Methods and Styles for Proposing Source Code Changes in Free and Open Source Software

Mark Hill

March 7, 2018

Abstract

Writing proposals is a key part of a software engineer's job; however, there is no previous research on how to write software proposals in industry. Rather, sites and forums encourage newcomers to read previous proposals and figure out the conventions on their own. This can be challenging to people with no industry experience or for whom English is not their first language. This paper explores the writing conventions of open source software proposals, such as the use of problem statements and solution explanations, by examining emails from the Linux kernel project. Comparisons between proposals from new members and previous contributors are made to show the requirements for establishing oneself in the Linux developer community.

Keywords: Linux, open source, request for comments, software development, pull request, mailing list

1 Introduction

Free and Open Source Software[1], better known in the free software community as FOSS, aims to fix the problems associated with the rise of proprietary software beginning in the late 1970s, namely: expensive software licenses, lack of extended support, and an inability to extend existing programs for custom use cases[1]. As a consequence of the GNU Project in 1983, free software experienced a revival, spurring numerous projects, including the GNU/Linux operating system, the BSDs[2], the Apache web server, Netscape Communicator[3], Android, Chromium[4], and countless other projects that form the basis for modern computing architecture and run our interconnected world.

Although some have argued that companies would never devote resources to products they could not directly monetize or own as intellectual property, such concerns have not been realized. For example, the Linux kernel, the underlying software managing hardware and resource allocation for the GNU/Linux operating system, Android, and countless Internet of Things[5] (IoT) products, receives 91.8% of its source changes from corporate-sponsored developers, with Intel, IBM, Samsung and Google among the biggest contributors of engineers and funding[2]. GitHub, a company offering a web front-end to the git version control system, sponsors Open Source Friday[3] where its employees contribute to open source projects rather than their normal work for GitHub. Even Apple, the tech giant known for closed source software and walled-garden ecosystem, has launched open source projects such as its Swift programming language[4].

As such cases indicate, contribution to free software programs is an important part of software developers' jobs, whether they contribute directly through paid contribution or as hobbyists developing skills to advance their career. It is unfortunate, then, that universities do not emphasize contribution to open source projects, minimizing opportunities for students to practice their coding skills, gain real world experience working with teams, and develop their professional writing.

At a general level proposals, requests for comments (RFC)[6], and pull requests all serve a similar purpose: informing other project members of your contributions and allowing the community to discuss the advantages and disadvantages of adopting that standard. Unfortunately, previous research on writing these materials is sparse if non-existent, likley due to its niche role and decentralized set of standards. Samsung Open Source Group recommends new contributors to the Linux kernel "search the subsystem mailing list archives for older threads to learn [unwritten rules]" [5], a sentiment that many blog posts covering the Linux kernel development process share. Universities in general and UC Berkeley in particular

[1]The word "free" in "free software" refers not to price–though FOSS is free from cost as a consequence of FOSS licensing–but to freedom[1], as in freedom to view, modify, and distribute source code.

[2]Berkeley Software Distribution

[3]The software for the Netscape Communicator suite was initially proprietary but was later released as free software due to the influence of the GNU manifesto

[4]Chromium is the free-software version of Google Chrome. It has an identical code base besides the lack of automatic updating and usage tracking

[5]Internet of Things is a term that applies to physical devices connected to the internet such as wireless routers, appliances, televisions, cars, and trucks, among others

[6]Requests For Comments are a common method of proposing a completed change for a software project, but they also find extensive use in outlining technology standards such as the internet protocols for addressing networked computers–IPv4 and IPv6.

only increase the problem by displaying the software development process in the absence of writing.

This paper looks at how proposals, RFCs, and pull requests (hereafter referred to collectively as proposals) are structured and aims to identify and describe the features that should be present in order to receive useful feedback.

2 Methodology

Due to FOSS development taking place over the internet, there are numerous proposals available to compare and analyze. In order to control for different contributor populations, source code development rates, and contributor motivations, only data from the Linux Kernel Mailing List (LKML) [6] Archive was used. This dataset included a wide range of information, much of it irrelevant to the study of proposals.

To find relevant information that contains formal proposals and RFCs, data was filtered by email thread metadata; only emails from the year 2017 whose subject lines contained "RFC" were examined. The tone, content, and volume of direct email replies to threads fitting the aforementioned criteria were used to determine community reaction to a proposal. A binary standard of success was used, with proposals that were eventually merged[7] or that received a positive community reaction classified as successful, while proposals that were not merged in their current form or were rejected by the community classified as unsuccessful. None of the proposals in this sample were eventually merged as they were early in the development process, so community reaction was the sole indicator of success.

A categorization of a proposal's structural components (problem statement, solution explanation, etc.) and the creator's responsiveness in the ensuing email chain (timeliness, level of detail in responses) was created, allowing proposals to be judged by specific criteria and the results compared to other proposals in the dataset. Given proper funding and department support, a larger sample size would have been used and proposals would have been judged by members of the technology discipline, defined as someone currently employed by a company that makes 25% or more of its revenue from software or computer hardware. However, due to to lack of funds, it was not possible to gather responses from enough industry professionals. As a result, I personally examined and judged proposals for this paper.

Eleven proposals were selected after sampling emails that met the aforementioned criteria. Of the eleven sampled, one became unavailable when I attempted to collect data from it, which reduced the sample size to ten. The following are the questions used to collect data for each email thread:

1. Has the author previously contributed code to the linux kernel?
2. Has the author previously given feedback on other contributions?

[7] Merging code is an operation in software version control systems that makes it possible to include changes from multiple authors across all files in a project. This process is complicated and works automatically for most of the changes, but requires manual work for the remaining portions. Project leads and qualified contributors are given the unfortunate burden of merging changes into the canonical source code. In return, they are given the title maintainers.

3. Is this submission a work in progress or mostly complete change?
4. Was there a problem statement?
5. Was there a solution explanation?
6. How many words was the proposal (excluding code)?
7. How many responses did the submission receive?
8. How many different people responded?
9. Did the responses indicate the proposal was a good idea?
10. Did the author reply within 24 hours to feedback from others?
11. On average, how many words did the author use to respond to first level feedback?[8]
12. Was the code accepted and merged into the linux kernel source?

The data for each proposal is given in the appendix.

3 Results

In this paper, a "new member" or "new contributor" is someone who has not previously submitted code to the linux mailing lists, while people who have submitted work are classified as "previous contributors."

Documentation for the linux project indicates that the linux community strongly recommends new members prove their coding ability and trustworthiness by contributing small bug fixes before they submit major changes such as adding new features. I used three different comparisons to see the differences in the community reaction to work from previous contributors and new members.

Table 1 shows the difference between new members and previous contributors in the amount of work put into implementing a proposed change before submitted it to the mailing list for review. Results show that previous contributors are more likely to submit proposals with limited functionality code.

	Proposal code functionality		
Previous contributor	Nearly complete	Work in progress	Totals
Yes	2	5	7
No	2	1	3
Totals	4	6	10

Table 1: Functionality of code at the time of the proposal vs. whether the author has previously contributed to the Linux project

[8]First level feedback is an email reply from someone other than the author that asked questions about the proposal or made suggestions to improve it

The number of people providing feedback for proposals also differs between new members and previous contributors as show in table 2, with new members receiving feedback from a greater number of people on average. This could be due to numerous factors specific to new members including: they have more complete code when asking for feedback, they are not as trusted by the community, or they do not have established relationships with community members who review code in their area of focus.

	Number of unique respondents			
Previous contributor	0 - 2	3 - 5	6+	Totals
Yes	3	3	1	7
No	0	2	1	3
Totals	3	5	2	10

Table 2: Number of unique respondents vs. whether the author has previously contributed to the Linux project

While new members may provide a nearly complete implementation of their proposal, they face more resistance from the community about their approach. Table 3 indicates previous contributors receive positive feedback on their ideas more often than new members. This could be due to a few things, including new members' unfamiliarity with the project's internals, distrust of new members from the community, or new members having less coding experience. Regardless, new members have an increased barrier to acceptance and, as a result, must closely follow the community's conventions.

	Community opinion of the proposal		
Previous contributor	Positive	Negative	Totals
Yes	5	2	7
No	1	2	3
Totals	6	4	10

Table 3: Supportive feedback vs. whether the author has previously contributed to the Linux project

Next, I looked at what feedback contributors should expect from their proposals: specifically, whether feedback was about clarifying information missing in the proposal, responding to information in the proposal write up, or commenting on the author's approach as found in their code. To do this, I looked at the correlation between word count in the proposal and total word count of the author's responses. A negative correlation would suggest feedback is mainly composed of clarifying missing information, a positive correlation would suggest reviewer feedback is based mainly on the content present in the non-code portion of the proposal, and no correlation would suggest community feedback is primarily focused on the code implementation of the proposal.

As figure 1 shows, there is no correlation between the two word counts, indicating that feedback came from issues or questions about the code attached to the proposal. This was verified by reading the proposal feedback and noting the sections quoted by respondents.

Out of the sample, all proposals started with a 1-2 paragraph problem statement where they

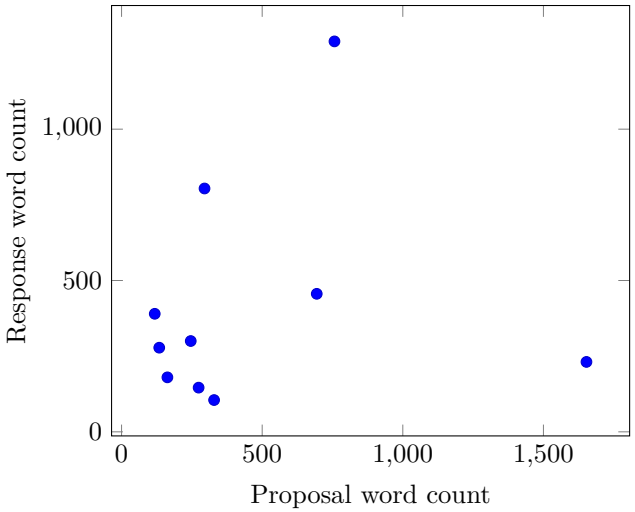

Figure 1: Proposal word count vs. response word count

indicated at a high level what issues or deficiencies were present in the linux source code. All but one of the proposals followed the problem statement with a 1-3 paragraph solution explanation which described their approach to solving the problem. They then ended with a short paragraph that contained a brief list of changes since the last version of their proposal or questions for reviewers about specific parts of their implementation.

4 Discussion

The proposals in this sample displayed a simple, formulaic structure, which is expected as the project handles hundreds of changes a month, so a standardized writing process would help reviewers quickly identify components of interest.

Proposals begin with a problem statement, which can be stated implicitly as in "This RFC focuses on changes needed to support A for B hardware," or it will be stated explicitly. In the explicit case, a problem statement will start with an undesirable situation (excessive CPU usage, high latency), explain the affected devices or configurations, and then suggest the change the author believes will solve that problem.

Problem statements are followed by solution explanations, which are often, though not always, the longest section of the proposal. Here, the author describes at a high level what their code does or intends to do. In the case of a new contributor, code that implements the change is expected, while previous contributors are allowed to simply describe their planned change.

A proposal will receive its harshest criticism when it is first submitted, especially if the author does not have a demonstrated track record with the Linux community. At this point, the author has a few options. If the feedback is mostly negative, they may decide that their proposal is not worth pursuing and abandon it. In the case that there are issues voiced over the way the author planned to implement their changes, the author can work with the

community to come up with a better plan and submit a follow-up proposal. Finally, if the community reaction is mostly positive, the author can begin working on a patch[9] to fully implement their proposal.

To go from the initial proposal (RFC) to a patch and finally to inclusion in the production version of Linux takes at least a year, assuming the proposal is accepted by the community. This allows for dozens of people to review changes and consider alternative approaches before committing the proposed change. In addition, it has the side effect of filtering out engineers not willing to work through the long process of developing a change and maintaining their code in future releases.

5 Conclusion

Software proposals submitted to the Linux kernel mailing list contain a problem statement followed by a solution explanation. Software engineers, whether they've contributed to Linux or not, are expected to understand the subsystem they are changing and be prepared to defend and alter their implementation choices. However, experienced contributors are not required to provide complete code with their proposals, whereas the community unofficially requires new contributors to submit well-written code to demonstrate their ability.

If students looking to contribute to open source projects that utilize mailing lists for communication are both aware of and able to follow this structure in submitting their work for review, they will be better prepared to anticipate a long, difficult development and review period while they establish themselves in the community. In order to generalize these findings to other forums of software development, future work can examine proposals found on GitHub.com or internal communication from software teams in corporations to look for similar biases against new contributors or new employees.

6 Appendix

6.1 Responses for Each Source

As a reminder, here are the questions being used. For the sake of brevity they will not be repeated for each source; instead, the number next to the answer indicates the question it responds to. Sources are grouped as subsections with a title generated from their subject line.

1. Has the author previously contributed code to the Linux kernel?
2. Has the author previously given feedback on other contributions?
3. Is this submission an early work in progress or mostly complete change?
4. Was there a problem statement?

[9] A patch is a set of files that describe the changes needed to a project in order to add in new code. They allow multiple people to create changes on the same code without knowing the changes others are making

5. Was there a solution explanation?
6. How many words was the proposal (excluding code)?
7. How many responses did the submission receive?
8. How many different people responded?
9. Did the responses indicate the proposal was a good idea?
10. Did the author reply within 24 hours to feedback from others?
11. On average, how many words did the author use to respond to first level feedback?
12. Was the code accepted and merged into the Linux kernel source?

Question	1[10]	2[11]	3[12]	4[13]	5[14]	6[15]	7[16]	8[17]	9[18]	10[19]
1	No	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes
2	No	Yes	Yes	No	No	No	Yes	Yes	No	No
3	MC	WIP	Early WIP	Early WIP	MC	Complete	Complete	WIP	WIP	MC
4	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
5	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
6	329	757	1653	295	694	134	246	163	118	274
7	3	11	4	6	8	7	36	3	5	4
8	3	6	1	2	5	6	5	3	2	3
9	Yes	Yes	none	Yes	No	No	Yes	No	Yes	Yes
10	Yes	Yes	No	Yes	No	Yes	Yes	Yes	No	Yes
11	105	215	77	268	228	139	228	180	390	73
12	Not yet	Not yet	Not yet	Not yet	No	No	Yes	No	No	Not yet

Table 4: Data collected from the Linux Kernel Mailing List Archive

References

[1] R. Stallman, *The GNU Manifesto*, 1987. [Online]. Available: `https://www.gnu.org/gnu/manifesto.en.html`,
Establishes the goals and values of the free and open source software movement. By responding to common concerns in a faux question and answer format, Stallman argues for the existence of software not tied to a corporate entity or used for profit to keep software free to use and change.

[10] NOVA file system, Source: https://lkml.org/lkml/2017/8/3/85
[11] I3C subsystem, Source https://lkml.org/lkml/2017/7/31/532
[12] Utilization estimation for FAIR tasks, Source https://lkml.org/lkml/2017/8/25/195
[13] Adding Virtual Box guest drivers to the mainline kernel, Source https://lkml.org/lkml/2017/8/25/195
[14] WhiteEgret LSM module, Source https://lkml.org/lkml/2017/5/30/376
[15] Check Tasks for Uninterruptible Sleep State, Source https://lkml.org/lkml/2017/8/21/208
[16] AMD Secure Encrypted Virtualization, Source https://lkml.org/lkml/2017/7/24/599
[17] Hardware Flow Offloading, Source https://lkml.org/lkml/2017/7/21/429
[18] Add new mdev interface for Quality of Service, Source https://lkml.org/lkml/2017/7/26/330
[19] Add the secdata Section to the Setup Header, Source https://lkml.org/lkml/2017/5/12/111

[2] J. Corbet and G. Kroah-Hartman, *2017 Linux Kernel Development Report*, 2017. [Online]. Available: https://go.pardot.com/l/6342/2017-10-24/3xr3f2/6342/188781/Publication_LinuxKernelReport_2017.pdf,
Shows the status of the Linux kernel project and its accomplishments in 2017. Using statistics about the developers and sponsors of the project, this document reveals the health of the Linux project, its impact on the industry, and hints at its future goals.

[3] M. McQuaid, *Contribute on Open Source Friday*, 2017. [Online]. Available: https://github.com/blog/2386-contribute-on-open-source-friday,
Introduces Github's Open Source Friday to the rest of the world. This blog post covers the reasoning behind supporting open source development on company time and the way employers can benefit from sponsoring it.

[4] *Apple Swift*. [Online]. Available: https://swift.org/contributing/.

[5] J. M. Canillas, *A Survivor's Guide to Contributing to the Linux Kernel*. [Online]. Available: http://events.linuxfoundation.org/sites/events/files/slides/klf2015_slides_javier_martinez_0.pdf,
Gives an overview of the process for contributing to the Linux kernel and goes of the different requirements for each stage of the process. Using the existing documentation as well as personal experience, Canillas provides helpful software commands and important steps for interacting with developers to help new members understand their duties.

[6] *Linux Kernel Mailing List*. [Online]. Available: https://lkml.org,
Stores an archive of email conversations on various Linux kernel mailing lists and provides access to individual messages. The offical method for communication in the Linux project is the mailing lists, so this archive provides information about all changes, suggestions, and debates about the project.

[7] *Linux - Elixir*. [Online]. Available: https://elixir.free-electrons.com/linux/latest/source,
Publishes source code for the Linux kernel and organizes it by version. By storing source code with hyperlinks to other files in the project, this site makes it easy to find relations between parts of code and definitions for variables, which is useful when reviewing how changes impact the structure and flow of data for the kernel.

[8] *Vim Pull Requests*. [Online]. Available: https://github.com/vim/vim/pulls,
Displays pull requests for the Vim project, a popular text editor in the free and open source software community. The pull requests here provide another example of proposals in computer science and how different communication mediums (email vs. GitHub) influence the amount and style of writing.

[9] *RFC Index*. [Online]. Available: https://tools.ietf.org/rfc/index,
Records requests for comments related to internet infrastructure and network security. This provides an archive and reference for internet specifications for organizations implementing the proposals in software or groups involved in setting standards for internet communication.

[10] *Early Stage*. [Online]. Available: https://github.com/torvalds/linux/blob/master/Documentation/process/3.Early-stage.rst,
Explains the recommended process for contributing to the Linux project with a focus on preparation before work begins. This is an official document in the source code file tree written by veteran contributors and kept up to date with process changes. It explains how to submit a proposal to the community and engage corporate sponsors and recieve feedback on ideas.