_EXTRUDER and SWITCHING_EXTRUDER~~
- #4222 : Add P parameter to M302 (more like RepRapFirmware)
- ~~#4226 : Add EMERGENCY_PARSER to allow override commands~~
- #4241 : Add a serial transfer buffer option (TX_BUFFER_SIZE)
- #3992 : Add error-checking of E parameter in M303
- #4013 : Add S parameter to stay in place on tool-change. Example: T1 S1

- #4053, #4060, #4094, #4158 : Add support for the Cartesio UI (BOARD_CNCONTROLS_12)
- #4159 : Support for RigidBot V2 and its digipot (BOARD_RIGIDBOARD_V2)
- #4192: Support for Vellemann K8400 (BOARD_K8400)
- #4244 : Dyze High Temperature Thermistor support (up to 500°C)
- #4271, #4279 : Add X_DUAL_STEPPER_DRIVERS option
- ~~#4299 : Add NOZZLE_PARK_FEATURE~~
- #4305 : Drop-in custom boot screens
- ~~#4336 : Add support for BLTouch sensor (BLTOUCH)~~
- #4362 : Add DUAL_NOZZLE_DUPLICATION_MODE
- #4408 : Add support for REPRAPWORLD_GRAPHICAL_LCD

B

Appendix Feature Collection Result

This is the summary of the feature location result. The first column records the name of each feature. and second indicate whether the feature is mandatory or not. From column RL to CWi records the data on which source of information are used to locate the feature. Column FIT and FCT indicate the different time for identifying and comprehending the feature. From Column CM to LD records facts such as how many files and commit messages each feature is retrieved from Github. The rest of the columns records the feature characteristics.

Table B.1: Feature Collection Result

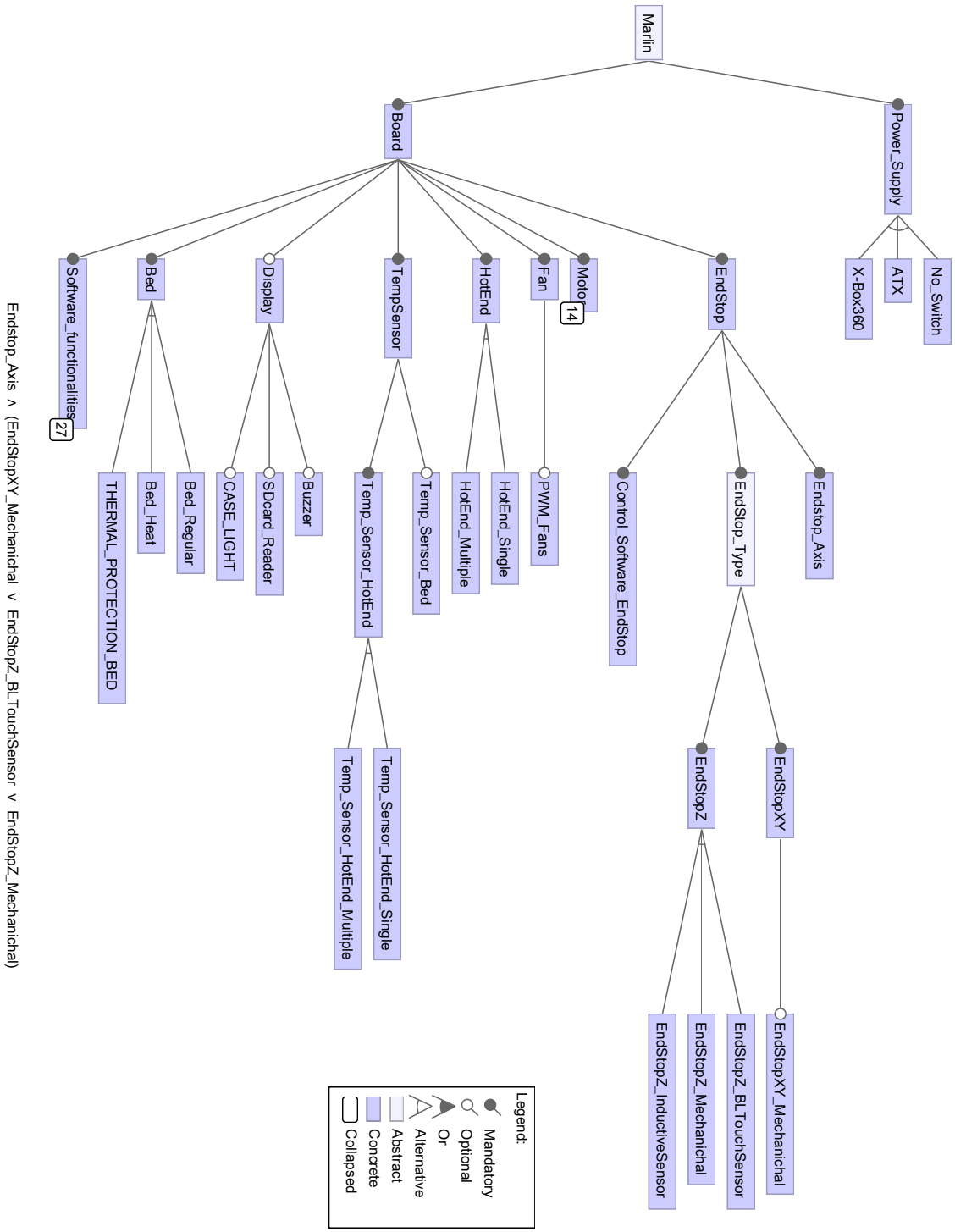
Feature Name	MF	RL	PR	CM	Ifdef	DK	CC	SC	GD	CWi	FIT	FCT	CM	Files	LA	LD	LOF	SD	TD
Allow distinct factors for multiple extruders	n	y	y	y	y	n	n	n	n	y	10	40	3	29	495	131	164	22	0
Arc Movement	y	n	n	n	y	y	y	y	n	n	NA	NA	NA	NA	NA	NA	262	5	5
Auto Bed Leveling Bilinear	n	n	n	n	y	y	y	y	n	y	NA	NA	NA	NA	NA	NA	226	1	6
Auto filament change	n	y	y	y	y	n	n	n	n	y	35	60	3	26	791	253	397	15	2
BLTouch Sensor for Homing	n	y	y	y	y	n	n	n	n	y	35	60	2	24	86	5	65	5	3
Board	y	n	n	n	n	y	y	y	n	n	NA	NA	NA	NA	NA	NA	214	72	0
Buzzer	n	n	n	n	y	y	y	y	n	n	NA	NA	NA	NA	NA	NA	25	3	0
Case Light Menu	n	y	y	y	y	n	n	n	n	y	25	30	2	20	129	0	51	6	0
Command Input Process	y	n	n	n	n	y	y	y	n	n	NA	NA	NA	NA	NA	NA	823	2	42
Emergency Command Parser	n	y	y	y	y	n	n	n	n	y	35	90	6	29	249	56	236	11	3
Endstop	y	n	n	n	n	y	y	y	y	n	NA	NA	NA	NA	NA	NA	198	14	17
Extended Capabilities Report	n	y	y	y	y	n	n	n	n	y	30	60	1	21	157	9	322	7	20
Extruder	y	n	n	n	n	y	y	y	n	n	NA	NA	NA	NA	NA	NA	149	15	1
G20 Set units to inches	n	y	y	y	y	n	n	n	n	y	20	30	1	23	531	181	43	4	0
Heated Bed	n	n	n	n	y	y	y	y	n	n	NA	NA	NA	NA	NA	NA	211	4	5
Hotend	y	n	n	n	n	y	y	y	y	n	NA	NA	NA	NA	NA	NA	355	7	16
Input and Output Process	y	n	n	n	n	y	y	y	n	n	NA	NA	NA	NA	NA	NA	87	12	0
Linear Advance Extrusion Algorithm	n	y	y	y	y	n	n	n	n	y	20	60	8	26	360	57	279	18	7
M108 Cancel Heat Up	n	y	y	y	n	n	n	n	n	n	2	2	1	21	13	0	4	2	0
M149 set temperature units	n	y	y	y	y	n	n	n	n	y	20	60	1	23	531	181	39	3	0
M155 Auto temp report	n	y	y	y	y	n	n	n	n	y	5	10	1	19	128	0	39	4	0
M211 Enable/Disable Software Endstops	n	y	y	y	y	n	n	n	n	n	20	25	5	9	154	107	90	6	1
M43 Pin report and debug	n	y	y	y	y	n	n	n	n	n	35	60	14	59	2758	752	217	8	2
Minimum Stepper Pulse Option	n	y	y	y	n	n	n	n	n	n	5	15	1	1	97	15	6	1	0
Mixing Extruders	n	y	y	y	y	n	n	n	n	y	10	60	4	38	1402	399	775	42	23
Move to Destination	y	n	n	n	n	y	y	y	n	n	NA	NA	NA	NA	NA	NA	357	14	16
Move to Home Position	y	n	n	n	n	y	y	y	n	n	NA	NA	NA	NA	NA	NA	686	9	46
Nozzle Clean	n	y	y	y	y	n	n	n	n	y	35	60	8	54	1450	200	31	3	0
Park Nozzle	n	y	y	y	y	n	n	n	n	n	30	40	2	25	683	85	43	4	0
Power Supply	y	n	n	n	n	y	y	y	n	n	NA	NA	NA	NA	NA	NA	80	4	5
Print Counter	n	y	y	y	y	n	n	n	n	n	45	90	13	34	772	93	84	11	0
Print Job Timer	n	n	n	n	y	y	y	y	n	y	NA	NA	NA	NA	NA	NA	11	2	2
Servo Motor	n	n	n	n	y	y	y	y	y	n	NA	NA	NA	NA	NA	NA	64	3	0
Single Nozzle Multiple Extruders	n	y	y	y	y	n	n	n	n	n	15	20	5	75	644	465	18	5	0
Stepper Motor	y	n	n	n	n	y	y	y	y	n	NA	NA	NA	NA	NA	NA	262	12	2
Support for an RGB LED light using 3 pins	n	y	y	y	y	n	n	n	n	y	15	15	1	24	238	10	72	5	1
Support for COREXY, COREXZ, and COREYZ	n	y	y	y	y	n	n	n	n	y	30	60	4	26	133	108	182	3	6
Support for multiple PWM fans	y	y	y	y	n	y	n	n	n	n	30	60	3	11	326	89	356	30	1
Support G2/G3 with R parameter	n	y	y	y	y	n	n	n	n	y	10	15	1	2	51	9	32	1	0
Switching Extruders	n	y	y	y	y	n	n	n	n	y	10	60	4	38	1402	399	137	12	18
Temperature Control	y	n	n	n	n	y	y	y	y	n	NA	NA	NA	NA	NA	NA	142	8	1
Temperature watch protection for heated bed	n	y	y	y	y	n	n	n	n	y	15	30	3	21	277	59	73	8	1
TMC2130 Silent StepStick support	n	y	y	y	y	n	n	n	n	y	20	120	7	25	6137	1832	709	16	53
WatchDog	n	n	n	n	y	y	y	y	n	n	NA	NA	NA	NA	NA	NA	20	6	0

MF: Mandatory Feature; Source used for Feature location (**RL**: Release Log; **PR**:Pull Request; **CM**: Commits; **Ifdef**: ifdef with feature expression; **DK**: Domain Knowledge; **CC**: Code Comment; **SC**: Source Code; **GD**: G-code Documentation;) **CWi**: Completely Wrapped by ifdefs; Feature measurement result (**FIT**: Feature Identification Time (min); **FCT**: Feature Comprehension Time (min) ;**CM**: Commit; **LA**: Lines Added; **LD**: Lines Deleted; **LOF**: Lines of Feature Code; **SD**: Scattering Degree; **TD**: Tangling Degree;)

C

Appendix Feature Model

This is the feature model that we have build iteratively, it includes all the mandatory features and optional features that we found, and also other features we learned from domain analysis.



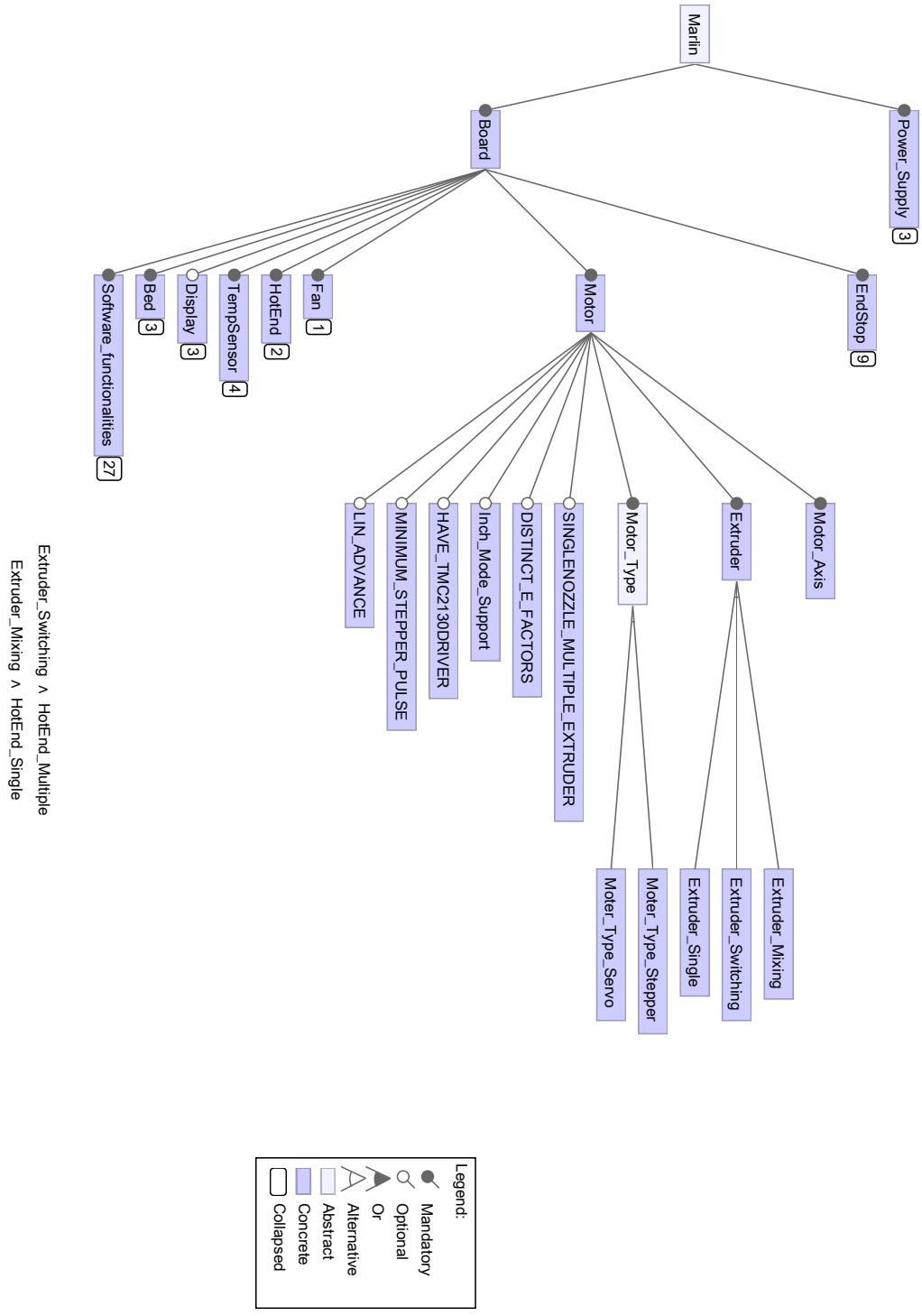
Endstop_Axis ∧ (EndStopXY_Mechanichal ∨ EndStopZ_BLTouchSensor ∨ EndStopZ_Mechanichal)

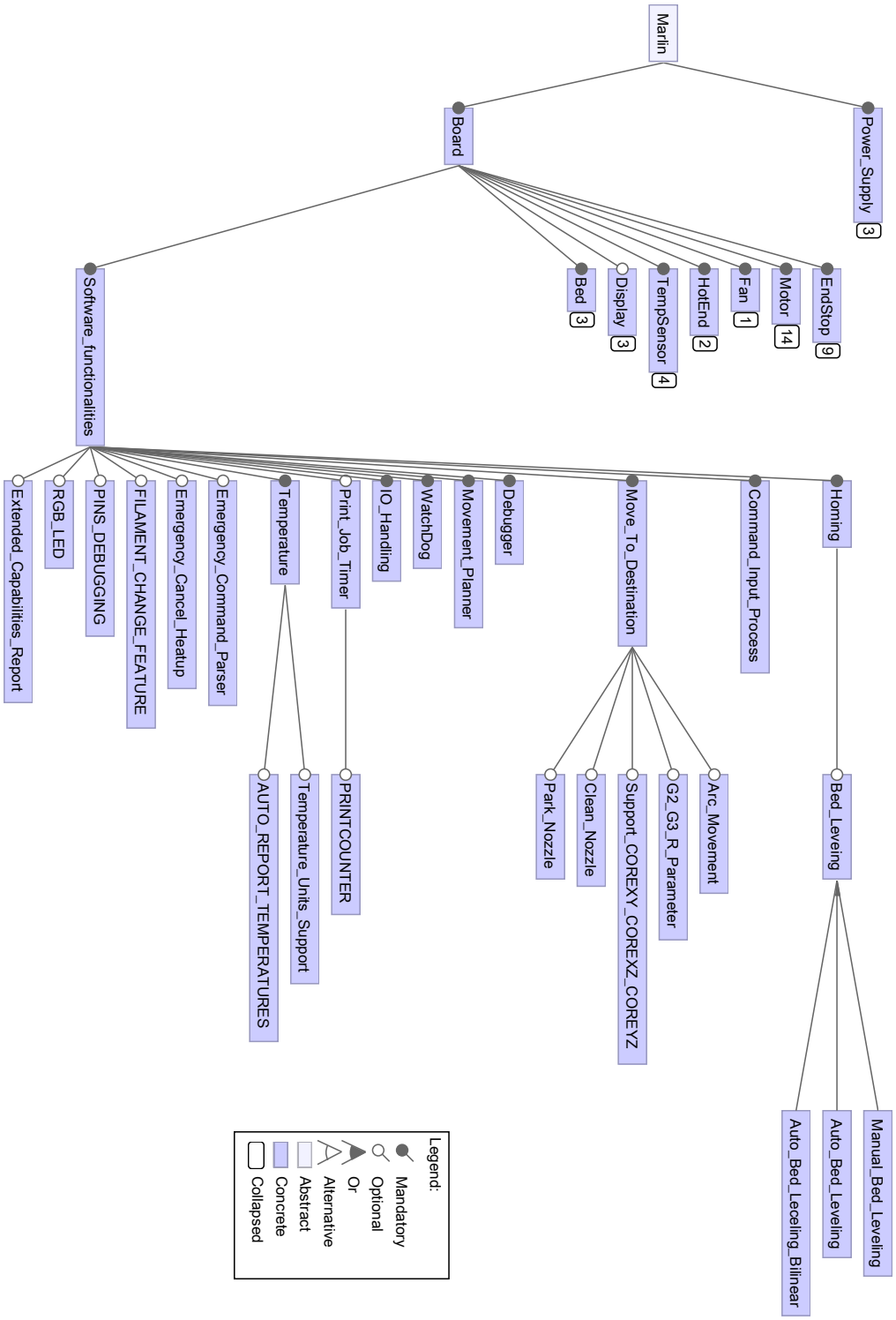
Extruder_Switching ∧ HotEnd_Multiple

Extruder_Mixing ∧ HotEnd_Single

HotEnd_Multiple ∧ Temp_Sensor_HotEnd_Multiple

Temp_Sensor_Bed ∧ Bed_Heat


Marlin
Board
Power_Supply 3
Software_functionalities 27
Bed 3
Display 3
TempSensor 4
HotEnd 2
Fan 1
Motor
EndStop 9
LIN_ADVANCE
MINIMUM_STEPPER_PULSE
HAVE_TMC2130DRIVER
Inch_Mode_Support
DISTINCT_E_FACTORS
SINGLENOZZLE_MULTIPLE_EXTRUDER
Motor_Type
Extruder
Motor_Axis
Moter_Type_Servo
Moter_Type_Stepper
Extruder_Single
Extruder_Switching
Extruder_Mixing
Extruder_Switching ∧ HotEnd_Multiple
Extruder_Mixing ∧ HotEnd_Single
Legend:
Mandatory
Optional
Or
Alternative
Abstract
Concrete
Collapsed


Marlin
Board
Power_Supply 3
Software_functionalities
Bed 3
Display 3
TempSensor 4
HotEnd 2
Fan 1
Motor 14
EndStop 9
Extended_Capabilities_Report
RGB_LED
PINS_DEBUGGING
FILAMENT_CHANGE_FEATURE
Emergency_Cancel_Heatup
Emergency_Command_Parser
Temperature
Print_Job_Timer
IO_Handling
WatchDog
Movement_Planner
Debugger
Move_To_Destination
Command_Input_Process
Homing
AUTO_REPORT_TEMPERATURES
Temperature_Units_Support
PRINTCOUNTER
Park_Nozzle
Clean_Nozzle
Support_COREXY_COREXZ_COREYZ
G2_G3_R_Parameter
Arc_Movement
Bed_Leveing
Auto_Bed_Leceling_Bilinear
Auto_Bed_Leveling
Manual_Bed_Leveling
Legend:
Mandatory
Optional
Or
Alternative
Abstract
Concrete
Collapsed

D

Appendix Statistical Test Result

This statistical test result records the process and result obtained from R. We use this to analyze the metrics to obtain correlations among the feature characteristics.

```
summary(features)
                  Annotation_Name   Mandatory_Feature
 196,2727273               : 1     10,11363636: 1
 8636                      : 1     445        : 1
 ARC_SUPPORT               : 1     n          :31
 Auto_Bed_Leveling_Bilinear: 1     y          :13
 AUTO_REPORT_TEMPERATURES  : 1
 Bed_Heated                : 1
 (Other)                   :40
 FeatureIdentificationTime FeatureComprehensionTime      LOF
 NA    :18                  NA    :18                 Min.   :  4.0
 20    : 5                   60    :11                 1st Qu.: 43.0
 35    : 5                   15    : 3                 Median :139.5
 10    : 4                   30    : 3                 Mean   :196.3
 30    : 4                         : 2                 3rd Qu.:262.0
 15    : 3                   40    : 2                 Max.   :823.0
 (Other): 7                  (Other): 7                 NA's   :2
       SD              TD
 Min.   : 1.00   Min.   : 0.000
 1st Qu.: 3.75   1st Qu.: 0.000
 Median : 6.00   Median : 1.000
 Mean   :10.11   Mean   : 6.932
 3rd Qu.:12.00   3rd Qu.: 6.000
 Max.   :72.00   Max.   :53.000
 NA's   :2       NA's   :2

> FeaturesData = data.frame(features[3:7])
> cor(FeaturesData,use = "pairwise.complete.obs")
```

	FeatureIdentificationTime	FeatureComprehensionTime	LoFC	SD	TD
FeatureIdentificationTime	1.000000000	0.6267697	0.09751381	-0.001107099	-0.06730504
FeatureComprehensionTime	0.626769726	1.000000000	0.62389647	0.402473097	0.60503966
LoFC	0.097513811	0.6238965	1.000000000	0.345745100	0.84588717
SD	-0.001107099	0.4024731	0.34574510	1.000000000	0.08384359
TD	-0.067305040	0.6050397	0.84588717	0.083843588	1.000000000

```
#It has a value between +1 and −1, where 1 is total positive linear correlation, 0 is no
#linear correlation, and −1 is total negative linear correlation.

> fit <- lm(LoFC~TD)
> summary(fit)

Call:
lm(formula = LoFC ~ TD)
```

```
Residuals:
    Min      1Q  Median      3Q     Max
-214.13  -69.79  -37.78   48.25  353.91

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   99.285     19.447   5.106 7.55e-06 ***
TD            13.992      1.361  10.278 4.91e-13 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 112.8 on 42 degrees of freedom
Multiple R-squared:  0.7155,    Adjusted R-squared:  0.7088
F-statistic: 105.6 on 1 and 42 DF,  p-value: 4.915e-13

> plot(TD,LoFC)
> abline(99.285,13.992,col="red")

> fit <- lm(Feature_Identification_Time~FeatureComprehensionTime)
> summary(fit)

Call:
lm(formula = FeatureIdentificationTime ~ FeatureComprehensionTime)

Residuals:
    Min      1Q  Median      3Q     Max
-20.484  -6.399   2.055   7.203  12.311

Coefficients:
                        Estimate Std. Error t value Pr(>|t|)
(Intercept)              9.30255    3.60954   2.577 0.016533 *
FeatureComprehensionTime 0.25985    0.06594   3.941 0.000612 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9.214 on 24 degrees of freedom
  (18 observations deleted due to missingness)
Multiple R-squared:  0.3928,    Adjusted R-squared:  0.3675
F-statistic: 15.53 on 1 and 24 DF,  p-value: 0.0006122

> plot(FeatureComprehensionTime,Feature_Identification_Time)
> abline(9.30225,0.25985,col="red")

> fit <- lm(FeatureComprehensionTime~LoFC)
> summary(fit)

Call:
lm(formula = FeatureComprehensionTime ~ LoFC)

Residuals:
    Min      1Q  Median      3Q     Max
-39.313 -15.054  -4.778  14.633  50.163

Coefficients:
```

```
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 32.60658    5.77836   5.643 8.25e-06 ***
LoFC         0.08607    0.02201   3.911  0.00066 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 22.29 on 24 degrees of freedom
  (18 observations deleted due to missingness)
Multiple R-squared:  0.3892,    Adjusted R-squared:  0.3638
F-statistic:  15.3 on 1 and 24 DF,  p-value: 0.0006596

> plot(LoFC,FeatureComprehensionTime)
> abline(32.60658,0.08607,col="red")

> fit <- lm(FeatureComprehensionTime~TD)
> summary(fit)

Call:
lm(formula = FeatureComprehensionTime ~ TD)

Residuals:
    Min      1Q  Median      3Q     Max
-37.536 -15.193  -2.562  17.208  50.464

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  39.5357     4.9277   8.023    3e-08 ***
TD            1.4473     0.3888   3.723  0.00106 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 22.71 on 24 degrees of freedom
  (18 observations deleted due to missingness)
Multiple R-squared:  0.3661,    Adjusted R-squared:  0.3397
F-statistic: 13.86 on 1 and 24 DF,  p-value: 0.001058

> plot(TD,FeatureComprehensionTime)
> abline(39.5357,1.4473,col="red")

hist(LOF)
hist(SD)
hist(TD)
```