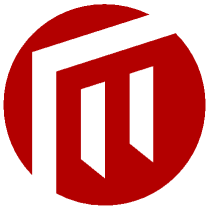


MALMÖ HÖGSKOLA

Teknik och samhälle
Datavetenskap

Examensarbete
15 högskolepoäng, grundnivå

The differences in requirement elicitation between community- and firm-driven open source software projects on Github.

Skillnaden i framtagningen av mjukvarukrav mellan community- och företagsdrivna open source projekt på Github.

Teddy Andersson
Filip Harald

Examen: kandidatexamen 180 hp
Huvudområde: datavetenskap
Program: systemutvecklare
Datum för slutseminarium: TBD

Handledare: Nancy Russo
Assisterande handledare: Aleksander Fabijan
Examinator: Carl Magnus Olsson

Sammanfattning

Kunskap om olika utvecklingsmetoder vid start av ett nytt mjukvaruutvecklingsprojekt är avgörande för utvecklarna, styrorganen och slutprodukten. Därför prioriteras ofta nya och okända metoder ned för att säkerställa att arbetet blir gjort och att lösningen kommer att levereras i tid och med hög kvalitet. Detta beteende gör på lång sikt att mjukvaruutvecklingsprojekt går miste om nya och bättre utvecklingsmetoder.

För belysa nya utvecklingsmetoder och upplysa de som behöver, valde vi att undersöka skillnaderna i krav framställning inom området Open Source Software(OSS)[1]-utveckling. I vårt arbete ställer tre forskningsfrågor som ska belysa ämnet dessa bevarar vi genom att utföra en fallstudie. I fallstudien undersöker vi hur och av vilka som krav framställts i ett företagsstyrt projekt jämfört med ett projekt drivet av en frivilligorganisation.

Fallstudien visade att externa användare i frivilligorganisation OSS-projekt har lägre delaktighet, det vill säga bidrag till projektartefakter, jämfört med företagsdrivna projekt där deltagandet av externa användare är högre. Slutligen diskuterar vi implikationerna av resultaten för både OSS-projekt drivna av företag och frivilligorganisationer. Vi kan för båda styrorganen dra slutsatsen att det är möjligt att öka både utvecklingshastighet och produktens värde för kunden.

Keywords—open source software, development process, requirement elicitation, git, Github, Atom, Neovim

[1] öppen källkods mjukvara

Abstract

Knowledge about different development methods when starting up a new software development project is crucial for the developers, the governing bodies and the end product. Therefore new and unfamiliar options are taken out of the equation to make sure that the work gets done and that the solution will be delivered on time and with high quality. This behaviour in the long term does, however, exclude new and better ways of executing the work in the process.

To shine light upon new development methods and enlighten those who are in need of insight into a new viable option we chose to investigate the differences in requirement elicitation within the area of Open Source Software development. By examining how and by who requirements are elicited in a firm-driven project compared to a community driven project, we framed a total of three research questions to base our case study on.

The case study showed that in community driven Open Source Software projects external users have low participation, in other words contributions to project artefacts, compared to firm-driven projects where the participation of external users is high. Finally, we discuss the potential implications of the findings for both community- and firm-driven OSS projects. We could conclude for both types that it's possible to increase both development speed and customer product value.

Keywords—open source software, development process, requirement elicitation, git, Github, Atom, Neovim

Contents

1 Introduction

Today we are constantly introduced to new software applications and systems that we install on our smartphones, tablets and computers. Behind each software that we choose to install and use on our devices is a development process with several defined steps to be completed before we can put our hands on the software and try it out. The steps that take a software from an idea to a final product are many and the different methods to solve this process are often hard to choose from for those who are set to develop the software. Choosing a method to follow throughout this process is therefore important in the aspects of expenses, time and software quality. Despite the choice of method there will always be a constant work to understand the user needs, from the needs necessary functionality can be written down in paragraphs and modelled in diagrams, in the area of computer science we call these paragraphs requirements, so the software developers know what to implement into the software during the process.

The elicitation of these requirements in a development process based on an Open Source Software(OSS) development method, a development method where the source code of a software is available and open to anyone, are faced with several challenges. First, Curtis et al [11] explained in 1986 that software development is a knowledge-intensive activity with a large number of potentially confounding factors, in 2016 Minghui et al. [26] further explain that this makes it difficult or impossible to discern the impact of commercial involvement from different types of companies. Second, Minghui et al. [26] noted that to observe the impact of commercial involvement, it is important to compare the differences in contributions from developers, and to observe the work impact on projects over a long period of time. Third, there is no easy way to learn companies' and individuals' intentions motivating their involvement in the Open Source Software (OSS) community, and it is even more difficult to examine the effects of such involvement.

The challenges described above are key factors for this thesis purpose, to enlighten the decision of developing software with an open source software (OSS) development approach. By providing this information about governing bodies, regardless if there is a firm or a community managing the project, the initial decision of choosing a software development approach could be made easier. This thesis will however not make a comparison between open source and proprietary software. Therefore this thesis will also not provide pros and cons of the transition from proprietary software to OSS. Bottom line, the thesis will help governing bodies realise that OSS truly can deliver quality software, enthusiastic developers, better customer relationships and commercially available software.

This thesis presents a comparative case study of differences and commonalities in requirement elicitation between firm-driven and community-driven OSS projects on Github: Atom and Neovim. We address key questions about their differences overall and within the area of requirement elicitation, based on data gathered from Github. Based on the work of [20] and [22] we framed a number of hypotheses that we conjectured would be true generally for both development and requirement elicitation within the area of OSS-development.

As a result of this study we will have made the following three contributions. (a) When studying the characteristics of two OSS-projects the results will of course contribute to the general knowledge about OSS development. (b) As mentioned above our results can be used as support for a firm when deciding if they want to use OSS development. (c)

Lastly the scripts we have programmed and used to process the project artefacts of the two projects will contribute by enabling others to do similar studies or recreate our study. These scripts are modular and can be used to retrieve information about a large group of open source projects.

1.1 Background

> *"A great babbling bazaar of differing agendas and approaches out of which a coherent and stable system could seemingly emerge only by a succession of miracles."* [3]

In the evolution of software development different types of methods have been used to manage initiation, planning, execution, monitoring and closeout of a software project. The project methods which were first to adopt these phases are known as traditional development methods.

The traditional development approach identifies a sequence of steps to be completed, where the work of a new step doesn't begin until the previous step is completed. However, methods were not optimal for software development, where the need for continuous integration is important [14]. Therefore on February 11, 2001, at The Lodge at Snowbird ski resort in the Wasatch mountains of Utah, 17 people met to talk, ski, relax and try to find common ground [12]. What emerged from this meeting was the Agile Software Development Alliance. From this alliance, the Agile methods evolved based on a manifesto where the effort was to overcome perceived and actual weakness in conventional software engineering [12]. This alliance wrote down the famous Agile Manifesto and their reason was simply to uncover better ways of developing software by doing it and helping others do it [12].

In contrast to the traditional methods the agile methods are based on incremental methods, managing the design and build activities of engineering [14]. Agile methods are also claimed to encourage developers to be more flexible and efficient by means of arrangements in the development teams physical and social environment [14].

Apart from the traditional and agile methods the open source software (OSS) development method started to receive a lot of attention in the late 90's. OSS development is the process by which open-source, or similar software whose source code is publicly available, is developed. These are software products available with its source code under an open-source license to study, change and improve its design. For this reason, OSS was in the late 90's characterised as a fundamentally new way to develop software and was seen as a challenge to the commercial software business that dominated the vast majority of the software market [20]. Over time, OSS evolved into a method that can be applied to develop commercial software and to maintain quality. Examples of some popular open-source software products are Mozilla Firefox, Google Chromium, Android, Linux, the Apache OpenOffice Suite, Atom and Neovim. Open-source software development has been a large part of the creation of the World Wide Web as we know it, with Tim Berners-Lee contributing his HTML code development as the original platform upon which the internet is now built [8].

Regardless of the development process a project chooses to use when developing a software system they would most likely need a way to handle requirements in one way or another. While requirements in traditional and agile development processes are elicited

from discussions with a client/customer, OSS projects handle the elicitation through the users/developers that are involved in the OSS community. People in these communities cooperate via the Internet and never, or seldom, meet face to face. The number of developers can differ from a handful to thousands and are often geographically distributed [18]. Krishnamurthy et al. [17] describe the two vital roles of developers in an OSS project, where the so-called core developers are developers with a substantial, long-term involvement, and in an abstract sense these developers play an essential role in developing the system architecture and forming the general leadership structure. In contrast, peripheral developers are typically involved in bug fixes or small enhancements, and they have irregular or short-term involvement. Krishnamurthy et al. further describe that the motivation for both core and peripheral to get involved into OSS projects are based on the reward that they can get from signalling to stakeholders with eyes on OSS projects, and the OSS area in general. The signaling could reward the developer with a job or better reputation, which is a large reward for contributing to a project which they most likely find interesting to work on.

In recent years, we have seen an increased interest in using open source alongside with the Agile development [9, 14]. Large-scale companies like Google, Microsoft, Facebook etc have been striving towards a more open environment in both development projects but also in general work process [25]. Alongside with an interest and momentum of OSS it is now also considered to be, in a commercial setting, a more viable approach [9]. From a commercial standpoint OSS development also promises many advantages. Including reduced salary costs; reduced cycle time arising from' 'follow-the-sun" software development; cross-site modularisation of development work; access to a larger skilled developer pool; innovation and shared best practice; and closer proximity to customers [9].

1.2 Related Work

In previous work within the area of OSS and requirement elicitation there are two highly relevant studies for this thesis. The first article [20] presents two case studies of the development and maintenance of major OSS projects: the Apache server and Mozilla. Mockus et al. [20] address key questions about the development process and about the software that is the result of the processes. Based on results from an earlier study made on Apache, covered in [19], they framed a number of hypotheses that they conjectured would be true generally of open source developments. The second case study in the article [20] examined data from the Mozilla project based on the analyses and hypotheses that were framed from the Apache project. Mockus et al. came to the conclusion that the essential differences by which elements of commercial and open source processes coordinate, select, and assign work could be combined. To make their generalisation trustworthy they selected the two cases which can be considered typical instances for an OSS-project, Apache and Mozilla. The authors present 6 research questions they used to compare the projects [20]. The 6 questions are presented in Table 1.

1. What were the processes used to develop Apache and Mozilla?
2. How many people wrote code for new functionality? How many people reported problems? How many people repaired defects?
3. Were these functions carried out by distinct groups of people, that is, did people primarily assume a single role? Did large numbers of people participate somewhat equally in these activities, or did a small number of people do most of the work?
4. Where did the code contributors work in the code? Was strict code ownership enforced on a file or module level?
5. What is the defect density of Apache and Mozilla code?
6. How long did it take to resolve problems? Were high priority problems resolved faster than low priority problems? Has resolution interval decreased over time?

Table 1: Research questions for exploring OSS-projects [20].

In the second highly relevant article, [22], Noll et al. presents a case study of the OpenEMR, an open source project developing electronic medical record (EMR) software. The goal of the study was to understand how requirements are elicited, documented, agreed and validated in a small open source software project. The study followed the same approach as an earlier study of the Firefox web browser [21]. The comparison between the both projects showed that, similar to the Firefox study, the majority of OpenEMR requirements are asserted by developers. Documentation was informal, consisting mainly of archived discussions. Contributors to the OpenEMR project were medical practitioners such as doctors or clinic administrations, who use the product in their practices and also developers which do not use the product at all. The implication for software development, in general, is that developers can be a significant source of innovation. In order to conduct the study the authors present a method including 5 steps for retrieving the data upon which they will make the comparison. The method is presented in Table 2.

1. Identify the set of features delivered for OpenEMR after release 2.8.0, up to and including release 2.9.0.
2. Select a subset of these features for examination.
3. Examine Internet resources related to OpenEMR, such as archives of discussion forums, the OpenEMR issue database, the OpenEMR "tracker" on Sourceforge.net, and other online forums, to discover when the feature was first proposed, and what role the person proposing the feature played (such as user or developer).
4. Determine the initial implementation of the feature (prototype by a developer, patch submitted to the tracker, or enhancement committed directly to the codebase).
5. Categorise the requirement as asserted by a developer, either from his or her personal experience or knowledge of user needs; proposed by an end-user, for example by posting a request to one of the discussion forums, or filing a bug report or "Request for Enhancement" in the issue database; or derived from features found in competing products.

Table 2: 5 steps for gathering data from OSS-projects [22].

The articles together summarise the differences between open-source projects in different aspects. Mockus et al. [20] investigate two cases from which they try to form theories on what it is that defines OSS-projects. They conclude the study by comparing their results with commercial, proprietary, projects. Noll et al.OSS-projects [22] on the other hand investigate the differences in requirement elicitation between large and small OSS-projects. These articles are the two most relevant studies in comparing requirement elicitation in OSS projects due to the contributions in form of methods that can be applied to several aspects in comparing different types of OSS-projects.

1.2.1 Case Study

Oates refers to a case study as being a research strategy in which one tries to describe a case, or phenomenon, from within its natural context [23]. Rather than just confirming a phenomenons existence, which could be done by for example conducting a survey, a case study aims to discover why it occurred. This is done, as a researcher, by aiming to give a detailed description of not just the phenomenon but also its context. It is from the context the researcher can identify factors which might have caused the phenomenon to occur. The number of factors varies from case to case and the probability of multiple causing factors should be taken into account.

Oates says that case study can favourably be used when "studying the 'life' on the Internet" [23]. However, the author mentions particular issues with conducting this kind of research. The first is the problem of the boundary. Namely, how does one define

the boundaries for which the study is being conducted. The second is the problem of offline/online existence. That is one might not get all the details for a case by only studying it online, since some information might only be available offline, such as communications between participants of the studied case. These problems are taken into consideration when constructing our method for conducting the case study. However, the first problem will have minimal effect on our research since it's easy to define what is part and not of the development process of a product and what is not. The second problem, on the other hand, could affect our research to a greater extent. The nature of OSS is that everything related to the project is stored and displayed online. But even though developers might never meet face-to-face, they may still communicate on other channels than those provided through the project. These channels are not always openly displayed online and can therefore be seen, from a researcher's perspective, as offline.

1.3 Research Question

Based on our research interests in studying the commonalities and differences between community and firm driven OSS projects, we present our research questions below. The first research question (RQ) addresses the characteristics of community-driven OSS development. The second RQ addresses the characteristics of firm-driven OSS development. The third and last RQ addresses the results of the first two and what they imply.

RQ1 *What are the characteristics of community-driven OSS development?*

RQ2 *What are the characteristics of firm-driven OSS development?*

RQ3 *What are the implications of the differences between firm- and community-driven OSS development?*

We answer the first two questions by conducting a case study on two OSS-projects with different governing bodies, a firm-driven and a community-driven. We answer the third research question by analysing the results from the first two.

The remainder of this paper is organised as follows: the next section will present our systematic literature review which we conducted to familiarise ourselves with the subject; the next section will present the chosen method for this study; next are the results; they are followed by an analysis and discussion of their implications; lastly we present a conclusion and suggestions for future work.

2 Systematic Literature Review Process

To ensure that our thesis is credible we first have to make sure that we are well informed on the subject we're researching. In order to do this we have performed a systematic literature review which is presented in this section.

When searching we used three different databases: *ACM Digital Library* (`ACM`), *IEEE Xplore Digital Library* (`IEEE`) and *Google Scholar* (`GOOGLE`). The first two, `ACM` and `IEEE`, are well-known within the area of computer science and they were used for searching for articles using keywords and search phrases. The last, `GOOGLE`, was used for forward and backward snowballing.

When searching in `ACM` and `IEEE` we used different combinations of keywords and sometimes filters to refine the search. As we read more papers on the subject the list of keywords was extended. We used the following keywords: *"open source", open, source, success, software, development, community, requirements, corporate, firm, project, "case study"* and *case, study*. Used search phrases with their results are presented in table 3.

Search Phrase	**Filters**	`ACM`	`IEEE`
"open source" AND success	-	333	330
"open source" AND success	only matching in title	6	21
open AND source AND software AND development	-	2618	260
open AND source AND software AND development	only matching in title	99	0
open AND source AND development AND community	-	1129	684
open AND source AND development AND community	only matching in title	25	0
open AND source AND development AND requirements	-	734	137
open AND source AND development AND requirements	only matching in title	5	0
"open source" AND corporate AND project	-	1689	28
"open source" AND corporate AND project	only matching in title	67	0
"open source" AND firm AND project	-	20	2264
"open source" AND firm AND project	only matching in title	0	45
"open source" AND firm AND community	-	14	1412
"open source" AND firm AND community	only matching in title	0	39
"open source" AND corporate AND project AND community	-	540	0
"open source" AND corporate AND project AND community	only matching in title	15	0
Evolution in open source software AND case	only matching in title	0	169
case AND study[2]	only matching in title	384	156

Table 3: Used search phrases and filters and amount of results.

When a number of search results were maximum 100 we went through the studies doing the first pass of Keshavs *three-pass approach* [16]. If the article was found interesting we added it to our reference manager to later perform the second and third pass on. For each article which after the second pass still was found relevant we noted the references to snowball more articles, this process is described in the next section.

We used `GOOGLE` to snowball more studies. This was done in three parts for every study: (1) search for all references used in the study, (2) search for all studies that referenced to

[2]We read through most of these even though they were more than 100

the study and (3) search for more studies written by the same author(s). Searches in GOOGLE sometimes generated a huge amount of results and also the results did not always have the article itself available for download. In those cases we used ACM and IEEE to limit the result or find the article. Keshavs *three-pass approach* was also here used for selecting relevant articles [16]. All our relevant studies are presented in Table 4.

Name of Article	**How it was found**	**How many articles was snow-balled from this**	**Used search phrase**
A Case Study of Open Source software Development: the Apache server	ACM or IEEE	1	open AND source AND software AND development
Agila projektledningsmetoder och motivation	GOOGLE	0	open source + konventionell + project + management + case study
An analysis of requirements evolution in open source projects: recommendations for issue trackers	ACM or IEEE	0	open AND source AND development AND requirements
Balancing act: community and local requirements in an open source development process	ACM or IEEE	0	open AND source AND development AND community
Case Study Research: Design and Methods	Snowballed	0	-
Challenges and recommendations for the design and conduct of global software engineering courses: A systematic review	Snowballed	0	-
Classifying Developers into Core and Peripheral: An Empirical Study on Count and Network Metrics	Personal recommendation	0	-
Comparative Case Studies	Snowballed	0	-
Evolution in open source software: a case study	ACM or IEEE	0	Evolution in open source software AND case
Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code	Snowballed	2	-
Free/Libre open-source software development	ACM or IEEE	1	”open source” AND success
From a Firm-Based to a Community-Based Model of Knowledge Creation: The Case of the Linux Kernel Development	Snowballed	0	-
How Can Open Source Software Development Help Requirements Management Gain the Potential of Open Innovation: An Exploratory Study	ACM or IEEE	0	open AND source AND development AND requirements
How Google Works	Snowballed	0	-
Inflow and Retention in OSS Communities with Commercial Involvement	ACM or IEEE	0	”open source” AND corporate AND project AND community
Information systems success in free and open source software development: Theory and measures	Snowballed	0	-
Innovation Model : Issues for Organization Science Open Source Software and the ”Private-Collective” Innovation Model : Issues for Organization Science	Snowballed	0	-
Open Sources: Voices from the Open Source Revolution	Snowballed	0	-
Peripheral Developer Participation in Open Source Projects	ACM or IEEE	1	”open source” AND corporate AND project
Requirements acquisition in open source development: Firefox 2.0	Snowballed	0	-
Requirements elicitation in open source software development: a case study	ACM or IEEE	3	open AND source AND software AND development
Researching information systems and computing	Course literature from previous course	0	-
Software Psychology: The Need for an Interdisciplinary Program	Snowballed	3	-
The agile manifesto	GOOGLE	0	The + agile + manifesto
Two case studies of open source software development: Apache and Mozilla	Snowballed	0	-
Why do users contribute to firm-hosted user communities? The case of computer-controlled music instruments	Personal recommendation	1	-

Table 4: Shows the names of the articles relevant to our thesis and how they were found.

Out of the studies presented in Table 4 there are seven which we mainly use in our thesis. The studies and how they were found are presented below.

1. **Software Psychology: The Need for an Interdisciplinary Program [11]**

   The search phrase `acmdTItle:(+"open source" +corporate +project +community)` on `ACM` resulted in 15 hits. One of them was [26] which after the second pass we still found relevant. When going through its references we found the following relevant articles: [24], [13] and also [11] which is used in our thesis.

2. **Agila projektledningsmetoder och motivation [14]**

   When researching on how we could introduce the reader to agile project management and open source we also used `GOOGLE` to do some searches in Swedish, our native language. The search phrase `(''open source'' + konventionell + projektledning + ''case study'')` then resulted in 8 hits. One of the was [14] which we found relevant in the first pass.

3. **Classifying Developers into Core and Peripheral : An Empirical Study on Count and Network Metrics [15]**

   This article was recommended to us by our supervisor. Other than [15], the references also yielded [10].

4. **Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Codeă [18]**

   This article was recommended to us by our supervisor.

5. **Requirements elicitation in open source software development: a case study [22]**

   This article was found out of the 99 hits we got from `acmdlTitle:(+open +source +software +development)`.

6. **Two case studies of open source software development: Apache and Mozilla [20]**

   The search phrase `acmdlTitle:(+open +source +software +development)` on `ACM` resulted in 99 hits. One of them was [22] which after the second pass we still found relevant. When going through its references we found the following relevant articles: [21] and also [20] which is used in our thesis.

7. **Outsourcing to an Unknown Workforce: Exploring Opensourcing as a Global Sourcing Strategy [9]** The search phrase `ACM: acmdlTitle:(+"open source" + corporate + project)` on `ACM` resulted in 67 hits. One of them was [17] which after the second pass we still found relevant. When Going through its references we found the relevant article: [9] which is used in our thesis.

3 Methodology

In section 1, we introduced OSS development and related studies comparing the different ways that these projects are executed. In this section, we present our research methodology.

The purpose of this study is to compare two different types of governing bodies for OSS-projects, firm driven and community driven. And with these results try to identify what they imply. For OSS projects, in general, all information regarding development process is public, this includes both documentation and code. The communication between project participants also tends to be public. This public access to data creates an opportunity for researchers to study the development process over time. With all this detailed information of a project's development process we have chosen to conduct a comparative case study on two OSS-projects, one of each type of governing bodies.

With all the given data mentioned in the previous paragraph one could argue that conducting a survey would be more suitable. However, such a survey would risk to miss out on detailed information that is unique for each project which a comparative case study would be able to discover.

In the subsections below, we first present the two projects we have chosen for our case study, followed by a motivation for why we choose them. Next, we describe what a case study is and continue by presenting previous case studies that were similar to ours. Next, we present what we will do in our case study, both how we will gather data and what data we will gather. Finally, we present what tools we will use, how we will analyse the data and present identified threats to the validity of our report.

3.1 Case Description

This section describes the two projects that we have conducted our case studies on. The two projects we investigated, Atom and NeoVim, are open-source text editors, programs that are used for editing text files such as the source code of a software, that are free to use and the source code from these projects is available to investigate on GitHub.

3.1.1 Case 1: Atom

Atom is a free and open-source text and source code editor for macOS, Linux and Microsoft Windows. The Atom project was initialised by Github on Github on August 14 2011 and has since then grown to be a very popular text editor for developers all around the world. The project has today a total of 349 contributors that have made one or more contributions from the start of the project. Atom is licensed under the MIT license[3]. Github is still in control of the development process. Atom also have a huge amount of extending packages developed by its own users under free software licenses and are community-built and maintained. Atom was released from beta, as version 1. 0, on June 25, 2015. Its developers call it "a hackable text editor for the 21st Century" [2, 1].

[3]A short and simple permissive license with conditions only requiring preservation of copyright and license notices

3.1.2 Case 2: NeoVim

Neovim is a free and open-source text and source code editor for MacOS, Linux and Microsoft Windows. The Neovim project started in 2014 on Github, with some Vim community members offering early support of the high-level refactoring effort to provide better scripting, plugins, and integration with modern GUIs. Vim (Vi Improved) is a text editor which was first released in 1991 under a special license compatible with the GNU General Public License[4] It was made to streamline creating and changing text-files. Neovim is, therefore, a refactor of Vim and strives to be a superset of Vim except for some intentionally removed features. It is built for users who want the good parts of Vim, and more. The Neovim Project today is a total of 309 contributors that have been involved in writing code in one way or another. Neovim is driven by an independent community that is able to support the project through monthly subscriptions. Neovim had its first public release on 1 November 2015 [6, 7].

3.1.3 Motivation for Selection of Cases

To ensure our results reflect only the requirement elicitation process of the projects we selected projects that are similar in a number of aspects, especially on product type, project age and project size. Both projects develop a product that is classified as a text-editor. Atom has a more graphical user interface than Neovim, but they are still both used for the same purpose. The projects were created five (Atom) and three (Neovim) years ago. When comparing project size we used the data available on the main page for each project on Github [1, 4] and also counted project Lines of Code (LoC) , which is a common metric for determining how big a software is, and number of files. The data is presented in the Table 5.

Data	**Atom**	**Neovim**	**Largest**	**Difference**
Contributors	349	303	Atom	46
Commits	31382	7925	Atom	23457
LoC	57089	246741	Neovim	189652
Number of files	388	579	Neovim	191

Table 5: Project data found on Github as of March 2017 [1, 4].

Based on the data presented in table 5, we see that (1) the projects are similar in relation to the number of contributors, (2) both projects have several thousands of commits, and (3) both projects consists of several hundred of files.

With Atom having more than 3 times as many commits as Neovim there is a difference. However, the usage of how one would commit code could differ between the two projects. For example the amount of LoC committed could be approximately the same even if amount of commits differ.

Lastly, is the difference in the amount of LoC and files. As mentioned in section 3.1.2 Neovim is a product that uses a lot of the functionality from Vim. The result of this is that Neovim has a lot of code which has been transferred from Vim. Vim has 328722 LoC and 230 files. This justifies the difference in the LoC.

[4] A free, copyleft license for software and other kinds of works.

The two addressed differences were considered when constructing and conducting the case study. And even with these differences, Atom and Neovim are the most similar firm- and community-driven OSS projects of the ones we considered in our case study. Other candidates were, Google Chrome and Mozilla Firefox (difference in age and location of the codebase) and Codelite instead of Neovim (Codelite was considerably smaller and older than Atom).

3.2 Data Collection

3.2.1 Sources

Several data sources have been used to collect different types of data for this thesis. The biggest source of data for this thesis is Github, a web-based version control and collaboration platform for software developers, which have provided us with data from both Atom and Neovim projects. The data is, for example, project participants, developer discussions, contribution history and feature suggestions. Data from Github have been used on two occasions. The first occasion is the gathering of overview data from the Github API for both Atom and Neovim. The second occasion is gathering of detailed data from Atom and Neovim. The data gathering for the both issues is presented in section 3.2.2.

We have also turned to Atom and Neovim's forums for development discussions. Within these forums proposals for features and issues alongside with instructions and tips to the users can be found. Atom uses its own discussion forum that can be found on their official website [2]. Neovim, on the other hand, used both Reddit[5] and Github as discussion channels. The gathering of this detailed data is presented in section 3.2.2.

3.2.2 Data Schema

Below we present a schema for what data we will collect with its description. The schema consists of two parts The first, D1-D4, is overview data that can be used to compare the two projects themselves and with other popular OSS projects. The second part, D5-D8, is sampling data focusing on the requirements elicitation process of the two projects. A more extensive description and technical definition of each overview data point can be found in Appendix A.

[5]Reddit is a social news aggregator, web content rating, and discussion website [5]

Overview Data

D1 *Code contributors*

How many contributors are there in the project, and what is their distribution? We define a contributor as someone who has written code to the project.

D2 *Feature proposers*

How many individuals proposed new features that got implemented to the project and what is their distribution?

D3 *Problem reporters*

How many people reported problems they had with the product? And how was the distribution of problem reports over these people? (Did everyone report an equal amount of problems?)

D4 *Defect repair time*

How long did it take for a reported problem to be repaired?

Detailed Data

D5 *Feature proposals*

Where were features first proposed?

D6 *Feature origin categories*

Which of these categories did the feature belong to?

a) asserted by a developer, either from his or her personal experience or knowledge of user needs.

b) proposed by an end-user, for example by posting a request to one of the discussion forums, or filing a bug report or "Request for Enhancement" in the issue database.

c) or derived from features found in competing products.

D7 *Feature acknowledgement*

When and by whom was a feature acknowledged?

D8 *First implementation of feature*

When and by whom was a feature first implemented?

3.2.3 Collection process

Below we present a schema for our method of collecting the data. Every step which will generate data will refer to for which which of the data presented in Section 3.2.2 it will be. We are then referring to the schema found in section 3.2.2.

1. Select a time period between release M and N for project X. (Where $M < N$)

   Project X is either Atom or Neovim. M and N are two versions of the software developed in project X. In order to compare the two projects M and N should be approximately within the same timespan.

2. Extract the data specified in items D1-D4 for the selected time period.

3. Identify a set of features delivered for project X after M, up to and including N.

4. Select a subset of these features for examination.

5. Examine resources[6] related to the project to discover how the feature was first proposed, what role the person proposing the feature played, and to what category the proposal belongs to. (D5-D6)

6. Determine when the feature was acknowledged as a requirement. (D7)

7. Determine when the feature was submitted. (D8)

3.2.4 Tools

In our research we make use of two tools that need further explanation. They are the Github API[7] and the scripts that we programmed for the purpose of this research. First, we describe the Github API, then we will describe the scripts.

In order to retrieve the desired data from the two projects we will use the Github API. This is a service offered by Github for users to programmatically extract, and to some extent process, data hosted on Github. The API is accessible with the use of HTTP[8]. The API have different addresses, called end-points, for different kinds of data. Our research requires us to make use of multiple end-points to have enough data.

To minimise the risk of making manual HTTP-request (in the command prompt) or manual processing of data we wrote scripts to automate as much as possible. The scripts are written in the programming language Python. Our scripts are described in Appendix B.

3.3 Threats to Validity

When conducting a case study it is important to select a case that is representative for the researched phenomenon in order to use the results to draw generalised conclusions. If a representative case is not selected this poses a threat to the external validity of the thesis. This is because our results could then not be used to draw a generalised conclusion. To minimise this threat we collect the overview data, D1-D4 in section 3.2.2. This data can

[6] Project resources are presented in section 3.2.1.

[7] Application Programming Interface

[8] Hyper Text Transfer Protocol, one of the most common protocols used on the Internet today.

be used to compare to other projects and the projects that are researched in the previous studies, [20] and [22].

Furthermore, it is important when selecting cases that they have similar attributes that are not the focus of the study. In our study we want to see the differences depending on the governing body and not the size of the project, product type or project age. Ensuring the similarity on these factors increases the internal validity of our study. This is initially ensured in section 3.1.3 and also with the overview data[9].

Lastly, a threat to validity when conducting a case study is that the researcher affects the results by being involved in the context for which he/she researches [23]. This, however, is not applicable to our case since we are not researching on a present phenomenon but rather on something that is in the past. Therefore we do not consider it as a threat to our study.

3.4 Data Analysis

The analysis of the data will consist of three parts, which will be described in the following paragraphs. They are: (1) validate the selection of cases, (2) identify similarities and differences in the requirement elicitation process and (3) other potential findings.

In section 3.3 we mentioned that selecting two different projects will greatly affect the results of this study. Therefore we start by analysing the results from overview data, to compare the two projects on both size compared to each other and also similarities with other OSS projects.

In order to answer our first two research questions, RQ1 and RQ2, presented in section 1.3 we need to analyse the results from detailed data. From the analysis of this data combined with other potential findings we will draw a conclusion that can answer the last research question, RQ3.

When conducting a case study it is possible that one can discover things that were not initially intended to research. In our case that would be if we discovered something not related to the data types presented in section 3.2.2. These discoveries we categorise as other potential findings.

[9] D1-D4 in section 3.2.2

4 Results

In the previous section 3, we presented our research methodology. In this section the results from the methods specified in section 3 are presented. They are presented in the same order as they occur in section 3.2.2 which is first the overview data and then the detailed data. Lastly we will include a small summary of the results.

4.1 Overview Data

In this section we present the overview data results for Atom and Neovim and also other popular OSS projects. We also write a brief description of our observations of the results.

4.1.1 Code Contributors (D1)

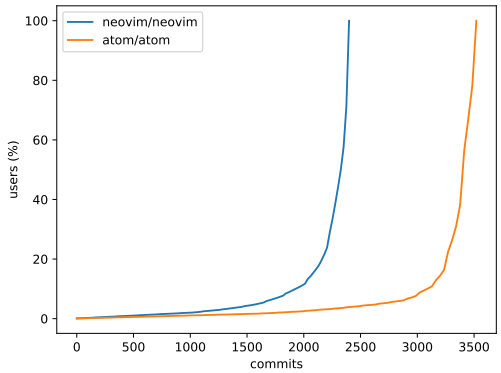
The number of code contributors was similar for the two projects as seen in Table 6. Around 5% of the code contributors did approximately 90% of the contributions, as seen in Figures 1a, 2a and 3a. There is a difference between amounts of commits between the two projects as shown in Figure 1a. And there is a similarity in amount of additions and deletions in LoC, Figures 2a and 3a. Compared to other popular OSS-projects both Atom and Neovim can be considered similar as seen in Tables 1b, 2b and 3b. Note that the tables comparing to other OSS-projects is with percentage on the X-axis as well. This is due to the fact that the other projects vary in size and not using percentage would make the comparison more difficult.

Table 6: Showing amount of code contributors.

Data Type	**Atom**	**Neovim**
Amount of contributors	128	113

Figure 1: Commits distribution compared to...

(a) ...each other.



(b) ...other popular OSS-projects.

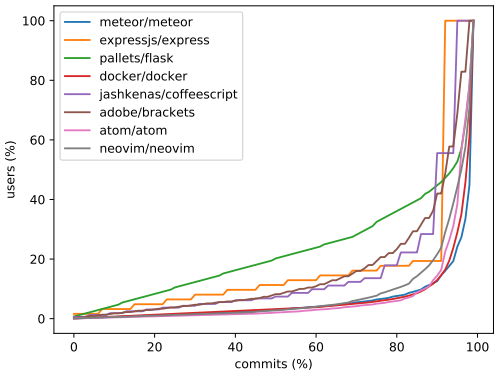
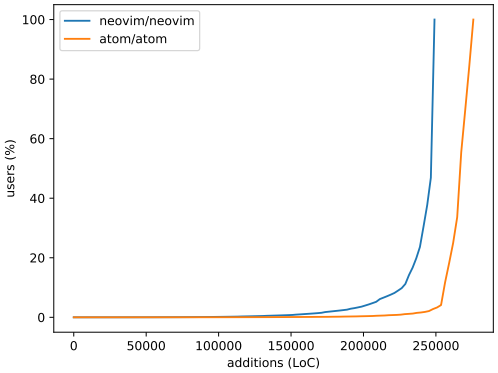


Figure 2: Additions distribution compared to...

(a) ...each other.



(b) ...other popular OSS-projects.

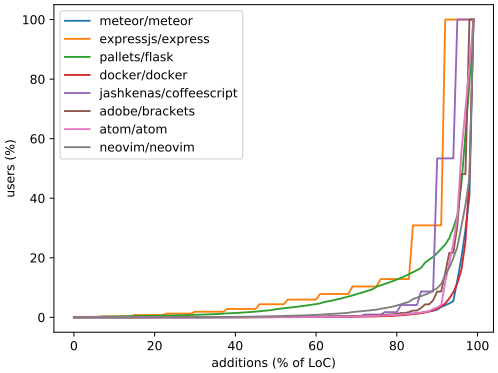
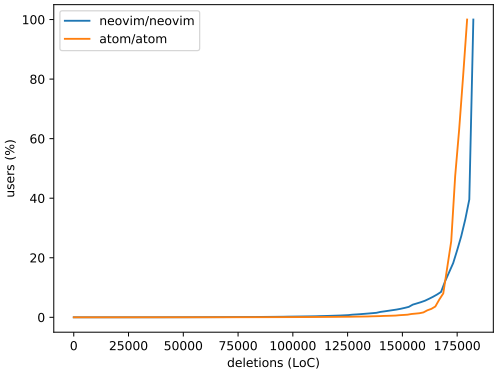
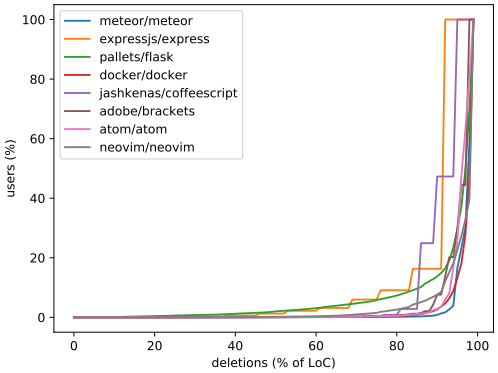


Figure 3: Deletions distribution compared to...

(a) ...each other.



(b) ...other popular OSS-projects.



4.1.2 Feature Proposers & Problem Reporters (D2 & D3)

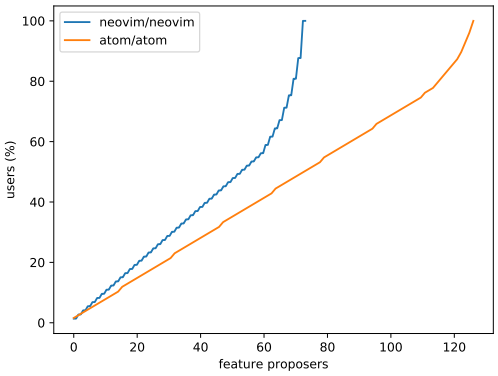
The amount of feature proposers were 108 for the two projects as seen in Table 7. The amount of problem reporters were 50 for the two projects as seen in Table 7. A majority, around 60%, of the feature proposers and problem reporters did more than one proposal or report and the remaining 40% did only one report each as seen in Figures 4a and 5a. Compared to other popular OSS-projects Atom and Neovim can be considered similar as seen in Figures 4b and 5b.

Table 7: Showing amount of feature proposers and problem reporters.

Data Type	Atom	Neovim
Amount of feature proposers	108	50
Amount of problem reporters	230	141

Figure 4: Distribution of feature proposals over users compared to...

(a) ...each other.



(b) ...other popular OSS-projects.

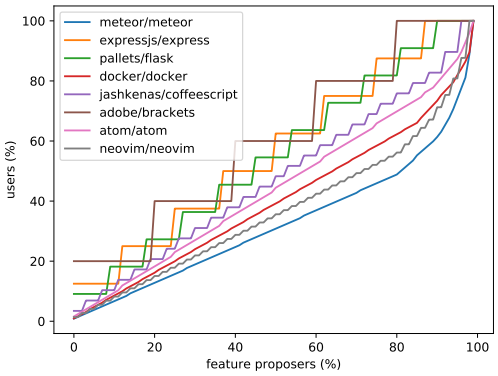
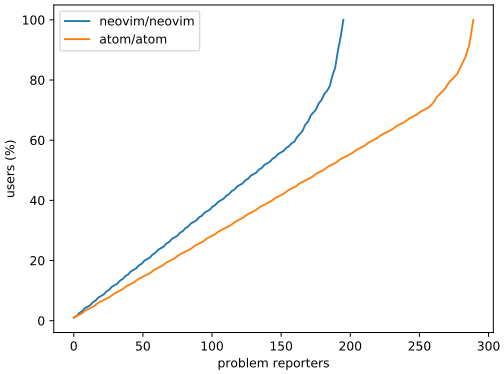
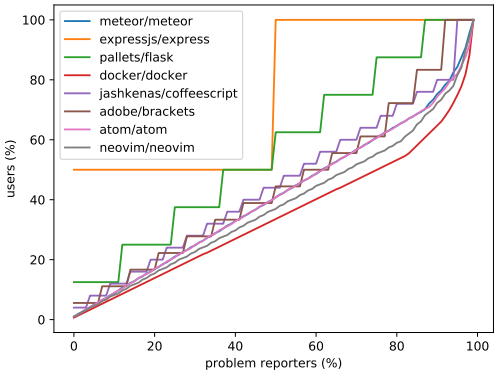


Figure 5: Distribution of problem reports over users compared to...

(a) ...each other.



(b) ...other popular OSS-projects.



4.1.3 Defect Repair Time (D4)

The two projects differed in the amount of defects repaired and the repair time, with Atom having almost twice as many defects, seen in Table 8 and a longer repair time, seen in Table 9.

Table 8: Showing amount of reported defects.

Data Type	**Atom**	**Neovim**
Amount defects	271	148

Table 9: Showing repair time measured in days.

Data Type	**Atom**	**Neovim**
Average	200	151
Median	81	82
Highest	1098	807
Lowest	<1	<1

4.2 Detailed Data

In the previous section Overview Data 4.1 we described overview data to show you that the the projects are on the same level for this comparison to be more accurate. We also showed you the two projects in a bigger picture with a comparison towards several other generally known open source projects that can be found on Github.

In this section we will present to you the detailed data that we have collected from the Atom project and the Neovim project. The different data categories that we have investigated are described in section 3.2.2. The raw collected data can be found in Appendix C.

4.2.1 Feature Proposals (D5)

The results of feature proposals displayed in Tables 10 and 11 shows the amount of proposals made by individual users categorised as core developers, peripheral developers and end users. From this data we can summarise that there was a total of 20 feature proposals in the Atom project and 26 proposals in the Neovim project. Proposals from core developers in the Atom project add up to a total of 4 and in the Neovim project they are a total of 12. Proposals made from contribution developers were in the Atom project a total of 12 and in the Neovim project 11. Regarding proposals from end users the Atom project had 4 and the Neovim project had a total of 3.

In Table 12 we can also see the number of unique users that that proposed a feature in each category in each project.The total amount of unique users among core developers (core dev) were 2 in the Atom project and 7 in the Neovim project. The biggest spread of unique users proposing features in both projects are from the category peripheral developers (dev) where Atom had a total of 9 unique peripheral developers proposing features and Neovim had a total of 10. In the category of end users all feature proposals were made by to unique users. Atom had 4 end users proposing features and Neovim had 3.

Table 10: Showing Atom users with amount of feature proposals.

Username	**Feature Proposals**	**User Status**
damieng	3	core developer
celrenheit	2	peripheral developer
codingbelief	2	peripheral developer
fracalo	2	peripheral developer
mnquintana	1	core developer
bgriffith	1	peripheral developer
caleb531	1	peripheral developer
astrojie	1	peripheral developer
alhadis	1	peripheral developer
kevindeleon	1	peripheral developer
MichaelAquiliana	1	peripheral developer
boustanihani	1	end user
jamesonquinn	1	end user
jesseleite	1	end user
karai17	1	end user

Table 11: Showing Neovim users with amount of feature proposals.

Username	**Feature Proposals**	**User Status**
bfredl	4	core developer
justinmk	2	core developer
tweekmonster	2	core developer
shougo	2	peripheral developer
Splinterofchaos	1	core developer
jamessan	1	core developer
elmart	1	core developer
tarruda	1	core developer
mtortonesi	1	peripheral developer
shazow	1	peripheral developer
rygwdn	1	peripheral developer
tony	1	peripheral developer
cyphar	1	peripheral developer
joshtriplett	1	peripheral developer
nhooyr	1	peripheral developer
ZyX-I	1	peripheral developer
equalsraf	1	peripheral developer
sunaku	1	end user
abstiles	1	end user
mmlb	1	end user

Table 12: Showing amount of feature proposers in user groups.

User Group	**Atom**	**Neovim**
core devs	2	7
devs	9	10
end-users	4	3

4.2.2 Feature Origin Categories (D6)

In this section we present the results for Feature Origin Categories presented in section 3.2.2. The categories are: (A) asserted by a developer, either from his or her personal experience or knowledge of user needs. (B) Proposed by an end-user, for example by posting a request to one of the discussion forums, or filing a bug report or "Request for Enhancement" in the issue database. And (C) derived from features found in competing products.

Table 13: Showing amount of proposals for each category.

Proposal Category	**Atom**	**Neovim**
A	16	18
B	2	6
C	4	3
tot.	22	27

4.2.3 Feature Acknowledgement (D7)

The two tables 14 and 15 below presents detailed data for feature acknowledgement 3.2.2. The tables include data regarding unique user status (peripheral developer or core developer), the amount of features they have acknowledge during the time period and the unique username. Table 14 presents detailed data from the Atom project and Table 15 present detailed data from the Neovim project.

Table 14: Showing Atom users with amount of feature acknowledgements.

Username	**Feature Acknowledgements**	**User Status**
fracalo	4	peripheral developer
bastillian	3	peripheral developer
kuychaco	2	core developer
damieng	2	core developer
50Wliu	2	core developer
svanharmelen	2	peripheral developer
as-cii	1	core developer
lee-dohm	1	core developer
simurai	1	core developer
izuzak	1	core developer
codingbelief	1	peripheral developer
benogle	1	peripheral developer
nathansobo	1	peripheral developer

Table 15: Showing Neovim users with amount of feature acknowledgements.

Username	Feature Acknowledgements	User Status
justinmk	12	core developer
bfredl	4	core developer
tarruda	2	core developer
jamessan	2	core developer
tweekmonster	2	core developer
alexgenco	2	peripheral developer
mhinz	1	core developer
tjdevries	1	peripheral developer
shougo	1	peripheral developer

4.2.4 First Implementation of Feature (D8)

This section presents the results of how many first implementation of a feature was done by each developer. Table 16 presents this data for the Atom project and Table 17 presents the data for the Neovim project. Both Table 16 and Table 17 includes data regarding the amount of first implementation from a unique user, the status of that unique user (core developer or peripheral developer) and the unique username.

Table 16: Showing Atom users with amount of first feature implementations.

Username	First Feature Implementations	User Status
damieng	3	core developer
bastillian	3	peripheral developer
50Wliu	1	core developer
kuychaco	1	core developer
wil93	1	core developer
alhadis	1	peripheral developer
astrojie	1	peripheral developer
caleb531	1	peripheral developer
codingbelief	1	peripheral developer
danjordan	1	peripheral developer
esdoppio	1	peripheral developer
fracalo	1	peripheral developer
jtokoph	1	peripheral developer
kevindeleon	1	peripheral developer
MichaelAquilina	1	peripheral developer
timkelty	1	peripheral developer

Table 17: Showing Neovim users with amount of first feature implementations.

Username	**First Feature Implementations**	**User Status**
justinmk	10	core developer
bfredl	8	core developer
tweekmonster	2	core developer
jamessan	2	core developer
alexgenco	1	peripheral developer
nhooyr	1	peripheral developer
shougo	1	peripheral developer
ZyX-I	1	peripheral developer

5 Analysis

In the previous section, we presented and described the results from our overview and detailed data gathering that we have conducted. In this section we analyse our results presented in the previous section. The analysis consists of two parts: (1) validate the selection of cases and (2) identify similarities and differences in requirement elicitation process.

5.1 Validity of Selected Cases

Based on the results presented in section 4.1 we can identify two findings. First, the two projects are similar in size on amount of contributors, additions[10] and deletions[11] as seen in Table 6 and Figures 2a, 3a, 4a and 5a. Second, the distribution of the gathered overview data are very similar both between the projects, but also compared to other popular OSS-projects as seen in Figures 1b, 2b, 3b, 4b and 5b. The difference in amounts of commits as seen in Figure 1a could, as mentioned before in section 3.1.3, be a difference in how the two projects use commits. With this information the two selected projects can be considered suitable for comparison on requirement elicitation.

5.2 Identify Similarities and Differences from Detailed Data

In general, participants in an open source software development show a strong sense of engagement in, and ownership of, the project, where the time they spent on contributing to the development in most cases are rewarded with signalling as mentioned in section 1.1. In the case of Atom and Neovim there are no exceptions. However within this engagement among participants we have seen certain differences and similarities within feature proposals, origin, acknowledgement and in the first implementation of a feature among participants.

In Tables 10 and 11 we can see the amount of proposals made by individual users categorised as core developers, peripheral developers and end users. From this data we can summarise that there was a total of 20 feature proposals in the Atom project and 26 proposals in the Neovim project. Proposals from core developers in the Atom project adds up to a total of 4 and in the Neovim project there are 12. Proposals made from peripheral developers were a total of 12 in the Atom project in contrast to the Neovim project which had a total of 11. Regarding proposals from end users the Atom project had 4 and the Neovim project had a total of 3.

> In Neovim around 50% of the feature proposals was from core developers. In Atom they were only 20%.

In Table 12 we can see the number of unique users that proposed a feature in each category for each project. The amount of unique users among core developers were 2 in the Atom project and 7 in the Neovim project. Peripheral developers was category for

[10] Added Lines of Code.
[11] Deleted Lines of Code.

which most of the feature proposals were made, and also had the most unique users, Atom had a total of 9 and Neovim had a total of 10. In the category of end users all feature proposals was made by unique users, Atom had 3 and Neovim 4.

The amount of suggestions per user is approximately the same for the two projects.

Looking into the detailed data for feature origin, Table 13, it was in both cases clear that the majority of proposals originated from a developer's personal experience or his knowledge of user needs, 72,7% in the Atom Project and 66,7% in the Neovim Project. There were 9.1% proposals coming from end users in the Atom project and 2,2% in the Neovim project. In the aspect of features that derive from competing products there was also a minor difference, where Atom had 1,8% and Neovim had 1,1% of feature proposals derived from competing products.

Feature origin categories was for both projects approximately the same.

Acknowledgment of features is the single most interesting aspect of the results from the detailed data. In our results, Tables 14 and 15, we can see that core developers in the Atom project stands for 45,5% of the acknowledgments compared to the 85,2% in the Neovim project. This means that 55,5% of the feature acknowledgments came from peripheral developers in the Atom project and 14,5% in the Neovim. There are two aspects that make these numbers interesting. First, several peripheral developers in both projects have the authority to acknowledge a feature for implementation, from what we can see, without a core developers involvement. The second interesting finding is the fact that the Atom project allowed more than half of their features to be acknowledged by peripheral developers while the Neovim project only had 14,5% acknowledgments by peripheral developers. Furthermore, the developers in the Atom project that made the most acknowledgments were peripheral developers while the Neovim project had core developers at the top.

Peripheral developers are in both projects were allowed to acknowledge feature proposals.

In Atom 55,5% of the feature acknowledgments came from peripheral developers.
In Neovim the number was only 14,5%.

In Tables 14 and 15 we can see that there is a significant spread among acknowledgement from unique developers in the projects. In Neovim the developer with the most acknowledgements, "justinmk", stands for 44,4% while in Atom it was "fracalo" who had the most feature acknowledgement which was 18,2%.

Moving on to analyse when a feature was first implemented (Tables 16 and 17), we can confirm that from the features that had been proposed in the Atom project 27,3% of them

were first implemented by a core developer and 72,7% were implemented by a peripheral developer. The Neovim project, on the other hand, had 86,6% of first feature implementation done by core developers and 15,4% from peripheral developers. Furthermore, the data also shows that that the core developer ”justinmk” stands for 38,5% of the first feature implementations in the Neovim project and the top spot with 13,6% in Atom is shared between the core developer ”damieng” and the peripheral developer ”bastillian”. Overall there are a total of 12 peripheral developers and 4 core developers in the Atom project doing the first implementation of a feature while Neovim has a total of 4 core developers and 4 peripheral developers.

In Atom the majority of feature implementations were done by peripheral developers. For Neovim it was the opposite.

6 Discussion

In section 1 we presented the problem, purpose, contribution and approach for this thesis. Furthermore in section 1.1 we introduce the research area and describe the related work within the area. Based on these facts and thoughts a total of three research questions were framed and presented in section 1.3.

In this section we will discuss our findings from the previous section, their implications and their relevance to our first two research questions. Then we will discuss, based on the first two research questions, the answer to our third research question. We consider the answers to the first two questions to be conclusive, and the answer to our third research question to be speculative.

6.1 Characteristics of Atom's and Neovim's Requirements Processes

The research questions *What are the characteristics of community-driven OSS development?*(RQ1) and *What are the characteristics of firm-driven OSS development?*(RQ2) will be discussed in this section. A summary of the differences between the both cases will be presented and furthermore a conclusion will be framed for each research question based on the result data in section 4.2.

When comparing the detailed data results form Atom and Neovim in this thesis, presented in section 4.2, to the work that Noll et al. [22] made on requirements elicitation in open source software development. The comparison shows that the majority of features are asserted by developers, based on either their personal experience or knowledge of users needs. Therefore we can conclude that our results are general for several areas within the area of OSS development. Furthermore, we also compare the results from our statistical data from section 4.1 with the results from [20], where we can see a similar pattern based on the schema we framed for overview data in section 3.2.2.

The case study we have conducted throughout this thesis are investigating the process of requirement elicitation. We have divided to investigate the process we did divided the requirement elicitation into two major steps (1) the proposal of a feature and (2) the acknowledgment of a feature.

The first step involves several depending factors such as when a suggested implementation for a feature where made and where the feature origin came from based on three categories where the first category are if a proposal where asserted by a developer, either from his or her personal experience or knowledge of user needs, the second category are if the feature where proposed by an end-user, for example by posting a request to one of the discussion forums, and the last category focus if the feature where find while filing a bug report or ?Request for Enhancement? in the issue database or derived from features found in competing products. The investigation of the first step is to us interesting due to the fact that you can se how high the contribution from non core developers are and therefore you can see if there are a general interest in proposing features and how that interest are handeld from the central group of core developers.

The second step has no depending factor more than the first step because to acknowledge a feature there must be a feature proposal to acknowledge. So the second step is straightforward to investigate due to the fact that a proposal can either be acknowledged or not. We found this step interesting to investigate because we wanted to se if there where

more or less feature acknowledgment in a firm-driven OSS project due to the high pressure on delivering a quality product that are associated with a company reputation.

To investigate the requirement elicitation process we first compared the two cases in the first step of a requirement elicitation process which are feature proposals. In this step we found that Neovim had around 50% of the feature proposals coming from core developers whereas In the Atom project only 20% of the proposals came from core developers. These facts tell us that the core group of developers in a community driven project are a bit more involved in the a early stage of the requirement elicitation process. Furthermore the proposals were in most cases followed up by a first implementation suggestion from developers and here we can see the same differences between the projects. The Neovim project had 72,7% of the first implementation suggestions coming from core developers whereas the Atom project had 27,3%. These numbers in itself implicate that there are a differences in early involvement between the projects. Where the core developers working on the Atom project allow more responsibility towards the peripheral developers and the core developers at Neovim don?t give as much space to the peripheral developers possibly due to their high interest in further implementing features into the Neovim project. So feature proposals and first implementations of a feature showed us that community and firm driven OSS project have some initial differences in the requirement elicitation. However we also found similarities in these differences such as the amount of feature proposals per user is approximately the same for the two projects. We also conclude that the feature origin categories was for both projects approximately the same, the categories can be found in section 3.2.2. All of the above findings where findings that we expected to find based on the systematic literature review that we have conducted and presented in section 2.

However looking more into each project we found interesting results based on the engagement and involvement among the individuals within the project. In both projects there were certain individuals that did stand out in contribution to the development. In the Neovim project one core developer did stand for 44,4% of all acknowledgments and the same individual had a total 38,5% in the first feature implementation. The Atom project did not have an individual with such high involvement however the individual with highest and second highest feature acknowledgment where peripheral developers and not a core developers. These findings along with the earlier mentioned findings are interesting due to the fact that the core developers in the firm-driven project Atom project in this case don?t have high involvement in the suggestions of requirements whereas the core developers in the Neovim project has a high involvement in these early stages. Despite the early involvement in the requirement elicitation process one would definitely think that the acknowledgment of the proposed features would strictly be traced back to a core developer. This was however not the case in our results where both projects had peripheral developers making acknowledgements. Even more interesting are the fact that Atom had approximately 50% of the acknowledgments coming from peripheral developers. For us this is extremely interesting finding but we can?t fully support this conclusion due to lack of insight into the projects, simply we don?t know if the peripheral developers actually have some sort of permission to do these kind of acknowledgments. This finding are further explained in the section 6.2.

Overall we can clearly see a pattern of motivated and engaged core developers within the Neovim project in comparison to the peripheral developers and the end-users with high presence in suggesting-, acknowledging- and implementing features.The presence in

the requirement elicitation could however impact the product in way that might be negativt to the overall usage among users (core developers, peripheral developers and end-users) but positive for the usage of the product among core developers. It would be negative due to the fact that core developers don?t take in the overall users suggestions into consideration and therefore leads the product in the direction they want.

When it comes to Atom we can from the findings conclude that core developers overall tend to have low presence in suggesting-, acknowledging- and implementing features in a firm-driven OSS development projects. The low involvement from core developers could be due to several reasons. For an example the core developers for a project could be core developers in several projects within the firm and might just be there to make the project move in the direction that best befits the firm the work for.

Last but not least communities and firms have approximately the same end-user involvement which means that when firms consider choosing OSS-development they can look at any OSS-project to see how end-users are involved.

6.2 Implications of the differences between firm- and community-driven OSS development

From the differences that we concluded in Section 6.1 for research questions one and two, one could speculate in what they imply. Looking at peripheral developer participation in firm-driven projects we can see that it's higher. That implies that communities could have a higher peripheral developer participation which could result in; (1) increasing product customer value by having more developers working on improving the product; (2) increasing the actual development speed by having a greater number of developers; and (3) enabling core developers to focus on managing project and its future goals by relieving them of development work.

Furthermore, from a firm's perspective the participation from peripheral developers is already high. This could mean that firms choosing to use an OSS development approach instead of a proprietary development approach could (1) increase their development speed without increased costs by having more developers and (2) increase product customer value by having more developers working on improving the product.

6.3 Unforeseen complications

When analysing the results we found that the definition that Krishnamurthy et al. [17] used to categorise core and peripheral developers based on count-based operationalisations and that we also used when we collected the data might not be accurate, as mentioned in the previous section 6.2. We notice this due to the fact that there where a high amount of peripheral developers that made acknowledgments in both of the projects, a task that according to Krishnamurthy al. [17] should be in the hands of a core developer.

After further investigation we found a recent study by Joblin et al. [15] that looks at the issue. Joblin et al. concludes that the count-based operationalisations, which we used in this thesis, does not give an accurate distribution of core and peripheral developers when compared to the project participants perceptions of the distribution. Joblin et al. instead suggests a network-based operationalisation, which, when compared to project participants perceptions of the distribution, resulted in a more accurate distribution of core and peripheral developers. The network-based operationalisation focus on the communication and contributions among the developers instead and then classify the developers based on their network activity and impact rather then just looking into if a developer are a part of the core organisation or not.

7 Conclusions and Future Work

In this section we will summarise the answers for all three research questions. Then we will make suggestions for future work.

With our study we the can give the following answers for our presented research questions:

RQ1 *What are the characteristics of community-driven OSS development?*

In community-driven OSS development, core developers in projects tend to have a **high** presence in suggesting-, acknowledging- and implementing features.

RQ2 *What are the characteristics of firm-driven OSS development?*

In firm-driven OSS development, core developers in projects tend to have a **low** presence in suggesting-, acknowledging- and implementing features.

RQ3 *What are the implications of the differences between firm- and community-driven OSS development?*

- Communities could, by allowing more peripheral developers to participate, increase product customer value and development speed and also allow core developers to focus on managing the project.
- Firms could, by choosing to use an OSS-development approach, increase product customer value and increase development speed without increasing development costs.
- Both firms and communities could, by encouraging contribution developers in an OSS-development with goals and opportunities, increase engagement among involved developers and improve the reputation towards developers not jet involved.

In section 6.3 we mentioned that the count-based operationalisations we used in this thesis to categorise core and peripheral developers is not optimal according to Joblin et al. [15]. Therefore it would be appropriate to conduct a similar study to ours but instead use the, by Joblin et al., suggested network-based operationalisation.

Given the scripts we programmed for conducting our research it also possible to replicate this study for the same projects but different time periods. But most importantly it creates an opportunity to study any OSS project, hosted on Github, for any time period.

Furthermore our results indicate that there is a possibility for firms to reduce their development costs by using an OSS development process. This could be further studied by comparing the actual costs between a firm-driven OSS development project with a firm-driven proprietary development project. It can also be done by studying a firm-driven proprietary development project which is doing a transition into an OSS development process.

References

[1] Atom project page on github. https://github.com/atom/atom. Accessed: 2017-04-02.

[2] Atom web page. https://atom.io/. Accessed: 2017-04-02.

[3] The cathedral and the bazaar. www.catb.org/ esr/writings/cathedral-bazaar/. Accessed: 2017-02-28.

[4] Neovim project page on github. https://github.com/neovim/neovim. Accessed: 2017-04-02.

[5] Neovim subreddit on reddit. https://www.reddit.com/r/neovim/. Accessed: 2017-04-17.

[6] Neovim web page. https://neovim.io/. Accessed: 2017-04-02.

[7] Neovim wiki on github. https://github.com/neovim/neovim/wiki. Accessed: 2017-04-02.

[8] Tim berners-lee on the web at 25: the past, present and future. http://www.wired.co.uk/article/tim-berners-lee. Accessed: 2017-03-01.

[9] Pär J Ågerfalk, Brian Fitzgerald, Par J Agerfalk, and Brian Fitzgerald. Outsourcing to an Unknown Workforce: Exploring Opensourcing as a Global Sourcing Strategy. *MIS Quarterly*, 32(2):385–409, 2008.

[10] Kevin Crowston, James Howison, and Hala Annabi. Information systems success in free and open source software development: Theory and measures. *Software Process Improvement and Practice*, 11(2):123–148, 2006.

[11] Bill Curtis, Elliot M. Soloway, Ruven E. Brooks, John B. Black, Kate Ehrlich, and H. Rudy Ramsey. Software Psychology: The Need for an Interdisciplinary Program. *Proceedings of the IEEE*, 74(8):1092–1106, 1986.

[12] Martin Fowler and Jim Highsmith. The agile manifesto. *Software Development*, 9(August):28–35, 2001.

[13] Delwyn Goodrick. Comparative Case Studies. Technical report, 2014.

[14] Tomas Jansson. Agila projektledningsmetoder och motivation. *Karlstad University Studies, ISSN 1403-8099 ; 2015:9*, 2015.

[15] Mitchell Joblin, Sven Apel Claus Hunsen, and Wolfgang Mauerer. Classifying Developers into Core and Peripheral: An Empirical Study on Count and Network Metrics Mitchell. *CoRR*, abs/1604.0, 2016.

[16] S Keshav. How to Read a Paper. 37(3):2, 2010.

[17] Rajiv Krishnamurthy, Varghese Jacob, Suresh Radhakrishnan, and Kutsal Dogan. Peripheral Developer Participation in Open Source Projects. *ACM Transactions on Management Information Systems*, 6(4):1–31, 2016.

[18] Alan MacCormack, John Rusnak, and Carliss Y Baldwin. Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code. *Management Science*, 52(7):1015–1030, 2006.

[19] a. Mockus, R.T. Fielding, and J. Herbsleb. A Case Study of Open Source software Development: the Apache server. *ACM. Proceedings of the 21st International Conference on Software Engineering. ICSE 2000, Los Angeles*, pages 263–272, 2000.

[20] Audris Mockus, Roy T. Fielding, and James D Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Trans. Softw. Eng. Methodol.*, 11(3):309–346, 2002.

[21] John Noll. Requirements acquisition in open source development: Firefox 2.0. In *IFIP International Federation for Information Processing*, volume 275, pages 69–79, 2008.

[22] John Noll and Wei-Ming Liu. Requirements elicitation in open source software development: a case study. In *Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development - FLOSS '10*, pages 35–40, 2010.

[23] Briony J Oates. *Researching information systems and computing*. Sage, 2005.

[24] Yin Robert K. *Case Study Research: Design and Methods*, volume 5. ed. SAGE, cop. 2014., London, 5. ed. edition, 2013.

[25] Eric Schmidt and Jonathan Rosenberg. *How Google Works*. Grand Central Publishing, 2014.

[26] Minghui Zhou, Audris Mockus, Xiujuan Ma, Lu Zhang, and Hong Mei. Inflow and Retention in OSS Communities with Commercial Involvement. *ACM Transactions on Software Engineering and Methodology*, 25(2):1–29, 2016.

A Detailed Data Schema for Statistical Data

In this appendix we will give a detailed description for how we define each of the data type presented in the thesis. Before presenting the definitions we will explain some Git-terms.

Git-terms

Git is, as mentioned in the thesis, a version control system. Below we present the Git-terms used to present the data schema definitions.

- `commit`

  A set of changes to the source code. The `commit` always has an author and an ID, typically a message describing the `commit` is also included.

- `issue`

  An `issue` is a suggestion for improvement, task or question related to the project. An `issue` can be created by anyone. An `issue` can have the status `open` or `closed`.

- `label`

  An `issue` can optionally also have one or many `labels`. A `label` provides meta data to the `issue`. Projects can define their own labels, popular `labels` are for example `duplicate`, `enhancement` and `bug`.

Code contributor

Any author of a `commit` to the project is a code contributor.

Feature proposer

Any author of an `issue` to the project with the following criterias: (1) the `issue` is `labeled` as an enhancement, (2) the `issue` is not labeled as a duplicate and (3) the `issue` is `closed`.

2. If there are two `issues` proposing the same thing. One of them is labeled as a duplicate. We only need to count the one that is not.

3. To ensure that the `issue` is in fact not a duplicate it needs to already be `closed`. This means that a user has seen the issue, and if it is a duplicate the user would have labeled it as such.

Problem reporters

Any author of a problem report which we define as an `issue` to the project with the following criterias: (1) the `issue` is `labeled` as an bug, (2) the `issue` is not labeled as a duplicate and (3) the `issue` is `closed`.

2. If there are two `issues` reporting the same thing. One of them is labeled as a duplicate. We only need to count the one that is not.

3. To ensure that the `issue` is in fact not a duplicate it needs to already be `closed`. This means that a user has seen the issue, and if it is a duplicate the user would have labeled it as such.

Defect repair time

Defect repair time is the time between the creation of a problem report and when it was `closed`.

B Libraries and Programmed Scripts

In this appendix we will present the libraries and main scripts used. To see the complete system please see our public Github repository found at: `https://github.com/FilipHarald/bachelor-thesis`.

Libraries

- `PyGithub`

  This python library was used to communicate with Github through the Github API.

- `Matplotlib`

  This python library was used to visualize our results from the statistical data.

Main Scripts

Below we present the two main scripts used to collect the data. The first, `code_contributors.py`, collected data for D1. The second, `other_contributors.py`, collected data for D2-D4.

`code_contributors.py`

```
import json
import os
from collections import defaultdict
from datetime import datetime

from utils import cache, analyzer
from utils.pretty_printer import nc_print
from utils.init import config
from github.GithubException import RateLimitExceededException

file_name = os.path.basename(__file__)  # file cache key
json_file = os.path.join(os.path.dirname(__file__), os.pardir) + 'commits_counter.json'


def get_contributors_data(g, repo_name, since, until):
    try:
        commits = g.get_repo(repo_name).get_commits(since=since, until=until)
        commit_data = get_commits_data(commits)
    except RateLimitExceededException:
        print("RATE_LIMIT_EXCEEDED_TRYING_WITH_NEW_ACCOUNT")
        g = config.get_other_g(g)
        commits_rest = g.get_repo(repo_name).get_commits(since=since, until=until)
        with open(json_file, "r") as file:
            counter = json.load(file)
        one = commits[:counter]
        two = commits_rest[counter:]
        print(type(one))
        new_commits = one + two
        commit_data = get_commits_data(new_commits)
    return commit_data


def get_commits_data(commits):
    unique_users = defaultdict(int)
    addition_dict = defaultdict(int)
    deletion_dict = defaultdict(int)
    counter = 0
    for commit in commits:
        with open(json_file, 'w') as file:
            json.dump(counter, file)
        unique_users[commit.commit.author.name] += 1
        addition_dict[commit.commit.author.name] += commit.stats.additions
        deletion_dict[commit.commit.author.name] += commit.stats.deletions
        counter += 1
    return {'commits_dict': unique_users,
            'additions_dict': addition_dict,
            'deletions_dict': deletion_dict}
```

```
def run(g, config):
    nc_print('------------------------------Code_contributors_START------------------------------')
    contributors_data = {}
    for repo in config.repos:
        print(repo['color'] + repo['name'])
        contributors_data[repo['name']] = cache.cache(get_contributors_data,
                                                      key=file_name + '_' + repo['key'] + '
                                                          _contributors',
                                                      g=g,
                                                      repo_name=repo['name'],
                                                      since=datetime.fromtimestamp(int(repo['
                                                          since'])),
                                                      until=datetime.fromtimestamp(int(repo['
                                                          until'])))
    analyzer.visualize_results(config.repos, 'commits', contributors_data, 'code_contributors/
        commits')
    analyzer.visualize_results(config.repos, 'additions', contributors_data, 'code_contributors/
        additions', 'LoC')
    analyzer.visualize_results(config.repos, 'deletions', contributors_data, 'code_contributors/
        deletions', 'LoC')

    nc_print('------------------------------Code_contributors_END------------------------------')
    pass
```

other_contributors.py

```
import os
from collections import defaultdict
from datetime import datetime

from github.GithubException import RateLimitExceededException

from utils import cache, analyzer
from utils.pretty_printer import nc_print
from utils.init import config

file_name = os.path.basename(__file__)  # file cache key


def get_issues_data(g, repo_name, since, until, labels):
    try:
        issues = get_issues(g, repo_name, since, labels, 'closed')
        unique_users = get_unique_users(issues, until)
    except RateLimitExceededException:
        g = new_g(g)
        issues = get_issues(g, repo_name, since, labels, 'closed')
        unique_users = get_unique_users(issues, until)
    return unique_users


def get_defect_repair_time(g, repo, since, until):
    try:
        labels = [g.get_repo(repo['name']).get_label(repo['bug'])]
        issues = get_issues(g, repo['name'], since, labels, 'closed')
        repair_times = get_repair_times(issues, until)
    except RateLimitExceededException:
        g = new_g(g)
        labels = [g.get_repo(repo['name']).get_label(repo['bug'])]
        issues = get_issues(g, repo['name'], since, labels, 'closed')
        repair_times = get_repair_times(issues, until)
    return repair_times


def get_issues(g, repo_name, since, labels, state):
    return g.get_repo(repo_name).get_issues(since=since, labels=labels, state=state)


def get_unique_users(issues, until):
    unique_users = defaultdict(int)
    for issue in issues:
        if int(issue.created_at.timestamp()) < int(until):
            if not any(label.name == "duplicate" for label in issue.labels):
                unique_users[issue.user.login] += 1
    return unique_users


def new_g(g):
    print("RATE_LIMIT_EXCEEDED_TRYING_WITH_NEW_ACCOUNT")
    return config.get_other_g(g)


def get_repair_times(issues, until):
    repair_times = []
    for issue in issues:
```

```
        if int(issue.closed_at.timestamp()) < until:
            if not any(label.name == "duplicate" for label in issue.labels):
                repair_times.append(((int(issue.closed_at.timestamp()) - int(issue.created_at.
                    timestamp())) / 3600)/24)
    return repair_times if len(repair_times) > 0 else [0]


def run(g, config):
    nc_print('----------Other Contributors START----------')
    problem_reporters_data = {}
    feature_proposers_data = {}
    repair_times_data = {}
    for repo in config.repos:
        print(repo['color'] + repo['name'])
        since = datetime.fromtimestamp(int(repo['since']))
        until = int(repo['until'])
        pr_dict = cache.cache(get_issues_data,
                              key=file_name + '_' + repo['key'] + '_problem_reporters',
                              g=g,
                              repo_name=repo['name'],
                              since=since,
                              until=until,
                              labels=[g.get_repo(repo['name']).get_label(repo['bug'])])
        problem_reporters_data[repo['name']] = {}
        problem_reporters_data[repo['name']]['problem_reporters_dict'] = pr_dict
        fp_dict = cache.cache(get_issues_data,
                              key=file_name + '_' + repo['key'] + '_feature_proposers',
                              g=g,
                              repo_name=repo['name'],
                              since=since,
                              until=until,
                              labels=[g.get_repo(repo['name']).get_label(repo['enhancement'])])
        feature_proposers_data[repo['name']] = {}
        feature_proposers_data[repo['name']]['feature_proposers_dict'] = fp_dict
        repair_times = cache.cache(get_defect_repair_time,
                                   key=file_name + '_' + repo['key'] + '_repair_times',
                                   g=g,
                                   repo=repo,
                                   since=since,
                                   until=until,)
        repair_times_data[repo['name']] = {}
        repair_times_data[repo['name']]['repair_times_array'] = repair_times
    analyzer.visualize_results(config.repos,
                               'problem_reporters',
                               problem_reporters_data,
                               'problem_reporters/problem_reporters',
                               x_axis='problem reporters')
    analyzer.visualize_results(config.repos,
                               'feature_proposers',
                               feature_proposers_data,
                               'feature_proposers/feature_proposers',
                               x_axis='feature proposers')
    analyzer.analyze_repair_time(config.repos,
                                 'repair_times',
                                 repair_times_data,
                                 'repair_times/repair_times')
    nc_print('----------Other Contributors END----------')
    pass
```

C Detail Results Raw Data

In this appendix we present the raw data collected for D5-D7. The data can also be found at: `https://docs.google.com/spreadsheets/d/1HuNUuB8UeVbQUn4qeKSg7NY_QTHRG5_PlzuXJua196w/edit?usp=sharing`. First we present the data for Atom and then Neovim.

Atom	D5 first proposed					D7 feature acknowledemer			D8 first implementation of fea				
feature name	**D6**	**timestamp**	**user**	**user status**	**asserted**	**timestamp**	**user**	**user statu**	**timestamp**	**user**	**user status**	**days to acknowledgement**	**days to implementation**
Allow easier clicking on tabs when in fullscreen mode	C	10-05-2015	wil93	core dev	-	08-07-2016	kuychaco	core dev	11-06-2015	wil93	core dev	**425**	**32**
Always open search results on single click	A	12-04-2016	kuychaco	core dev	Experience	12-04-2016	kuychaco	core dev	12-04-2016	kuychaco	core dev	**0**	**0**
Add find in parent directory functionality	A	05-10-2015	codingbelie	phe dev	Need	16-03-2016	codingbelief	phe dev	28-12-2015	timkelty	phe dev	**163**	**84**
Wait for updated package to load and then try to activate	A	09-06-2015	mnquintana	core dev	Experience	07-07-2015	benogle	phe dev	20-04-2016	jtokoph	phe dev	**28**	**316**
Unify icons between tabs and titles	A	27-06-2016	damieng	core dev	Experience	28-07-2016	damieng	core dev	27-07-2016	damieng	core dev	**31**	**30**
Emit events when directories are toggled	A	11-05-2016	alhadis	phe dev	Need	22-07-2016	as-cii	core dev	11-05-2016	alhadis	phe dev	**72**	**0**
Added support for inline block comment	A	28-04-2016	fracalo	phe dev	Experience	28-04-2016	fracalo	phe dev	22-06-2016	50Wliu	core dev	**0**	**55**
Add stroke properties	A	20-07-2016	fracalo	phe dev	Experience	20-07-2016	50Wliu	core dev	20-07-2016	fracalo	phe dev	**0**	**0**
Add syntax highlighting for drop command of interactive reba	A	07-07-2016	astrojie	phe dev	Experience	07-07-2016	50Wliu	core dev	07-07-2016	astrojie	phe dev	**0**	**0**
Adds snippet for Go starter template	A	23-01-2016	kevindeleor	phe dev	Need	24-01-2016	svanharmelen	phe dev	23-01-2016	kevindeleon	phe dev	**1**	**0**
Add Snippet to handle Context, ResponseWriter, *http.Reque	A	24-01-2016	celrenheit	phe dev	Need	24-01-2016	svanharmelen	phe dev	24-01-2016	celrenheit	phe dev	**0**	**0**
Add picture element to selectors	A	10-06-2016	celrenheit	phe dev	Need	10-06-2016	fracalo	phe dev	10-06-2016	celrenheit	phe dev	**0**	**0**
Maps in maps	A	07-07-2016	bgriffith	phe dev	Experience	15-07-2016	fracalo	phe dev	07-07-2016	esdoppio	phe dev	**8**	**0**
Add glade to list of xml file types	A	19-07-2016	MichaelAqu	phe dev	Experience	19-07-2016	fracalo	phe dev	19-07-2016	MichaelAquilina	phe dev	**0**	**0**
Introduce autoWidth for TextEditors	A	30-05-2016	damieng	core dev	Experience	17-08-2016	nathansobo	phe dev	17-08-2016	damieng	core dev	**79**	**79**
Make Atom executable as a standard Electron app	A	27-07-2016	damieng	core dev	Experience	27-07-2016	damieng	core dev	27-07-2016	damieng	core dev	**0**	**0**
Allow different versions of Atom to run simultaneously on Wir	A	08-06-2016	codingbelie	phe dev	Experience	08-06-2016	lee-dohm	core dev	08-06-2016	codingbelief	phe dev	**0**	**0**
Show settings after install	B	05-09-2015	jesseleite	end-user	Need	06-04-2016	bastilian	phe dev	06-04-2016	bastilian	phe dev	**214**	**214**
Sort packages search	B	17-02-2016	boustanihar	end-user	Need	05-04-2016	bastilian	phe dev	05-04-2016	bastilian	phe dev	**48**	**48**
Add background to close icon	C	13-04-2016	jamesonqui	end-user	-	21-06-2016	simurai	core dev	21-06-2016	bastilian	phe dev	**69**	**69**
Resize tabs when mouse moves away from tab bar	C	07-04-2015	karai17	end-user	-	08-04-2015	izuzak	core dev	12-04-2016	danjordan	phe dev	**1**	**371**
Recognize `source` keyword in grammar (for Gemfiles)	C	24-06-2016	caleb531	phe dev	-	24-06-2016	bastilian	phe dev	24-06-2016	caleb531	phe dev	**0**	**0**
												51,77	59

Neovim	D5 first proposed					D7 feature acknowledement			D8 first implementation of f			days to acknowledgement	days to implementation
feature name	D6 f	timestamp	user	user statu	asserted	timestamp	user	user status	timestamp	user	user statu		
Ruby support	B	11-12-2015	mtortonesi	end-user	Need	12-12-2015	alexgenco	phe dev	12-06-2016	alexgenco	phe dev	1	184
Horizontal Scrolling	B	25-5-2015	shazow	end-user	Experience	9-10-2015	justinmk	core dev	13-10-2015	justinmk	core dev	137	141
Implement timers	C	05-11-2014	splinterofchaos	core dev	-	05-11-2014	justinmk	core dev	21-04-2016	bfredl	core dev	0	533
support "special" highlight (undercurl)	B	20-02-2015	rygwdn	end-user	Experience	22-02-2015	tarruda	core dev	20-05-2016	justinmk	core dev	2	455
error reports/dumps	B	20-03-2016	tony	end-user	Experience	15-06-2016	tjdevries	phe dev	16-06-2016	tjdevries	core dev	87	88
allow setting cwd in jobstart(), termopen()	A	25-05-2016	cyphar	phe dev	Experience	25-05-2016	justinmk	core dev	10-06-2016	justinmk	core dev	0	16
api_info()	A	15-06-2016	bfredl	core dev	Experience	16-06-2016	justinmk	core dev	17-06-2016	bfredl	core dev	1	2
Always resize the :terminal	B	25-05-2016	mmlb	end-user	Experience	26-05-2016	mhinz	core dev	21-06-2016	justinmk	core dev	1	27
Use buffered reading/writing for ShaDa files	A	04-03-2016	shougo	phe dev	Experience	04-03-2016	justinmk	core dev	01-06-2016	ZyX-I	phe dev	0	89
tui: Assume 256 colors in most cases.	A	03-07-2016	justinmk	core dev	Experience	03-07-2016	justinmk	core dev	03-07-2016	justinmk	core dev	0	0
terminal: Ensure b:term_title always has a value	A	09-07-2016	joshtriplett	phe dev	Experience	09-07-2016	justinmk	core dev	13-07-2016	justinmk	core dev	0	4
allow stderr handler for rpc jobs	A	08-05-2016	bfredl	core dev	Experience	08-05-2016	bfredl	core dev	12-05-2016	bfredl	core dev	0	4
Better handling of mouse-clicks on concealed cha	A	28-07-2016	tweekmonster	core dev	Experience	28-07-2016	justinmk	core dev	28-07-2016	tweekmonster	core dev	0	0
man.vim rewrite	A	05-05-2016	nhooyr	phe dev	Experience	05-05-2016	justinmk	core dev	12-05-2016	nhooyr	phe dev	0	7
Restore ":browse oldfiles".	A	11-08-2016	jamessan	core dev	Experience	11-08-2016	jamessan	core dev	11-08-2016	jamessan	core dev	0	0
capture() function (renamed to execute())	C	03-05-2016	shougo	phe dev	-	03-05-2016	shougo	phe dev	03-05-2016	shougo	phe dev	0	0
rplugin manifest: default to XDG dir'	B	09-07-2016	sunaku	end-user	Need	12-07-2016	tweekmonster	core dev	18-08-2016	justinmk	core dev	3	40
API: external UIs can render custom popupmenu	A	09-03-2016	bfredl	core dev	Experience	09-03-2016	tarruda	core dev	29-08-2016	bfredl	core dev	0	173
API: call any API method from vimscript	A	24-01-2016	bfredl	core dev	Experience	24-01-2016	justinmk	core dev	01-09-2016	bfredl	core dev	0	221
API: nvim_call_atomic(): multiple calls in a single	A	05-04-2016	ZyX-I	phe dev	Experience	13-04-2016	bfredl	core dev	22-10-2016	bfredl	core dev	8	200
API: nvim_win_get_number(), nvim_tabpage_get	C	27-01-2015	abstiles	end-user	-	04-10-2016	jamessan	core dev	15-10-2016	jamessan	core dev	616	627
has("nvim-1.2.3") checks for a specific Nvim versi	A	05-05-2016	justinmk	core dev	Experience	05-05-2016	justinmk	core dev	05-05-2016	justinmk	core dev	0	0
:CheckHealth checks tmux, terminfo, performance	A	05-09-2016	alexgenco	phe dev	Experience	05-09-2016	alexgenco	phe dev	06-09-2016	justinmk	core dev	0	1
events: allow event processing in getchar()	A	26-10-2015	tarruda	core dev	Experience	25-09-2016	bfredl	core dev	15-10-2016	bfredl	core dev	335	355
API: metadata: Nvim version & API level	A	05-11-2014	elmart	core dev	Need	25-09-2016	justinmk	core dev	25-09-2016	justinmk	core dev	690	690
API: metadata: "since", "deprecated_since"	A	26-09-2016	equalsraf	phe dev	Experience	26-09-2016	bfredl	core dev	27-09-2016	bfredl	core dev	0	1
Added QuickFixLine highlight group	A	03-08-2016	tweekmonster	core dev	Experience	03-08-2016	tweekmonster	core dev	03-08-2016	tweekmonster	core dev	0	0