Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik

Harisree Radhakrishnan
MASTER THESIS

A Theory of Open Source Engineering Processes

Submitted on 29. November 2017

Supervisor: Prof. Dr. Dirk Riehle, M.B.A.
Professur für Open-Source-Software
Department Informatik, Technische Fakultät
Friedrich-Alexander University Erlangen-Nürnberg

Versicherung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Harisree Radhakrishnan

_____

Nuremberg, 29. November 2017

License

Harisree Radhakrishnan

_____

Nuremberg, 29. November 2017

Acknowledgements

I would like to express my gratitude towards Prof. Dirk Riehle for his support and encouragement. I am also grateful to him for connecting me with the interview partners from Open Source Communities. I would also like to thank Andreas Kaufmann for his useful insights throughout the course of my research. In addition, I would like to thank Michael Dorner for helping during the initial stages of my thesis and preparing for the interviews.

I am also grateful to the interview partners who were kind enough to take time to share their experience and knowledge.

Abstract

Open Source communities are largely people centric and work on customized software processes created by people while trying to solve a problem. Hence, most Open Source projects do not have formal processes or do not follow software engineering best practices. But at the same time, they are successful and the processes followed are instrumental in their success. The objective of this thesis is to build a theory of Open Source Engineering processes. This theory can be used by Open Source communities to design their own processes and to compare their processes with that of other communities. The theory is presented as categories and sub-categories and is derived from qualitative data analysis of interviews and supplemental materials. The model is then applied to three polar Open Source communities.

Keywords

Open Source Engineering Process, Open Source Development Process, Qualitative Research, Decision-making in Open Source

Contents

1 Introduction

1.1 Original Thesis Goals

The goal of the thesis was to build a theory of open source engineering processes using three very different examples. To better understand the Open Source engineering processes, three polar examples were chosen - The Linux kernel, the PostgreSQL database, and the Tiki (Wiki CMS) software. These communities were chosen depending on how they were organized and how decision-making worked in these communities. Interviews with 3 to 6 practitioners from these communities and subsequent qualitative data analysis of these interviews would be used to develop the theory. The theory would then be cast as a multi-dimensional model and the three processes described as instances of the model.

1.2 Changes to Thesis Goals

The initial goal was to interview 3 to 6 practitioners from all the three open source communities. During the course of the thesis, a total of 4 practitioners were interviewed in 3 interviews as two practitioners from the last community participated in the third interview. This was due to the difficulty in scheduling interviews with practitioners due to their busy schedules.

2 Research Chapter

2.1 Introduction

Open Source Software has evolved over the years and it now invites considerable commercial interest. Once considered maverick and unconventional, Open Source now has ubiquitous presence and acceptance. It is economically viable and there are many examples of highly successful Open Source projects. This has led to significant corporate interest in the Open Source topic. (Joseph Feller, 2000) Growth in infrastructure and connectivity led to the fast dissemination of Open Source. Because of the decentralized nature of the communities, Open Source differs from traditional software engineering processes in that there is no single, centralized software engineering setting (Scacchi, 2006).

The term "Open Source" was coined in 1998 to avoid confusion over the term "Free Software" which was widely used before and which was promoted by the Free Software Foundation. The Open Source Initiative defines Open Source as software which is free and provided under open source licenses and which uses practices of open collaboration. (Open source initiative, 2007). Open Source Software is usually developed by loosely organized communities who may not meet face-to face on a regular basis, but who are rather motivated by a strong sense of community feeling.

In traditional "closed source" development approach there is always the need to develop systems which work well but which also take less time and cost to be developed. This is undertaken by means of better processes and tools. Although engineering processes are rarely considered when Open Source is referred to, Open Source processes reflect many of the basic tenets of software engineering (Fitzgerald, 2011). One can see that Open Source projects have very little formal processes, very few of the projects have an explicit model for design and development. Processes are usually restricted to issue-trackers and communication tools (Boldyreff, 2003). Some may even argue that Open Source projects defy traditional software engineering best practices of measurable goals, risk management, monetary incentives for performance and formal control.

But it is also interesting to note that these practices may not, after all matter, as open source projects function equally, if not exceedingly well producing high quality output at obvious cost advantages (Fitzgerald, 2011). According to a survey of the European automotive industry conducted by management and technology consulting firm BearingPoint, the "drivers (of Open Source) included competitive differentiation, reduced development costs, increased customization agility and avoidance of vendor lock-in" (Bock, 2012). The growing interest of corporate majors in the industry like IBM and Hewlett-Packard to start open source consulting shows that Open Source is very much mainstream. Internet giants like Google and Facebook have used scalable infrastructure using open source technologies. The Future of Open Source Survey received over 1300 responses out of which 67 percent of respondents report actively encouraging developers to engage in and contribute to open source projects. The survey also revealed an active corporate open source community that delivers value, triggers innovation and shares camaraderie. (The Tenth Annual Future of Open Source Survey, 2016)

Early on, to solve the software crisis (which refer to problems related to time, cost and quality of software delivered), many "silver bullet solutions" were suggested. Frederick Brooks said that "there is no single development, in either technology or management technique" that may increase productivity. (Frederick P Brook, 1987). Many proponents of Open Source Software believe that Open Source can act as the silver bullet that allows improvement along cost, schedule and features.

Open Source communities are not particularly interested in a software process model. This could be due to several reasons. The first could be that the “hacker culture” and a “bazaar” model of development is inherent to Open Source which is fundamentally against the principles of software engineering. Another reason could be that the constantly evolving processes may not be conducive to a fixed process model. Just as closed source relies on no single process model, neither is there one process model in the Open Source world. Nevertheless, there exists some common features of all fully-fledged Open Source projects which can be thought of as a generic model. (Lonchamp, 2005)

Some efforts have been made in the past to develop a process model for Open Source Software(OSS). For instance, Mauerer and Jaeger present an engineering process with best practices, examples and comparisons with traditional methods. (Wolfgang Mauerer, 2013). The three-layered development model proposed by Lonchamp specify OSS features as “definitional”, “generic” and “specific”. (Lonchamp, 2005)

This thesis aims to provide a theory of Open Source Engineering Processes based on inputs from three different communities, supplemental materials and existing scientific literature. The categories of the Engineering processes are elicited in a tabular format with features and constraints. The model is then applied to the three Open Source Software communities. The differentiating feature of the communities is the organisation and collaboration within the community. The model can be used by Open Source Software communities to understand, compare, re-use and improve their processes.

The main contributions of this thesis are:

- A model to describe the open source engineering processes
- Application of the model to compare the engineering process of Linux kernel development, Postgresql and Tiki.

The main data sources were the three interviews with 4 practitioners from the three open source communities. Qualitative data analysis was done on these interviews where coding was applied to bring out the concepts. Other supplemental data from the project websites and articles were also used to build the theory. The engineering processes were grouped under six main categories “Decision-making”, “Product management”, “Engineering Management”, “Software Development”, “Patch Flow” and “Quality Assurance”. This is also applied to the three Open Source communities.

This Chapter presents the research in a systematic manner. It covers the Related Work which includes the literature review done as part of the thesis in Section 2.2. The high-level research question and the more granular question are discussed under Section 2.3. The steps within research and general approach is covered in Section 2.4. The data sources are outlined in Section 2.5 and the data analysis process is detailed in Section 2.6. The Section 2.7 finally presents the model of Open Source engineering processes in a tabular format and a descriptive format. It also applies the model to the three communities - Linux, Postgresql and Tiki which helps to underscore the model. Section 2.8 discusses the limitations of the research and 2.9 concludes the Research Chapter.

2.2 Related Work

A Software Engineering process allows the timely development of software. It also holds all the technology layers together for the effective delivery of software. It is a framework of the activities which lead to the development. (Pressman, 2005)

There are several accepted and popular software process models followed by traditional software projects which are prescribed to outline a process flow. The framework may be linear or incremental or evolutionary depending on the specific needs of the project and the software that is being developed. For example, the Capability Maturity Model Integration (CMMI) framework acts as an instrument for measuring performance on specific "process areas". It is detail-oriented and difficult to emulate. Increasingly, researchers are recognizing that the people and their actions influences the performance of a software process. (Alfonso Fuggetta, 2014). But most of these processes were not designed with OS communities and projects in mind.

Open Source projects are characterized by voluntary contributors and a distributed and virtual team. There has been prior research and many scientific articles have been published on Open Source Software Engineering Processes. As part of this thesis the existing literature was classified as (i) "community-based" where the literature largely dealt with people and community side of the Open Source processes like decision-making, motivation of contributors and participation and (ii) process-based where focus of the papers was process of the projects or sometimes segments within the process. (Table 4 and Table 5 in Appendix A).

It can be noted that a number of research papers concentrate on the social mechanism which exists within Open source software communities. How Open Source communities adjust themselves to function effectively has been widely studied. Also of interest, is how decision-making plays a role in the code review process and the success of Open Source projects. The role of the "core" developers and reviewers are crucial here.

When some of the existing work compares traditional engineering process models with Open Source process models, some others concentrate on the roles and responsibilities of individuals and how they fit into the process. Frame work of activities, best practices and simulation model are proposed from studying specific Open Source projects and applying it back to them. Case studies, previous literature and surveys are the main methods of research used.

While the existing research deals with the decision-making feature of the Open Source communities as a mutually exclusive factor separate from the process model, this thesis tries to understand the impact of decision-making in Open Source engineering processes. The scope of this thesis is to build an engineering process model from interviews with experts from three polar Open Source communities with respect to hierarchy and decision-making and apply the model to these three projects.

2.3 Research Question

The fundamental research question of the thesis is:

"How to model open source engineering processes?"

The initial review of existing literature and supplemental materials helped to formulate a more granular question: "How does decision-making work in open source engineering processes?". The polar sampling of cases was chosen on the dimension of maximum diversity of collaboration and how decisions are taken in these communities.

Suggestions from literature review which included aspects of Open Source adoption of enterprises were not considered as the relations and exchanges of these open source communities with commercial organizations and companies was marked as out-of-scope of the research.

2.4 Research Approach

The case study research methodology according to Yin was chosen for this research due to the reasons stated below. The type of research question which investigates "how" to model open source engineering processes called for a case study research method. The topic chosen was contemporary and "allowed to maintain the holistic and meaningful characteristics of real life events such as organizational and managerial processes". (Yin)

A multiple-case study design was chosen for the thesis. Unit of analysis was the process in the three open source communities that were chosen. Each open source community - Linux kernel, Postgresql and Tiki CMS was chosen as a separate and contrasting/polar case.

During the designing of the case study research a case study protocol was created according to Yin which is detailed in the Appendix C. This helped with design of units of analysis, defining the type of case study and the procedure to be followed.

The research thesis started out with the apriori knowledge that the processes in the three Open Source projects would be fit into the model which would be developed, although the processes of the three projects are considered different mainly because of the difference in the styles of collaboration. So, essentially theoretical sampling of projects on the dimension of decision-making was to be done.

Three open source communities were chosen for investigating the engineering processes. The choice was a polar sampling of cases based on the dimension of maximum diversity of collaboration. Hence, the three open source projects chosen were:

- Linux - single-rooted collaborative hierarchy
- PostgreSQL - core team of equals (peers) with supporting contributors
- Tiki - free for all and everyone can write

For building the theory, qualitative data analysis of interviews and supplementary materials was chosen as the main technique. Following the case study design and protocol creation, a systematic literature review was done on scientific articles and research papers. This helped as background reading and also in refining the interview guidelines for practitioners. Interview guidelines helped to organize the questions which would be used to steer the course of the interviews (Appendix D). The data was primarily collected using semi-structured interviews with practitioners from the three projects mentioned above. The Table 1 below depicts the profiles of the practitioners interviewed.

Role	Open Source Community	Experience
Core contributor	Linux Kernel	~20 years
Major contributor and Committer	Postgresql	~9 years
Project Admin	Tiki CMS	~15 years
Project Admin	Tiki CMS	~15 years

Table 1: Interviewee Profiles

The audio files of the recorded interviews were then transcribed to text format. Qualitative data analysis was done on the transcripts of the interviews using QDAcity - an in-house tool developed by the Open Source Research chair at the FAU. An integrated method (Grounded Theory as well as Deductive) was employed to do coding of the interview transcripts. This involved "microanalysis" and "constant comparison" of the interview transcripts, labelling "concepts" and grouping them into "categories". Figure 1 depicts the research process.

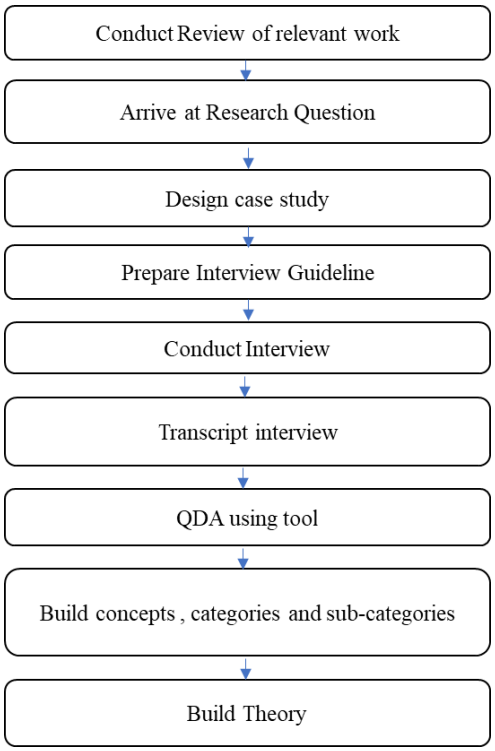


Figure 1: Research Process

For formulating the model into a tabular form and applying the model to the three communities, the article "A model of open source developer foundations" (Dirk Riehle, 2012) was taken as reference.

2.5 Used Data Sources

Semi-structured interviews with practitioners of the three open source communities were the primary data source for the thesis. After short-listing the candidates for interview based on compatibility of relevant experience, availability and willingness, they were reached out via e-mail. Once the practitioners confirmed to participate in the interviews a second e-mail was sent to them giving them more details on the goal and method of research of the thesis (Appendix E). Their permission to record the call was requested and the interview guidelines were shared when asked for.

An initial data assimilation and comparison was done for the three projects based on information gathered from the corresponding websites and articles on the projects. Please refer to the Appendix B for the comparison. This helped to understand the processes specific to each of these Open Source projects more distinctly. Some of the papers, also mentioned under the section Related Work talked directly about the engineering process modelling in the Open Source projects and also specifically on the decision-making mechanisms existing in the Open Source communities. Such data was also directly used in the building of the model.

2.6 Data Analysis Process

Qualitative data analysis(QDA) is used to unearth phenomena by understanding the underlying concepts and the connections among these concepts, thus generating a theory. This form of research is employed when data cannot be measured. Data for research is gathered through different means like interviews or focus groups and the collected data is iteratively analyzed.

To analyze the data in an organized manner, coding process is followed. It helps not only in classifying data but also in understanding the relationships that lie underneath.

Different methods of coding may be used in qualitative research. The most popular Grounded Theory approach is purely inductive and does not force code systems. The deductive method of coding starts with an initial set of codes which may be taken from literature review or already well-known concepts. An integrated approach uses both the emerging codes as well as the pre-determined code structure.

This thesis concerns itself with software engineering processes which is already a well-established subject with a wealth of knowledge. Hence when coding the transcripts of interviews, some of the pre-known concepts of software engineering processes was used for labelling. For example, *"Roles and Resource allocation", "Coding", "Review", "Testing" "Version Control", "Release Planning"*. These were processes which were discussed during the interview and hence could be easily labelled. The definitions on when to use these codes were straightforward and they were constantly compared within different interviews.

Other than these pre-conceived codes, some other themes or codes became apparent during the analysis. Examples of these codes are *"Patch commit", "Patch submission", "Philosophy that drives the project"*

The table in the Appendix F shows the QDA output of the coding done using the tool QDAcity. Once these codes were labelled against the instances from the interview, they were grouped appropriately under a code group. *"Patch Submission"* and *"Patch commit"* were grouped under *"Patch Flow"*. The codes *"Review", "Testing"* and *"Release Management"* come under *"Quality Assurance"*

To build a model from this code structure, the codes and code-groups were further analyzed and re-grouped. Some of the codes were combined and some were dropped. For example, when coding *"Product Road-mapping"* and *"Product Specifications"* were coded separately. But later it was combined as a sub-category "Specifications/Features" for better coherence.

The code-group *"Collaboration"* emerged from the codes *"Communication", "Conflict-resolution" and "Decision-making"*. Here, the overarching concept that reflected throughout was the *"Decision-making"* and hence it emerged as a Category. Through constant comparison and referring to the online literature of the three projects, six categories emerged very evidently. These were listed down to form the resulting model.

2.7 Research Results

The non-linear process of literature review, interviews and the data analysis resulted in the emergence of categories and sub-categories. The six main categories which emerged were *Decision Making, Product Management, Engineering Management, Software development, Patch flow* and *Quality Assurance.* The relationship between different categories of the model are represented in the Figure 2 below.

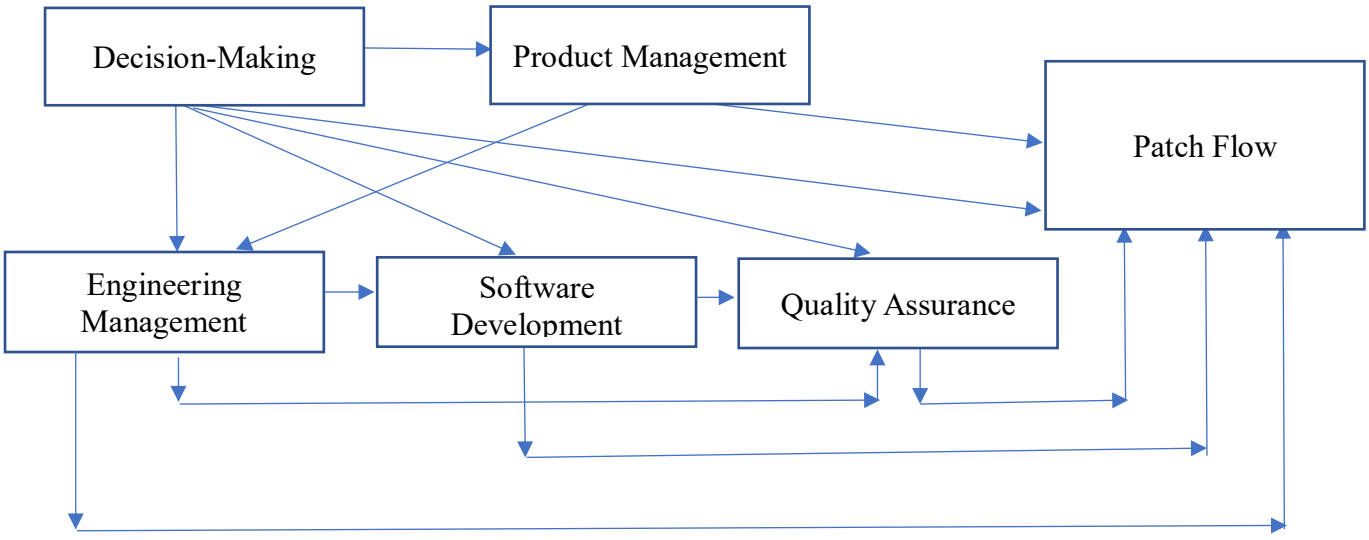


Figure 2: Categories in the model

In this section, the tabular form of the model is presented. It is organized in a tabular form (Table 2) as:

Column 1: Categories,

Column 2: Sub-categories

Column 3: Possible Features

Column 4: specifies the constraints among the features of each sub-category. The model is further explained on the basis of categories and sub-categories in the following sub-sections.

2.7.1 The Decision-Making Category

This category broadly reflects how the Open Source community functions in terms of how it is organized, how the community collaborates and how the power dynamics work within a community.

The *Organisation* sub-category: The "onion model" is used to represent the social structure of an Open Source community. The Open Source community may choose to be hierarchical with a definite path of escalation and a Benevolent Dictator for Life (BDFL) with users, committers, core-committers and a Project Leader. It may also have an organisation of users, contributors, major contributors and a Core team. The other possibility is to have a free-for-all model where anyone can commit and there is no defined path of escalation. There may be other options of organisation which lie outside these three possibilities. But for sake of simplicity, these alone are considered here. It may be noted that even in the BDFL model, many duties are delegated to the core team of "lieutenants". Similarly, in the free-for-all model, when taking crucial decisions, an experienced contributor/committer gets to enforce more influence than the less experienced contributor.

Category	Sub-category	Possible features	Constraints
Decision Making	Organisation	Hierarchical	Single choice
		Core Team	
		Free for all	
	Conflict resolution	Mostly through discussion	Multiple choice
		Can be escalated	
		Prominent members may have more say	
	Collaboration	Mailing lists	Multiple choice
		Other online communication (forum, webinars)	
		Traditional face-to-face occasionally	
	Chance of a new-comer's patch getting committed	High	Single choice
		Low	
Product Management	Specifications/Features	Picked by companies	Multiple choice
		Picked by individuals	
		May arise out of external triggers	
	Release planning	Centralized timelines	Multiple choice
		Features in each release depend on the patches	
Engineering Management	Release management	Hierarchical- What goes in is decided by some committers	Single choice
		Release management team in place	
		Self-organized release management	
	Roles	Strict roles of committer and contributor	Single choice
		Contributor is also the committer	
	Process improvement	Ad-hoc	Multiple choice
		Done in isolation	
Software development	Design and coding	Done privately	Multiple choice
		Done publicly	
		Need to confirm to coding guidelines	
	Version control	Distributed (like Git)	Single choice
		Centralized (like SVN)	
Patch flow	Patch submission	Via mailing list	Single choice
		Direct commit	
	Patch commit	Follows review	Multiple choice
		Done by a committer	
		Done by contributor	
		Commit at regular intervals	
Quality Assurance	Review	According to review checklist/tool	Multiple choice
		Done by a non-contributor of the patch	
		No specific review process	
	Testing	Regression testing is done	Multiple choice
		Use of testing tools	
		Prioritized according to use of feature	

Table 2: Model – tabular representation

Some of the excerpts from the interviews on the hierarchy are listed below. *"..all the discussion will also be done on the mailing lists as a reply or follow up to the patch you sent…. with that everyone reading the mailing lists can infer the status of the patch. So, if there is an agreement on that patch and you got the reviewers, then yes it will be applied. If there is not an agreement then this patch will typically not be applied. There is only very, very, very rare cases when a patch will be applied despite being, haven't reached an agreement... very infrequently."* Linux

"I think having multiple people involved rather than a single leader whose decisions can't be challenged is probably a good thing because it does leave room for Debate and disagreement particularly on a small Project. But you have a project which one committer... and that person is the only authority figure in the community, then, you know if that person makes a bad decision...there is nobody to come back and say... whoa...whoa…whoa...And by having multiple people involved you kind of avoid that. At the same time, by limiting it to a core team of people, rather than a very large group, you know, you retain some control as long as we have a group of committers who broadly agree among themselves about what the, eh. . the goals are" Postgresql

"So basically, some people could think that's ok, let us say we are 7 people on the board of directors or admin group in this case, if we add up an 8th one, like I am losing part of my power. but that's not the way to look at.. the way is that we have someone that we feel is at least as good as everybody else that's on the group. so, we are having more wisdom and more people that can play that role, and if some more people come along we take them and even if some people are bit less active, we still trust their judgements." Tiki

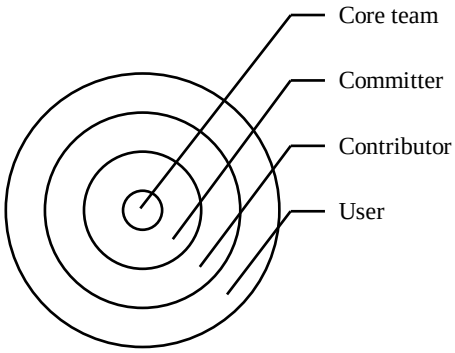


Figure 3: The simplified Onion Model

The *Conflict Resolution* Sub-category: Open Source communities are more informal compared to the traditional closed source software development and the developers have a high degree of autonomy. But invariably, conflict situations do arise. As the Linux practitioner pointed out "conflicts are resolved through discussion and more discussion". This is rightly the case with all three communities. The difference primarily arises if an issue can be possibly escalated to a "higher" authority in a hierarchical community. In others, there may be very less chance of a conflict because everyone has their way and there is no waiting for approval. But in all communities, it is noted that in case of an issue the experienced developer gets their way almost always. The Linux example quote says:

"If you fall out with a maintainer or whatever, maybe on a completely unrelated issue and the maintainer does not apply your patch, there is nothing you can do about it. You can try to apply to a higher authority but a higher authority in this case will be Linus, so it's not something that you do lightly or virtually impossible for a typical contributor"

".. we kind of count the votes... So, we say, you know, 6 people weighed in on this issue 5 of them wanted one thing and one of them wanted the other thing so we're going to go with 5 people wanted. ...em ...some people's voice carries more weight than others.." Postgresql

".. the wiki way is that we are building something together… and in the wiki culture it is very acceptable for someone else to say oh let's clean that up, organize and together we build a plan. and that changes the philosophy vs people that object to something. like in a discussion forum, someone could say, I disagree. ok. but in the wiki way it's like ok what do you propose? you don't like my proposal, rewrite it, rewrite it" Tiki.

The *Collaboration* Sub-category: Different channels of communication are used by Open Source communities to work in a distributed environment. The most widely used platform for communication is the mailing list. Other online communication like forums and webinars are also preferred for discussion and knowledge sharing. Occasionally community members get together for real-life meetings. These communication events help to bring in a personal context to the exchanges.

The *Chance of a new-comer's patch getting committed* Sub-category: In a hierarchical Open Source community as well as in an Open Source community with a core group, the only chance for a new code getting submitted is by a committer picking up the code change (patch). This is relatively easier if it is submitted by an established developer within the community. New-comers' code is subjected to stricter reviews and is usually picked up, if at all, after a lot of back and forth communication. In a Free-for-all model, this problem does not arise as the developer is by default, a committer too.

2.7.2 The Product Management Category

This category concerns itself with the requirements engineering, product road-mapping and the feature-wise implementation.

"there is an, well, overall roadmap where the communities/the most active developers agree how the development should progress. But there is typically no time limit attached to it when certain features will be or should be ready or should be incorporated" Linux

"*Individual companies sometimes post things about what they intend to work on in the coming year. We are on a 1-year release cycle and sometimes people who work for particular company will say that they're going to work on this topic. But that's, you know something which is done by the individual companies not by the project"* Postgresql

"So, someone comes and say hey we think you should do this and we want to do it well there is nobody there to say oh no we shouldn't. that's like Tiki has grown to having tons of features, and it is actually the free open source web application with most built-in features and scope is very large." Tiki

The *Specifications/Features* Sub-category: Open Source communities do not have a formal specifications document. What they do have is a bug-tracker or a wish-list. The features of the product depend on the contributions that are submitted. Organizations may push for their desired features by submitting patches through their developers. Users or developers themselves may submit changes while "scratching their personal itch". Changes in the ecosystem may sometimes trigger some changes to the features.

The *Release Planning* Sub-category: It is noted that as open Source communities evolve and grow, they find the right interval and schedule for major releases. The release timelines are planned centrally by the core or admin group. The features or changes which go into each release depend on the contributions submitted by developers in that interval.

2.7.3 The Engineering Management Category

This category talks about the project management, resource allocation and release planning activities.

The *Release Management* Sub-category: In a hierarchical community, the leaders make the decision on which contributions or patches actually go in to the actual code-base for each release. The contributors - especially the less experienced ones - may not have much influence on this decision other than requesting for reviews and hoping that their code gets "pulled in". Some communities have a release management team in place to make decisions on last-minute submissions and to oversee the releases. Certain Open Source communities also have a more informal and self-organized release management.

"at the end of each release cycle we choose a 3-person release management team and that group of three people is allowed after the feature freeze date to you know, basically by fiat, by a vote of those three people, are allowed to make whatever decisions they need to make in order to get the release out on time" Postgresql

The *Roles* Sub-category: Although Open Source communities do not have strict roles assigned, the two most important roles played by practitioners are that of Contributor and Committer (elite team of capable developers who have write access). The committer can also be a contributor. The committer role is either explicitly assigned for experienced contributors or the community may decide that any contributor can also be a committer.

The *Process Improvement* Sub-category: Process improvement initiatives are done in an ad hoc manner and in isolation. It is done more as a better way of doing things or as a solution to a problem encountered. It is not a formal process which carried out and earmarked as "process improvement".

"We try to have a yearly conference for the... for the subsystem. ..where those issues will and should be discussed but again, there is no real process enforced for that" Linux

"there is no centralized command and control and therefore there is nothing like you know, a continuous process improvement plan or anything like that because that would require a central office control which we do not have and we do not want." Postgresql

"every year or two especially when we get together there's a huge plan to move to git. and there is a road map for that. and we definitely need to do that at some point but it is just a question when and because so much stuff will break" Tiki

2.7.4 The Software Development Category

All activities and processes related to the development are included here.

The *Design and Coding* Sub-category: The development activities can be done privately by a person, but the preferred method is to do it openly involving a larger audience. Projects may enforce a strict adherence to a coding style or direct the contributors to follow some specific programming language guideline. As a best practice of respecting the environment, committing early and frequently is encouraged as opposed to working in isolation.

"first thing you do is checking ...Alright maybe someone else did a similar thing. So, you look at other drivers, other subsystems, alright how did they fix it..and then you, more often than not you find.. ah yeah that sounds similar, that looks similar to what I need. You copy it over, adapt it to your needs and there you go which means that you really have some sort of fashion how things are coded.. into... in the kernel itself" Linux

"that's really again up to individual developers. Generally, we do encourage people to post designs or ideas on the mailing list before they go and write the code because if they don't then they may find that they spend a lot of work developing a feature which is not something that people feel is a good idea." Postgresql

"so, it was deliberately very declarative and simple so that people could get stuff done and write new stuff and add new features without having to know object oriented programming and hierarchies and inheritance and so on and then gradually as times got on, there's been a need because things got more and more complex and there has been a need for a proper, you know structured code design" Tiki

The *Version Control* Sub-category: Distributed or centralized version control tools may be used in Open Source development. There is a tendency to move towards distributed version control.

2.7.5 The Patch Flow Category

This category may be the most important process in an Open Source project. The planning, creation and submission of source code is done by the contributor, whereas review and commit to code-base is the responsibility of the Committer.

"the official workflow is that, you...actually the developer submits your patch with proper description. Then, this patch will be getting reviewed and if it's being reviewed then the maintainer will be picking up the patch and it will be, then merged in first with the maintainer subtree and then subsequently with the next pull request into the Linux kernel proper" Linux

"someone will create a patch and they will post it to the mailing list...eh and they will add it to our patch tracker which is commitfest.postgresql.org. and hopefully someone will take an interest in it and review it. If it's a simple patch, then a committer such as me may take a quick look at it….. if it's a more complex patch, then typically couple of committers will, not necessarily look at it right away, although they may, in some cases if they happen to be particularly interested in what the patch does, eh, hopefully but other people will come along and help with, with code reviews, with testing…when at least one non-committer reviewer thinks that the patch is ready for the committer to look at it then the status in the patch tracker gets set to "ready for committer" and then hopefully a committer will look at it and often provide

more review feedback but sometimes not, sometimes they will just commit it directly" Postgresql

"Most of the time people just get commit access. We don't have any kind of commit team or really any appraisal system" Tiki

The *Patch Submission* Sub-category: Contributors can submit their code changes to a mailing list where they will be picked up for review and feedback is provided which may lead to a possible commit or more discussion.

The *Patch Commit* Sub-category: The submitted changes or patches undergo a review and feedback process. The guidelines for review and the process may vary according to the subsystem within a project. The contributor can request another contributor or committer for review or wait for the patch to be picked up for review and submission. When the contributor also has commit access, the patch can be directly committed by the contributor

2.7.6 The Quality Assurance Category

Reviews and different types of testing which ensures the quality of the product fall under this category.

"best practices are supposed to be caught by the reviewers" Linux

"We do have a checklist of things to review and that's very often used especially by new reviewers, to kind of help them understand what the expectations are" Postgresql

"We have unit tests which are mainly for the backend parts of the system. ……. we have got plans for a continuous integration testing system." Postgresql

The *Review* Sub-category: Review of contributions may be done as per a review checklist. Code review is implicitly done in every Open Source project. This checklist may depend on the project or the sub-system. Reviews are carried out by anyone other than the person who submitted the code. That said, some projects do not have a formal or specific review process and it depends on the type of change that is submitted

The *Testing* Sub-category: Other than the unit tests done, regression testing is part of the Open Source development process and is usually automated. Use of testing tools are common. Testing is largely proportionate to the complexity and priority of the feature.

In addition to the categories mentioned above, one other factor that emerged throughout the course of the research was the overarching "Philosophy of the Open Source communities" which cannot perhaps be categorized under the processes but is more abstract and defines the purpose which binds the communities together and helps them grow.

2.7.7 Application of the Model to the three Open Source Projects

The derived model could be applied to the three Open Source communities as shown Table 3. The possible features could be appropriately mapped to the projects as the model itself was derived from data concerning the three projects.

Category	Sub-category	Linux Kernel	Postgresql	Tiki
Decision Making	Organisation	Hierarchical	Core Team	Free for all
	Conflict resolution	Mostly through discussion	Mostly through discussion	Mostly through discussion
		Can be escalated	Prominent members may have more say	Prominent members may have more say
		Prominent members may have more say		
	Collaboration	Mailing lists	Mailing lists	Mailing lists
		Other online communication (forum, webinars, chats)	Other online communication (forum, webinars, chats)	Other online communication (forum, webinars, chats)
		Traditional face-to-face occasionally	Traditional face-to-face occasionally	Traditional face-to-face occasionally
	Chance of a newcomer's patch getting committed	Low	Low	High
Product Management	Specifications/Features	Picked by companies	Picked by companies	Picked by companies
		Picked by individuals	Picked by individuals	Picked by individuals
		May arise out of external triggers	May arise out of external triggers	May arise out of external triggers
	Release planning	Centralized timelines	Centralized timelines	Centralized timelines
		Features in each release depend on the patches	Features in each release depend on the patches	Features in each release depend on the patches
Engineering Management	Release management	Hierarchical- What goes in is decided by some committers	Release management team in place	Self-organized release management

	Roles	Strict roles of committer and contributor	Strict roles of committer and contributor	Contributor is also the committer
	Process improvement	Ad-hoc	Ad-hoc	Ad-hoc
		Done in isolation	Done in isolation	Done in isolation
Software Development	Design and Coding	May be done privately	May be done privately	May be done privately
		Done publicly	Done publicly	Done publicly
		Need to confirm to coding guidelines		
	Version control	Distributed (like Git)	Distributed (like Git)	Centralized (like SVN)
Patch flow	Patch submission	Via mailing list	Via mailing list	Direct commit
	Patch commit	Follows review	Follows review	May be done by contributor
		May be done by a committer	May be done by a committer	Commit at regular intervals
Quality Assurance	Review	According to review checklist/tool	According to review checklist/tool	No specific review process
		Done by a non-contributor of the patch	Done by a non-contributor of the patch	
	Testing	Regression testing is done	Regression testing is done	Use of testing tools
		Use of testing tools	Use of testing tools	Prioritized according to use of feature
		Prioritized according to use of feature	Prioritized according to use of feature	

Table 3: Application of the model to three open source communities

2.8 Limitations

The three case studies chosen are polar in terms of hierarchy which is why they were chosen but they were also totally different in terms of the application domains or the products offered. While the Linux Kernel development concerns with an operating system kernel, Postgresql is a relational database and Tiki is a web application platform which makes it hard to compare the development processes followed within these communities. The size of the communities compared are also varied with Linux having thousands of contributors, Postgresql and Tiki in the hundreds range. The research does not claim to be applicable to all Open Source Communities but rather depicts the types of communities that were studied. The fact that only four practitioners were interviewed as part of the research is another limitation of the research.

2.9 Conclusion

This Master thesis proposes a theory of Open Source Engineering Processes. Using Qualitative Data Analysis, it provides a tabular model with Categories, Sub-categories and Possible values of the engineering processes followed in the open source communities. In order to validate the model and to establish its effectiveness, it was applied to three Open Source communities. This theory aims to help Open Source communities in designing and developing their own processes.

Appendix A Related Work

Literature	Focus
(MRÓWKA, 2012)	Outlines the specific nature of decision-making in open source projects, the probability of success of an Open Source project and the advantages and disadvantages of a group decision-making model.
(Gläser, 2012)	Discusses the social mechanism creating social order in Open Source communities. Compares Open Source communities to scientific communities in that both work on common product and are self-adjusting communities
(Yan Li, 2012)	Proposes a model to assess the relationship between leadership style and developer's motivation to contribute in Open Source communities.
(O'Mahony, 2007)	Main principles critical to community-managed governance are identified with respect to open source communities.
(Toshiki Hirao, 2015)	Studies the collective decision-making in code review process. Majority method of voting is only used as a reference by the core reviewer who makes the final decision.
(Christopher Oezbek, 2010)	The "onion model" of gradually varying degrees of participation is validated against participation in OSS project mailing list traffic

Table 4 Category: Community-based

(Han Lai, 2012)	Establishes a framework to define the metamodel of Requirements Elicitation process. Creates a template for the process
(I. P. Antoniades, 2002)	Proposes a dynamic simulation model for the development processes of Open Source software projects. The model is applied to an Apache case study to produce indicative simulation results
(Keng Siau, 2013)	Uses Grounded Theory Approach to propose a Phase-Role-Skill-Responsibility Open Source Software Development Process Model. Different roles in the Open Source community are required to have certain skills and responsibilities which correspond to phases of the Open Source development process.
(Tiwari, 2011)	Studies existing literature on Open Source models, compares traditional software engineering development model with Open Source model.
(Wolfgang Mauerer, 2013)	Compares traditional approaches to software engineering and Open Source methods. Proposes best practices with examples.

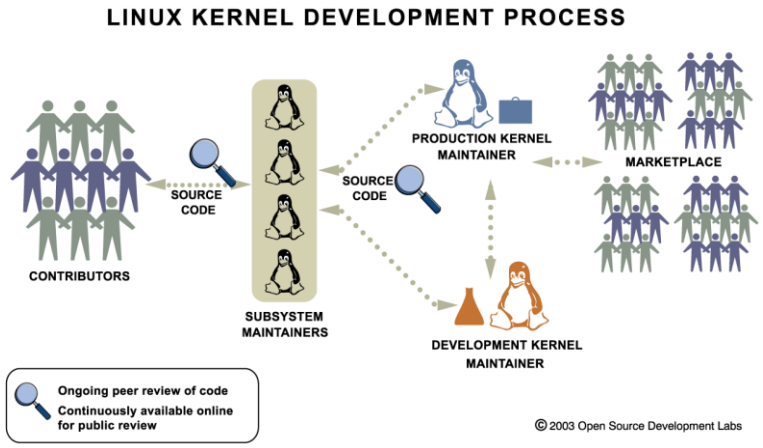
(Timo Koponen, 2005)	Proposes a framework for open source maintenance process. Four activities within Open Source maintenance were found similar to ISO/IEC framework.
(Mehrdad Nurolahzade, 2009)	By studying the development process of the Mozilla foundation, how different roles involved affect the patch-review process are analyzed.
(Audris Mockus, 2002)	Data from two major Open Source projects are used to quantify several aspects of Open Source software development process. A hybrid model is then proposed.
(Lua Marcelo Muriana, 2014)	Uses a survey to study how Knowledge Management stimulates Quality Assurance in developing Open Source settings.
(Kevin Crowston, 2003)	Success of Open Source and range of measures to assess the success are identified considering the Open Source development process.
(Bahamdain, 2015)	Quality assurance and quality control within Open Source - stakeholders, Quality Assurance frameworks, problems affecting quality of Open Source software development, comparison to closed source software

Table 5 Category: Process-based

Appendix B Comparison of the 3 Open Source Communities

The Linux Kernel Development project

The Linux Kernel is an operating system kernel. The open source community is hierarchical with Linus Torvalds at the top and subsystem maintainers or his trusted lieutenants. Working with the community, especially for a beginner, involves being able to take criticism, requests for changes or even silence. The development process involves a major release every 2 or 3 months with the timelines being managed by Linus. It has a merge window, fixes to problems window and then a final stable release. The patches merged are tested and staged before the merge window. Coding guidelines are strictly followed. Patches are submitted to the mailing lists and may receive feedback. Once reviewed, it may be taken up the subsystem maintainer. Here, it receives more feedback. Maintainers ask Linus to pull their changes. Mailing Lists and the Internet relay chats are preferred form of communication.



The Postgresql project

Postgresql is a relational database system. The development process of Postgresql is not elaborated or illustrated as that of the Linux Kernel Development. The general work flow for a patch is explained in the Postgresql website as follows

Patch review and commit

There's basically three different workflows a patch can follow after it's been submitted that lead to it being commited:

Workflow A:

1. You post patch to pgsql-hackers
2. A committer picks it up immediately and commits it.

Workflow B:

1. You post a patch to pgsql-hackers
2. You add the patch to the open commitfest queue
3. A committer picks up the patch from the queue, and commits it

Workflow C:

1. You post a patch to pgsql-hackers
2. Bruce adds the patch to a list of unapplied patches
3. At the beginning of the next commit fest, Alvaro (with the help from others, I hope) goes through the list, and adds the patch to the open commitfest queue
4. A committer picks up the patch from the queue, and commits it

At any of these stages, your patch might instead be rejected for technical, style, or other reasons. These rejections will normally come with feedback on whether an improved version of that patch would be more acceptable. In those cases, you should consider updating your patch based on that feedback and re-submit.

Postgresql has a core team which co-ordinate releases and act as an Admin team for website. It follows coding guidelines and uses some checklists for review and testing. Patches are submitted to mailing list and follows review and commit by a committer.

The Tiki project

Tiki has a development model which is different than most of the other comparable projects. (really!) In a nutshell: **Software made the wiki way** and the FLOSS Web Application with the most built-in features.

The "Tiki model" consists of:

- Open Wiki community (do-ocracy) (Sort of like Wikipedia but for software instead of content)
- **Wiki Way** participation to the code (● 500+ with full write access to the complete code base)
- ● Scheduled releases (2 major releases per year with Long Term Support versions, like Ubuntu)
- All-in-one codebase (1 000 000 lines of code, with everything bundled. Each feature is optional)
  - Inherent synchronized releases (all features have to be ready at the same time)
  - Half the code comes from other projects such as Zend Framework, jQuery, Smarty, etc.
- Lots of features, but no duplication (in a wiki, similar/related content is merged, so the same is applied to features)
  - Tiki is the Free/Open Source Web Application with the most built-in features.
- Dogfood (Tiki is a community recursively developing a community management system)

Tiki is a web application platform with the most built-in features. Any reasonable person can get commit access. It follows respecting the environment by committing early and often and making the features optional. So, there is no plug-in architecture. The project believes in recruiting collaborative people as contributors and eliminating the initial hostility. Testing is based on priority of features.

Appendix C Case Study Protocol

1. Background

a) identify previous research on the topic - *Literature review to be done as two parts - Prior art w directly related to the thesis and which will help in creating the model. Related work which is literature related to the thesis and help as background reading. Some of the key-words to be used for searching are "open source" "engineering process models in open source" "decision-making in open source communities"*
b) define the main research question being addressed by this study - *How to model open source engineering processes?*
c) identify any additional research questions that will be addressed - *How does decision making work in open source engineering processes?*

2. Design

a) identify whether single-case or multiple-case and embedded or holistic designs will be used, and show the logical links between these and the research questions - *multi-case holistic as three different open source communities are chosen to study their engineering processes.*
b) describe the object of study - *To create a model of engineering processes in open source communities*
c) identify any propositions or sub-questions derived from each research question and the measures to be used to investigate the propositions - *From the initial literature review and information obtained from the community websites, software engineering practices of product development will be used for creating the interview guideline and subsequent qualitative data analysis of interviews.*

3. Case Selection

a) Criteria for case selection -*3 polar cases selected based on the dimension of collaboration*.
4. Case Study Procedures and Roles

a) Procedures governing field procedures - *Interviews of 2 practitioners each from each community is planned. Interview questions to guide and steer the interview to be prepared*

b) Roles of case study research team members - not applicable

5. Data Collection

a) identify the data to be collected - *Data concerning the three open source communities selected from semi-structured interview with practitioners and from their websites as well as from articles.*

b) define a data collection plan - *interviews to be recorded as audio files and later to be transcribed as text files. Other relevant data to be collected from websites and online publications*

c) define how the data will be stored - *interviews to be stored as audio (.m4a) files. Transcripts to be stored as .rtf files.*

6. Analysis

a) identify the criteria for interpreting case study findings - *Qualitative data analysis using coding. Ordering the data in concepts, categories and subcategories.*
b) identify which data elements are used to address which research question/sub question/proposition and how the data elements will be combined to answer the question-*Answers of specific questions guide the analysis in the direction of the research question. Also, constant comparison of coding each interview and refining the coding results help to identify the emerging theory.*

7. Plan Validity
a) construct validity - show that the correct operational measures are planned for the concepts being studied. Tactics for ensuring this include using multiple sources of evidence, establishing chains of evidence, expert reviews of draft protocols and reports - *Data from interviews as well as supplementary sources will be used to build the theory. The developed model will be sent to the interviewees for feedback*

c) external validity – identify the domain to which study finding can be generalized. Tactics include using theory for single-case studies and using multiple-case studies to investigate outcomes in different contexts - *Multiple case study method is used to predict contrasting results.*

8. Study Limitations

Specify residual validity issues including potential conflicts of interest (i.e. that are inherent in the problem, rather than arising from the plan). - *The open source communities although are similar in the sense that they are "open source", vary in the technical solutions they offer and therefore the processes may vary leading to difficulties in forming correlations.*

9. Reporting

Identify target audience, relationship to larger studies (Yin, 2003) - *Practitioners in the open source communities especially the leaders and steering groups can use the model to compare with their own processes and help them create or improve their processes.*

10. Schedule

Give time estimates for all of the major steps: Planning, Data Collection, Data Analysis, Reporting. - *Six months are allocated for the project and will be used in the planning, data collection and analysis as well as reporting.*

11. Appendices

a) Validation: *Review by supervisor, member checking,*

b) Divergences: update while conducting the study by noting any divergences from the above steps. *Divergences are updated in the thesis goals*

Appendix D Interview Guidelines/Questions

Interview guideline for semi-structured interview about open source processes.

Introduction
This interview is part of my research on "A theory of three open source engineering processes". The multidimensional model of the theory will be derived using qualitative data analysis applied to the interviews of practitioners from three very different examples - The Linux kernel, the PostgreSQL database, and the Tiki (Wiki CMS) software. The three examples are then described as instances of this model.
Could you please provide permission to record this interview? Thanks.

Main questions

General
- Could you please tell me what your main role in the project is?
- Who are the key players involved in the process and what are their responsibilities?
- Let's assume, a developer created a patch. How should they submit it and what will happen until the patch makes it into the final codebase?

Product management
- How are the requirements picked for a particular release?
- How is feedback from user community integrated into the feature list/ bug tracker?
- Is there a roadmap for new features triggered by changes in the ecosystem?
- Who makes the decisions about the release features?

Engineering management
- What are the development processes (design, coding, testing) followed?
- Which activities/processes make the development better or more efficient?
- Which processes need improvement?
- What is the version control process within the project?
- Is there some kind of a resource planning (especially for contributors)?
- Are there continuous process improvement initiatives? Are processes updated (lessons learned)?

Software Development
- Adherence to coding philosophy - How is it enforced?
- What is the process for creating and maintaining user documentation (how-tos)?
- Who creates the coding guidelines? How is it maintained?

Quality Assurance
- How important is testing in the process? What testing methods are used?
- How is regression testing ensured? Who decides the test cases?
- How are code reviews enforced?

Collaboration
- How do the contributors/committers collaborate?
- How are conflicts resolved?
- How transparent and open is the decision-making process?
- What is the influence of contributors in the decision-making process?

Conclusions
According to you, how does this process of "hierarchical/peer group/free-for-all" method affect the overall functioning?

Thank you for your time and agreeing to co-operate for my research.

Appendix E Communication to Interviewees seeking consent

Thank you for agreeing to let us interview you as part of our research.

My name is Harisree Radhakrishnan and I am doing my master thesis under Prof. Dr. Dirk Riehle's chair of Open Source Research at the Friedrich-Alexander University, Erlangen-Nuremberg.

This interview is part of my research thesis on open source engineering processes. Three very different open source communities are chosen for this research - The Linux kernel, the PostgreSQL database, and the Tiki (Wiki CMS) software. We are interviewing practitioners from these three communities. Using qualitative data analysis applied to the interviews and additional materials, we are aiming to build a theory of open source processes. The dimension which is of most interest to us in the theory building is how the decision-making process works in open source engineering process.

Please also find attached the interview prep questionnaire for your reference. The interviews will take about 40-50 min.

Since we would like to do qualitative data analysis on the interviews and are focused on high quality input material, we just wanted to check with you if it would be ok that we record the interview. The recordings and later the transcriptions are confidential and will never be published.

Please let us know which communication mode is convenient for you. I would suggest a Skype audio call.

To proceed further, could you please let us know **what date(s) and time(s)** would be convenient to you for the interview.

Please let me know if you would like to have any further information.

Thank You.

Appendix F Code System

Code group	Code	When to use	Coding instances
Product Management			
	Product road mapping	This code is associated with processes which help define how the project should progress or evolve	15
	Product specifications	This code is used for practices related to the requirements gathering and definition of specifications of the product	23
	Release planning	This code is used for processes which plan the feature wise implementation	11
Engineering Management			
	Roles and Resource Allocation	This code is used for practices of how work gets done by contributors and how a task gets picked up	11
	Process improvement	This code is used for practices which improve the project, product or the processes itself or shows potential for improvement	27
Software development			
	Coding	This code is used for the programming practices	14
	Coding guidelines	This code is used for processes which describe the coding guidelines, how it is imposed and its benefits	11
	Version control	This code is associated with the configuration management of source code and other configurable items	4
Quality Assurance			
	Review	This code is associated with the practices of reviewing the code and related changes that are submitted	14
	Testing	This code is related to all the testing practices employed within the project. It encompasses manual and automated testing and also the pre-requisites for a successful testing	12
	Release management	This code is associated with the processes of deploying the product across releases.	10
Collaboration			
	Communication	This code is used for describing how the people working on the project communicate among themselves	31
	Conflict resolution	This code is used to describe the possible causes of conflicts and how they are resolved	20
	Decision making	This code is used for practices of decision-making, escalation of conflicts, overrides and final decision. It also is used to describe the openness and transparency of decisions	26
Patch flow			

	Patch submission	This code is used for the processes of planning, creation of source code and submission by a contributor	18
	Patch commit	This code is used for the processes of picking up, reviewing and committing code	19
Barrier to enter the project		This code is used where references are made about the ease or difficulty to be part of the community	10
Philosophy that drives the project		This code is used where references to the larger goal of the community is made	12

Table 6: The Code System

References

Alfonso Fuggetta, E. D. (2014). Software Process. *Proceedings of the on Future of Software Engineering* (pp. 1-12). Hyderabad: ACM.

Audris Mockus, R. T. (2002). Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology, Vol. 11, No. 3*, 309-346.

Bahamdain, S. S. (2015). Open Source Software (OSS) Quality Assurance: A Survey Paper. *Procedia Computer Science, Vol 56*, 459 – 464.

Bock, A. (2012, March 26). *BearingPoint Study Reveals Broad Use of Free and Open Source Software in Automotive Industry* . Retrieved from BearingPoint : https://www.bearingpoint.com/en/about-us/news-and-media/press-releases/bearingpoint-study-reveals-broad-use-of-free-and-open-source-software-in-automotive-industry-need-to-improve-compliance-across-software-supply-chains/

Boldyreff, C. L. (2003). Open-Source Development Processes and Tools. *Taking Stock of the Bazaar: Proceedings of the 3rd Workshop on Open Source Software Engineering ICSE'03 International Conference on Software Engineering* , (pp. 15-18). Portland, Oregon.

Christopher Oezbek, L. P. (2010). The onion has cancer: some social network analysis visualizations of open source project communication. *Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development* (pp. 5-10). Cape Town: ACM.

Dirk Riehle, S. B. (2012). A Model of Open Source Developer Foundations. *Proceedings of the 8th International Conference on Open Source Systems.* Springer.

Fitzgerald, B. (2011, October). Open Source Software: Lessons from and for Software Engineering . *IEEE Computer Society*, pp. 25-30.

Frederick P Brook, J. (1987). No Silver Bullet : Essence and Accidents of Software Engineering. *IEEE Computer Society*.

Gläser, J. (2012). The Social Order of Open Source Software Production. *International Journal of Open Source Software and Processes*, 1-15.

Han Lai, R. P. (2012). A Lightweight Forum-based Distributed Requirement Elicitation Process for open source community. *International Journal of Advancements in Computing Technology(IJACT)*, 138-145.

I. P. Antoniades, I. S. (2002). A Novel SimulationModel for the Development Process of Open Source Software Projects. *SOFTWARE PROCESS IMPROVEMENT AND PRACTICE*, 173-188.

Joseph Feller, B. F. (2000). A Framework Analysis of the Open Source Software Development Paradigm. *W. Orlikowski et al (Eds) Proc. of 21st International Conference on Information Systems.* Australia.

Keng Siau, Y. T. (2013). Open Source Software Development Process Model:A Grounded Theory Approach. *Journal of Global Information Management, 21(4),*, 103-120.

Kevin Crowston, H. A. (2003). Defining Open Source Software project success. *Proceedings of the 24th International Conference on Information Systems (ICIS).* Seattle.

Lonchamp, J. (2005). Open Source Software Development Process Modeling. In J. Lonchamp, *Software Process Modeling* (pp. 29-64). Boston: Springer.

Lua Marcelo Muriana, C. M. (2014). Development of Open Source Software, a Qualitative View in a Knowledge Management Approach. *Proceedings of the 16th International Conference on Enterprise Information Systems* (pp. 391-399). Lisbon: Scitepress.

Mehrdad Nurolahzade, S. M. (2009). The role of patch review in software evolution: an analysis of the mozilla firefox. *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software*

evolution (Evol) (pp. 9-18). Amsterdam: ACM.
Morkel Theunissen, D. K. (2007). Corporate-, Agile- and Open Source SoftwareDevelopment: A Witch's Brew or An Elixirof Life? *Balancing Agility and Formalism in Software Engineering, Second IFIP TC 2 Central and East European Conference on Software Engineering Techniques, CEE-SET* (pp. 84-95). IFIP.
MRÓWKA, R. (2012). Decision-making in the process of implementation of open source projects. *ISSN 2029-8234 , Business Systems and Economics*.
O'Mahony, S. (2007). The governance of open source initiatives: what does it mean to be community managed? *Journal of Management and Governance*, 139-150.
Odence, P. (2013). *It's No Myth: Compliance Is Good Business, Linux Collaboration Summit.* Black Duck.
Open source initiative. (2007, 3 22). Retrieved from Open source definition: https://opensource.org/osd
Pearl Brereton, B. K. (2008). Using a Protocol Template for Case Study Planning. *12th International Conference on Evaluation and Assessment in Software Engineering (EASE).* University of Bari, Italy: Electronic Workshops in Computing eWiC.
Pressman, R. S. (2005). *Software Engineering: A Practitioner's Approach Sixth Edition.* McGraw Hill.
Scacchi, W. (2006). Understanding Open Source Software Evolution. In J. F.-R. Nazim H. Madhavji, *Software Evolution and Feedback: Theory and Practice* (pp. 181-202). Wiley Publishers.
Strauss, A., & Corbin, J. (1998). *Basics of Qualitative Research : Techniques and Procedures for Developing Grounded Theory.* Sage Publications, Inc.
The Tenth Annual Future of Open Source Survey. (2016). Retrieved from Blackduck software: https://www.blackducksoftware.com/2016-future-of-open-source
Timo Koponen, V. H. (2005). Open source software maintenance process framework. *5-WOSSE Proceedings of the fifth workshop on Open source software engineering* (pp. 1-5). Missouri: ACM.
Tiwari, V. (2011). Software Engineering Issues in Development Models of Open Source Software. *International Journal of Computer Sci ence and Technology Vo 2, Issue 2*, 38-44.
Toshiki Hirao, A. I.-i. (2015). Pilot study of collective decision-making in the code review process. *CASCON '15 Proceedings of the 25th Annual International Conference on Computer Science and Software Engineering*, (pp. 248-251). Markham, Canada.
Wolfgang Mauerer, M. J. (2013). Open Source Engineering Processes. *it - Information Technology. 55. . 10.1515/itit.2013.1008*, 196-203.
Yan Li, C.-H. T.-H. (2012). Leadership characteristics and developers' motivation in open source software development. *Information & Management*, 257-267.
Yin, R. K. (2009). *Case Study Research: Design and Methods (4th Edition).* Sage Publications.