THE POSITION OF QUALITY ASSURANCE CONTRIBUTORS IN FREE/LIBRE OPEN SOURCE SOFTWARE COMMUNITIES

BARHAM, Adina

Doctoral Dissertation
Graduate School of Social Sciences
Hitotsubashi University

一橋大学審査学位論文

博士論文

品質保証の貢献者がフリー・オープンソースソフトウェア・
コミュニティに占める位置に関する実証研究

バーハム・アディナ

一橋大学大学院社会学研究科博士後期課程

SD112005

Abstract

The Free/Libre Open Source Software (FLOSS) has created new ways for end users to interact with software. However, with the growth of FLOSS and diversification of its user base, communities are having to implement a more rigorously controlled development process in order to produce high quality products. Such a process includes verification steps and implementing formal quality assurance (QA) procedures that are increasingly present in FLOSS development. These changes in development affect the community structure as new groups of contributors emerge. Although FLOSS communities have been the main focus of many studies, little research has been done on how this emerging group of individuals performing QA tasks is integrated in the community structure. This study aims to start filling this gap by answering whether this emerging group is a separate layer of contributors in the community. In addition, considering the correlation between access to information and productivity, communication patterns between members of this emergent layer and between these members and members of other community layers will be analysed.

Research was conducted in two phases. First, a pilot case study (the Mozilla community) was conducted and based on the findings a set of hypotheses was proposed with respect to QA contributors in FLOSS communities. The second phase consisted in analysing four case studies (the Ubuntu, Plone, KDE and LibreOffice communities) in order to test and refine the hypotheses proposed in the the first phase. In each case the study employed social network analysis techniques to a dataset of mailing list and issue tracker communication. The findings suggest that in the Ubuntu and LibreOffice communities QA represents a somewhat separate layer of contributors whereas in the Plone and KDE communities QA does not seem to form a separate layer. QA teams do not display consistent growth over time but show more irregular patterns with spikes and troughs in metrics such as size, degree and betweenness values. With respect to communication patterns, findings suggest that members performing QA tasks communicate both among themselves and directly with members contributing in other areas of the project.

Acknowledgements

First of all I would like to thank my advisor Jonathan Lewis for all the great advice, support and guidance that he provided for the last 4 years. I would also like to thank everyone at Lero – The Irish Software Engineering Research Centre (Klaas-Jan Stol, John Noll, Brian Fiztgerald, Padraig O'Leary, Howell Jordan, and others) for hosting me in March 2013 and for the invaluable feedback. I am also grateful to Sulayman Sowe for his useful suggestions. Last but not least I would like to thank all the participants and organisers of the Open Source Software Doctoral Consortium at OSS 2011, 2012 and 2013 for providing an excellent environment for exchanging research ideas.

Publications

Parts of this thesis (ideas, figures, results, and discussions) have appeared previously in the following publications:

1. Adina Barham. The Emergence of Quality Assurance Practices in Free/Libre Open Source Software: A Case Study. In *Proceedings of the 9th IFIP WG 2.13 International Conference, OSS 2013, Koper-Capodistria, Slovenia, June 25-28, 2013*, volume 404, 2013, pages 271–276. Springer, 2013.

2. Adina Barham. The impact of QA practices on FLOSS communities. In *Proceedings of the OSS 2013 Doctoral Consortium*, volume 22, pages 37–56. Skövde University Studies in Informatics, 2013.

3. Adina Barham. The Impact of Formal QA Practices on FLOSS Communities The Case of Mozilla. In *Proceedings of the 8th IFIP WG 2.13 International Conference, OSS 2012, Hammamet, Tunisia, September 10-13, 2012*, volume 378, 2012, pages 262–267. Springer, 2012.

4. Adina Barham. QA practices and FLOSS communities. In *Proceedings of the OSS 2012 Doctoral Consortium*, volume 21, pages 11–25. Tampere University of Technology, 2012.

5. Adina Barham. The emergence of quality assurance in open source software development. In *Proceedings of the OSS 2011 Doctoral Consortium*, volume 20, pages 1–19. Tampere University of Technology, 2011.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the last three decades, computers have permeated society and daily life. They have become critical to every kind of infrastructure and organisation, and software or hardware malfunctions can have far-reaching and potentially devastating effects. At the same time, computers, or programmed devices from cars to mobile telephones to cash machines to pacemakers are being used by people with no understanding of the software controlling those devices. There is thus a requirement for, on the one hand, software that is extremely reliable, and on the other, software that is extremely easy to use. These twin requirements have prompted software companies to develop and implement quality standards and quality assurance (QA) methods for achieving those standards. QA practices have become a core part of software development.

In the early days of computing it was common practice for researchers to share the source code of programs they wrote. However, with the rise of the proprietary software industry, access to much source code became restricted. This frustrated many users and developers, and led to the Free/Libre Open Source Software (FLOSS)[1] movement. Two of the most important features of FLOSS are the availability of source code to all users, and the possibility for anyone to contribute to development by joining a project community. Today, software developed by FLOSS communities is used in many parts of the software industry.

The development of QA practices by software companies and the growth of FLOSS are thus two major phenomena in the domain of software development over the last 30 years. As FLOSS products became more widely

[1]Throughout this study, the term Free/Libre Open Source Software will be used although there are differences between free/libre software and open source software [51]. The motivation behind this decision is that the development process is similar and this is the main interest of this research. In other words, FLOSS will refer to software which allows inspection, modification and redistribution of its code source.

adopted, it was unavoidable that the two phenomena would "meet". In recent years, more and more FLOSS communities have adopted formal QA procedures in their development process in an attempt to improve stability and usability. This study seeks to investigate the relationship between QA and FLOSS, and more specifically to investigate the impact of QA adoption on the structure of FLOSS communities.

1.1 Significance and Contributions of this Thesis

1.1.1 Significance

FLOSS has introduced not only innovative software development methodologies but also new ways through which users can interact with software by creating a new type of community. These communities have "brought a unique approach to intellectual property and ownership from political, sociological as well as philosophical perspectives and managed to produce unique artefacts such as norms, values and beliefs" [15]. FLOSS has had a great impact on both the software industry and software users. Its diffusion, however, has triggered an increased need for FLOSS communities to ensure quality standards. Therefore, a deeper understanding of QA adoption in the context of FLOSS is needed.

Communities are the driving force behind FLOSS development. A change in the structure of communities can therefore be an important factor in a project's success or failure. Hence, monitoring the evolution of FLOSS communities is important for both practitioners and researchers. Academic studies of FLOSS communities have focussed on a variety of topics ranging from community structure and communication patterns, career advancement and role migration, to developers' motivations and the role of voluntary contributions. However, studies of QA adoption by communities are lacking.

The stereotyped image of FLOSS communities is of a bunch of überhackers developing for their own gratification. However, research shows that FLOSS communities are comprised of individuals with diverse skill sets who contribute for various reasons ranging from hobbyist interest to professional advancement. Individuals without advanced programming skills have hitherto been able to contribute to FLOSS projects by translating or writing documentation. The emergence of QA activities in FLOSS development may offer a new area in which such non-programmers can contribute. Therefore, it is important to analyse who is contributing and how they are communicating regarding QA in FLOSS communities.

Finally, FLOSS communities, which capitalize on the close-to-zero costs of storing, processing and sharing information to create and share knowledge in a distributed, open, and informal fashion, are, as Aigran argues, a "paradigm of the new information world" [2]. The ways in which FLOSS communities adapt to ensure greater quality may therefore have implications for other areas of information-intensive economic and social activity.

1.1.2 Contributions

This study offers the first systematic empirical research on the impact of QA adoption on FLOSS communities. With the growing popularity of FLOSS products and the diversification of its user base, communities have realised the need for a more rigorously controlled development process that can ensure higher quality. As a consequence, methodologies previously used exclusively in proprietary software development such as formal quality assurance processes are increasingly present in the FLOSS landscape. These transformations in the development process suggest that FLOSS is heading towards a hybrid model that integrates elements from both the open and proprietary paradigms. The question of how and how far FLOSS communities achieve quality standards has already attracted the attention of academics; as will be shown in Chapter 2, they have conducted a variety of studies focusing on topics including the QA practices that are employed under the FLOSS model, comparisons of quality levels between proprietary and open systems, sustainability, fault prediction, and the relationship between social network metrics and quality. Thus a new question arises with respect to whether the implementation of formal QA procedures is affecting the structure of FLOSS communities. However, little research has sought to position QA activities within project communities. One reason for this gap might be the novelty of dedicated QA teams within FLOSS projects. The main objective of this research is to start filling this gap by establishing the impact of formal QA practices adoption on the structure of FLOSS communities. This goal will be achieved by analysing communities that have implemented a formal QA step in their development process and investigating how this emergent layer in the community affects its structure.

Second, this study contributes a set of hypotheses that can be validated or tested in future studies. This research consists of two phases: in the first a pilot case study of a FLOSS community in which QA is employed formally is carried out, and a set of hypotheses generated from the findings. In the second phase these hypotheses are tested on four more FLOSS communities with formal QA practices, and modified accordingly based on the results. The resulting set of hypotheses and the method used to generate them can be used in analysis of further FLOSS communities that decide to employ QA practices by assigning a dedicated QA team.

Third, this thesis contributes to research on organisations and governance by deepening our understanding of communication within voluntary knowledge communities. Open source and open source communities represent social experiments in building political economies that are based on innovative organisational structures to coordinate behaviour [87]. Weber, for example, considers that open source raises three issues from the perspective of political economy: motivation, coordination and complexity. Furthermore, he states that in a company authority mechanisms are "means of controlling specialised knowledge in highly differentiated division of labour" which is not the case for FLOSS communities in which contributions are not bound by monetary renumeration and tasks are chosen based more on personal preference rather than assigned by a superior [87]. In other words, these communities represent a community whose members develop high quality products without conforming to the traditional hierarchy of a commercial institution or market mechanism—a new kind of organisation. However, the adoption by FLOSS projects of practices and methodologies including QA from the proprietary sector may be causing changes in the way these projects operate. An empirical study of the impact of the adoption of QA practices by FLOSS projects can therefore offer a useful perspective on potentially important changes in the community-driven innovation organisation. From the perspective of FLOSS governance, an understanding of how QA-related practices are absorbed by FLOSS communities can provide insights for practitioners. For example, it would be useful for both project leaders and also potential users of the software to know if information flows within projects are dependant on a few key contributors; such dependence might imply that a project is vulnerable to the departure of those key contributors.

1.2 Research Questions

Q1: Is QA a separate layer in FLOSS communities?

FLOSS communities are usually described as following a concentrical layered model that resembles an "onion". In this FLOSS onion model, communities are formed of four main layers: passive users, active users, developers and core developers; passive and active users are together considered to be the periphery. Although this model fails to capture some complexities of FLOSS communities—see [50] and [90]—the onion model is still regarded as generally valid. This research aims to investigate how QA contributors fit into this onion model. More specifically, it asks whether QA contributors form a separate layer or whether members who belong to other layers perform QA related activities. It would be interesting to note, if all projects taken into consideration share the same structure with respect to the QA contributors. Previous research on the Firefox project has shown that peripheral members (i.e. members of the periphery as defined in the onion model of FLOSS communities) perform some QA tasks, for example posting 20 to 25% of bugs on the issue tracker [29]; this research will deepen our understanding of this question by establishing the extent of peripheral involvement for several other major FLOSS projects.

Q2: What are the communication patterns between QA members as well as with other project participants?

Previous research has shown that information access correlates with productivity and participants who have better access to information are able to contribute more efficiently [3]. Hence it would be useful to know whether there are participants who control the information flow; such dependence on a few participants may be risky for a project's long-term sustainability. This project will analyse communications between QA members to identify the central figures, in other words members with high activity levels within the community, and observe their evolution over time within the project. This requires us to analyze communication across different channels, most notably e-mail lists and issue trackers.

1.3 Outline of the Research Project

Based on existing literature regarding QA, the following working definition of QA is used for the purposes of this thesis: Testing, contributing code to automated testing tools or any test related activity, triaging bugs or any activity performed on the projects issue tracker, participating on the QA dedicated communication channels. QA team members are considered members who perform any of the tasks described in this definition. The number of FLOSS projects that are implementing formal QA activities in their development process is increasing. A preliminary assessment of the top 100 FLOSS projects listed on www.ohloh.net shows that more than a quarter of these projects include some form of QA in their development.

The approach chosen for conducting this research is the case study approach as it allows both theory building and validation. The research consists of two phases:

Phase I: A FLOSS community that has implemented QA practices is analysed as a preliminary case study. The community analyzed is Mozilla, which is a large and mature community with a long history in developing successful FLOSS products such as Firefox and Thunderbird. QA mailing list data and issue tracker data are retrieved, cleaned and analysed using first simple statistics and then social network analysis techniques. A number of hypotheses regarding the position of QA activities within FLOSS communities are formed based on the findings.

The findings of the preliminary case study suggests that in the case of Mozilla a smaller percentage of peripheral members (occasional contributors) are active on the mailing list as opposed to the issue tracker where a larger number of members have only one non-repeated act of communication. Activity on the QA mailing lists seems to be independent from activity on the issue tracker presenting peaks that are not directly related to time progression. Furthermore, a small group of people seems to be highly active in comparison to the rest of the community on both issue tracker and mailing lists. Social network analysis of the Mozilla community shows a large group of people spanning both issue tracker and mailing lists. However, almost two thirds of the connections are created by single acts of communication from one member to another. Furthermore, the existence of a highly active small group of people is also supported by the social network analysis of the Mozilla community. As regards information flow within the Mozilla community, the risk of a small group of people brokering information seems to be very low. As far as the communication carried out only on the QA lists, the patterns seem to display similarities to the whole network communication but on a smaller scale, i.e. one group of people working together where a small subgroup is highly active.

Phase II: Four other case studies are carried out. The projects studied differ from Mozilla in size and history and are: Ubuntu, Plone, KDE, and LibreOffice. The hypotheses proposed in Phase I are examined and adjusted in the light of the findings for those case studies.

More details regarding these communities and the reasons they were chosen are given in the Research Methodology chapter on page 53 as well as the Case Studies chapter. The analysis of the Mozilla community revealed that issue tracker data and QA mailing lists' data is insufficient to determine if QA represents a separate layer in the community. As a result, in addition to retrieving communication data conducted on these venues, all mailing lists associated with each project were retrieved as far as possible. A full description of the mailing lists taken into consideration is given in Annex A attached to this thesis on page 197. In addition, a list of community members contributing code to the project was downloaded from www.ohloh.net[2] which was used for both data cleaning and contributor layer identification. Thus, data triangulation is used in the sense that data associated with various venues is downloaded.

The analysis of these four case studies included general statistics methods and social network analysis techniques applied in a similar manner as for the Mozilla preliminary case study. The findings seem to validate some of the hypotheses proposed in the first phase of the research. For example, all four communities contained a large group containing most of the projects' participants that spanned mailing lists and issue trackers. Furthermore, all communities had a small number of participants with a higher than average activity that displayed strong connections among themselves and were connected to a large number of members. However, some case studies displayed particularities. For example, while in the Mozilla, Plone and KDE cases the QA contributors seemed to merge with other layers in the case of Ubuntu the QA group tended to be more separate. In other words, a smaller number of members contributing to QA activities in the latter two communities were contributing with other activities to the project while members contributing with QA activities in Mozilla, Plone and KDE seemed to bring a variety of contributions to the project by submitting code and participating to a large number of non-QA mailing lists. These findings may suggest that in some cases less technically knowledgeable individuals are finding a new way to contribute to FLOSS development aside from documentation and localisation. Further study can be conducted in this direction considering the targeted user base for these projects. Ubuntu and LibreOffice may display a more segregated and rigorous QA due to the fact that the intended end-users for these software products are not necessarily technically “savvy”.

1.4 Thesis Structure

The following section outlines the structure of this thesis as follows:

1. *Chapter 1: Introduction.* This chapter introduces the main goals of this thesis and is divided in four sub-chapters:

    1. 1 *Significance and Contributions of this Thesis.* This sub-chapter states the significance and the main contributions of this thesis to the field.

[2]Ohloh: https://www.ohloh.net/

1. 2 *Research Questions.* In this sub-chapter the main research questions are stated and explained.
1. 3 *Outline of the Research Project.* This sub-chapter summarises the research conducted in this thesis.
1. 4 *Thesis Structure.* This sub-chapter details the chapter and sub-chapter structure of this thesis.

2. *Chapter 2: Literature Review.* This chapter reviews the appropriate literature and is divided in four main sub-chapters:

   2. 1 *Free/Libre Open Source Software.* This sub-chapter presents a short history of FLOSS and essential concepts characterising the FLOSS development paradigm.
   2. 2 *Software Quality Assurance and FLOSS.* This sub-chapter presents a review of literature focusing on Software Quality Assurance from both the standpoint of proprietary as well as FLOSS development.
   2. 3 *Communities and FLOSS development.* This sub-chapter contains a review of literature focusing communities under the FLOSS development model including structure and dynamics.
   2. 4 *Summary.* This sub-chapter summarises the current chapter.

3. Chapter 3: Research Methodology. This chapter describes the research methodology and the processes used for data retrieval and analysis in the following sub-chapters:

   3. 1 *The Case Study Approach.* This sub-chapter describes the research approach and methodology in the context of research objectives. Justifications are made for the choice of the case study methodology with respect to potential criticism of said method.
   3. 2 *Data.* This sub-chapter presents the datasets included in this research as well as data models, tools used for data retrieval, chosen venues and data time span.
   3. 3 *Social Network Analysis.* This sub-chapter presents essential Social Network Analysis metrics and concepts as well as justifies the choice of said method in the context of this research.
   3. 4 *Summary.* This sub-chapter summarises the current chapter.

4. *Chapter 4: Preliminary Research and Pilot Case Study.* This chapter presents the first stage of this research and is organised in the following sub-chapters:

4. 1 *Working Definition of Quality Assurance.* This sub-chapter presents the working definition of quality assurance used in this thesis for identifying project participants contributing with QA activities.

4. 2 *Preliminary Study of QA Adoption in FLOSS Projects.* This sub-chapter presents a preliminary assessment of QA presence within popular FLOSS projects.

4. 3 *Pilot Case Study: Mozilla.* This sub-chapter presents the justifications for choosing the Mozilla community as a pilot case study and presents general statistics about the QA teams within this community as well as a detailed social network analysis.

4. 4 *Summary.* This sub-chapter summarises the current chapter.

5. *Chapter 5: Case studies.* This chapter presents the second phase of the research and is organised in the following sub-chapters:

   5. 1 *Ubuntu.* This sub-chapter presents general statistics about the QA team as well as a detailed social network analysis of the Ubuntu community.

   5. 2 *Plone.* This sub-chapter presents general statistics about the QA team as well as a detailed social network analysis of the Plone community.

   5. 3 *KDE.* This sub-chapter presents general statistics about the QA team as well as a detailed social network analysis of the KDE community.

   5. 4 *LibreOffice.* This sub-chapter presents general statistics about the QA team as well as a detailed social network analysis of the LibreOffice community.

6. *Chapter 6: Conclusions and Discussion.* This chapter presents the main conclusions of this research and is structured in the following sub-chapters:

   6. 1 *Comparative Analysis.* This sub-chapter presents a comparative analysis of all the case studies in order to verify the hypothesis presented in Chapter 4.

   6. 2 *Answers to the Research Questions.* This sub-chapter presents the general conclusions of this research, and answers to the research questions.

6. 3 *Discussion and Limitations.* This sub-chapters addresses research limitations, potential threads to validity and proposes future research directions.

7. *Appendix A:* This appendix contains tables associated with the preliminary analysis of the top 100 projects on Ohloh.

8. *Appendix B:* This appendix contains tables that enumerate all the mailing lists associated to each community and which mailing lists were retrieved and stored for each case study.

9. *Appendix C:* This appendix contains tables that detail the activity of QA member participants on all communication venues taken into consideration for this study. In addition, this appendix contains tables that describe activity on a yearly basis for each of the case studies.

Chapter 2

Literature Review

"We can't run the modern world without software." [76]

"Software is virtually inescapable in a modern world." [67]

The two sentences above, both from books that are essential reading for anyone interested in the complexities of software development processes, describe well the current relation between software and society. Not only are citizens using computers to complete more tasks everyday, but governments, banks, international corporations, universities and public transportation systems all rely on software. Moreover, this phenomenal spread of software is only just getting started: "as we move into the twenty-first century, [software] will become the driver for new advances in everything from elementary education to genetic engineering" [67]. We will become more and more dependent on software, whose potential has no natural limits due to the fact that software is independent from manufacturing processes or even the laws of physics [76]. Is it is generally accepted that software products should respect explicitly stated functional requirements, provide explicitly documented development standards and have implicit characteristics [67]. However, the Free/Libre Open Source Software (FLOSS) development paradigm has demonstrated that it can produce high quality software without adhering strictly to these requirements.

This chapter is organised into three sections: an outline of the key concepts and history of Free/Libre Open Source Software; a literature review of research conducted on software quality assurance (SQA) in the context of FLOSS; and a literature review of research conducted on communities in the context of FLOSS development.

2.1 Free/Libre Open Source Software

In the 1940s, according to Feller and Fitzgerald, computers were mainly used for scientific purposes by researchers with a mathematical or engineering background who were writing their own programs [33]. More than a decade later, computer use was starting to spread beyond the academic world, and the IBM corporation developed "a line of products that met the information-handling needs of American business" [16]. However, due to the difficulty of writing and successfully running programs, it was not unusual for software to be shared. The era of proprietary software, or closed source software as some call it, began in 1969 when IBM adopted a new marketing policy that charged separately for software products, thereby starting the multi-billion dollar software industry[1]. Despite the arising complications with monopoly battles and trials, people continued to share source code.

The free software movement was triggered in 1983 when Richard Stallman launched GNU, an alternative to the proprietary UNIX operating system. However, the movement got underway officially two years later when Stallman established the Free Software Foundation (FSF)[2] as a non-profit organisation to promote the use and development of free software. The FSF defines free software as "software that respects users' freedom and community" where "users have the freedom to run, copy, distribute, study, change and improve the software". Additionally, the FSF defines the following "essential freedoms" for a program to be considered as free software:

1. The freedom to run the program, for any purpose.

2. The freedom to study how the program works, and change it so it does your computing as you wish. Access to the source code is a precondition for this.

3. The freedom to redistribute copies so you can help your neighbour.

4. The freedom to distribute copies of your modified versions to others. By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this[3].

[1]IBM archive—timeline: http://www-03.ibm.com/ibm/history/ history/year_1969.html

[2]Free Software Foundation: http://www.fsf.org/about/ what-is-free-software

[3]The of definition free software as maintained by the FSF: http://www.gnu.org/philosophy/free-sw.html

However, the term “free” was confusing to many despite the FSF’s clarification that became a well known motto in the hacker community: “think of free as in ‘free speech’, not as in ‘free beer’ ”[4]. In addition, many associated the term “free software” with hostility towards intellectual property or even communism, which somewhat hindered the success of the movement [68].

Another key moment in the spectacular evolution of the open source movement was the launch of the Linux operating system kernel launched in 1991 by Linus Torvalds, a Finnish computer science student [60]. Linux’s success proved that high quality software can be produced under the open source paradigm—hundreds of hackers from all over the world worked together on a piece of software without expecting any financial compensation in return.

In 1997, Eric Raymond gave a paper at the Linux Kongress in Bavaria called “The Cathedral and Bazaar” that described the methodologies used in free software development [68]. The paper aimed to offer guidelines for developing software products under the bazaar model by gathering aphorisms from both the Raymond’s personal experience and common knowledge among the hacker community. Some of these aphorisms, such as “Every good work of software starts by scratching a developer’s personal itch” and “Given enough eyeballs, all bugs are shallow,” became well-known statements about open source software. Raymond’s paper was highly acclaimed in the hacker community and became a source of inspiration for many including Jim Barksdale, the CEO of Netscape who decided to release the Netscape Communicator source code—which was later named Mozilla. Barksdale’s action was unprecedented and had a great impact on both the hacker community and the software industry. The stakes were high for everyone involved in the free software movement, as the success of the Netscape project would mean proving the viability of the bazaar development model in a commercial environment. On the other hand, failure would have meant that companies would be more reluctant to employ the bazaar model. Raymond, Bruce Perens and others recognised that a strategy was needed in order to ensure both the growth of the bazaar model and the success of the Netscape browser. They decided to re-brand “free software” as “open source,” and in 1998 established the Open Source Initiative (OSI). They also used the Debian Free Software guidelines for devising the Open Source Definition (OSD)[5], which quickly became one of the most widely used definitions of open source.

In recent years FLOSS products, like software generally, have become so

[4]The definition of free software as maintained by the FSF: http://www.gnu.org/philosophy/free-sw.html

[5]Open Source Initiative—the Open Source Definition: http:// opensource.org/osd

complex that projects rarely build entirely from scratch but instead integrate code created, or components maintained, by other communities. One of the unique characteristics of the FLOSS paradigm is that it allows this reuse of source code by third parties. However, from a legal perspective, it is important to note that the FLOSS paradigm is not in contradiction with the concept of intellectual property but actually relies on licences that guarantee the freedom to create and distribute derived products [38]. There are multiple licensing schemes available but perhaps the best-known FLOSS license is the GNU General Public Licence which is "termed "Copyleft" a play on words that refers to its formal provision of lasting freedom to subsequent users, as opposed to traditional restrictions applied by copyright law" [62]. The GPL was proposed by Richard Stallman out of a desire to ensure certain freedoms to the users of GNU.

There are major differences between the "free" and the "open" paradigms as well as the organisations behind them. The first is more philosophical and ideological while the latter is more business-oriented and as a result there has been some conflict between the two camps [15]. In an attempt to capture the commonalities between the two paradigms, the acronym FLOSS was coined in 2000 by Rishab Ghosh [91] and then later that year, used by the European Commission (EC) when it funded a study on the topic. Proponents of the term argue that FLOSS incorporates both "Free" and "Open" thus giving the possibility to be neutral in the debate between the two camps. Another advantage of using FLOSS as an acronym is that the word "Libre" lacks the ambiguity of "Free"—"libre" means (only) "free" as in liberty or "free speech". Proponents of the term also point out that parts of the FLOSS acronym can be translated into other languages, with for example the F representing free (English) or frei (German), and the L representing libre (Spanish or French), livre (Portuguese), or libero (Italian), liber (Romanian) and so on. However, this term is not often used in official, non-English, documents, since the words in these languages for free (as in freedom) do not have the ambiguity problem of "free" in English.

Perhaps one of the most intriguing features of FLOSS is that this development model seems to function in spite of Brook's law that states that "adding manpower to a late software project makes it later" [13], because the more people who are brought into project the more complex it will become and the more difficult to coordinate. But if this is the case, then how did the FLOSS model become so successful? Despite the "bazaar" metaphor, the secret is high modularity—small teams of people working on separate components of the project. As Dibona et al. put it: "The bazaar looks less like a bustling, homogenous mass and more like a structured community, [...] a tribe" [30].

Mockus et al. in 2000 were among the first academics to investigate whether FLOSS has the ability to compete with or even overcome proprietary software by offering faster, better and cheaper development methods [58]. Although academic researchers studying FLOSS development at that time differed on many points, they did agree that one of FLOSS's key characteristics is that its source code is freely available for anyone to use. According to Mockus et al., this has led to a unique development process with the following main characteristics:

- FLOSS products are developed by a large number of volunteers;
- tasks are not assigned, but instead participants choose what they want to contribute to the project;
- there is no explicit system-level design, or even detailed design;
- there is no project plan, schedule or list of deliverables.

The fact that FLOSS tended to be developed by geographically distributed teams without formal coordination mechanisms made its success even harder to explain. One example of such success was the Apache server which was the most deployed web server at the time of Mockus et al.'s article (2000). Mockus et al. started with a set of six research questions aimed at both understanding the development process behind the Apache server as well as the outcomes of this process. One of the authors, who was a core member of the Apache community, wrote a description of the development process, and Mockus et al. also downloaded the developers' mailing list archive, the project's issue tracker archive, as well as the code repository. Based on their findings they formulated the following hypotheses:

Hypothesis 1: Open source developments will have a core of developers who control the code base. This core will be no larger than 10-15 people, and will create approximately 80% or more of the new functionality.

Hypothesis 2: For projects that are so large that 10-15 developers cannot write 80% of the code in a reasonable time frame, a strict code ownership policy will have to be adopted to separate the work of additional groups, creating in effect, several related FLOSS projects.

Hypothesis 3: In successful open source developments, a group larger by an order of magnitude than the core will repair defects, and yet a larger group (by another order of magnitude) will report problems.

Hypothesis 4: Open source developments that have a strong core of developers but never achieve large numbers of contributors beyond that core will be able to create new functionality but will fail because of a lack of resources devoted to finding and repairing defects in the released code.

Hypothesis 5: Defect density in open source releases will generally be lower than commercial code that has only been feature-tested, *i.e.*, received a comparable level of testing.

Hypothesis 6: In successful open source developments, the developers will also be users of the software.

Hypothesis 7: FLOSS developments exhibit very rapid responses to customer problems.

Mockus et al. conducted a follow-up study in 2002 with Mozilla as a second case study in order to test and confirm these hypotheses [59]. After testing the hypotheses within the Mozilla project the first two hypotheses were modified as following:

Hypothesis 1a: Open source developments will have a core of developers who control the code base, and will create approximately 80% or more of the new functionality. If this core group uses only informal, ad hoc means of coordinating their work, it will be no larger than 10-15 people.

Hypothesis 2a: If a project is so large that more than 10-15 people are required to complete 80% of the code in the desired time frame, then other mechanisms, rather than just informal, ad hoc arrangements, will be required in order to coordinate the work. These mechanisms may include one or more of the following: explicit development processes, individual or group code ownership, and required inspections.

The third hypothesis was not modified at the time since Mozilla did not deviate significantly from it and it was not considered to be a pure FLOSS project by Mockus et al. The fourth hypothesis was not tested because Mozilla had clearly achieved a large number of contributors. The fifth hypothesis was supported considering that defect density within the Mozilla code was comparable to density in the Apache code. However, Mockus et al. found that in the case of Mozilla there seemed to be a team of people performing testing and bug reporting activities similar to proprietary development methodologies. The sixth and seventh hypotheses were also supported, however, in the Mozilla case the response time was much longer than in the case of Apache.

Mozilla was only one project among several that divided opinions about whether it was a FLOSS project or a new hybrid type of FLOSS/proprietary development. This showed the need for a clear set of criteria for defining FLOSS projects. Feller and Fitzgerald argued for the adoption of the OSD, in other words that open source software should be defined as any software meeting the criteria laid out in the OSD [33]. However, determining what a FLOSS project actually is proved to be a challenging task due to the variety of criteria used to describe FLOSS projects and the particular characteristics of projects. Gacek et al. [37] aimed to clarify this issue by providing a set of criteria common to most FLOSS projects as well as a set of attributes that vary. This aim was achieved by investigating several open source projects, literature studies, and conducting interviews with contributors to such projects. The characteristics that were found to be common for all the considered projects were:

- OSD—the criteria as given in the OSD;
- community—behind each FLOSS project there is a community that develops the product and/or uses it;
- development improvement cycles—which can include adding significant functionality or minor changes/bug fixes;
- motivation—there are various motivations behind the contributors' decisions to join a FLOSS project;
- developers are users—developers are themselves a subset of the community behind a FLOSS project;
- process of accepting submissions—FLOSS projects use a process composed of two main parts: decision-making and dissemination of information on submissions. The implementation of this process is unique to each FLOSS project;
- modularity—a prerequisite considering the geographical distribution of FLOSS developers. [37]

One reason for the huge impact of the FLOSS movement was that its core philosophy of free access to information reached out to other domains such as education, biotechnology [47], innovation and even entertainment. Carillo and Okoli argue that FLOSS is growing not only as a development methodology but also as an alternative approach to intellectual property [15]. One key aspect of this approach is "copyleft" which not only gives the user rights to distribute and modify the original source code, but requires the user to redistribute any derivatives under the same license as the original software [6]. Initially, there were concerns that copyleft would be difficult to be implement in profitable business models. However, FLOSS projects such as Apache, Linux and Mozilla showed that this was not the case.

In addition to these economic aspects, Carillo and Okoli argued that FLOSS should be regarded as a movement "because it has significant effects on social structures related to software creation and use at all levels of society". They also discussed the negative and positive influences of FLOSS communities on the software industry.

However, as FLOSS systems became more complex, their integrity became a major issue. Capiluppi and Beecher investigated whether the repository in which a FLOSS project was maintained had any influence on its structure and decay [14]. Based on their comparison of 50 projects in two FLOSS repositories (Debian and SourceForge), they concluded that projects hosted on the Debian repository had a higher level of complexity and received higher rates of control. Therefore, "FLOSS repositories act as exogenous factors in the evolution of projects they contain".

In an age where almost every computer user and organization is, consciously or unconsciously, using FLOSS every day, FLOSS is clearly no longer the mere byproduct of a bunch of super hackers scratching their own itches. However, Fitzgerald points out that many still perceive FLOSS in a somewhat mythical and outdated way [34]. FLOSS has undergone important changes since its inception, becoming a hybrid form that is more mainstream and commercially viable, a form labeled "OSS 2.0" by Fitzgerald. One of the consequences of this transformation is that FLOSS development in itself can no longer be characterised by Raymond's bazaar metaphor but has become a deliberately designed and rigorously analysed process. However, some processes within FLOSS production, such as product delivery or support, can still fit the bazaar metaphor [68].

Another issue approached by Fitzgerald is the potential impact of OSS 2.0 on the software market as it could end the proprietary driven model that has dominated the market since the 1970s. The FLOSS movement has been changing both the supply and the demand side by offering an alternative to "traditional options" as well as new and profitable business models. Fitzgerald argues that throughout history radical movements have been slowly assimilated into the mainstream and it is likely for the same process to occur with FLOSS as it becomes an important influence in the software industry. Fitzgerald proposes a framework describing FLOSS in order to illustrate the

[6]The GPL license: http://www.gnu.org/copyleft/gpl.html

changes that it has undergone.

The OSS 2.0 hybrid form includes methodologies and practices from both open and proprietary paradigms, and has attracted the interest of researchers as it impacts the software market offering a new array of possibilities [75], [73], [12], [15], [41]. As a result, the user base of FLOSS products has been increasing, along with the demands for higher quality from FLOSS products. For this reason, a process implementing more rigorous procedures "borrowed" from the proprietary development model is emerging. One example of such procedures would be formal quality assurance.

2.2 Software Quality Assurance and FLOSS

2.2.1 Software Quality Assurance in Proprietary Development

Due to the fact that software quality assurance (SQA) is a relatively new element in the development process within FLOSS projects, it is important to clarify first what SQA represents in the context of proprietary software.

Software development processes have evolved substantially in recent decades, becoming more complex and requiring a more structured and rigorous quality assurance process in order to produce stable solutions that meet the high standards demanded by users and customers. For this purpose a clear definition of quality was needed, and it took the shape of ISO/IEC 9126-1[7]—updated and replaced in March 2011 by ISO/IEC 25010[8]—which defines a quality model in use and a product quality model that are relevant for any software product or system. According to ISO/IEC 25010, " a quality model in use is the degree to which a product or system can be used by specific users to meet their needs to achieve specific goals". Quality in use is further described by five main characteristics: effectiveness, efficiency, satisfaction, freedom from risk and context coverage. The product quality model describes product quality properties as eight characteristics: functional suitability, reliability, performance efficiency, usability, security, compatibility, maintainability and portability. Considering the complexity of both software development and quality assurance, we should understand quality assurance not only as testing but as a complex set of activities performed throughout the development process.

[7]ISO/IEC 9126-1: http://www.iso.org/iso/catalogue_detail. htm?csnumber=22749

[8]ISO/IEC 25010: http://www.iso.org/iso/iso_catalogue/ catalogue_tc/catalogue_detail.htm?csnumber=35733

Quality plays a very important role in the software industry even to the extent that quality concerns have become part of marketing strategies employed by software companies. Most software companies employ methodologies to ensure a certain quality level in their products [67]. Pressman defines software quality assurance as an “umbrella activity that is applied throughout the software process” [67]. More concretely, he states that “software quality assurance encompasses a quality management approach, effective software engineering technology (methods and tools), formal technical reviews that are applied throughout the software process, a multitiered testing strategy, control of software documentation and the changes made to it, a procedure to ensure compliance with software development standards (when applicable), and measurement and reporting mechanisms”. Pressman also stresses the following three points regarding software quality:

1. Quality is measured as conformity with the software requirements. A software product in which these requirements are not followed will usually lack quality.

2. Development criteria outline the software development process. A software product in which there criteria are not followed will usually lack quality.

3. Additional to software requirements, there is a set of implicit requirements that often goes unmentioned such as the desire for good maintainability. A software in which the explicit requirements are met but the implicit requirements are not, will usually lack quality.

Software quality assurance includes a variety of tasks that are carried out usually by two groups: software engineers and SQA groups. The software engineers usually perform technical work while the SQA groups are responsible for QA planing, oversight, record keeping, analysis and reporting. According to Pressman, an SQA team should be independent from the software engineers and should assist in producing a high quality product. A QA team should carry out the following activities:

- prepare a QA plan for the project;

- participate in the development of the project’s software process description;

- review the process description for compliance with organisation policy, internal software standards, externally imposed standards; (for example the ISO-9001), and other parts of the software project plan;

- review software engineering activities to verify compliance with the defined software process;
- identify, document, and track deviations within the project;
- verify that corrections have been made to deviations;
- coordinate the control and management of change;
- help collect and analyse software metrics [67].

Pressman argues that testing is a crucial factor in SQA and "represents the ultimate review of specification, design, and code generation". Testing activities can be classified as black-box and white-box testing. White-box, or "glass-box", testing, is usually performed by software engineers, requires high level of technical knowledge and is defined by Pressman as "a test case design method that uses the control structure of the procedural design to derive test cases". On the other hand, black-box testing is meant to verify the software's functionalities without accessing the code (the software is tested as it would be used by an end user). As black-box testing is intended to uncover a different array of errors and issues, it does not represent an alternative to white-box testing, but a complementary method. As Dijkstra pointed out, "program testing can be used to show the presence of bugs, but never to show their absence" [31].

One would expect that as a project matures so does the testing process around it, and according to Dibona [30], this is indeed true for both open source and proprietary software. According to Sommerville, a respected authority on software engineering, in proprietary software development testing usually contains three stages [76]. These three stages are complex phases in the development phase but can be described briefly as follows:

1. development testing—concurrently with the development phase and is usually performed by developers and/or system designers;
2. release testing—prior to system release and is usually performed by an independent team that ensures that the system meets the stakeholder requirements;
3. user testing—after release and is usually performed by users or potential users in their own environment [76].

Sommerville recommends that release testing should be performed by a team independent of the system's development so that the quality evaluation is performed objectively and free from the influence of development issues.

For Sommerville, the QA team should be responsible, in most cases, for carrying out the release testing, ensuring that the testing process covers the system requirements, verifying whether the system is usable, and providing records of the testing process. After the release testing phase, the system is usually delivered to the end-users, however there are cases when the release is delivered to other development teams that are developing related systems or to other departments that prepare the software product for sale. Regarding release testing, Sommerville points out that its main purpose is to "show that the system delivers its specified functionality [...] and that it doesn't fail during normal use" [76]. The testers usually perform black-box testing which is testing the system's functionalities without concerns for implementation methods.

When software quality assurance was being performed by software companies and/or was carried out in a shallow fashion using black-box testing techniques, it was difficult for researchers to obtain enough data to evaluate quality attributes or the effectiveness of QA practices. However, Spinellis et al. argue that the emergence of QA in open source software, with its transparent communications and results, promises to allow scholars to study quality assurance processes more thoroughly than was previously possible [78].

2.2.2 Evaluating Quality in FLOSS

In 2002 Stamelos et al. measured the quality characteristics of 100 randomly selected applications developed for the Linux platform and compared the results with applications developed under the proprietary paradigm. Logiscope®, a tool popular at the time of the study among large organisations for controlling the programming process, was used for performing operations such as code measurement and comparison [79]. The analysis was limited to the component level and ten metrics were used to measure code quality for a total of 606,095 lines of code (i.e. number of statements, cyclomatic complexity, maximum levels, number of paths, unconditional jumps, comment frequency, vocabulary frequency, program length, average size, and number of inputs/outputs). The findings suggested that structural code quality of FLOSS applications provides results higher than expected considering the nature of development and more specifically the limited control over the development process. On the other hand, the structural quality of the FLOSS applications analysed provided results lower than the quality implied by the industrial standard. Another interesting observation was that the average component size of an application is negatively related to the user satisfaction index.

However, the question of whether FLOSS software quality was comparable to that of proprietary software still lacked a solid empirical basis, and there was a need for a survey of QA practices [93]. Zhao and Elbaum argued that under the FLOSS model contributors can be coding, testing or writing documentation in parallel, triggering faster software evolution. They also argued that the success of FLOSS products appears to defy the recommendations of quality assurance practices employed within proprietary development. Zhao and Elbaum surveyed 229 projects active on two repositories in late 2001, using a random sample controlled for programming language, environment, and application domain.

One of their findings was that documentation of development procedures was not considered an important issue within FLOSS projects as just 32% of projects used design documents, and 20% had release plans. On the other hand, the survey confirmed the existence of testing activities within FLOSS projects considering that 58% of the projects spent more than 20% of the time on testing, while more than 15% of the projects spent more than 40% of their time on testing. Regarding the types of testing performed, the survey showed that 68% of the respondents "provide inputs trying to imitate user behaviour", 25% "provide extreme values as inputs", 25% "use assertions (assert, Junit, etc.)", 26% people adopt other validation methodologies, and 5% employ tools to assess coverage. In addition, almost 30% of the projects, regardless of size or maturity, had a testing coverage estimated at less than 30%. Zhao and Elbaum argued that the lack of mature testing techniques was surprising considering the large amount of time allocating to testing. In the case of assertions, the situation seemed to be more evolved in the sense that almost 35% of the large projects employed some sort of validation activities. With respect to user participation, the survey showed that user suggestions generated over 20% of the changes on almost 50% of the projects. In addition, in almost 20% of the projects, the users posted 40% of the bugs, and 44% of the respondents thought that users found "hard" bugs (not likely to be found by the developers). In line with the findings on end-user participation, only 14% of the respondents thought the users "don't help too much". User contribution in mature projects was higher, which was to be expected considering that these projects have been running for longer periods of time and generally tend to have larger communities. This survey also confirmed the life cycle described by Raymond [68] considering that 60% of the projects started from a need"to scratch a developer's itch" before shifting to the community and incorporating more features. Furthermore, an increasing number of projects developed under the proprietary paradigm are shifting to the FLOSS model. Zhao and Elbaum concluded their survey by arguing that we still lack knowledge with respect to the FLOSS model and that further empirical studies are needed in the field.

2.2.3 Researching QA Practices in FLOSS

The number of FLOSS products has shown a steady increase over the years, which suggests that software produced under this paradigm can offer a viable alternative to proprietary software [57]. Michlmayr et al. conducted exploratory interviews with seven FLOSS practitioners in order to identify common quality practices and issues in the FLOSS paradigm. They concluded that even mature and successful projects display quality issues that are caused by the FLOSS development model itself. Michlmayr et al. classified various practices into three categories: infrastructure (bug tracking systems, version control systems, mailing lists, etc.), processes (release, peer review, testing, etc.), and documentation (coding style, etc.). However, they noted that we lack tools and metrics to measure quality accurately. Their study showed that practices employed in FLOSS vary greatly from project to project and that more investigation is needed to define relationships between quality and practices. In some FLOSS projects, the importance of quality assurance has been recognised and dedicated teams are assigned to perform QA tasks. In addition, the following issues have been identified: unsupported code, configuration management, security updates, users without proper bug reporting skills, attracting volunteers, lack of documentation, and coordination and communication issues.

Schmidt and Porter recognised that FLOSS development has been effective in reducing both cycle-time and design, implementation and quality assurance costs [72]. However, they argued that the two main challenges facing FLOSS development were containing the long-term maintenance and evolution costs associated with QA, and ensuring the coherency of system-wide properties. The FLOSS paradigm functions, as argued by Schmidt and Porter, from two perspectives: the end-user and the software process. From the end-user perspective the benefits of the FLOSS paradigm are reduced software acquisition costs, enhanced diversity and scale and simplified collaboration. From the software process perspective the benefits are a scalable division of labour, short feedback loops, effective leverage of user community expertise and computing resources, inverted stratification, and greater opportunity for analysis and validation.

Under the FLOSS development model a large amount of resources is allocated to bug triaging activities and it is often the case that these resources are not sufficient [46]. Hooimeijer and Weimer present a basic linear regression model that predicts whether bugs are resolved in a certain period of time using a dataset of 27,000 bug reports associated with the Mozilla

Firefox project. The model uses bug report features that can be retrieved from a report a day after it has been posted. However, their results show that this model, with respect to F_1-score (a metric that is the weighted harmony mean between precision and recall, two measures used in information retrieval), achieves a slightly better performance than bug triaging. In addition, using data from more than one day does not affect the performance of the model in a significant manner. Their results also suggest that "early" features have the largest impact on the model's performance despite the fact that more data is be available. An interesting finding is that the severity metric is an important factor in the model's efficiency while severity changes made in time do not improve performance. In other words, changes made to the severity of an issue might not be a precise indicator of the bug's impact on the overall system. However, analysis has shown that the presented model is viable in the sense that it can reduce software maintenance costs, in cases where the average cost of triaging a bug is greater than 2% of the cost of ignoring such a bug.

Chengalur-Smith et al. examined the factors influencing the long-term sustainability of FLOSS projects and developed and empirically tested a model of project sustainability [17]. One of their most provocative arguments is that a small number of successful FLOSS projects are responsible for most of the movement's success. In addition, they found that abandonment is a common issue within FLOSS projects and is the result of inefficient allocation of effort. Furthermore, repeatedly contributing to a project that is then abandoned can cause disappointment and decrease developers' motivation to participate in subsequent FLOSS projects, thereby hindering the evolution of the movement. Hence, improving sustainability under the FLOSS development model would benefit both contributors and the movement itself. Chengalur-Smith et al. attempted to discover factors associated with the project and the project's environment that can affect its sustainability. For this purpose they proposed a number of hypotheses and tested them on a dataset of 2,772 projects retrieved from SourceForge.net. They found that the size of the development base in early stages of a project was a predictor of later success in attracting new contributors. In addition, project size and the niche it occupies seem to have a significant importance in attracting future contributors. With regard to niches occupied by projects, they argued that a project "occupying a larger niche is more likely to become connected to other projects from the same niche through its developer network" and thus increasing the project's legitimacy and sustainability in the same time. Chengalur-Smith et al. also found that the ability to attract new contributors affects greatly a project's sustainability in the long-run and recommend potential contributors to consider not only the current situation of a project but also its history. Another interesting finding is that the age of a project seems to influence potential users but not developers, which suggests a different assessment process for different stakeholders. The authors also conclude that predicting a project's sustainability includes understanding "both project's internal characteristics as well as how it relates to the broader FLOSS community".

In the context of an increasing user base of FLOSS products, the number of non-technical users is also increasing which produces a higher expectation of quality[45]. Hedberg et al. reviewed FLOSS literature focusing on quality and usability issues, analysing current practices, identifying gaps in the literature and proposing further research directions. They found that typically, within FLOSS development, end-users are regarded as co-developers who perform testing (comparable to beta-testing in proprietary software development), report bugs or even submit bug fixes. However, users rarely have the skills to find bugs, let alone submit fixes. "Naive" end users cannot be expected to tolerate crashes, submit bug reports or contribute patches. For this reason quality assurance activities must be pushed back into the development stages, prior to releasing a product to the general public. Hedberg et al. acknowledged the existence of high quality FLOSS products such as Apache or Linux, however they point out that the quality of FLOSS projects currently depends on the skill set of the developers involved. Just because a few large and active communities generally produce quality software, does not mean that the same is true for the majority of FLOSS projects. In most projects development lacks a strict process regarding quality and testing is usually very limited. In addition, modification requests coming from the community are rarely integrated in the final product. The lack of strict development methodologies, they argue, is also becoming an issue considering the increase of developers contributing to FLOSS projects. One impediment to implementing QA procedures within FLOSS development is that usually each community member chooses how and what to contribute, and QA activities are considered as trivial or boring by most developers. One of the conclusions of their study is that peer reviews and testing, considered by Hedberg et al. as the most efficient QA procedures, are not well suited for the FLOSS development model and more research should focus on adapting these methodologies.

Halloran and Scherlis argued in 2002 that FLOSS had reached a level of quality and dependability comparable to software produced under the proprietary development paradigm. They conducted a preliminary survey on quality practices employed by FLOSS practitioners focusing on adoptability issues [43]. Halloran and Scherlis examined eleven successful FLOSS projects by analysing data from November 2001 through March 2002. They observed that responsibility for quality was given to contributors with commit privileges to the projects' repositories, and that all the projects taken into consideration used nightly builds and visible bug tracking tools. In addition, most projects offer appropriate resources and tools for potential contributors to become visible and acknowledged by the community. Based on their observations, Halloran and Scherlis suggested criteria shared by successful FLOSS projects:

- an incremental model for quality investment and payoff;
- incremental adoptability of methods and tools both behind the server wall and in the baseline client-side tool set;
- trusted server-side implementation that can accept untrusted client-side input;
- a tool interaction style that is adoptable by FLOSS practitioners.

Five years later, Aberdour reviewed prior research on quality in FLOSS [1]. He found that substantial studies had been conducted on the issue and that, in order to achieve quality, actionable conclusions should be drawn from these empirical studies. He defined a quality model as follows:

- *quality assurance* which occurs throughout the software organisation and focuses on process and procedure learning from mistakes and ensuring good management practices;
- *quality control* which is the process of verification and validation, usually within a structured testing process, using high-level plans and detailed test scripts to document and manage the testing process.

According to Aberdour, the quality model in FLOSS differs from that employed in proprietary development in various ways. For example, in view of the distributed nature of FLOSS development, coordination tasks are essential. On the other hand, FLOSS development practices included practices "borrowed" from proprietary development such as: central management, code ownership, task ownership, planning and strategy, testing, leadership and decision-making. An important aspect in achieving quality in FLOSS is having a large and sustainable community eager to "rapidly develop code, debug code effectively and build new features".

English et al. focus on quality by conducting empirical research on identifying locations of faults in a project's source code, the number and severity of faults as well as the number of modifications associated with each part of the project [32]. Testing activities within a large system can consume a lot of resources, so concentrating testing efforts on code areas that are prone to produce faults would improve development costs in the long run without compromising quality. The authors propose four research questions that investigate whether the Pareto principle applies in fault distribution, and whether Chidamber and Kremer [18] metrics can be correlated to fault prediction. Pareto's Law states that 80% of the effects can be contributed to 20% of the causes, while Chidamber and Kremer cover a wide set of metrics from the Object Oriented model such as: weighted methods per class, depth of inheritance tree, number of children, coupling between object classes and response for a class. The findings seem to support the application of Pareto's law in fault distribution: 82% were detected in 20% of classes. Thus, identifying these problematic classes could improve product quality and reduce costs by focusing testing efforts and reducing development cycles. On the other hand, the inheritance based metrics NOC (Number Of Children) and DIT (Depth of Inheritance Tree) were not predictors of faults within the project. A reason for this finding might be that these metrics showed little variation and thus could not be used to distinguish between classes. On the other hand, the coupling metrics CBO (Coupling Between Object classes), LOC (Lines Of Code) and RFC (Response For a Class) proved to be accurate predictors of fault-prone classes.

Barbagallo et al. conducted an empirical study in order to define relationships between social network structure and quality within FLOSS projects [8]. They proposed three hypotheses, two of which described the impact of contributor centrality degree on project success, development and maintenance efforts, while a third correlated the relationship between success and project quality. Social network analysis techniques were applied by creating two-mode undirected affiliation networks. The dataset used for generating the social network contained data associated with 57,142 contributors listed on the SourgeForge.net repository as participants in various projects developed in the Java programming language. The findings seemed to support the hypothesis that centrality values correlate positively to a project's success. In addition, Barbagallo et al. found that centrality values are positively correlated with the ability to attract new contributors. However, the study also suggested that more successful projects tend to have a lower design quality, and a more "bazaar"-like development style. Thus Barbagallo et al. concluded that centrality measures represent an important metric in FLOSS projects and that centrality should be monitored. However, their finding that success negatively affects design quality raises a potentially serious issue in the long-term management of FLOSS projects.

2.2.4 Applying the Results of FLOSS QA Research

The quality of FLOSS is a matter of great economic and social importance, so a number of efforts have been made to exploit the findings of research on FLOSS quality to improve the quality of software. Schmidt and Porter launched the Skoll project[9], which they defined as "a long-term, multi-site collaborative research project that leverages key open-source assets" [72]. The goals of the Skoll project are to develop tools and methodologies that improve quality of FLOSS systems incrementally, automate processes and minimise end-user involvement. These improvements are applied to ACE and TAO, two widely deployed FLOSS toolkits used by programmers. The reason why ACE and TAO were chosen is that they share the same attributes as any other successful FLOSS project (large and mature code bases, heterogenous platform support, highly configurable, core development teams, comprehensible source control and bug tracking, large and active user communities, continuous evolution, frequent beta releases and occasional "stable" releases). However, a unique aspect of these projects is that they employ a highly automated regression testing procedure. Schmidt and Porter argue that the widespread adoption of these projects in commercial environments is due to the high quality resulting from these regression tests and FLOSS development methodologies. Their research hypothesis is that "a systematic QA process based on distributed continuous testing and profiling would prove markedly superior to the ad hoc QA processes" and is tested in the ACE and TAO projects. Schmidt and Porter's seems to support this hypothesis, as their methods found bugs that were not initially identified using the traditional ad-hoc QA process.

More recently, Spinellis et al. presented a technical and research overview of software quality assurance within FLOSS in order to establish an observatory for open source. Their research goals were to:

- create a metric plugin-based architecture and a corresponding processing engine;
- establish new product and process software metrics that take an advantage of the SQA-OSS infrastructure;
- provide an interface through the web, web services, and an Eclipse plugin that developers can use to improve the quality of their application;
- publish concrete values of product and process metrics for popular open source software;

[9]http://www.cs.umd.edu/~aporter/html/skoll.html

- set up a league of open source software applications based on user-specified criteria [78].

Spinellis et al. argued that by combining product and process metrics, new issues in software quality could be approached. One of the outcomes of their research is an "integrated platform for large scale software engineering studies" called Alitheia[10] which consists of a data collection system, a computation component and a presentation layer in the form of a website.

The topic of quality under FLOSS also attracted the interest of various organisations and governments, which have started initiatives such as the Qualipso[11] or Qualoss[12] projects. QualiPSo, the first consortium focused on FLOSS quality, ran from November 2006 to October 2010. The purpose was to aid industries and government agencies supporting innovation by providing a way to use FLOSS in developing reliable systems. To achieve this goal, technologies, processes and policies were defined and implemented. QualiPSo was the largest FLOSS initiative ever funded by the European Commission as part of its Information Society Technology (IST) initiative. Concurrently, the Qualoss project ran from September 2006 to February 2009 with the goal of "enhanc[ing] the competitive position of the European software industry by providing methodologies and tools for improving their productivity and the quality of their software products". To achieve this goal, a method to assess FLOSS quality was built in order to facilitate integration with commercial software systems[13].

2.3 Communities and FLOSS development

People themselves represent the most important resource in software development. Pressman reminds us that "the cultivation of motivated, highly skilled software people has been discussed since the 1960s" [20][28] [89]. In fact, the "people factor" is so important that the Software Engineering Institute has developed a people management capability maturity model (PM-CMM), "to enhance the readiness of software organisations to undertake increasingly complex applications by helping to attract, grow, motivate, deploy, and retain the talent needed to improve their software development capability" [26] [67].

[10]http://www.sqo-oss.org

[11]Qualipso (trust and quality in open source systems): http://www.qualipso.org/

[12]Quality in Open Source Software: http://www.qualoss.org

[13]The Qualoss project related information hosted on libresoft.es—an important online repository for libre software research: http:// libresoft.es/research/projects/qualoss

However, the unique characteristic of FLOSS projects is that it builds communities of people with similar goals who share knowledge and obey the same norms. “Gemeinschaft” or community in general is defined as a group of individuals brought together based on shared feelings of togetherness and on mutual bonds. These feelings represent a goal to be achieved while community members represent the means for achieving this goal [82]. In the context of virtual space, Rheingold defines online communities as “social aggregations that emerge from the Net when enough people carry on public discussions long enough, with sufficient human feeling, to form webs of personal relationships in cyberspace” [69], a set of criteria fulfilled by FLOSS communities.

Weber considers open source as an experiment in “social organisation for production around a distinctive notion of property” [87]. He also argues that the open source process suggests four principles for distributed innovation:

- empower people to experiment;
- enable bits of information to find each other;
- structure information so it can recombine with other pieces of information;
- create a governance system that sustains this process—social concepts expressed in rules and law. Governance of a distributed innovation system also needs to scale successfully (organised effectively, more eyeballs really are better than fewer) at relatively low overhead costs [87].

Carillo and Okoli argue that the emergence of FLOSS has radically affected the dynamics of the software landscape both within software companies and between developers, end-users and even hardware manufacturers [15]. Furthermore, this movement has created a new type of community, creating new kinds of interaction between users and software. However, despite the many positive aspects of this development, “some effects must be worthy of caution” [15]. With regards to property rights, Carillo and Okoli believe that the open source movement is changing the way property rights are being used to promote software innovation.

Furthermore, FLOSS communities have an increasing role in the software industry and can be defined as groups of individuals who convene online with the purpose of creating software products, useful for both the community itself as well as for the general public, by using open source development methodologies [15]. Communities themselves represent the driving force behind the open source movement, and studies have shown that having a sustainable community should be a priority [65][68][79]. As a result, academics and industry representatives alike have focused on understanding FLOSS communities, and on how to create and maintain communities capable of delivering "success stories" such as Firefox, Linux, and the Apache web server.

A well covered research topic concerns the actual profile of the members of these communities, more specifically, code contributors or developers. Several surveys [39][55][44] have profiled FLOSS contributors and although there are some discrepancies between these surveys the common denominators are that:

- FLOSS contributors are almost exclusively male;
- the average age a FLOSS contributor is around 27;
- around 20% of FLOSS contributors are students;
- over 50% of FLOSS contributors have IT related jobs;
- around 16% are paid to contribute to FLOSS projects.

However, in 2002, Feller and Fitzgerald recognised that there was a gap in the literature regarding FLOSS developers considering the differences between estimates given by previous research [33]. Despite the lack of demographic information, Feller and Fitzgerald were able to describe FLOSS developers as tending to be professionals and specialists, and in their opinion, this observation could explain how largely uncoordinated efforts were able to create high quality software products. Another assumption was that professionals came with a knowledge of best practices when joining a FLOSS project, a knowledge which they could use in development. Feller and Fitzgerald also observed that community members regularly switched roles between client, actor and owner and that the lines between these roles were often blurred. In addition, they observed a higher FLOSS penetration in the corporate market, although they forecasted increasing private use of FLOSS products.

Another relevant area of academic research regarding FLOSS communities is the investigation of various factors that can affect the levels of success of projects. For example, Crowston and Fogel both found that an active mailing list can be a reliable indicator of success within FLOSS projects [23][36]. Taking this research a step further, Sowe et al. conducted a study to correlate developers' activity on mailing lists with their efforts in submitting code to the project by presenting a methodology for data mining and analysis from a FLOSS repository of repositories (RoRs) [77]. RoRs are collections of source code commit data and mailing list data associated with many FLOSS project that are publicly available. Sowe et al. argued that developers' presence and activity on mailing lists is important because their activity influences the way both individuals outside and inside the community view the commitment levels of project leaders and core contributors. For the purpose of this study, Sowe et al. chose 14 random projects with two criteria in mind: the projects should cover as many domains as possible and the developer contribution should vary as much as possible in terms of commits and postings. In explaining their mining methodology, Sowe et al. raised the difficult problem of how to identify the same individuals on both mailing lists and in source code repositories. To address this issue, Sowe et al. compared data from three tables as follows:

- people table—containing name, username and e-mail address;
- messages table—containing e-mail address;
- commit table—containing committer name.

The people table was used as a common denominator in order to identify unique developers. After data analysis, Sowe et al. reached the conclusion that FLOSS developers contribute equally to code repositories and mailing lists. However, this conclusion pointed to new questions that should be answered in future studies. For example, regarding code commits, it is important to establish what kind of commits developers are submitting due to the fact that they may be modifying, deleting or even adding important new functionalities. Another question would be how exactly developers are contributing to the mailing lists, i.e. whether they are actively contributing by replying to previous messages and engaging in substantive discussions, or simply posting information and ignoring other community members.

Open source is not only growing as a way of developing software but also as a business model, with open development communities springing up in other areas of innovation [52]. Kilamo argues that as the number of companies that are choosing to go "open" is increasing, so is the demand for communities to attract contributors. However, despite the fact that these communities have a key role in deciding the success or failure of a project, there are "few guidelines on how to create and maintain a sustainable open source community". In her doctoral research, Kilamo seeks to identify issues that must be addressed to grow such communities at different aspects of open development. Her goal was not to provide a recipe for success, but to offer a set of key aspects necessary for growing open communities. She used a case study approach to investigate three different types of community: software business, educational and innovation. Kilamo et al. argued that, although companies involved in FLOSS development are heavily dependent on the communities surrounding the projects, business decisions are often made based on little information [53]. The main reason is that existing methods for measuring community metrics, such as social network analysis or business readiness rating, tend to be hard to implement. With these considerations in mind, Kilamo et al. presented a "measuring model for onion-based measuring of open source communities" using data collected continuously via multiple channels. The model was then validated by applying a measuring model to an open source community, and Kilamo et al. argued that it offers a viable method for companies to assess the community structure in order to support decision making.

Open source systems can differ greatly from one another with differences ranging from implementation of the system to the communities themselves [54]. Compared to evaluating the implementation, which can be performed using a number of methods, evaluating a community has proven to be a somewhat more difficult task. Despite the fact that some community measuring models exist, Mikkonen et al. argue that having a researcher participate in the community as a developer for a period of time is one of the best methods. They present their experiences using this method in a company-centric community with an onion-layered structure. Valuable community related data was collected despite the fact that it tends to be subjective as only one developer has participated in community activities. Mikkonen et al. found that in the sample community, experience and technical knowledge are prerequisites for participation, which may cause the community to be less approachable. In addition, their experiment suggested that participation can reveal insights that are beyond the reach of measurement and that this type of approach is suitable for collecting developer related data.

In recent years, as the number of FLOSS products and communities has increased so has the number of individuals using FLOSS products [42]. Both practitioners and academics have been in interested in the questions of how communities form, evolve and succeed. Community members often participate in more than one FLOSS project, thus creating collaboration ties with other FLOSS contributors. The strength of collaboration ties can vary, and is influenced by various factors such as intimacy or emotional intensity [40]. Hahn et al. analysed whether these collaboration ties can influence a developer's decision to join a FLOSS project [42]. They used social network analysis techniques and focussed only on factors influencing an individual's decision to join an emerging/new project and not an already established project. They defined FLOSS communities as "collaborative network[s] of developers working together in different project teams". FLOSS communication is usually asynchronous and carried out over "computer-mediated distributed networks" which might be a concern for a developer looking to join a new FLOSS project. Hahn et al. formulated three hypotheses regarding the likelihood of a developer joining a FLOSS project and the factors that affect it. The dataset used for their research consisted of data associated with newly registered projects on SourceForge.net (between September 30 and November 11, 2005), as well as data associated with all individuals involved in at least one project hosted on SourceForge.net (including projects registered prior to new project data retrieval). The final dataset contained 2,349 projects and 170,741 developers.

Hahn et al. distinguished two cases: the first where variables associated with the project initiator influenced a developer's decision to join a new project, and the second where variables associated with non-initiator contributors influenced a developer's decision to join a project. In the first case, developers were more likely to join projects initiated by individuals with whom they already had strong collaboration ties, regardless of the outcome of previous projects. On the other hand the developer's decision was not influenced by the initiator's status cues. In other words, developers did not rely on recommendations provided by other FLOSS contributors, but relied on their own experiences when joining a new project. In the second case, authors were not influenced by previous collaboration ties when choosing a new project. Hahn et al. argued that this might have been due to the importance of the project leader role within FLOSS development and communities. In addition, whether previous projects in which developers created collaboration ties were successful or not did not seem to influence their decision when joining a new one. However, the status cues played an important part of the developer's decision in the sense that developers tended to join projects where non-initiators had more experience. A somewhat surprising aspect however, was that developers were less likely to join a project where developers had a high number of past collaboration ties. Another factor that could influence a developer's decision to join a project was the perceived probability of success considering that developers were more likely to join a project with more developers than one with fewer.

In his book "The Art of Community", Bacon (a community manager for Ubuntu) proposes a list of checklists for creating and managing a thriving community [6]. Bacon covers a wide variety of topics ranging from the use of social media to ways of organising events and even methods of mediating conflict. Bacon argues that FLOSS communities function as a coordinated effort of multiple teams and that one of the keys to their success is ensuring a strong information flow between these teams so that they do not become isolated components. However, he warns that "the flow of communication between teams is a lot more complex than you would first imagine". It is important to identify the information brokers (people who belong to more than one team), observe communication patterns and optimise this communication for the teams that communicate the most in order to ensure a clear and effective exchange of information. To illustrate his recommendations, Bacon uses the development and the bug squad teams from the Ubuntu community. After an observation period, the lines of communication and how both teams interacted on the issue tracker were optimised, and the intersection of meetings and events was coordinated more efficiently.

Au et al. argued that FLOSS communities display characteristics essential to successful virtual communities such as self-governance, effective work structure and process or technology for communication and coordination [5]. They chose a sample of 118 projects of various sizes and types hosted on SourForge.net and analysed data associated with bugs in order to study the factors that affect learning processes in FLOSS projects. They proposed five hypotheses relating bug fixing activities to other factors such as community size. Based on their analysis Au et al. concluded that a number of factors can affect efficiency within a project. For example, performance is optimal in moderately sized teams, while smaller teams display a greater variability in efficiency. Team size depends in turn on other factors such as the manager's "perception of the relative importance of efficiency versus bug resolution time" [5]. Another factor that can improve efficiency is assigning bugs to specific developers; when possible assigning should be based on the developers' expertise and experience. One of the most interesting observations made by Au et al. is that developers who participate in a large number of projects are more efficient in bug fixing and contribute more to knowledge sharing.

Aral and Brynjolfsson conducted a study not within a FLOSS community but in a proprietary software company. However, their study might have a great impact if its findings were applied to FLOSS development. They examined the diffusion of information within email networks and how it affects productivity by analysing the transmission of various types of information within a company over a span of two five-month periods [3]. Their study offers valuable evidence as how access to information increases productivity and reduces costs. Aral and Brynjolfsson argue that the more individuals interact, the more likely it is for them to exchange information and for this reason the strength of the relationship between two individuals is measured as the number of e-mails exchanged between them. Two types of information are identified: "event news" and "discussion topics" where the first is simple, declarative, likely triggered by external events and is of interest for the majority of the individuals. The latter is more complex, specific, procedural, and is of interest for smaller groups of individuals. Aral and Brynjolfsson propose three hypotheses which they proceed to validate using data gathered from three sources: accounting data on co-work relationships, positions and so on, email data retrieved from the company server and surveys of demographic statistics. After analysing the data, the authors reached the conclusion that men are 55% more likely to receive information of any type. In addition, as was expected, the strength of ties is an important factor in receiving information as it increases the possibility: ten additional e-mails raises the likelihood of receiving information by 2%. In addition, path length reduces the possibility of receiving information in the sense that each additional step reduces the possibility of diffusion by 29%. Another observation is that having many relationships in common reduces the chance of receiving an information cascade. In addition, discussion topics are more prone to diffuse vertically within the hierarchy across stronger ties, while news is more prone to diffuse both laterally and vertically without being influenced by factors such as relationship strength. An important conclusion is that access to information predicts productivity and affects productivity in the sense that faster and better access to information can generate a higher revenue for the company as these individuals finish tasks faster than their counterparts that don't receive so much information or receive it later. "Employees who are aware of new ideas and information are better able to solve problems, improve decisions and conclude projects". For these reasons it is important to study how implementing new methodologies such as QA affects the structure and dynamics of FLOSS communities. In addition, investigating these matters might provide insightful information to corporations looking to adopt FLOSS or people considering contributing to FLOSS projects.

Nakakoji et al. conducted a case study on four FLOSS projects using interviews as well as quantitative data such as mailing lists. The focus of their paper was on the evolutions of systems and of communities and the relationship between the two [61]. They argued that as opposed to proprietary development, FLOSS community members can choose according to their own preferences one or more of the following eight roles: passive user, reader, bug reporter, bug fixer, peripheral developer, active developer, core member and project leader. Nakakoji et al. also pointed out that the percentage of users associated with each role can vary according to the FLOSS community and that some of these roles might not exist in some communities. The community structure is described as layered with a higher influence for roles closer to the core. Nakakoji et al. furthermore proposed three types of FLOSS project: Exploration-Oriented, Utility-Oriented and Service-Oriented.

The communities behind FLOSS projects produce not only software products but also models of and frameworks for open participation [64]. As a result the structure and dynamics of FLOSS communities have become a point of interest among researchers. The common conception is that FLOSS communities are organised in layers similar to an onion, hence the “onion” model which is the most common model of a sustainable community [1]. The onion model contains the following main layers: end-users, active users, developers (co-developers) and core developers. Another common conception is that participation and influence increase towards the inner circles/layers while the number of members of each of these layers decreases. The reason behind this layered structure is that FLOSS communities function as meritocracies in the sense that the individual’s involvement increases as they “migrate” towards the inner circles by committing bug reports, submitting patches or even new functionalities. However, only a few community members will be able to join the small core of developers. Bacon defines a meritocracy as “a system of governance in which its members are given responsibilities and recognition based on achievements, merit and talent” [6]. He also stresses the importance of having open, well-defined and enforced communication channels in order to ensure collaboration. As Aberdour puts it, “each type of member has certain responsibilities in the system’s evolution, which relates to the system’s overall quality” and “advancement through the member types is reward and recognition for each member’s abilities and achievements” [1].

Valverde et al. argued that while complex designs can emerge from a distributed process, the individuals involved will have only a limited understanding of the global pattern they are developing [83]. In the case of software development, it is only natural to expect developers to know the main purpose/goal of the software they are developing, however with the increase of project complexity the knowledge of the whole project decreases. Another issue in complex projects is that decisions depend on other project developments and will be made locally. Valverde et al. argued that understanding the principles behind self-organising FLOSS development can improve software process management. They conducted a comparative study social of organisation in wasp colonies and in FLOSS development communities by analysing patterns and functional significance. Two colonies of Polistes dominulus with a constant number of 13 wasps each (maintained artificially) and 12 small software communities with ten members each were taken into consideration for the study. However, the authors concluded that wasp communities are essentially different because in FLOSS communities reinforcement mechanisms distinguish a few core members.

In his essay “A twenty-first century science”, Duncan J. Watts argues that social science has failed to produce a coherent theoretical framework so far due to the complexity of social phenomena [86]. However, in recent years, with the possibility of accessing interactions between millions of people, researchers’ interest in social networks has grown substantially. Watts also points out that some issues need to be addressed when analysing such networks. For example, social networks are in a continuous state of change, and people engage in a variety of types of interactions. However, despite the fact that studies that consider all these issues are far from complete, progress is slowly being made.

Christley and Madey analysed the social positions as well as how the social distribution changes over time within the SourceForge.net community by social network analysis techniques [19]. They argued that community members can contribute to a project by performing various activities such as: submitting code, installing and testing the software, reporting and fixing bugs, writing documentation and posting messages on various communication channels. Their question was whether the social position system is static, in other words whether people maintain the same position in projects over time. In order to answer these questions, Christley and Madey extracted activity event data associated with 21 projects hosted on the SourceForge.net platform. Event data could be the submission or assignment of artefacts like bugs, feature requests and patches, posting of forum messages, modification of project settings, file releases, management of project tasks, documents or jobs. This activity data was used to generate a multi-relational, weighted, bipartite graph (social network) where the nodes were members of the community and projects, and the edges were links between projects and members. Christley and Madey conducted temporal analysis from two perspectives: global—in which they compared the state of the network as a whole in 2003 with the state of the network in 2004; and local—in which they took an egocentric point of view by looking at each person in the network. One of their conclusions was that insights can be extracted from different views of data. For example, for the global social network, "software user" is the largest social position despite the fact that when using temporal analysis this position does not exist. The reason is that members do not perform their primary activities very close in time in the sense that individuals may check out the source code and post a message on a forum at a much later time. Some social positions were common for both the local and global network which indicates that these positions are a common phenomena. In addition, it is possible that open source software communities are more efficient and adaptable considering that each member can select their own social position and/or change it over time to respond to the projects' needs. This aspect is clearly in contrast with proprietary software development where each individual is specialised with a clear distinction between roles such as tester, developer, architect and so on. However, Christley and Madey recognised the possibility that these projects may be not large enough to require dedicated positions and further insight could be provided by studying a number of large projects hosted on SourceForge.net (general results might have been skewed by the large number of small projects), or even the Mozilla or Apache projects.

Developer coordination after the release of a FLOSS project and management of activities such as knowledge sharing and bug fixing was investigated by Hossain and Zhu using social network analysis techniques [48]. Hossain and Zhu proposed two hypotheses describing variables that might affect FLOSS project metrics such as code quality or defect density. Data associated with 45 FLOSS projects hosted on SourceForge.net was used to test these hypotheses. Their first conclusion was that an increase in centrality or betweenness within the social network of the FLOSS community positively affects code quality. On the other hand, an increase in the density of the social network causes a decrease in code quality. However, the most important conclusion than can be drawn from this study is that social network characteristics can be correlated with performance metrics within FLOSS projects, underlying the importance of further research in this direction.

Dalle et al. analysed how bug reports and requests are handled, and more specifically whether ownership plays an important role in bug resolution [27]. In other words, they analysed whether bugs posted by a member of the core (insider) are treated differently from bugs posted by a member of the periphery (outsider). Dalle et al. argued that understanding the process through which bugs are fixed is essential for understanding the development process. For this purpose, meta-data associated with bugs that are mentioned in code commits was retrieved from both CVS (the project's code repository) and Bugzilla (the project's issue tracker). The meta-data used for this study included the number of lines added/deleted with each commit associated with a certain bug and the number of duplicate bugs. The next step consisted in retrieving two samples from the available 40,000 bugs, one from the early stages of the Firefox project and the second from a later and more mature stage. Their findings suggested that bugs reported by core members (insiders) tend to be fixed more rapidly than bugs reported by members of the periphery (outsiders), especially in the later stages of development. In addition, the number of duplicates a bug has seems not to affect the speed with which the bug is resolved.

Masmoudi et al. took this research a step further by analysing communication patterns between the core and the periphery as well as the periphery's involvement in bug related activities [29]. The motivation for this study was that if the periphery is viewed as the "eyes", as described by Raymond [68], then the mechanism through which the periphery communicates with the core is highly important. Masmoudi et al. assumed that most of the interaction between these two "layers" of the community takes place on the issue tracker. To assess the situation within the Firefox community, they only considered bugs that were mentioned in the code commit comments in order to rule out any unproductive activity or noise, and they distinguished three key elements: action, comments and affiliation. In order to differentiate between peripheral and core members, the following method was used: if the bug was posted directly with the status "New" then the reporter was considered as a member of the core as he/she had privileges while if a bug was posted with the status "Unconfirmed" the reporter was considered as a member of the periphery. Masmoudi et al. found that 20-25% of the bugs were posted by members of the periphery and that most of these peripheral members post only one bug. However, if all actions contributed (i.e. comments) were considered, 85% of bugs could be traced to core members, who on average post 16 bugs. Another interesting finding is that bugs with more interactions are usually associated with the involvement of core members. In addition, core and periphery activities were analysed using textual content analysis which revealed that a clear distinction could be made between people taking a black-box approach and people having a strong technical knowledge. Another conclusion was that members of the periphery are disproportionately marking duplicate bugs. By performing additional analysis, Masmoudi et al. concluded that people who participate in a discussion at a later time are less likely to be members of the core while as discussions contain a higher number of core members, more core members are likely to join in. Furthermore, after even further analysis, they concluded that actions performed by core members tend to be followed by actions performed by other core members and that, similarly, actions performed by peripheral members are generally followed by actions performed by other peripheral members. However, discussions around patches tend to be a common ground for interaction between peripheral and core members; this confirms Alan Cox's dictum "show me the code" [21] which highlights the importance of suggesting solutions in the case of "outsiders who want to be heard" [29]. Summarizing Masmoudi et al.'s conclusions, most bug resolution activities are performed by a small number of core members while peripheral members seem to contribute with duplicate identification and reporting bugs.

Crowston and Howison analysed the social structure of FLOSS projects by examining 120 projects hosted on the SourceForge.net repository [24]. They argued that "effective FLOSS projects [...] are organised like 'bazaars' but ideally not like 'town councils' or 'cliques' and certainly not like teams building a 'cathedral' ". One of their motivations was to understand team practices and more specifically team coordination, team control, socialisation, continuity and learning. Another motivation was to assess or manage risks due to the fact the some community members were making contributions so crucial that their withdrawal would jeopardise the whole project. Therefore, keeping an up to date assessment on who these contributors are might aid maintaining a project's sustainability in the long run.

Crowston and Howison argued that in FLOSS development there are two forms of centralisation: development centralisation and communication centralisation [24]. The first is an action measure while the latter is an interaction measure. Development centralisation refers mainly to code writing while communication centralisation refers to centralisation in communications channels such as e-mail lists, bug-reporting systems, and instant messaging. The common opinion among FLOSS practitioners is that FLOSS projects display a decentralised communication pattern as opposed to proprietary development, or "cathedral", where communication would be centred around an "architect". A high degree of communication centralisation implies that a small number of individuals interact with a large number of individuals who do not communicate among themselves. On the other hand, a high communication decentralisation implies that that all individuals interact with each other.

Despite the potential implications of various communication patterns within FLOSS projects, Crowston and Howison noted that academic research on the topic is scarce [24]. Previous FLOSS literature suggests that FLOSS networks have a few nodes with a number of relationships significantly higher than the network's average, called hubs [8]. Crowston and Howison, in order to test whether there is a consistency in community structure within FLOSS, analysed communication carried out on issue trackers (bug-reporting systems). Crowston and Howison used a dataset limited to active projects with over 100 bugs and more than seven developers. It contained 140 projects with 61,068 associated bug reports and a total of 14,992 unique contributors, of whom 1280 were involved in more than one project. Their findings revealed that communications within FLOSS projects do not conform to a single pattern as had been expected. However, they found a correlation between FLOSS project size and the communication pattern: the bigger the project the more decentralised the communications tended to be. This variance also suggested the communities looking to grow should be achieving higher modularity. In another paper, Crowston et al. analysed self-organisation within FLOSS communities by examining three active and successful projects [25]. One of the main motivations behind this study was that distributed development is increasingly present in the software industry and FLOSS development in particular faces specific challenges. Crowston et al. defined large-scale software engineering as "a social activity involving numerous developers and other professionals working together in a tightly coordinated process". An inductive multiple case study research approach was chosen to analyse task assignment due to the fact that the current literature did not describe such processes. With this approach, Crowston et al. developed new theories after analysing FLOSS contributors' interaction focusing on task assignment processes. Their dataset contained developer communication carried out either on e-mail lists or web-based forums as project coordination activities are usually performed by developers through these public channels. Their findings suggested that self-assignment is the most common mechanism as opposed to traditional software development where assignments are explicit and usually made by managers/leaders. Crowston et al. argued that this was probably due to the fact that FLOSS is based on volunteering in the sense that each contributor chooses tasks that he/she finds interesting. After self-assignment was removed from the equation, developers were found to assign most of the tasks, which might suggest a status hierarchy as users rarely assign other users. Another interesting finding was a broader participation in tasks compared to proprietary development where tasks are restricted to the development team and not to users.

Oezbek et al. used social network analysis of mailing list traffic to assess participation of layers within the "onion" model [64]. They chose visualisation as an output method after a careful comparison of the advantages and disadvantages of both visualisation and metrics. Among the advantages of visualisation is that it allows easy identification of central participants, comparison of similar networks and assessment of the general structure of the network. However, the main disadvantage with visualisation techniques is that when analysing large network the process must be automated which implies choosing a layout algorithm. Oezbek et al. considered only emails exchanged in the year 2007 and proceeded to manually extract data from publicly available mailing list archives associated with 11 projects. All projects displayed a similar structure of a tightly integrated core, a loosely connected set of co-developers which had strong connections with the core and some connections among themselves, and peripheral members who were mostly connected to members of the core or were isolated. In other words, contrary to prior assumptions of strong peer-to-peer assistance, members of the periphery rarely communicated with co-developers. Oezbek et al. called this co-developer layer "cancerous" in that it is strongly oriented to the core while paying little attention to the periphery. On the other hand, core developers were much more engaged in communicating with peripheral members. These findings, they argued, suggested serious limitations in the onion model of FLOSS communities.

FLOSS communities are not exclusively composed of developers [37] but also include members who contribute with: dealing with bug reporting and end-user documentation [15]. Migration within the hierarchy of FLOSS projects' communities and role migration within the community is another issue that has caught the attention of researchers. Jensen and Scacchi provided an empirical analysis of career advancement and role migration in three large FLOSS communities: Mozilla, Apache and NetBeans [50]. They argued that the organisational structure of FLOSS communities has not drawn the same amount of interest as FLOSS processes. They also question the ability of the concentric "onion model" to capture the complex structure of FLOSS communities which have multiple tracks of project career advancement and different role-sets. Another motivation behind their study was that beside recruitment related guidance, little information is provided by communities as to how one ascends and transcends the organisational hierarchy. Jensen and Scacchi used a mixture of qualitative methods (participant interviews, collection and cross-coding of FLOSS artefacts) and web site data mining in order to describe the processes behind a member's evolution within the community hierarchy for each of the three cases. Jensen and Scacchi proceeded with a comparative case analysis in which the peculiarities of FLOSS organisational structure were underlined. They concluded that compared to traditional software development where the hierarchy is more rigid and static, the FLOSS hierarchy tends to be more fluid with both horizontal and vertical migrations as well as overlapping roles. Another interesting aspect revealed by their study was that even if the number of layers comprising a community is fixed sometime in the early stages of a project, the size of these layers, especially the outer layers, is highly variable. In addition, in the case of FLOSS, the mixture between paid and volunteer developers drives a more complex career advancement path than that suggested by previous literature.

2.4 Summary

In conclusion, the previous research supports the idea that FLOSS is heading towards a hybrid model that employs some practices and methodologies from the proprietary development model. These practices include well-defined quality assurance practices such as testing. Meanwhile, quality under the FLOSS model has also attracted the attention of academics who have conducted a variety of studies to measure the quality of FLOSS, to survey the QA practices being implemented in FLOSS projects, and to reflect the findings of those studies back to FLOSS communities through various analytical tools. With respect to communities developing FLOSS systems research has also approached a variety of topics ranging from community structure or communication patterns, career advancement and role migration, to developer's motivation to voluntarily contribute to FLOSS projects.

However, despite extensive research on both QA and communities little research has sought to link QA with the structure and dynamics of FLOSS communities. One reason behind this gap might be the novelty of dedicated QA teams within FLOSS projects. This research aims to start filling that gap by analysing some communities that have successfully implemented a formal QA step in their development processes, investigating how this emergent activity affects community structure, and formulating hypotheses which can be confirmed in future research.

Chapter 3

Research Methodology

Considering the importance of communities as resource for FLOSS development, it is important to understand the mechanisms behind their evolution. Thus, the main goal of this study is to identify the impact of newly emerged groups of QA contributors on the overall structure and dynamics of FLOSS communities. In other words, the aim is to discover whether these groups represent a new layer of contributors to this project, who are the members of these groups and how they communicate with other groups of contributors. This chapter describes the methodologies used to conduct this study and is split into three sections. The first section explains the choice of the case study approach used in this study. The second section describes the data used in the study and explains how it was retrieved and processed. The third section explains the choice of social network analysis as the main tool used in this research, and introduces some of the key concepts of social network analysis.

3.1 The Case Study Approach

3.1.1 The Case Study as a Research Method

The main method used for conducting this research is the case study approach. A case study is an empirical inquiry that investigates a contemporary phenomenon in depth and within its real life context, especially when the boundaries between phenomenon and context are not clearly evident [92]. According to Robson, a case study should be carefully planned and should contain the following [70]:

- objective—the research goal;
- the case—the investigated phenomena;

- theory—the research framework/theory on which the research is based;
- research questions—questions associated with the investigated phenomena;
- methods—data collection and analysis methods; and
- selection strategy—data selection process.

Case studies have become an established research methodology that has enabled many contributions to fields including software engineering. Flyvbjerg examines common misunderstandings about case study research, arguing in line with the Kuhnian insight that "a scientific discipline without a large number of thoroughly executed case studies is a discipline without systematic production of exemplars, and a discipline without exemplars is an ineffective one" [35]. Flyvbjerg proceeds to define the five most common misunderstandings as follows:

1. general, theoretical (context-independent) knowledge is more valuable than concrete, practical (context-dependent) knowledge;
2. one cannot generalise on the basis of an individual case; therefore, the case study cannot contribute to scientific development;
3. the case study is most useful for generating hypotheses; that is, in the first phase of a total research process, whereas other methods are more suitable for hypotheses testing and theory building;
4. the case study contains a bias toward verification, that is, a tendency to confirm the researcher's preconceived notions; and
5. it is often difficult to summarise and develop general propositions and theories on the basis of specific case studies [35].

Flyvbjerg argues that humans need context-dependent knowledge in order to acquire a specialised set of skills as one only becomes an expert by acquiring knowledge from a multitude of concrete real life cases. Furthermore, Flyvbjerg contends that predictive theory does not exist in social sciences, which has produced only context-dependent knowledge. Therefore, one can draw the conclusion that in "the study of human affairs" practical knowledge is more valuable. Historically, Newton, Einstein, Galileo, Freud and many others have used case studies to make important contributions to their fields. Flyvbjerg thus argues that case study results have lead to scientific innovation and should not be disregarded as a formal research method. Furthermore, case studies should not be limited to generating hypotheses, as generalizability can be increased by a strategic selection of the case studies. In addition, as opposed to random sampling that can show only the symptoms and frequency of a problem, a few valid case studies can offer important insight as to the causes and consequences. With regards to bias in case studies, Flyvbjerg argues that these concerns apply to all research methods and that considering previous experience, the case study method contains "a greater bias towards falsification of preconceived notions than towards verification" [35]. Finally, Flyvbjerg argues that the difficulty in summarising and generalising case studies is due to the complexity of reality and that in itself is what adds value to this research method.

Many researchers have adapted the case study methodology to the study of technology or software development. For example, due to the fact that analytical methods offer a limited perspective on analysing human interaction with technology, Runeson and Höst attempt to provide a set of recommended practices by adapting the case study methodology from other research domains to the particularities of software engineering [71]. They argue that most research on software engineering aims to investigate how the wide variety of activities that it contains (development, maintenance, etc.), is performed and by which stakeholders. In other disciplines, such research goals are commonly achieved using the case study methodology, which suggests that this method is also suitable for software engineering. Runeson proposes a set of key characteristics that are essential to any type of case study [71]. A case study should be flexible in order to cope with complex real world phenomena—of which software engineering is clearly one. The conclusions of a case study approach should be based on a clear chain of evidence where the data (qualitative or quantitative) is collected from multiple sources in a planned and consistent manner. Furthermore, a case study should contribute to existing knowledge by being based on already existent theory and by building new theory. Runeson also defines the phases of a case study as follows:

1. case study design: objectives are defined and the case study is planned;
2. preparation for data collection: procedures and protocols for data collection are defined;
3. collecting evidence: execution with data collection on the studied case;
4. analysis of collected data; and
5. reporting.

One of the motives for choosing the case study approach in the current research is to gain insights into certain phenomena, such as the how the QA process is integrated into FLOSS projects. The objectives of the case study approach can be exploration, characterisation or even validation, while the subject can be either an intervention such as a new tool or method, or an existing process such as QA [66]. According to Perry, the case study method should be employed only under certain circumstances such as when there are more variables than data points or when the phenomena do not occur in a laboratory (for example a large FLOSS project) [66]. A case study approach, as defined by Perry, should fulfil the following conditions [66]:

- the research questions are set from the beginning;
- data is collected in a planned and consistent manner;
- inferences are made from the data to answer research questions;
- the study explores a phenomenon, or produces an explanation, description, or causal analysis of it; and
- threats to validity are addressed in a systematic way.

Furthermore, Perry outlines a five-part structure of the case study:

- research questions;
- propositions;
- units of analysis;
- logically linking the data to propositions; and
- criteria for interpreting the findings.

Propositions however, may be lacking in exploratory case studies as finding propositions is the objective of such studies. Another important step in a case study is defining the units of analysis, which depend on the research questions. A case study, or the object of the study, can contain multiple units of analysis.

3.1.2 The Case Study Approach in this Research

A widely cited paper in the field of FLOSS research that uses an exploratory case study approach is the study of Apache conducted by Mockus et al. [58]. Mockus et al. investigate the development process within the Apache server community and propose a set of hypotheses. These hypotheses are validated in a second study conducted by the same authors [59]. Mozilla was chosen as the second case study and the initial set of hypotheses was tested in this different context. The hypotheses were then modified based on the findings. The same approach will be used in this research, i.e. an initial pilot case study will be chosen after which a set of hypotheses will be proposed. The validity of these hypotheses will be tested in other FLOSS projects.

This research will be conducted in two phases. The first phase of this research will follow the observational path as a research framework. In other words the "goal is to collect a set of observations and to explain them them in terms of a set of meaningful concepts" [80]. This research goal of collecting a set of observations, or in this case hypotheses, will be achieved by starting with the substantive domain or area of interest and a technique. Applying this framework to this study means that proposing a set of hypotheses will start with the research questions associated with QA practices within FLOSS projects, after which the case study approach will be applied. The second phase of this research will follow the hypothetical path as a research framework which means that the goal is to "test theory rather than build it" [80]. This research goal of testing theory will be achieved by starting with the conceptual domain and then the substantive domain in order to collect the necessary data. Applying this framework to this study means that testing the hypothesis will start with the hypotheses before the actual FLOSS projects are selected in order to retrieve the appropriate data.

3.1.3 Research Questions

The main objective of this research is to define and position QA practices under the FLOSS development model. As a result the following research questions were set in the initial stages of this study:

Q1: Is QA a separate layer in FLOSS communities?

FLOSS communities are usually described as following a concentrical layered model that resembles an "onion". In the FLOSS onion model, communities are formed of four main layers: passive users, active users, developers and core developers; passive and active users are together included in the periphery. Despite the fact that research has shown that this model fails to capture some complexities of FLOSS communities [50] [90], the onion model is still regarded as generally valid. This research aims to investigate how the emerging QA teams fit into this onion model. More specifically, it asks whether QA is established as a separate category or whether members that belong to other layers perform QA related activities. Previous research has shown that participants ascend and descend the organisational hierarchy as well as moving laterally to other tracks but it has also been suggested that many QA tasks are assimilated by other roles [50]. It would be interesting to note, if all projects taken into consideration share the same structure with respect to the QA contributors. In addition, in the case of Firefox, peripheral members (i.e. members of the periphery as defined in the onion model of FLOSS communities) perform some QA tasks, for example posting 20 to 25% of bugs on the issue tracker [29]. I will compare the percentage of peripheral involvement for all the case studies.

Q2: What are the communication patterns between QA members as well as with other project participants?

The goal of analysing communication patterns between QA members is to find the central figures, or in other words members with high activity levels within the community and observe their evolution over time within the project. A related task is to compare central figures based on different data sets, for example central figures based on e-mailing activity and central figures based on issue tracker activity.

3.1.4 Units of Analysis and Research Design

The next step consists of defining the units of analysis within the case studies, or in other words within the objects of study. As the case study is for QA within FLOSS communities, multiple units of analysis must be selected as follows:

- individual project contributors—the study focuses on participation to the project itself;
- team—the study focuses on team interactions; and
- process—the study focuses on QA process as a whole.

The main objective of this research is to explore quality assurance practices employed by FLOSS communities. As case study design strategy is flexible enough to allow incremental data collection and analysis, this research was conducted in two phases as follows:

Phase I: An exploratory case study in order to devise hypotheses describing QA activities within a FLOSS project. The case study is Mozilla. The Mozilla project was launched in 1998 started when the source code for the Netscape Communicator was released under an open source licence—the Mozilla Public License, which was accepted by the open source community, including the OSI (Open Source Initiative). Mozilla was a hybrid project in the sense that a commercial entity employed the bazaar model and many argue that Mozilla is probably one of the most important initiatives in the history of open source as it had a huge impact in promoting the open source credo [68] [33]. The community behind Mozilla is responsible for "Firefox", one of the most popular web browsers. It is a mature community that is well known for stable releases and high quality standards.

Phase II: Four more case studies are chosen to validate the hypotheses generated in Phase I. The case studies are:

- *Ubuntu*—The main motivation for developing Ubuntu was creating an easy to use Linux operating system. It was launched officially in 2004 and quickly thousands of contributors from all over the world joined the community. Today Ubuntu is one of the most widely used Linux distributions;
- *Plone*—Plone is an enterprise content management system written in Python that was first released in 2003. The Plone community is rather small compared to the Mozilla and Ubuntu communities, but Plone has been widely adopted by government agencies, universities, companies and non-profit organizations around the world;
- *KDE*—KDE is a community that produces a variety of FLOSS products among which the most well known is "Plasma Desktop" – a desktop environment. The KDE community was founded in October 1996 and is another example of a successful and mature FLOSS project.
- *LibreOffice*—LibreOffice is a productivity suite developed under the FLOSS paradigm and thus community-driven. The community behind it is powered by the Document Foundation which was established in 2010 after the fork with OpenOffice, another FLOSS product.

The dataset retrieved during Phase I consisted of QA mailing list and issue tracker data associated with the Mozilla community, and will be described in more detail in the next section. After analysing this dataset using basic statistical analysis and social network analysis techniques a set of hypotheses was formulated. In addition, following Phase I the mailing list data gathered was expanded from QA-specific lists to all (or as many as possible) mailing lists related to the projecs. Similarly to Phase I, statistical analysis and social network analysis techniques were applied. However, the purpose in this phase was to validate the set of hypotheses.

Properly addressing potential threats to validity is an important part of any case study, thus such potential threats will be addressed in the last chapter of this thesis in the "Limitations" section on page 189.

3.2 Data

3.2.1 Data Selection and Collection

Open communication channels that are accessible to all community members are essential in all FLOSS projects considering the geographical distribution of their participants. Members communicate asynchronously through the internet by commenting on issue trackers, posting on forums, and sending e-mails. They can also communicate synchronously using Internet Relay Chat. Archives of the discussions on asynchronous media are publicly available and are usually hosted in an online repository.

Data retrieval techniques can vary based on research type and according to Lethbridge et al. these collection techniques can be divided into three levels:

- first degree—direct methods are used when the researchers are in direct contact with the subjects of the study and collect the data in real time;
- second degree—indirect methods are used when researchers directly collect data without actually being in direct contact with the subjects of the study; and
- third degree—researchers perform independent analysis on artefacts that already exist, e.g. in archives [56].

Messages are exchanged between community members independent of this research and are already available in archives hosted in online repositories. Thus the data collection techniques used in this research are of the third degree [56]. Data triangulation, which means approaching an issue from different standpoints, is particularly common when analysing data gathered using qualitative methods (case studies that are based on qualitative data). However, using triangulation can also compensate for errors in measurement or modelling when using quantitative methods [71]. As a result, data triangulation was used in this study by gathering data from more than one data source, i.e. mailing list data, issue tracker data and code repository metadata. Because all the data gathered for this research was downloaded from public FLOSS repositories there are no particular ethical issues to be addressed. However, posters' names have been anonymised in this thesis.

The data was downloaded and stored in local PostgreSQL or MySQL databases using either Python scripts that would perform "screen scraping" (web crawlers) or open source tools[1]. All the used scripts were made public and can be downloaded from: `https://github.com/adinabarham/crawlers` .In addition, www.ohloh.net[2] was used for retrieving code contributor associated metadata in order to perform data cleaning and establish contributor roles within the community. The tools used for retrieving each dataset are shown in Table 3.1 on page 55.

Table 3.1: Tools used for data retrieval

Case study	E-mails	Issue tracker	Contributors
Mozilla	manual	Python scripts	—
Ubuntu	Python scripts	Python scripts	Python scripts
LibreOffice	MailingListStats	Python scripts	Python scripts
Plone	MailingListStats Python scripts	Python scripts	Python scripts
KDE	MailingListStats	Python scripts	Python scripts

Table 3.2 on page 56 describes the dataset associated with each study. For the exploratory case study (Mozilla community) only QA related mailing lists and issue tracker data was downloaded while for the other case studies most mailing lists, issue tracker and contributor related data were downloaded. Most FLOSS projects have a large number of associated mailing lists. Depending on the size of the project these mailing lists are specialised in the sense that some can be dedicated for the developers of a certain component or for teams that handle certain tasks such as localisation or QA related activities. Some mailing lists are used by members speaking a certain language

[1]MailingListStats: https://github.com/MetricsGrimoire/MailingListStats
[2]Ohloh: https://www.ohloh.net/

or from a geographical area. For a more comprehensive description of which mailing lists were retrieved and the numbers of e-mails obtained, see Tables B.3, B.4, B.5 and B.6 in Appendix B starting on page 212.

Table 3.2: Communication channel data retrieved for each community

Project	QA e-mails	General e-mails	Bugs	Code contributors
Mozilla	✓	X	✓	X
Ubuntu	✓	✓	✓	✓
LibreOffice	✓	✓	✓	✓
Plone	✓	✓	✓	✓
KDE	✓	✓	✓	✓

Each FLOSS project used as a case study for this research started its activities and implemented QA at different times. Furthermore, the large amount of data associated with each project meant that data retrieval for the projects took place at different times. As a result, the data gathered is associated with various time periods, which are shown in Table 3.3 on page 56.

Table 3.3: Periods covered by datasets

Project	QA e-mails	General e-mails	Bugs
Mozilla	02/2006–07/2011	—	01/1998–02/2012
Ubuntu	08/2005–05/2013	09/2004–05/2013	05/2000–04/2013
LibreOffice	07/2011–05/2013	09/2010–05/2013	08/2010–06/2013
Plone	06/2011–07/2013	07/2002–07/2013	05/2002–12/2012
KDE	04/2012–04/2013	03/1999–05/2013	01/1999–06/2013

As the FLOSS projects studied vary in size and other characteristics, so do the sizes of the datasets associated with each project and with each communication channel. The total numbers of QA e-mails, e-mails bugs and bug comments for each project are given in Table 3.4 on page 57.

The initial design contained only PostgreSQL databases but the MailingListStats tool required MySQL, so mailing list data associated with the KDE, Plone and LibreOffice was initially stored in MySQL databases and

Table 3.4: Numbers of artefacts retrieved for each project

Project	QA e-mails	Total e-mails	Bugs	Bug comments
Mozilla	3,689	3,689	687,221	5834,507
Ubuntu	8,798	487,694	993,420	4,114,392
LibreOffice	3,381	192,709	19,432	127,208
Plone	170	176,144	13,026	55,883
KDE	954	529,488	312,847	1,364,871

then migrated to PostgreSQL using a migration script written in Python. A detailed description of each database is given in Appendix B on page 209.

3.2.2 Data Cleaning

After storing the Mozilla mailing list data on a local machine, it was cleaned manually by completing the following actions:

- spam was marked as such and removed;
- double posts were removed; and
- single usernames was assigned to participants posting under multiple usernames.

In view of the large amount of data associated with the other four projects, data cleaning was performed using Python scripts in the following steps:

- a list of code contributor names and usernames used to commit code was retrieved from the appropriate DB table;
- contributors were retrieved from both mailing list and bug tables and processed as follows:
  - if these contributor names contain "@" mark then the characters to the left of the sign are considered the same;
  - if the name contains "." then the dot is replaced with a space;
  - if the name contains the characters "a.k.a." then two names are attributed to the same contributor (before the "a.k.a." characters and after); and

  - if the name contains brackets then two names are attributed to the same contributor (outside and inside the brackets);

- the contributor names are then compared with the names and nicknames of code contributors; and

- names are then modified in the bugs and mailing list tables.

Unlike in the Mozilla pilot case study, a manual check of emails to remove spam was not conducted for the other four cases due to the large amount of data. Spam was unlikely to affect the results of the social network analysis because it is generally ignored by other mailing list participants and therefore does not create connections that appear in the network. However the activity level analysis did include spam e-mails. For example, the discrepancies between the activity levels and the social network analysis of the Plone community suggested a large amount of spam e-mails on the Plone-QA mailing list; for more details see page 119.

3.3 Social Network Analysis

3.3.1 Social network analysis and its applications in social science

A parlor game in early 18th century Könisberg (now Kaliningrad) was to try to find a way to walk through the town without crossing any of its seven bridges twice. In 1736, the famous mathematician Euler demonstrated the impossibility of such a walk [7]. Euler's demonstration marked the beginnings of graph theory, an important branch of mathematics that is the basis of social network analysis—the analysis of relationships and their implications between people, groups, organisations and other types of entities [40] [84] [88]. However, despite the fact that graph theory has long been an established branch of mathematics, networks (which are represented as graphs) were viewed as objects of "pure structure whose properties are fixed in time" [85]. Watts argued that this view is false because these networks can represent populations of individual components that are in a continuous state of change. Watts proceeds to argue that graph theory requires a more interdisciplinary approach, because mathematicians do not consider individual behaviour or cultural norms while social scientists lack the mathematical background to understand and apply graph theory.

In the years since Watts made that argument, interdisciplinary research has grown, and along with it interest in social network analysis and its applications in fields as varied as anthropology, organisational behaviour, chemistry and even epidemiology. Social network analysts have developed two frameworks for approaching the problem of complex networks. The first focuses on the relationship between network structure while the second focuses on the information flow within a network. On other words, social network analysis allows the analysis and observation of links between a network's actors as well as the social structure or the variety of roles and social groups within the network. Furthermore, social network analysis allows the study of information diffusion, identification of potential information brokers through whom much information is likely to flow (and whose departure would jeopardize the flow of information), and of individuals who are in a position to influence other network members.

When "stripped to its bare bones, a network is nothing more than a collection of objects connected to each other in some fashion" [85]. A network can be represented by a graph. A directed graph shows the direction of communications between community members, in the sense that if a community member "A" sends a message to community member "B", an arc will exist from vertex A to vertex B while an arc will not exist from vertex B to vertex A. The number of interactions between members can be represented as the weight (value) of the graph's arcs. For example, Figure 3.1 on page 60 describes interactions between four community participants. There have been three interactions between Ann and Pete, one interaction between Sara and Pete and two interactions between Pete and John. On the other hand, Mike is an isolate vertex as he did not interact with anyone in this particular community.

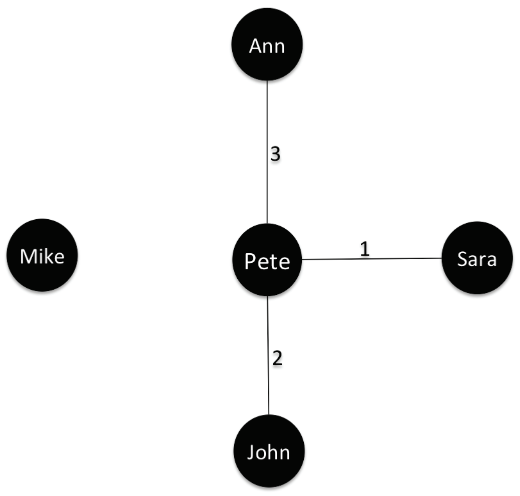
3.3.2 Basic SNA concepts

Here let us introduce some basic SNA concepts that will be used in this thesis. [3]

- vertex/node—an actor in the social network;
- edge/arc—a connection between two vertices/nodes. Edges are undirected lines while arcs are directed lines;
- graph—a set of vertices and a set of lines between pairs;
- oriented graph—containing at least one arc;

[3]For more details of these concepts, see for example [63]

Figure 3.1: Community graph example



- network—a graph and additional information associated with the lines and vertices it contains;
- vertex degree—the number of edges adjacent with the vertex;
- outer degree/outdegree—the number of arcs starting from the vertex to other vertexes in the graph (oriented graphs only);
- inner degree/indegree—the number of arcs pointing to the vertex from other vertexes in the graph (oriented graphs only);
- loop—an edge/arc that connects a vertex to itself;
- density—the number of lines in a simple network expressed as a proportion of the maximum possible number of lines;
- complete network—a network with maximum density (density is 1);
- semiwalk from vertex u to v—a sequence of lines such that the end of vertex of one line is the starting vertex for the next line and the sequence starts at vertex u and ends at vertex v;
- walk from vertex u to v—a semiwalk with the additional condition that none of its lines are an arc of which the end vertex is the arc's tail;
- semipath—a semiwalk in which no vertex in between the first and last vertex of the semiwalk occurs more than once;

- path—a walk in which no vertex in between the first and last vertex of the walk occurs more than once;
- (weakly) connected network—a network where each pair of vertices is connected by a semipath;
- strongly connected network—a network where each pair of vertices is connected by a path;
- (weak) component—a maximal (weakly) connected subnetwork;
- strong component—a maximal strongly connected subnetwork;
- geodesic—the shortest path between two vertices;
- betweenness centrality of a vertex is the proportion of all geodesics between pairs of other vertices that include this vertex;
- betweenness centralisation is the variation in the betweenness centrality of vertices divided by the maximum variation in betweenness centrality scores possible in a network of the same size; and
- k-core—a maximal subnetwork in which each vertex has at least degree k within the subnetwork.

3.3.3 Social network analysis and FLOSS

FLOSS researchers have made considerable use of social network analysis, for two main reasons. First, the fact that in FLOSS development, people collaborate in communities to plan and perform the construction of specialized information i.e. computer code, means that researchers will be interested in network structures in information flows, which are the two main focuses of social network analysis. Second, the fact that most of the information flows in FLOSS communities are public and archived allows researchers to reconstruct networks relatively easily. For example, Crowston and Howison analyse the social structure of FLOSS projects by examining 120 projects hosted on the SourceForge.net repository [24]. Another example is the analysis of social positions as well as the dynamics of social distribution over time conducted by Christley and Madey [19].

Previous research focusing on FLOSS communities has shown that interactions between community members can be defined using two methods: thread-based analysis and quotation-based analysis. Thread-based analysis assumes that relations are created between members who are exchanging messages under the same thread [90]. Quotation-based analysis assumes that relations are created between two members if one has quoted another member's message in his own message [9].

Barcellini et al. propose the use of quotation-based analysis when investigating thematic coherence [9]. They define FLOSS design as "a particular case of asynchronous, distributed, collaborative design". The FLOSS design process usually takes place in a public communication channel and is archived in an online repository (documentation space). New project contributors to a FLOSS project are commonly encouraged to go through these archives in order to gain knowledge regarding the project, its evolution, and what has already been tried and accomplished. Barcellini et al. hypothesise that quotation-based analysis is more appropriate than thread-based analysis when reconstructing thematic coherence in design-oriented discussions. They also argue that thread-based analysis is more appropriate for analysis of the community structure, member roles or measuring member centrality. They compared the two methods by applying them to the same FLOSS project data. They made the interesting observation that there is a real need for tools and methods to extract relevant data from the design discussions considering the large amount of data associated with each project. Their findings supported the proposed hypotheses, meaning that quotation-based analysis is a promising approach for studies focusing on thematic coherence-related topics and design-relevant information. The main reason is that while using thread-based analysis, some theme-related messages were incorrectly divided into different threads. In addition, some messages were categorised as peripheral contributions while they were in fact central contributions. Quotation-based analysis did not seem to display the same erroneous behaviour. The study also revealed that the links within the project's social structure influenced the shape of the discussion space. The roles of participants influenced whom the participant replied to, thus influencing "the unfolding of the design process within the discussion space". In other words, the structure of this particular community resembled a transitional organisation hierarchy opposing the idealistic views of FLOSS organisation.

Wiggins et al. advanced the application of SNA techniques to FLOSS research by investigating the social dynamics of FLOSS teams and raising concerns regarding the techniques used by previous exponents of SNA [90].

FLOSS practitioners have often claimed that FLOSS communities display a decentralised communication pattern, making centralisation metrics an important issue in community research. Wiggins et al. argued that in prior research social network analysis techniques were commonly applied to an aggregated representation of the network which may have provided a skewed and simplified view of FLOSS communication patterns and dynamics. Instead they proposed a dynamic approach to assess centralisation in FLOSS communities. Another issue that is commonly present in prior FLOSS community research that employs SNA techniques is the actual construction method of the social network. These methods utilise replies on public discussion threads as proxies for direct communication between participants. However, some reply messages may include not only the previous poster but also all the community members or all previous members active on the thread as a recipient. This is not always the case but it nonetheless represents a potential issue in assessing centrality. As a result, "the broadcast of these communication channels restricts the choice of SNA measures that are meaningful as the potentially public nature of the messages clearly violates the assumptions of information brokerage such as betweenness centralisation among others" [90].

To address this issue, Wiggins et al. proposed the use of the out degree centralisation for measuring inequalities in communication contributions to the projects. Higher values indicate that a small number of members reply to more participants while lower values indicate a more equal contribution on communication channels. Wiggins et al. also argued that collapsing events over a long period of time is another common problem in prior research. To address this issue, they propose using message timestamps to build a set of snapshots of the community structure at different times. Furthermore, prior research has usually focused on communication carried out on one communication channel and thus does not depict the community. Wiggins et al. argue that differences should be expected between discussions carried out on mailing lists versus discussions carried out on issue trackers or between developer-oriented communication channels versus user-oriented ones. After applying SNA techniques to two FLOSS communities, they found that the user-oriented communication venues displayed a more decentralised pattern than developer-oriented communication venues. Therefore, communication patterns may vary depending on communication channels and researchers should consider this aspect when characterising a community. However, both projects tended to display a decentralisation pattern over time with occasional centralisation peaks. Another interesting finding was a spike in activity on the issue tracker which was linked to periodic management activities performed by a few users who closed a large number of bugs.

Another study that investigates validity issues in the appliation of SNA techniques to data associated with FLOSS projects was conducted by Howison et al. [49]. They argued that social network analysis is "not a theory per se but more a set of analysis techniques" and propose a series of practices in order to improve the existing research framework. FLOSS associated data is considered digital trace data which can be defined as "records of activity (trace data) which undertake through an online information (thus, digital)" [49] that is both produced and stored by an information system. All trace data can be characterised as follows:

1. it is not produced for the sake of research as it already exists as a byproduct of various activities;

2. it is event-based data rather than summary data; and

3. it is longitudinal data as events occur over a period of time.

Furthermore, Howison et al. clarify the distinction between archival data and trace data. Archival data is stored in archives that may contain both trace data and data "that represents participants' summaries of their social relationships" [49]. Thus all digital trace data is archival data but not vice versa. Howison et al. suggest that researchers should triangulate with multiple measures of links between community members and assess consistency. In addition, researchers should consider whether the order of events affects the results of the study when aggregating links that occur at different times. Furthermore, researchers should specify the SNA tools or software used, the algorithms and even perform tool triangulation and compare the results. With regards to the datasets, researchers should consider if it is a data sample or a census of activity and, if the former is the case, explain the sample logic. Finally, they argue that one should explicitly define nodes, links and network measures based on the network processes.

3.3.4 Applying SNA in this Research

For the purposes of this research communities associated with each case study will be represented as directed graphs. Project participants will be represented as nodes (vertices) while interactions will be represented as edges (arcs). The interactions are messages exchanged by community members on mailing lists and issue trackers. Thread-based analysis is used because thematic coherence does not represent a focus in this research. Furthermore, the number of messages exchanged between two members will determine the weight of the arc connecting these two members.

The dataset associated with each FLOSS community will be exported into a Pajek-readable format. Pajek[4] is a free social network analysis application that allows both mathematical analysis and visualisation of community graphs. Mathematical analysis of the community graphs will produce metrics that describe structure and dynamics. The purpose of visualising

[4]Pajek homepage: http://pajek.imfm.si/doku.php

these community graphs is not to show details of communications between individual community members but rather to show general communication trends in the community. Visualizations of all connections in a community will show so many lines that most individual arcs will be indistinguishable from one another, but the purpose of showing these figures is to convey that there is a large amount of communication between members of the various layers.

Social netwrok analysis will applied to each dataset in multiple steps. First, social network analysis techniques will be applied to the whole dataset in order to compute general metrics such as degree values, betweenness centrality for the whole community. Each dataset will then be split into layers of contributors and then layers associated with QA will be analysed separately by computing general SNA metrics. The third step will consist of analysing the groups of contributors active on the QA mailing lists and computing various metrics. The analysis conducted in these first stages will be conducted on an aggregated forms of the graphs. Finally, due to the dynamic nature of communities and to the QA focus of this research, the graphs consisting of community members active on the QA mailing lists will be analysed using six month time frames. These time-frames will be used to track the evolution of QA teams, and SNA metrics will be computed for each window.

3.4 Summary

The main objective of this research is to define the impact of the adoption of formal QA practices on the structure of FLOSS communities. For this purpose two main research questions have been formulated. These questions focus on the network structure of QA activities in FLOSS, and on information flows regarding QA within FLOSS communities. The study will be conducted in two phases: first, an exploratory case study will be made and a set of hypotheses proposed based on the findings; second, these hypotheses will be tested and refined in four more case studies of FLOSS projects. The data will be from mailing lists, bug trackers and will be further triangulated using metadata from code repositories. The datasets will be cleaned and statistical and SNA techniques applied both on aggregrate and time-series data.

Chapter 4

Preliminary Research and Pilot Case Study

4.1 Working Definition of Quality Assurance

"The problem of quality management is not what people don't know about it. The problem is what they think they know" [22]. In other words, quality assurance is often misunderstood and is actually a process that requires a complex set of methodologies and procedures [67].

In view of the complexities of quality assurance processes, a clear definition of what QA entails under the FLOSS model is necessary for the scope of this research. This thesis uses the following working definition of QA activities within FLOSS projects:

Testing, contributing code to automated testing tools or any test related activity, triaging bugs or any activity performed on the projects issue tracker, participating on the QA dedicated communication channels.

The main purpose of this definition is to help identify QA contributors within FLOSS communities by describing current observed QA practices employed in the open development model. However, the FLOSS model itself is still evolving and so is the role of QA within this model. Therefore it is to be expected that as FLOSS methodologies mature and become more complex, so will the QA procedures employed [30]. One potential problem for this research is that there could be QA contributors performing QA activities that are not observable by monitoring public communication channels. This problem is addressed by comparing observed QA team activities in the traditional development model [76] and previous FLOSS literature [93] [57].

However, future research should include further triangulation by conducting interview or questionnaire surveys of key project members.

4.2 Preliminary Study of QA Adoption in FLOSS projects

Before the main research, a preliminary study was carried out to assess the presence of QA procedures within popular FLOSS projects. The top 50 FLOSS software products ranked by number of downloads were accessed on Ohloh, which is a directory of FLOSS and the contributors who create and maintain it[1]. It was found that almost one third had QA procedures. QA was considered to be present if there were clear references to its existence (for example, mailing lists, IRC channels, webpages, wikis and so on) and they were easily identifiable. The results of this step were published previously [10]. In the next phase of this preliminary assessment, the top 100 software projects ranked by number of users were analysed; these results were also published previously [11]. The conclusions were consistent with the findings of the previous phase: more than a quarter (27 projects) had some form of QA present in the development process. In addition it was noted that almost half of these projects had dedicated communication channels for the QA teams. The remaining projects either had channels for contributors to post test results or other resources dedicated to describing QA within the project such as wikis or websites. These projects are listed with general statistics regarding contributor numbers and lines of code in Table A.1 of Appendix A on page 197. Furthermore, a detailed description of which projects had dedicated QA communication channels and QA teams is provided in Table A.2 of Appendix A on page 197.

4.3 Pilot Case Study: Mozilla

4.3.1 The Mozilla project

The Mozilla project was launched in 1998 when the source code for the Netscape Communicator was released under an open source licence—the Mozilla Public License, which was accepted by the open source community, including the OSI (Open Source Initiative). The "Mozilla Organization" was founded to ensure the transition from proprietary development to the open development model and manage the project. Mozilla was a hybrid project

[1]Ohloh: https://www.ohloh.net/

in the sense that a commercial entity employed the bazaar model and even if it was not the first of its kind, as Mitchell Baker puts it,"the combination of open source techniques with an active, focused commercial management structure was uncharted territory" [30]. Feller and Fitzgerald argue that Mozilla is one of the most important initiatives in the history of open source as it had a huge impact in promoting the open source paradigm [33]. As a pioneer FLOSS project Mozilla has also attracted the attention of researchers [59].

The importance of QA within the Mozilla project was recognised as an important issue from the early days and as a result Christine Beagle was hired to coordinate the QA effort. The result, according to Baker, was an active and effective quality assurance community [30]. In 2003 the "Mozilla Foundation" was established as an independent non-profit organisation in order to manage the Mozilla project. Soon after, a transition began from the Mozilla Application Suite to new products such as Firefox and Thunderbird.

The pioneering nature of Mozilla as a hybrid project, the wealth of previous research on it, the size and maturity of its community, the widespread adoption of its products, and its early adoption of formal QA practices: all these factors combine to make Mozilla a suitable subject for our pilot case study.

4.3.2 General analysis of the Mozilla dataset

The Mozilla community was chosen as a pilot case study in order to generate a set of hypotheses and create a framework for further research. Mailing list data was downloaded in July 2011. At that time, according to the Mozilla Quality Assurance (QMO) website[2], there were five sub-teams: Web QA, Desktop Firefox, Browser Technologies, Automation and Services. The Web QA, Desktop Firefox, Browser Technologies and Services teams used the mozilla.dev-quality mailing list while the Automation team used the Mozmill developer mailing list.

Bugzilla data were acquired in February–March 2012 using a web crawler that accessed pages associated with each bug (the actual bug page and the change history page associated with each bug) and stored the issues directly into a PostgreSQL database. The next phase consisted of cleaning both datasets by removing double posts, removing spam and normalising name formats. As expected, the QA mailing list was created much later than the issue tracker. For the purpose of analysing activity levels issue tracker and mailing list data should be retrieved from the same periods of time,

[2]Mozilla quality assurance: https://quality.mozilla.org/

however by doing so important data might be lost. For example, if activity pertaining to a certain period of time is not be taken into account then there is a risk that we might miss migration of certain members between layers of the community.

The Mozilla.dev-quality mailing list data contains 2,535 e-mails exchanged between February 2006 and June 2011, while the Mozmill developer mailing list data contains 1,155 e-mails exchanged between October 2008 and July 2011. The traffic and number of users is higher on the Mozilla.dev-quality mailing list than the Mozmill developer mailing list—see Table 4.1 on page 70. This is to be expected as the Mozmill developer list is addressed to more technically aware users, as Mozmill is an automated testing tool produced by the Mozilla community primarily to test their own products.

Table 4.1: QA mailing list activity in the Mozilla community

	Mozilla.dev–quality	Mozmill developer	Total
Topic	1,042	313	1,299
Messages	2,535	1,155	3,689
Thread initiators	199	47	233
Distinct authors	293	61	332

The issue tracker (Bugzilla) dataset covers all Mozilla products from 1998 to 2012 and contains 687,221 bugs with 5,834,507 associated comments. Bug IDs range from 0 to 724,339; the collected bugs thus represent 94.87% of the id range. The remaining 5.13% were not collected because they were not publicly available or could not be parsed due to bad HTML.

Approximately 4,400 distinct project members were identified as assigned to fix bugs. Without getting the data associated with code commits it is not safe to assume that these members were also the members who posted the bug fix. However, these individuals assume the role of code committers or are viewed by other community members as code committers. These users are also active when it comes to posting bug comments as well as sending e-mails on the QA mailing lists. After cross-referencing members active on the mailing lists and code committers, 883 bugs were found, most pertaining to Firefox.

Of all the e-mails exchanged 152 (approximately 4%) were sent by authors who had sent only one e-mail throughout the period of the study. On the other hand 135,466 bugs (approximately 20%) were posted by members who had posted only one bug while 61,196 (approximately 1%) comments were posted by members who had posted only one comment. In other words occasional contributors, or peripheral members are more active in posting bugs than engaging in conversation. Table 4.2 on page 71 describes activity on various channels on a yearly basis. Most activity has steadily increased over time; however activity on the QA mailing lists has irregular spikes but does not seem to show an overall upward or downward trend. It would be useful to find out if the activity peak in 2009 is linked to interior events or external factors, but that is beyond the scope of this study.

Considering these findings the following two hypotheses are proposed:

Hypothesis 1: A smaller percentage of peripheral members are engaged in conversation than in posting bugs.

Hypothesis 2a: Activity levels on the QA mailing lists are independent of activity levels on other communication channels.

Hypothesis 2b: Traffic on the QA mailing lists does not show a consistent upward trend.

Table 4.2: Activity on a yearly basis in the Mozilla community

	2006	2007	2008	2009	2010	2011
Comments	328,846	335,323	467,087	528,199	658,030	703,857
Bugs	42,015	41,995	56,785	60,880	78,089	78,896
QA e-mails	343	361	556	1,307	739	384
Dev bugs	119,571	123,234	174,742	177,776	227,123	226,555
Dev comments	258,458	271,679	375,729	449,539	541,707	561,853
Dev e-mails	196	286	343	953	500	264

Developers were considered to be those to whom bugs were assigned on the tracker. Developers' activities comprised comments posted on the issue tracker as well as e-mails sent to the QA mailing lists. On the bug tracker only about 4% of participants posted more than the average of 39 comments, while approximately 9% posted more than six bugs. The average number of e-mails sent is 11.28 (sd = 37.23) and only about 17% of participants sent more than this average. Active members were considered to be those who sent eleven or more e-mails, i.e. more than the average. Based on these findings we generated two more hypotheses:

Hypothesis 3a: A small group of people is highly active on the QA mailing lists. These groups tend to represent less that 20% of the mailing list participants.

Hypothesis 3b: A large percentage of QA mailing list participants send e-mails only very occasionally.

4.3.3 Social network analysis of the whole dataset

In order to apply SNA techniques the dataset was then exported from the database into a Pajek-readable file containing community members, links between them and the periods in which they were active in the project. However, members who did not form any connections with other community members were eliminated from the dataset as it was most likely that the messages sent by these members were spam, general information messages or automated messages (i.e., messages sent automatically when a bug status is changed). After importing the data into Pajek all the loops were eliminated due to the fact that a message sent by a community member as a reply to one of his/her previous messages was usually either sent by mistake or was to add additional information and therefore did not contribute to our understanding of community structure or dynamics.

The resulting network contained 149,032 vertices and 1,132,413 arcs, of which 353,100 arcs had a value greater than 1 and 779,313 arcs had a value equal to 1; in other words, most of the connections were single, unrepeated acts of communication from one participant to another. We should note that it would be possible for there to be an arc of value 1 from member A to member B, and an arc of a higher value e.g. 5 from member B to member A, but that is unlikely to occur.

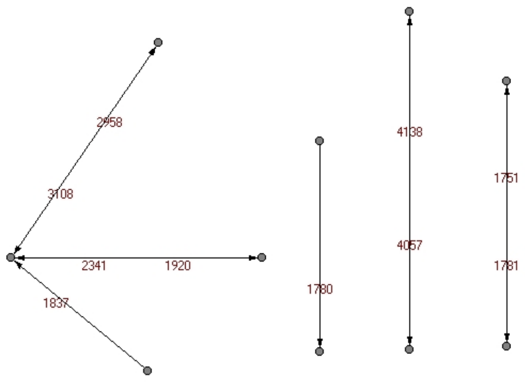
Hypothesis 4: Almost two thirds of connections within a project's graph are created by single acts of communication from one participant to another.

The network contained 786 components, of which the biggest component contains 148,180 vertices or approximately 99% of the community. The rest of the components contained between one and three vertices. One of the reasons for these small components could be that some messages are actually spam messages. Another reason would be that the messages received replies after the dataset was retrieved or that the messages were posted by members not active previously as they might be peripheral or new members who did not yet have the chance to create relations with more active users.

Hypothesis 5: The community consists of a large group of people that spans both issue tracker and mailing lists.

The average degree[3] of this directed network was 15.196, which means that the community members interacted on average with approximately 15 other members. The network's density was 0.00005099; this number expresses the ratio between all possible connections and the actual connections existing in the network. We will use this number later in the thesis to compare QA-related communications regarding Mozilla with those of other FLOSS projects. The heaviest ten arcs connected ten people and had values between 1,751 and 4,138; each of these represents one person sending many hundreds of communications to another person; see Figure 4.1 on page 73.

Figure 4.1: The heaviest ten arcs in the Mozilla graph connecting ten members—numbers represent e-mails and bug comments in each direction.



In order to calculate more exact degree values, the directed network was changed into an undirected graph by transforming all arcs into edges. The weights of the edges were the combined weights of the arcs between the relevant vertices. The resulting network had 505,132 edges with a value equal to 1, and 338,636 edges with values greater than 1. The network's density was 0.00007598. Compared to the directed graph, the average degree dropped to

[3]The average degree is calculated using the formula $2 * |E|/|V|$, where $|E|$ is the total number of arcs and $|V|$ is the total number of vertices in the network.

11.32. About 9% of individuals had connections with more than eleven (i.e. the average degree) other individuals. Looking at the vertices with degree value from 0 to 10, we found that they accounted for approximately 90% of the community. The remaining approximately 10% of vertices with the highest degrees were connected to—that is, communicated directly with—between 11 and 16,947 other members. Details of the approximately 90% of vertices in the Mozilla graph with degree value between 0 and 10 are given in Table 4.3 on page 74. In the table, Frequency is the number of vertices with that value, Frequency % is the percentage of vertices with that value from the whole community, CumFreq is cumulative frequency, i.e. the sum of the number of the vertices with that value added to the numbers of vertices with lower degree values and CumFreq % is the cumulative frequency expressed as a percentage of the whole community graph. The largest degree cluster is formed by vertices with a degree value of 1 and represents more than one third of the community.

Table 4.3: Vertices in the symmetrized Mozilla graph clustered by degree value

Degree value	Frequency	Frequency %	CumFreq	CumFreq%
0	724	0.4858	724	0.4858
1	5,9008	39.5942	59,732	40.0800
2	34,726	23.3010	9,4458	63.3810
3	1,4074	9.4436	108,532	72.8246
4	8,883	5.9605	117,415	78.7851
5	5,233	3.5113	12,2648	82.2964
6	3,859	2.5894	126,507	84.8858
7	2,799	1.8781	129,306	86.7639
8	2,058	1.3809	131,364	88.1448
9	1,555	1.0434	132,919	89.1882
10	1,381	0.9266	134,300	90.1149

Hypothesis 6a: Less than one tenth of the community members are connected to a higher than average number of other individuals.

Hypothesis 6b: More than a third of community participants have connections with only one other participant.

In any online community it is possible that some members may be sending more messages than they receive and some members receiving more than they send. In social network analysis this is measured by indegree and outdegree values that can be computed in a directed network. Returning to our directed Mozilla graph, and once again looking at the vertices with indegree values between 0 and 10, we found that they make up about 92% of the community—see Table 4.4 on page 75 for details. The remaining vertices had indegree values of between 11 and 8,853.

Table 4.4: Vertices in the directed Mozilla graph clustered by indegree value

Indegree value	Frequency	Frequency %	CumFreq	CumFreq%
0	2,582	1.7325	2582	1.7325
1	76,983	51.6553	79,565	53.3879
2	26,831	18.0035	106,396	71.3914
3	11,853	7.9533	118,249	79.3447
4	6,588	4.4205	124,837	83.7652
5	4,194	2.8142	129,031	86.5794
6	2,780	1.8654	131,811	88.4448
7	2,083	1.3977	133,894	89.8424
8	1,540	1.0333	135,434	90.8758
9	1,268	0.8508	136,702	91.7266
10	1,076	0.7220	137,778	92.4486

Hypothesis 7: More than half of community participants are connected to only one other community member from whom they received messages.

In the same directed Mozilla graph approximately 6% of vertices had outdegree values between 11 and 15,390; see Table 4.4 on page 75.

Hypothesis 8: Almost one third of community participants are connected to only one other community member to whom they had sent one or more messages.

The directed Mozilla graph contains 57,781 strong components[4]. The largest strong component contains 91,155 vertices which amounts to approximately 61% of the whole network.

[4]A strong component is a maximal strongly connected subnetwork. For details of this and other social network analysis concepts such as k-core, see page 59.

Table 4.5: Vertices in the symmetrized Mozilla graph clustered by outdegree value

Outdegree value	Frequency	Frequency %	CumFreq	CumFreq%
0	54,886	36.8283	54,886	36.8283
1	47,391	31.7992	102,277	68.6275
2	16,851	11.3070	119,128	79.9345
3	7,793	5.2291	126,921	85.1636
4	4,383	2.9410	131,304	88.1046
5	2,881	1.9331	134,185	90.0377
6	1,927	1.2930	136,112	91.3307
7	1,404	0.9421	137,516	92.2728
8	1,156	0.7757	138,672	93.0485
9	897	0.6019	139,569	93.6504
10	720	0.4831	140,289	94.1335

Hypothesis 9: More than half of community participants form a strongly connected subnetwork.

The symmetrized i.e. undirected network contains 786 components where the largest component contains 148,180 vertices or approximately 99% of the whole community. In the same graph, k-core values range between 0 and 166, in other words the most connected group of members contains individuals who interacted with a minimum of 166 other people. This group contains 450 community members or about 0.3% of the whole community. Table 4.6 on page 77 gives details of the members who have k-core values between 0 and 10, who make up approximately 91% of the graph.

Betweenness centrality within a social network represents the importance of a certain vertex in the information flow. In other words, if we consider geodesics as channels for transmitting important pieces of information then, an individual that is situated on a higher number of geodesics is more central according to the betweenness concept [63]. The Mozilla directed graph vertices have betweenness centralities of between 0 and 0.0626 where the network's betweenness centralisation value is equal to 0.0625. Table 4.7 on page 77 details the distribution of betweenness centrality vectors within the network.

In other words, these findings suggest that more than 60% of Mozilla community members do not represent a "step" in the information flow (i.e. have a centrality betweenness value equal to 0). The maximum possible value

Table 4.6: Vertices in the symmetrized Mozilla graph clustered by k-core value

K-core value	Frequency	Frequency %	CumFreq	CumFreq%
0	724	0.4858	724	0.4858
1	59,826	40.1431	60,550	40.6289
2	35,562	23.8620	96,112	64.4908
3	14,511	9.7368	110,623	74.2277
4	8,624	5.7867	119,247	80.0144
5	5,113	3.4308	124,360	83.4452
6	3,756	2.5203	128,116	85.9654
7	2,730	1.8318	130,846	87.7973
8	1,863	1.2501	132,709	89.0473
9	1,590	1.0669	134,299	90.1142
10	1,320	0.8857	135,619	90.9999

Table 4.7: Betweenness centrality clusters in the Mozilla community

Value intervals	Frequency	Frequency %	CumFreq	CumFreq%
0.000	91,688	61.5224	91,688	61.5224
0.000–0.021	57,338	38.4736	149,026	99.9960
0.021–0.042	4	0.0027	149,030	99.9987
0.042–0.063	2	0.0013	149,032	100.0000

of the betweenness centrality is 1 while the minimum is 0, so the highest value in the Mozilla community i.e. 0.063 is towards the low end of the spectrum of possible values. Less than 1% of vertices have a higher than average betweenness score, but they do not pose a high risk because of their relatively low values.

Hypothesis 10: Less than 1% of vertices have a higher than average betweenness centrality score.

To get a better understanding of communication patterns between QA team members and the rest of the community the graph was divided into four clusters as follows:

- Cluster 1—Members of the community who were assigned as bug fixers on the issue tracker and were active on the QA mailing lists.
- Cluster 2—Members of the community who were assigned as bug fixers on the issue tracker and were not active on the QA mailing lists.
- Cluster 3—Members of the community who were not assigned as bug fixers on the issue tracker and were active on the QA mailing lists.
- Cluster 4—Members of the community who were not assigned as bug fixers on the issue tracker and were not active on the QA mailing lists. In addition, members who could not be categorised were added to this cluster.

Each bug posted on the issue tracker has an associated field: assigned_to. The members whose names are displayed in this field are supposed to provide bug fixes. It is not safe to assume that these individuals did in fact provide the bug fixes, but they are regarded as code committers by the rest of the community. Table 4.8 on page 79 provides more details regarding cluster distribution within the community.

If each of these clusters is shrunk to one vertex then the resulting network has four vertices and is a complete network with a density value of 1. The graph contains a total of four loops and six lines and is depicted in Figure 4.2 on page 80. The values graph's lines are described in Table 4.9 on page 79.

The next step was to reduce all clusters to one vertex except for cluster 3. The resulting network contained 122 vertices of which three represented the other clusters. The graph contained a total of 377 arcs of which 219 had a value greater than 1 and 158 arcs had a value equal to 1. The network's density was 0.0253 while the average degree was 6.1803. The network is depicted

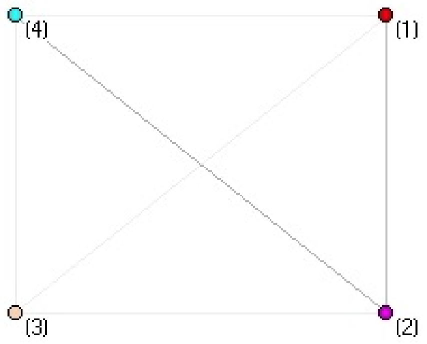
Table 4.8: Clusters in the Mozilla community

Cluster	Frequency	Frequency %	CumFreq	CumFreq%
1	101	0.0678	101	0.0678
2	4,017	2.6954	4,118	2.7632
3	119	0.0798	4,237	2.8430
4	14,4795	97.1570	149,032	100.0000

Table 4.9: Shrunk Mozilla network communication

Rank	Line	Value
1	4.4	1,575,648
2	4.3	519,218
3	3.4	372,553
4	1.4	339,663
5	4.1	325,703
6	3.3	254,976
7	1.3	178,328
8	3.1	107,031
9	1.1	99,245
10	4.2	1,212
11	2.3	1,142
12	1.2	1,036
13	2.4	982
14	2.1	876
15	3.2	772
16	2.2	147

Figure 4.2: Mozilla reduced graph—each cluster was reduced to one vertex in order to illustrate connections between various groups within the community

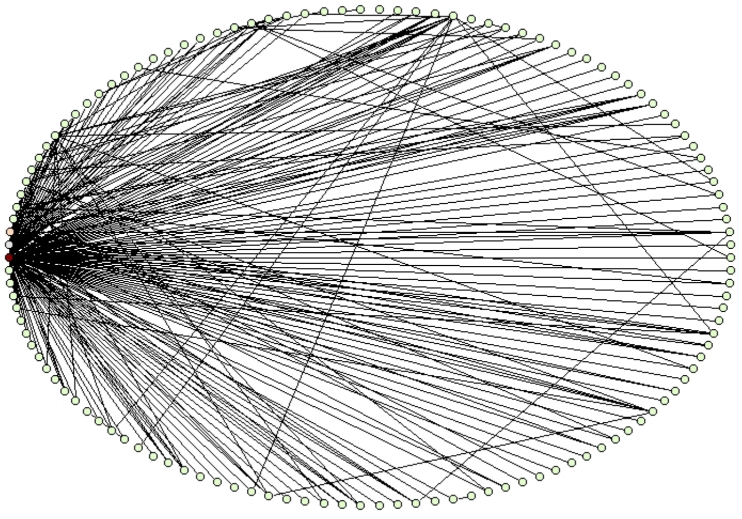


in Figure 4.3 on page 81. This figure shows that members of cluster 3 tend to direct their communication efforts to members of other clusters directly (i.e. there does not seem to be a small number of individuals who broker information between community groups). Figure 4.4 on page 82 conveys the same behaviour as the betweenness centrality of vertices representing other clusters is visibly greater. Furthermore, it shows that members of cluster 3 are also engaging in communication among themselves. If loops are eliminated, the density value drops to 0.0253 and the average degree to 6.1311.

The same operations were applied to the Mozilla graph in order to reduce all clusters except for cluster 1. The resulting network contains 104 vertices of which three represent the other clusters. The graph contains a total of 5,366 arcs of which 4,196 have a value greater than 1 and 1,170 have a value equal to 1. The network's density is 0.49611 while the average degree is 103.19. The network is depicted in Figure 4.5 on page 83. This figure shows a very dense communication (i.e. the large number of arcs is making the centre of the figure look almost completely black) both between members of cluster 1 as well as with members of other clusters. In Figure 4.6 on page 84 the variety of betweenness centrality values produces a variety of vertex sizes. This variety in size as well as the dense communication suggests that members of cluster 1 communicate both within the cluster as well as directly with members of other clusters. If loops are eliminated the density value drops to 0.5006 and the average degree to 103.13.

In order to visualise the communication patterns between the QA team as a whole and the rest of the community, clusters 1 and 3 were merged and clusters 2 and 4 were reduced. The resulting network contained 222 vertices of which two represented the other clusters. The graph contains a total of 6,099 arcs of which 4,455 have a value greater than 1 and 1,644 have a value equal to 1. The network's density is 0.1237 while the average degree

Figure 4.3: The Mozilla graph reduced except for cluster 3—depicts relations between members of cluster 3 as well as between these members and other clusters; the red, orange and grey vertices represent shrunk clusters.

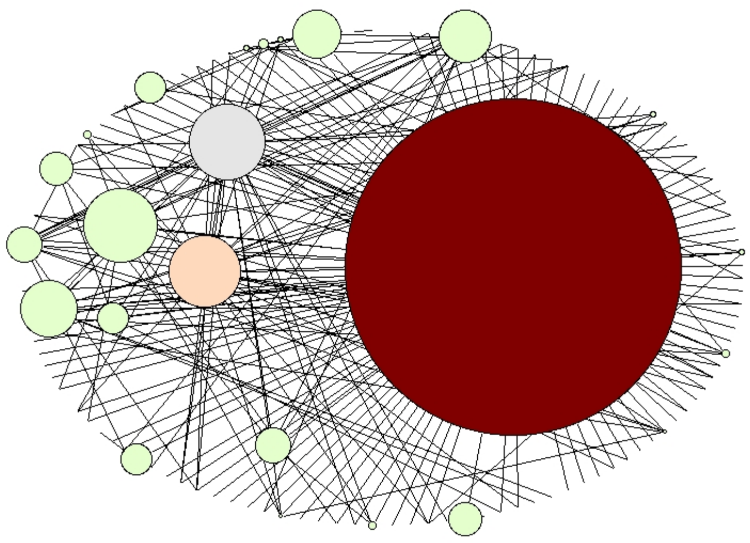


is 54.9459. The network is depicted in Figure 4.7 on page 86. This figure shows a very dense communication (i.e. the large number of arcs is making the centre of the figure look almost completely black) both between members of cluster 1 as well as with members of other clusters. In Figure 4.8 on page 87 the variety of betweenness centrality values produces a variety of vertex sizes. This variety in size as well as the dense communication suggests that members of cluster 1 communicate both within the cluster as well as directly with members of other clusters. If loops are eliminated the density value drops to 0.1242 and the degree to 54.9279.

The repeated reduction of clusters within the Mozilla community points to the fact that it is unlikely for a small number of individuals to broker information between members of the QA team and other groups of the community.

Hypothesis 11: The community graph does not contain a small number of people brokering information between the QA team and the rest of the community.

Figure 4.4: Mozilla graph reduced except for cluster 3—the size of each vertex is given by its betweenness centrality value; the red, orange and grey vertices represent shrunk clusters.

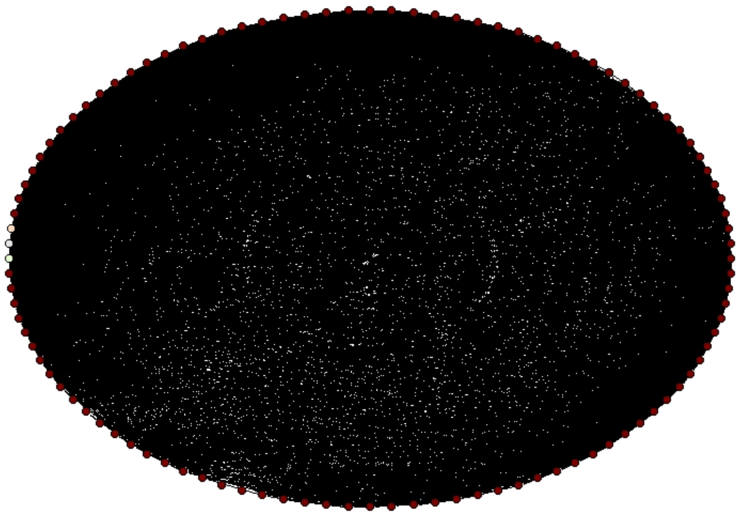


4.3.4 Social network analysis of QA-specific communications

Until this point we have analysed the graph of communications taking place on the issue tracker and QA mailing lists. The issue tracker includes communication between members not active in QA, so this broad approach allowed us to generate hypotheses about the relationships between members contributing to QA and other members. Next, however, we wanted to narrow our focus to the communications between QA members, so we excluded the issue tracker and applied SNA techniques to the graph formed by members active on the QA mailing lists. After eliminating loops, the network contained 297 vertices connected by 1,055 arcs, of which 731 had a value equal to 1 and 324 had a value greater than 1 (the highest arc value was 53). The average degree was 7.10 while the network's density was 0.0120. The graph is shown in Figure 4.9 on page 89.

The QA team graph contains nine components of which the biggest contains 285 vertices which represents approximately 96% of the whole network. Of the remaining eight components, four contain one vertex each while the other four contain two vertices each. The reason why these small components exist is similar to the reasons enumerated for the whole Mozilla community.

Figure 4.5: Mozilla graph reduced except for cluster 1—depicts relations between members of cluster 1 as well as between these members and other clusters; the green, orange and grey vertices represent shrunk clusters.



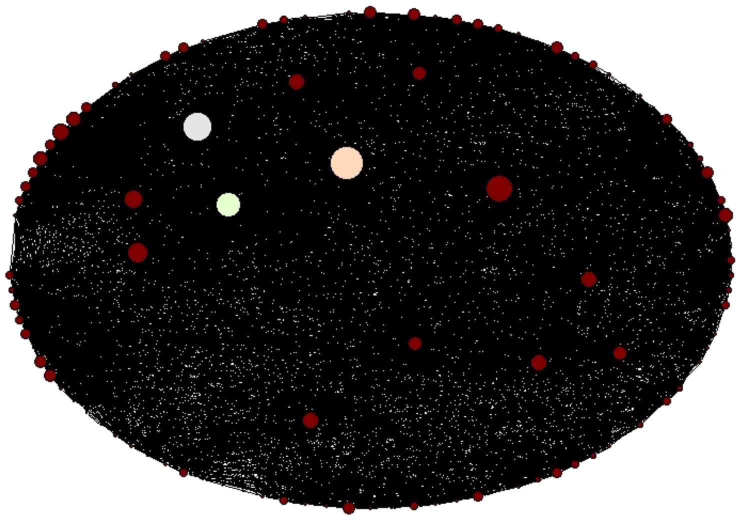
In other words, one possible reason for these small components could be that some messages are actually spam messages. Another reason would be that the messages received replies after the dataset was retrieved or that the messages were posted by members not active previously as they might be peripheral or new members who did not yet have the chance to create relations with more active users.

Hypothesis 12: The QA team forms a large group of people working together.

Next, we calculated indegree and outdegree values for the directed QA graph. The indegree value expresses the number of other members of the community an individual has interacted with by receiving messages. The outdegree value expresses the number of other members of the community an individual has interacted with by sending messages. It could be useful to observe if individuals create equal connections in the network by sending and receiving messages or whether there is some disparity.

In the directed QA graph the indegree values vary between 0 and 59. Table 4.10 on page 85 details the approximately 91% of QA team members who have indegree values between 0 and 10.

Figure 4.6: Mozilla graph reduced except for cluster 1—the size of each vertex is given by its betweenness centrality value; the green, orange and grey vertices represent shrunk clusters.



In the directed QA graph the outdegree values vary between 0 and 71. Table 4.11 on page 85 details the approximately 91% of QA team members who have outdegree values between 0 and 10.

The findings seem to suggest that members of the community active on the QA mailing lists have similar values of indegree and outdegree considering that approximately 91% of individuals receive and send from/to ten or fewer individuals. However, to portray the number of connections an individual has regarding of form of communication (i.e. sending or receiving) the graph must be symmetrized. For this purpose, the QA team graph was then symmetrized by transforming all arcs into edges and removing all multiple lines by adding the value of their weights. In this graph the degrees varied between 0 and 85. Table 4.12 on page 88 details the approximately 87% of vertices with a degree value between 0 and 10.

The number of community members with more connections has increased after normalising the degree values by not taking into consideration if the interaction consisted of sending or receiving a message. One explanation for this could be that some members send messages to a number of individuals and receive messages from a completely different number of individuals.

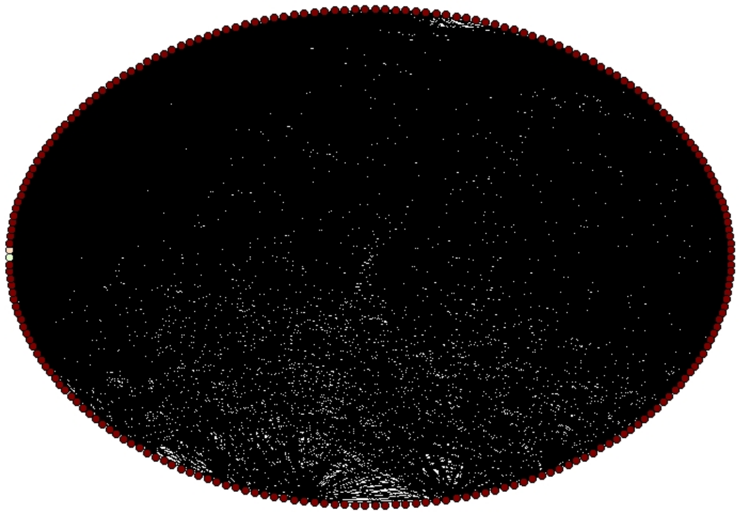
Table 4.10: Vertices in the directed QA graph clustered by indegree value

Indegree value	Frequency	Frequency %	CumFreq	CumFreq%
0	45	15.1515	45	15.1515
1	110	37.0370	155	52.1886
2	59	19.8653	214	72.0539
3	11	3.7037	225	75.7576
4	12	4.0404	237	79.7980
5	11	3.7037	248	83.5017
6	8	2.6936	256	86.1953
7	4	1.3468	260	87.5421
8	3	1.0101	263	88.5522
9	6	2.0202	269	90.5724
10	2	0.6734	271	91.2458

Table 4.11: Vertices in the directed QA graph clustered by outdegree value

Outdegree value	Frequency	Frequency %	CumFreq	CumFreq%
0	75	25.2525	75	25.2525
1	101	34.0067	176	59.2593
2	34	11.4478	210	70.7071
3	16	5.3872	226	76.0943
4	16	5.3872	242	81.4815
5	12	4.0404	254	85.5219
6	1	0.3367	255	85.8586
7	5	1.6835	260	87.5421
8	3	1.0101	263	88.5522
9	4	1.3468	267	89.8990
10	5	1.6835	272	91.5825

Figure 4.7: Mozilla graph reduced except for clusters 1 and 3—depicts relations between members of cluster 1 and 3 as well as between these members and other clusters; the orange and green vertices represent shrunk clusters.



Hypothesis 13: About one third of the QA team members create only one connection by receiving messages.

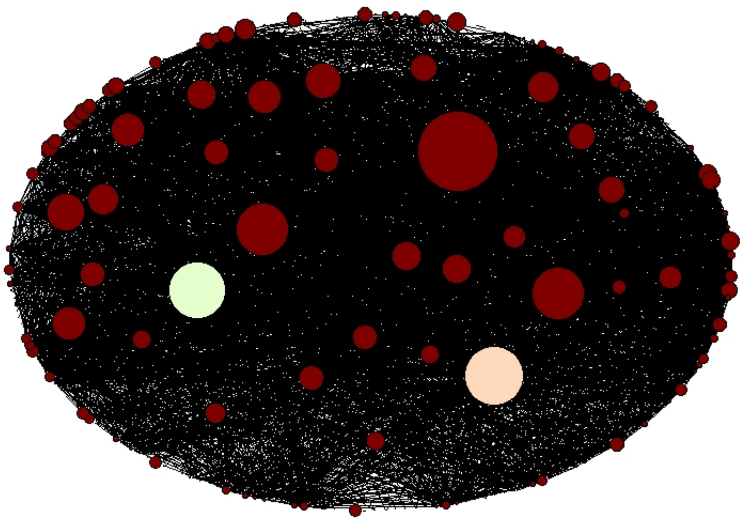
Hypothesis 14: About one third of the QA team members create only one connection by sending messages.

The symmetrized graph contains nine components, of which the biggest contains 285 vertices or approximately 96% of all vertices. However, if the network is not symmetrized, the directed graph contains 128 strong components of which the largest contains 170 vertices or about 57%. The other 127 components contain one vertex each. In other words, the largest component in which any two vertices are connected by a path[5] contains approximately 57% of vertices. The highest k-core in the symmetrized graph is a 9-core containing 20 vertices or approximately 7% of all vertices. This means that in this group (subgraph) of 20 individuals, 20 is the lowest degree within the subnetwork. The largest k-core is a 1-core containing about 41% of all vertices.

In order to understand communication patterns within the QA team, the graph was was divided into four clusters as follows:

[5]Please see detailed description of what a path is on page 59

Figure 4.8: Mozilla graph reduced except for clusters 1 and 3—the size of each vertex is given by its betweenness centrality value; the orange and green vertices represent shrunk clusters.



- Cluster 1—Members of the community who were assigned as bug fixers on the issue tracker and were active on the technical QA mailing list Mozmill.
- Cluster 2—Members of the community who were assigned as bug fixers on the issue tracker and were not active on Mozmill.
- Cluster 3—Members of the community who were not assigned as bug fixers on the issue tracker and were active on Mozmill.
- Cluster 4—Members of the community that were not assigned as bug fixers on the issue tracker and were not active on Mozmill. In addition, members who could not be categorised were added to this cluster.

Table 4.13 on page 88 provides more details regarding cluster distribution within the QA team graph.

After analysing the cluster distribution, we realised that the dataset is incomplete for the purpose of determining whether QA teams represent a separate layer in FLOSS communities. The full dataset should include the complete mailing lists archives (i.e. not only the QA mailing list archives) as well as a complete list of code contributors. However the following hypotheses

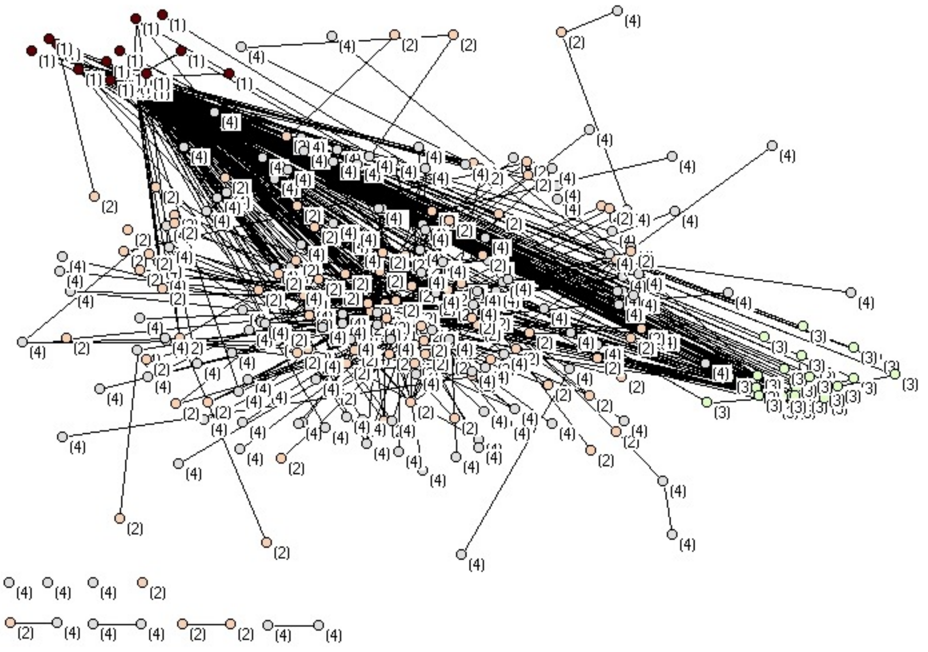
Table 4.12: Vertices in the symmetrized QA graph clustered by degree value

Degree value	Frequency	Frequency %	CumFreq	CumFreq%
0	4	1.3468	4	1.3468
1	116	39.0572	120	40.4040
2	63	21.2121	183	61.6162
3	28	9.4276	211	71.0438
4	13	4.3771	224	75.4209
5	9	3.0303	233	78.4512
6	6	2.0202	239	80.4714
7	7	2.3569	246	82.8283
8	4	1.3468	250	84.1751
9	4	1.3468	254	85.5219
10	5	1.6835	259	87.2054

Table 4.13: Clusters in the QA team

Cluster	Frequency	Frequency %	CumFreq	CumFreq%
1	22	7.4074	22	7.4074
2	99	33.3333	121	40.7407
3	27	9.0909	148	49.8316
4	149	50.1684	297	100.0000

Figure 4.9: QA team graph clusters—vertices that belong to a cluster share the same colour and are also labeled with the same cluster number



were proposed based on the Mozilla case and should be validated and/or modified accordingly after being tested in other case studies.

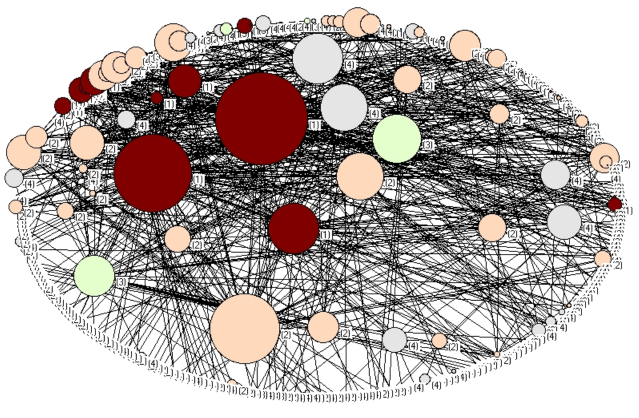
Hypothesis 15: More than half of the individuals active on the QA mailing lists are not active on other mailing lists and are not code contributors.

Hypothesis 16: More than one third of the individuals active on the QA mailing lists are listed as code contributors.

The value of betweenness centrality within the QA team graph is 0.1720. Vertices with a betweenness centrality equal to 0 represent approximately 60% of the vertices. Vertices with betweenness centrality values between 0.059 and 0.176 represent about 1% of the vertices. Figure 4.10 on page 90 shows clusters within the QA team; the size of the vertices reflects the centrality betweenness score.

Social network analysis techniques were applied so far on an aggregated form of the Mozilla graph. In other words the vertices and arcs were not separated considering time frames or various points in the network's evolution. If analysis techniques are applied without taking into consideration the dynamic nature of communities, the metrics such as degree values or even size might be skewed. For that reason, the Mozilla subnetwork formed of

Figure 4.10: QA team graph clusters—the size of each vertex is given by its betweenness centrality value



members active on the QA mailing lists was analysed using six-month time frames. The subgraphs for each time frame were as follows:

- SN_1 — 26 vertices connected by 33 arcs where the average degree was 2.53. The network betweenness centralisation score was 0.3377 where about 8% of vertices had a betweenness centrality score higher than average (i.e. 0.116). After symmetrising the network it contained 22 edges. The average degree dropped to 1.69.

- SN_2 — 35 vertices connected by 64 arcs where the average degree was 3.65. The network betweenness centralisation score was 0.1565 where about 11% of vertices had a betweenness centrality score higher than average (i.e. 0.057). After symmetrising the network it contained 45 edges. The average degree dropped to 2.57.

- SN_3 — 33 vertices connected by 29 arcs where the average degree was 1.75. The network betweenness centralisation score was 0.1211 where about 3% of vertices had a betweenness centrality score higher than average (i.e. 0.041). After symmetrising the network it contained 20 edges. The average degree dropped to 1.21.

- SN_4 — 15 vertices connected by 10 arcs where the average degree was 1.33. The network betweenness centralisation score was 0.0423 where about 27% of vertices had a betweenness centrality score higher than average (i.e. 0.016). After symmetrising the network it contained eight edges. The average degree dropped to 1.06.

- SN_5 — 36 vertices connected by 45 arcs where the average degree was 2.5. The network betweenness centralisation score was 0.1329 where about 11% of vertices had a betweenness centrality score higher than average (i.e. 0.048). After symmetrising the network it contained 32 edges. The average degree dropped to 1.77.

- SN_6 — 37 vertices connected by 44 arcs where the average degree was 2.37. The network betweenness centralisation score was 0.1626 where about 16% of vertices had a betweenness centrality score higher than average (i.e. 0.060). After symmetrising the network it contained 31 edges. The average degree dropped to 1.67.

- SN_7 — 41 vertices connected by 59 arcs where the average degree was 2.87. The network betweenness centralisation score was 0.1704 where about 7% of vertices had a betweenness centrality score higher than average (i.e. 0.061). After symmetrising the network it contained 41 edges. The average degree dropped to 2.

- SN_8 — 64 vertices connected by 77 arcs where the average degree was 2.4. The network betweenness centralisation score was 0.1619 where about 3% of vertices had a betweenness centrality score higher than average (i.e. 0.055). After symmetrising the network it contained 59 edges. The average degree dropped to 1.84.

- SN_9 — 32 vertices connected by 42 arcs where the average degree was 2.62. The network betweenness centralisation score was 0.0761 where about 3% of vertices had a betweenness centrality score higher than average (i.e. 0.026). After symmetrising the network it contained 29 edges. The average degree dropped to 1.81.

- SN_{10} — 31 vertices connected by 38 arcs where the average degree was 2.45. The network betweenness centralisation score was 0.1406 where about 3% of vertices had a betweenness centrality score higher than average (i.e. 0.048). After symmetrising the network it contained 28 edges. The average degree dropped to 1.8.

- SN_{11} — 25 vertices connected by 31 arcs where the average degree was 2.48. The network betweenness centralisation score was 0.1015 where about 8% of vertices had a betweenness centrality score higher than average (i.e. 0.037). After symmetrising the network it contained 21 edges. The average degree dropped to 1.68.

- SN_{12} — 5 vertices connected by 9 arcs where the average degree was 3.6. The network betweenness centralisation score was 0.4583 where about 20% of vertices had a betweenness centrality score higher than average (i.e. 0.167). After symmetrising the network it contained seven edges. The average degree dropped to 2.8.

The Mozilla QA team was active in all the time frames examined. The subnetworks associated with each time frame varied in size from 15 to 64 vertices[6]. The average degree after symmetrising the networks varied between 1.06 and 2.57. The findings, as predicted, seem to point to much smaller networks and lower degrees compared to the aggregated QA team graph. However, as the degree values were not exceptionally high in a given time frame compared to the aggregated graph it is safe to assume that a number of people made a sustained communication effort within the group and created connections between themselves as well as with members of the periphery or occasional contributors. These findings are consistent with the initial assumption that a small group of individuals within the QA team is communicating more and by extension performing most of the tasks. The value of the betweenness centrality value for the aggregated graph is 0.1720 where only 1% of participants have a higher than average score. The subnetworks' betweenness centralisation scores vary between 0.0423 and 0.3377 where the percentage of participants having a higher than average betweenness score varied between 3% and 27%. These findings confirm that subnetworks will display higher centrality values than the aggregated form of the graph.

Hypothesis 17: Size, degree and betweenness values of temporal subgraphs do not display a consistent growth over time but display a more irregular pattern.

4.4 Summary

After establishing a working definition of QA, an assessment of QA presence was carried out on the top 100 FLOSS projects listed on Ohloh. The assessment showed that more than a quarter of these projects include some form of QA in their development. The next step consisted in choosing the Mozilla community as a preliminary case study in order to generate a set of hypotheses that will be tested in the next stage of this research.

The dataset comprising data from QA mailing lists and the issue tracker was retrieved, stored on a local machine, cleaned and analysed. Simple statistical analysis techniques were first applied in order to analyse activity per

[6]As the dataset for SN_{12} was not complete, it was not taken into consideration.

year or activity performed by various layers in the community. The findings showed that a smaller number of peripheral members are active on the mailing lists than on the issue tracker. Furthermore, activity in the QA mailing lists showed peaks that are independent from activity carried out on other venues. Next, social network analysis techniques were applied to the data set. The social network analysis showed that in the case of Mozilla, the community forms one group that spans both mailing lists and issue trackers. Furthermore, a small group of people is highly connected within the network and also displays higher communication activities. Another interesting finding is that almost two thirds of connections were formed by single acts of communication from one individual to another, suggesting weaker ties. Members of the QA team show a distributed communication pattern in the sense that the QA team does not contain a small number of individuals who control the information flow. Furthermore, the QA team seems to display a similar structure to the whole Mozilla community in the sense that a small number of members are highly engaged in communication and thus highly connected.

Chapter 5

Case studies

In this chapter we will analyse further case studies in order to test the hypotheses proposed in the previous chapter (Phase I). Four case studies have been chosen on the basis of the success of their products, their maturity and the existence of a dedicated QA team in their community structure. Compared to the pilot case study, the datasets associated with these four cases were more extensive in the sense that a list of code contributors[1] as well as mailing lists not associated with QA activities were retrieved[2]. Analysis was performed similarly to the analysis of the Mozilla community with minor differences. For example, the community was split into clusters based on members' activity on QA mailing lists and by comparing members' names to the list of code contributors retrieved from Ohloh. For modularity and clarity purposes, each case study will be presented in a separate section which will contain: a short description of the project, the motivation behind choosing the project, general statistics, social network analysis of the whole community graph, and analysis of an aggregated form of the subgraph containing members active exclusively on the QA mailing lists as well as an analysis over six month time frames.

5.1 Ubuntu

5.1.1 The Ubuntu project

In 2004 the South African entrepreneur Mark Shuttleworth gathered a small team of developers from the Debian community with the purpose of develop-

[1]The list of code contributors was downloaded from www.ohloh.net

[2]Please see Appendix B on page 212 for a full list of mailing lists associated with each project.

ing an easy to use Linux operating system which he called Ubuntu[3]. Since its official launch the same year, thousands of FLOSS experts and enthusiasts alike have joined the community, and today Ubuntu is one of the world's most widely used Linux distributions: about 70% of the PCs shipped by the major PC companies are now certified to work with Ubuntu[4]. Ubuntu is clearly a mature and successful FLOSS project and is therefore a suitable case study for this research.

Ubuntu has a QA team[5] which uses a dedicated mailing list, an IRC channel, a blog and other resources for communication and for providing relevant information to potential contributors. QA team tasks, as described on the QA webpage, include writing test cases, executing tests, reporting bugs, giving instructional presentations, and encouraging best practices for quality. The Ubuntu community has a BugSquad team[6] which uses a dedicated mailing list, a wiki, an IRC channel and other resources for communication and for providing relevant information to potential contributors. BugSquad tasks include assigning bugs to packages, ensuring that bug reports are complete, finding duplicate bug reports, recreating bugs, and forwarding bugs to their upstream authors. In addition, the Ubuntu community has also had a laptop testing team that was using a dedicated mailing list. The activity of that mailing list decreased substantially after 2008 but for the purpose of this research it is regarded as QA related and included in our analysis as it might provide relevant information with respect to community structure.

5.1.2 General analysis of the Ubuntu dataset

In order to measure QA activity levels on more than one channel, issue tracker data as well as mailing list data were taken into account. Data were retrieved between April and May 2013 and stored locally in a PostgreSQL database. The issue tracker dataset contains a total of 993,420 bugs with 4,114,392 associated comments, and was downloaded using a web crawler. The mailing list dataset contains a total of 487,694 e-mails on all Ubuntu topics[7] that were downloaded using a Python script written specifically for the purpose. This dataset includes three QA-related mailing lists which contain a total of 8,798 e-mails. In addition, a list of contributors' usernames, nicknames

[3]Ubuntu—about: http://www.ubuntu.com/about/about-ubuntu

[4]Ubuntu—business: http://www.ubuntu.com/desktop/business

[5]Ubuntu quality assurance: http://community.ubuntu.com/contribute/quality/

[6]Ubuntu BugSquad: https://wiki.ubuntu.com/BugSquad

[7]The total number of analysed e-mails was 1,468,114, however due to bad HTML, double postings and other errors 980,420 e-mails were not stored in the database.

and (where available) real names was downloaded from Ohloh[8]. This list was used to perform data cleaning as follows: names contained in the issue tracker and the mailing list were compared with names from code repositories and unified.

Of all the e-mails exchanged 12,169 (2.49%) were sent by authors who had sent only one e-mail throughout the period of the study while of all the QA e-mails 839 (9.53%) were sent by authors who had sent only one e-mail. On the other hand 122,376 bugs (12.31%) were posted by members who had posted only one bug throughout the period of the study while 128,287 (3.11%) comments were posted by members who posted only one comment. The activity on mailing lists and the bug tracker on a yearly basis is shown in Figure 5.1 on page 97, while more detailed information is given in Table C.8 on page 271. The figure shows a steady increase in the number of bug comments from 2003 until 2010 followed by a slight drop. The number of bugs posted shows a much less pronounced rise and drop over the same period. Trends in mailing list activities are difficult to discern from the figure as the numbers are much smaller than those of bug postings and comments, but they show rises and drops from year to year with no discernible pattern.

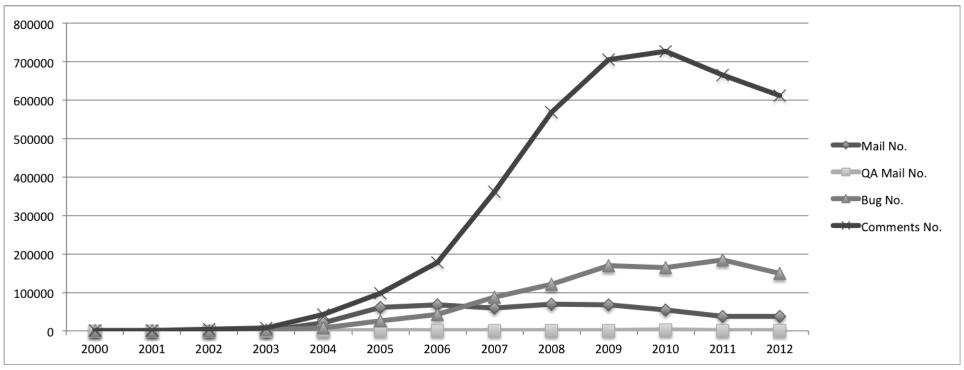


Figure 5.1: Ubuntu Activity Chart

The QA mailing lists started in 2005 and include 1,564 members of whom 53.64% (839 participants) sent only one e-mail. The average number of e-mails sent is 5.63 (sd = 22.5). Only 14.89% of QA mailing lists participants sent more than six e-mails. A detailed description of top QA e-mail participants' activity is given in Table 5.1 on page 98[9]. In the table the Author column represents a label assigned to each participant, the QA e-mails column represents the number of e-mails the participant sent to the QA mailing

[8]Ohloh: https://www.ohloh.net/

[9]A table containing all QA mailing list participants and their activity can be found in Table C.7 on page 225

lists, the Other e-mails represents the number of e-mails sent to other mailing lists, the Lists column represents the number of other mailing lists the participant has been active on, the Bugs column represents the number of bugs the participant has posted on the issue tracker, the Comments column represents the number of comments the participant has posted on the issue tracker, and the Code Contributor column represents whether the participant had submitted code to the project.

Table 5.1: Ubuntu QA mailing list participants' activity levels on other channels

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{352}	456	167	9	845	24231	Yes
X_{958}	397	42	5	133	0	No
X_{907}	343	0	0	0	0	No
X_{479}	249	12	1	0	0	No
X_{1468}	207	436	5	23	0	No
X_{624}	201	24	7	401	0	No
X_{723}	126	149	5	0	0	No
X_{1183}	120	1046	5	169	0	No
X_{916}	119	1256	14	1218	15392	Yes
X_{1421}	119	11	2	0	0	No
X_{1056}	110	258	9	185	0	No
X_{1029}	87	42	3	0	0	No
X_{1108}	86	62	2	44	0	No
X_{1223}	82	580	6	64	0	No
X_{179}	81	110	8	0	0	No
X_{691}	76	18	1	0	0	No
X_{47}	72	737	6	114	0	No
X_{1406}	69	6	1	36	0	No
X_{1508}	67	669	5	352	0	No
X_{772}	66	30	4	45	0	No

Of the total number of individuals active on the QA mailing list, 957 (61%) participants were not active on other mailing lists, and of these 957 members, 347 sent more than one message. Furthermore, of these 347 participants none was listed as a code contributor. In addition, 1,526 or about 98% of the total 1,564 participants were not listed as code contributors. These facts suggest that QA contributors form a separate layer in the Ubuntu community.

5.1.3 Social network analysis of the whole dataset

The next stage was to carry out a social network analysis of the Ubuntu community. The dataset was exported to a Pajek-readable file that contained community members, the links between them and the periods in which they were active in the project. Members who did not form any connections with other members were eliminated from the dataset. As in the Mozilla pilot case study, after importing this data into Pajek all loops were eliminated due to the fact that a message sent by a community member as a reply to one of his/her previous messages does not contribute to our understanding of community structure or dynamics.

The resulting network contains 294,988 vertices connected by a total of 1,530,150 arcs of which 323,313 have a value that is greater than 1 and 1,206,837 have a value equal to 1. This means that 1,206,837 connections (78.87%) are created by only one interaction (message). One could draw the conclusion that these members are occasional contributors or part of the periphery. The average degree is 10.37, which means that on average a person interacts with approximately ten other people. The network's density is 0.000017. The "heaviest" ten arcs in the network connected seven people and had values between 1,724 and 7,418; each of these represents one person sending many hundreds of communications to another person; see Figure 5.2 on page 99.

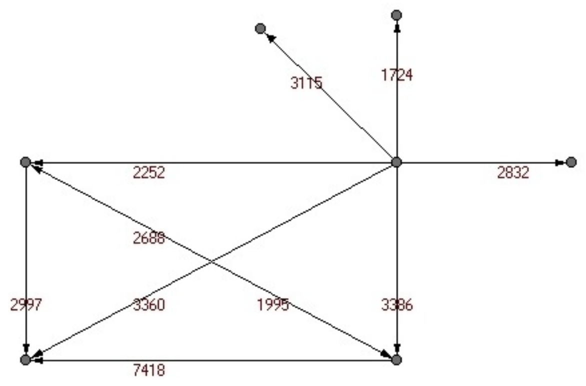


Figure 5.2: The heaviest ten arcs in the Ubuntu graph connecting ten members—numbers represent e-mails and bug comments in each direction

The network contains 2,796 components of which the biggest component contains 291,357 vertices or 98.77% of the community. Of the rest of the components, the largest contain nineteen, thirteen and ten vertices while the remainder contain less between one and seven vertices.

To get the precise values of vertex degrees within the Ubuntu community, the directed network was transformed into an undirected network by transforming all arcs into edges. This process was executed by adding the weights of arcs. The resulting network has 1,215,389 edges of which 374,015 have a value greater than 1 and 841,374 edges have a value equal to 1. The network's density is 0.000027. The average degree drops to 8.24 compared to 10.37 for the directed network. About 13% of individuals had connections with more than eight (i.e. the average degree) other individuals. Looking at the vertices with degree value from 0 to 10, we find that they account for approximately 89% of the community. The remaining approximately 11% of vertices with the highest degrees are connected to—that is, communicate directly with—between 11 and 34,003 other members. Details of the 89% of vertices in the Ubuntu graph with degree value between 0 and 10 are given in Table 5.2 on page 100. In the table, the Frequency column is the number of vertices with that value, the Frequency % column is the percentage of vertices with that value from the whole community, the CumFreq is cumulative frequency, i.e. the sum of the number of the vertices with that value added to the numbers of vertices with lower degree values and the CumFreq % is the cumulative frequency expressed as a percentage of the whole community graph. The largest degree cluster is formed by vertices with a degree value of 1 and represents approximately one third of the community.

Table 5.2: Vertices in the symmetrized Ubuntu graph clustered by degree value

Degree value	Frequency	Frequency %	CumFreq	CumFreq%
0	2,143	0.7265	2,143	0.7265
1	99,672	33.7885	101,815	34.5150
2	78,305	26.5451	180,120	61.0601
3	25,972	8.8044	206,092	69.8645
4	19,487	6.6060	225,579	76.4706
5	11,049	3.7456	236,628	80.2161
6	8,481	2.8750	245,109	83.0912
7	6,150	2.0848	251,259	85.1760
8	4,934	1.6726	256,193	86.8486
9	3,801	1.2885	259,994	88.1371
10	3,184	1.0794	263,178	89.2165

As it is possible for some members to be sending or receiving more messages than others, both the indegree and outdegree values were computed. In the directed Ubuntu graph vertices with values between 0 and 10 make up about 93% of the community—see table 5.3 on page 101. The remaining vertices have indegree values of between 11 and 14,987.

Table 5.3: Vertices in the directed Ubuntu graph clustered by indegree value

Indegree value	Frequency	Frequency %	CumFreq	CumFreq%
0	18,254	6.1880	18,254	6.1880
1	147,879	50.1305	166,133	56.3186
2	47,099	15.9664	213,232	72.2850
3	21,853	7.4081	235,085	79.6931
4	12,619	4.2778	247,704	83.9709
5	8,318	2.8198	256,022	86.7906
6	5,718	1.9384	261,740	88.7290
7	4,249	1.4404	265,989	90.1694
8	3,398	1.1519	269,387	91.3213
9	2,723	0.9231	272,110	92.2444
10	2,299	0.7794	274,409	93.0238

In the same directed Ubuntu graph approximately 6% of vertices have outdegree values between 11 and 30,535; see Table 5.4 on page 101.

Table 5.4: Vertices in the directed Ubuntu graph clustered by outdegree value

Outdegree value	Frequency	Frequency %	CumFreq	CumFreq%
0	74,912	25.3949	74,912	25.3949
1	113,240	38.3880	188,152	63.7829
2	37,182	12.6046	225,334	76.3875
3	17,548	5.9487	242,882	82.3362
4	10,599	3.5930	253,481	85.9293
5	6,831	2.3157	260,312	88.2449
6	5,071	1.7191	265,383	89.9640
7	3,759	1.2743	269,142	91.2383
8	3,025	1.0255	272,167	92.2638
9	2,397	0.8126	274,564	93.0763
10	1,883	0.6383	276,447	93.7147

The directed Ubuntu graph contains 94,586 strong components. The largest strong component contains 199,781 vertices which amounts to approximately 68% of the whole network. The symmetrized i.e. undirected network contains 2,796 components where the largest component contains

291,357 vertices or approximately 99% of the whole community. In the same symmetrized Ubuntu graph, k-core values range between 0 and 92, in other words the most connected group of members contains individuals who interacted with a minimum of 92 other people. This group contains 318 community members. Table 5.5 on page 102 gives details of the members who have k-core values between 0 and 10, and who make up approximately 92% of the graph.

Table 5.5: Vertices in the symmetrized Ubuntu graph clustered by k-core value

K-core value	Frequency	Frequency %	CumFreq	CumFreq%
0	2,143	0.7265	2,143	0.7265
1	102,667	34.8038	104,810	35.5303
2	82,414	27.9381	187,224	63.4683
3	28,811	9.7668	216,035	73.2352
4	18,614	6.3101	234,649	79.5453
5	11,589	3.9286	246,238	83.4739
6	8,040	2.7255	254,278	86.1994
7	6,091	2.0648	260,369	88.2643
8	4,628	1.5689	264,997	89.8331
9	3,575	1.2119	268,572	91.0451
10	3,026	1.0258	271,598	92.0709

Betweenness centrality values were then computed as they represent the importance of each vertex to the information flow. The Ubuntu directed graph vertices have betweenness centralities of between 0 and 0.118 where the network's centralisation value is equal to 0.1179. Table 5.6 on page 103 details the distribution of betweenness centrality vectors within the network. Findings suggest that approximately 47% of the Ubuntu community do not represent a "step" in the information flow. Furthermore, much fewer than 1% of vertices have a higher than average betweenness score and the maximum betweenness centrality is 0.118, which is higher than the maximum value for the Mozilla community which was 0.063.

Table 5.6: Betweenness centrality clusters in the Ubuntu community

Value Intervals	Frequency	Frequency %	CumFreq	CumFreq%
0.000	139,680	47.3511	139,680	47.3511
0.000–0.039	155,304	52.6476	294,984	99.9986
0.039–0.079	3	0.0010	294,987	99.9997
0.079–0.118	1	0.0003	294,988	100.0000

To get a better understanding of communication patterns between QA team members and the rest of the community the graph was divided into four clusters as follows:

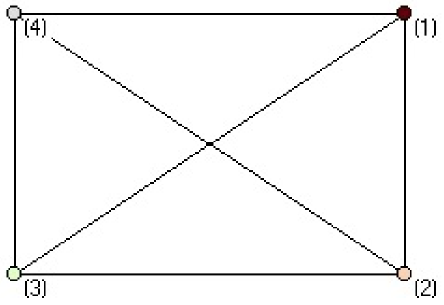
1. Cluster 1—Community participants who contribute code and are members of the QA mailing lists.
2. Cluster 2—Community participants who contribute code but are not members of the QA mailing lists.
3. Cluster 3—Community participants who do not contribute code but are members of the QA mailing lists.
4. Cluster 4—Community participants who do not contribute code and are not members of the QA mailing lists. Members who could not be categorised were also added to this cluster.

Table 5.7: Clusters in the Ubuntu community

Cluster	Frequency	Frequency %	CumFreq	CumFreq%
1	32	0.0108	32	0.0108
2	172	0.0583	204	0.0692
3	1,231	0.4173	1,435	0.4865
4	293,553	99.5135	294,988	100.0000

To categorise participants into clusters, a list containing Ubuntu code contributors was retrieved from Ohloh. A Python script was then written in order to compare all community members whose names contained more than four characters to the code contributor list and categorise them into clusters based on participation on the QA mailing list. Table 5.7 on page 103 provides more details regarding cluster distribution within the community. It will be observed that the cluster containing members who are neither code contributors nor QA contributors is the largest, representing approximately

Figure 5.3: Ubuntu reduced graph—each cluster was reduced to one vertex in order to illustrate connections between various groups within the community.



99.5% of the whole community. On the other hand people performing QA tasks appear to add up to only about 0.4% of the community.

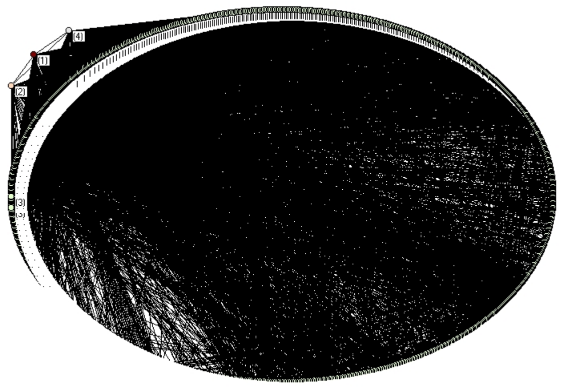
If each of the clusters is shrunk to one vertex then the resulting network has four vertices and is a complete network with a density value of 1. The graph contains a total of four loops and twelve lines and is depicted in Figure 5.3 on page 104. The values of the lines of the graph are described in Table 5.8 on page 104.

Table 5.8: Shrunk Ubuntu network communication

Rank	Line	Value
1	1.1	2,067,856
2	3.1	180,478
3	1.3	151,307
4	4.1	73,647
5	1.4	64,914
6	2.1	41,360
7	1.2	33,557
8	3.3	19,190
9	2.2	9,208
10	3.4	7,560
11	4.3	7,192
12	3.2	5,373
13	4.4	4,767
14	2.3	4,735
15	2.4	2,064
16	4.2	1,799

The next step was to reduce all clusters to one vertex except for cluster 3, the QA-only cluster. The resulting network contains 1,234 vertices of which

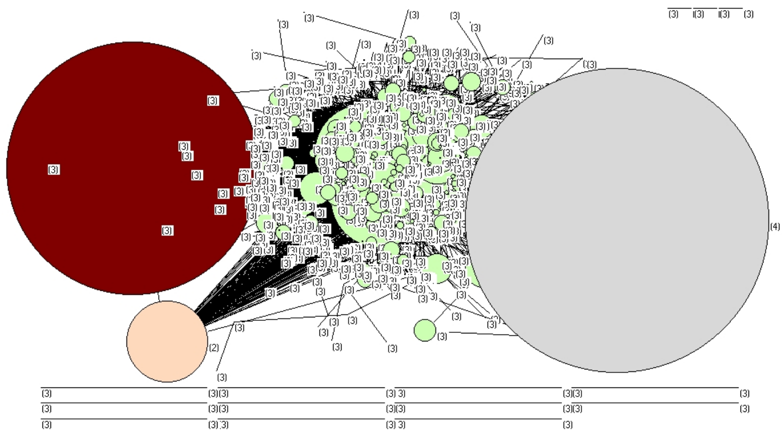
Figure 5.4: Ubuntu graph shrunk except for those who are active on the QA mailing lists but do not submit code (cluster 3)—depicts relations between members of cluster 3 as well as between these members and other clusters; the red, orange and grey vertices represent shrunk clusters.



three represent the other clusters. The graph contains a total of 7,145 arcs of which 2,914 have a value greater than 1 and 4,231 have a value equal to 1. The network's density is 0.0046 while the average degree is 11.58. The network is depicted in Figure 5.4 on page 105. This figure shows a very dense communication (i.e. the large number of arcs is making the centre of the figure look almost completely black) both between members of cluster 3 as well as with members of other clusters. In Figure 5.5 on page 106 the variety of betweenness centrality produces a variety of vertex sizes while a dense communication can be observed suggesting that members of cluster 3 communicate both within the cluster as well as directly with members of other clusters. If loops are eliminated the density value is 0.0046 and the degree drops to 11.57.

The same operations were applied to the Ubuntu graph in order to reduce all clusters except for cluster 1, i.e. those who both submit code and are active on the QA mailing lists. The resulting network contains 35 vertices of which three represent the other clusters. The graph contains a total of 870 arcs of which 767 have a value greater than 1 and 103 have a value equal to 1. The network's density is 0.7102 while the average degree is 49.71. The resulting network is depicted in Figure 5.6 on page 107. This figure shows a distributed communication among both members of cluster 1 as well as with members of other clusters. In Figure 5.7 on page 107 the variety of betweenness centrality produces a variety of vertex sizes while communication displays the same pattern suggesting that members of cluster 1 communicate

Figure 5.5: Ubuntu graph shrunk except for those who are active on the QA mailing lists but do not submit code (cluster 3)—the size of each vertex is given by its betweenness centrality value; the red, orange and grey vertices represent shrunk clusters.



both within the cluster as well as directly with members of other clusters. If loops are eliminated the density value is 0.7285 and the degree drops to 49.54.

In order to visualise the communication patterns between the QA team as a whole and the rest of the community, cluster 1 was merged with cluster 3 and clusters 2 and 4 were reduced. The resulting network contains 1,265 vertices of which two represent the other clusters (i.e. 2 and 4). The graph contains a total of 9,966 arcs of which 4,473 have a value greater than 1 and 5,493 have a value equal to 1. The network's density is 0.0062 while the average degree is 15.75. The resulting network is depicted in Figure 5.8 on page 108. This figure shows a dense communication among both members of cluster 1 as well as with members of other clusters. In Figure 5.9 on page 108 the variety of betweenness centrality produces a variety of vertex sizes while communication displays the same pattern suggesting that members of cluster 1 communicate both within the cluster as well as directly with members of other clusters. If loops are eliminated the density value remains 0.0062 and the degree remains 15.75.

The analysis of communication patterns between members of the community performed by reducing clusters suggests that members active on the QA mailing lists communicate among themselves and do not seem to be exclusively oriented towards other groups within the community. Communication seems to be somewhat decentralised in the sense that no individual or few individuals are central in the network. The fact that QA members commu-

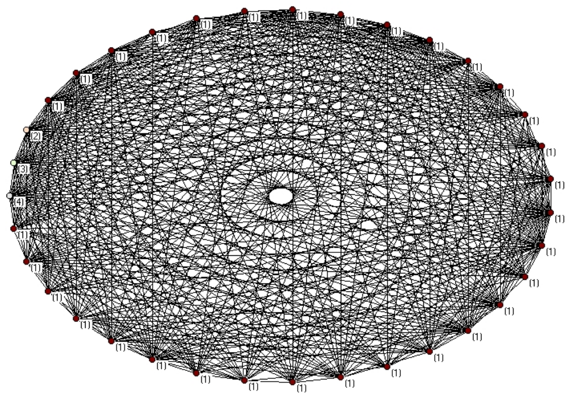


Figure 5.6: Ubuntu graph shrunk except for those who both code and are active on the QA mailing lists (cluster 1)—depicts relations between members of cluster 1 as well as between these members and other clusters; the green, orange and grey vertices represent shrunk clusters.

Figure 5.7: Ubuntu graph shrunk except for those who both code and are active on the QA mailing lists (cluster 1)—the size of each vertex is given by its betweenness centrality value; the green, orange and grey vertices represent shrunk clusters.

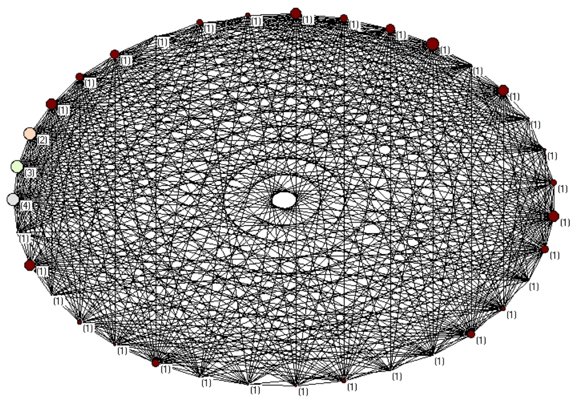


Figure 5.8: Ubuntu graph reduced except for cluster 1 and 3—depicts relations between members of cluster 1 and 3 as well as between these members and other clusters; the orange and green vertices represent reduced clusters.

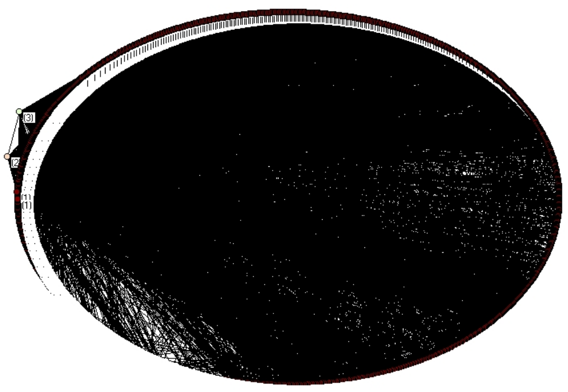


Figure 5.9: Ubuntu graph reduced except for cluster 1 and 3—the size of each vertex is given by its betweenness centrality value; the orange and green vertices represent reduced clusters.

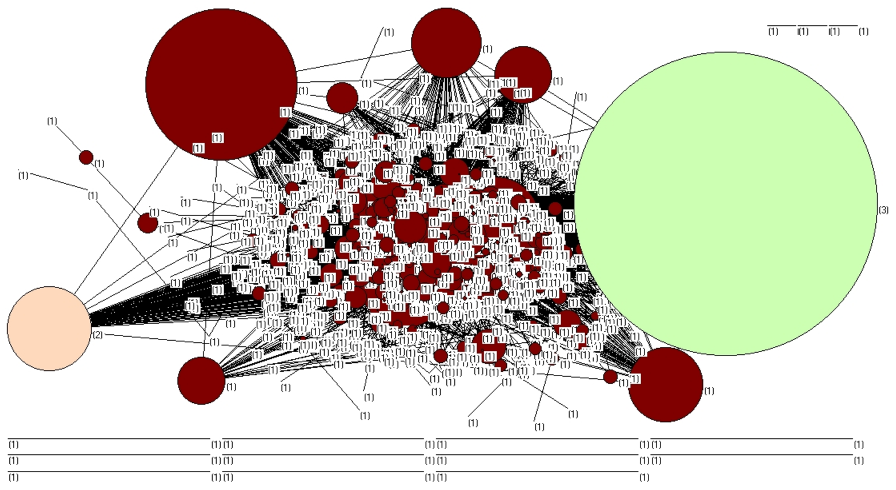
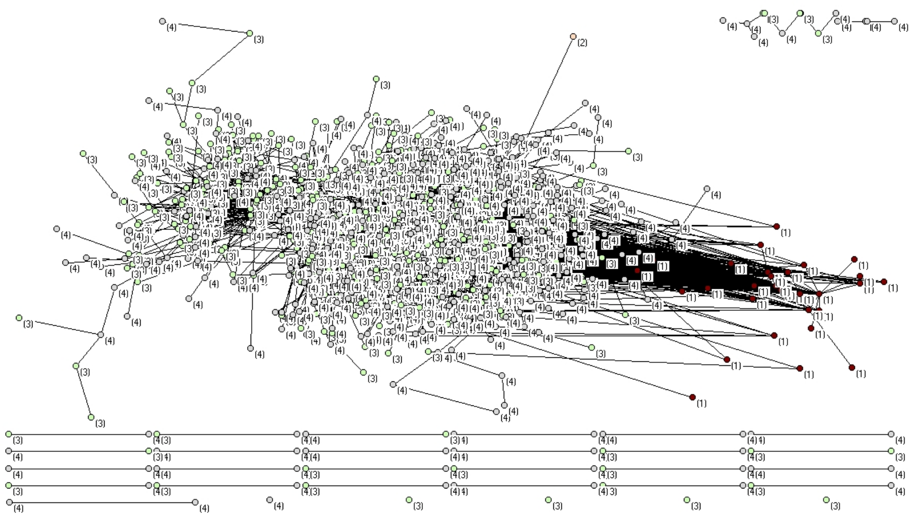


Figure 5.10: Ubuntu QA team graph clusters—each colour in the graph represents a cluster and is labeled as such.



nicate directly with members of other layers in the community and the lack of few central individuals also suggests that there are no information brokers between layers.

5.1.4 Social network analysis of QA-specific communications

So far we have analysed communications carried out on both issue tracker and mailing lists. Now, however, we want to focus on QA activities within the Ubuntu community, so we restrict our dataset to QA-related mailing lists. After eliminating loops the network contains 1,250 vertices connected by 3,267 arcs of which 2,620 have a value equal to 1 and 647 have a value greater than 1 (the highest value an arc had is 15). The average degree is 5.22 while the network's density is 0.00209. The graph is shown in Figure 5.10 on page 109.

The QA team graph contains 36 components of which the largest includes 1,181 vertices or approximately 94% of the whole QA team. The other components contain between one and three vertices.

In the directed QA graph the indegree values vary between 0 and 111. Table 5.9 on page 110 details the distribution of indegree values.

In the directed QA graph the outdegree values vary between 0 and 178. Table 5.10 on page 111 details the distribution of outdegree values.

Table 5.9: Vertices in the directed Ubuntu QA graph clustered by indegree value

Indegree value	Frequency	Frequency %	CumFreq	CumFreq%
0	184	14.7200	184	14.7200
1	582	46.5600	766	61.2800
2	204	16.3200	970	77.6000
3	85	6.8000	1055	84.4000
4	50	4.0000	1105	88.4000
5	23	1.8400	1128	90.2400
6	23	1.8400	1151	92.0800
7	12	0.9600	1163	93.0400
8	15	1.2000	1178	94.2400
9	13	1.0400	1191	95.2800
10	7	0.5600	1198	95.8400
11	2	0.1600	1200	96.0000
12	6	0.4800	1206	96.4800
13	6	0.4800	1212	96.9600
14	4	0.3200	1216	97.2800
15	3	0.2400	1219	97.5200
16	6	0.4800	1225	98.0000
17	1	0.0800	1226	98.0800
18	3	0.2400	1229	98.3200
19	3	0.2400	1232	98.5600
21	3	0.2400	1235	98.8000
24	1	0.0800	1236	98.8800
27	2	0.1600	1238	99.0400
28	2	0.1600	1240	99.2000
30	1	0.0800	1241	99.2800
31	1	0.0800	1242	99.3600
33	1	0.0800	1243	99.4400
36	1	0.0800	1244	99.5200
39	1	0.0800	1245	99.6000
40	1	0.0800	1246	99.6800
44	1	0.0800	1247	99.7600
52	1	0.0800	1248	99.8400
68	1	0.0800	1249	99.9200
111	1	0.0800	1250	100.0000

Table 5.10: Vertices in the directed Ubuntu QA graph clustered by outdegree value

Outdegree value	Frequency	Frequency %	CumFreq	CumFreq%
0	475	38.0000	475	38.0000
1	392	31.3600	867	69.3600
2	135	10.8000	1002	80.1600
3	58	4.6400	1060	84.8000
4	37	2.9600	1097	87.7600
5	29	2.3200	1126	90.0800
6	21	1.6800	1147	91.7600
7	16	1.2800	1163	93.0400
8	14	1.1200	1177	94.1600
9	7	0.5600	1184	94.7200
10	3	0.2400	1187	94.9600
11	5	0.4000	1192	95.3600
12	2	0.1600	1194	95.5200
13	7	0.5600	1201	96.0800
14	3	0.2400	1204	96.3200
15	3	0.2400	1207	96.5600
16	5	0.4000	1212	96.9600
17	4	0.3200	1216	97.2800
18	3	0.2400	1219	97.5200
19	1	0.0800	1220	97.6000
20	2	0.1600	1222	97.7600
21	1	0.0800	1223	97.8400
22	6	0.4800	1229	98.3200
24	3	0.2400	1232	98.5600
26	1	0.0800	1233	98.6400
28	1	0.0800	1234	98.7200
29	1	0.0800	1235	98.8000
30	2	0.1600	1237	98.9600
32	2	0.1600	1239	99.1200
35	3	0.2400	1242	99.3600
46	1	0.0800	1243	99.4400
53	1	0.0800	1244	99.5200
56	1	0.0800	1245	99.6000
71	2	0.1600	1247	99.7600
79	1	0.0800	1248	99.8400
86	1	0.0800	1249	99.9200
178	1	0.0800	1250	100.0000

Around 95% of individuals receive from and send to ten or fewer individuals, in other words the majority of members active on the QA mailing lists have similar values of indegree and outdegree. However, in order to portray the number of connections a member has regardless of direction of communication the graph was symmetrized. In this symmetrized i.e. undirected graph, the degree varies between 1 and 209. Table 5.11 on page 112 details the distribution of degree values. The results suggest that the number of individuals with more connections has increased after symmetrizing the graph. One explanation for this could be that some members send messages to one set of individuals but receive messages from a different set of individuals.

Table 5.11: Vertices in the symmetrized Ubuntu QA graph clustered by degree value

Degree value	Frequency	Frequency %	CumFreq	CumFreq%
0	5	0.4000	5	0.4000
1	580	46.4000	585	46.8000
2	259	20.7200	844	67.5200
3	113	9.0400	957	76.5600
4	66	5.2800	1023	81.8400
5	41	3.2800	1064	85.1200
6	31	2.4800	1095	87.6000
7	24	1.9200	1119	89.5200
8	15	1.2000	1134	90.7200
9	11	0.8800	1145	91.6000
10	15	1.2000	1160	92.8000
11	9	0.7200	1169	93.5200
12	7	0.5600	1176	94.0800
13	8	0.6400	1184	94.7200
14	1	0.0800	1185	94.8000
15	5	0.4000	1190	95.2000
16	4	0.3200	1194	95.5200
17	5	0.4000	1199	95.9200
18	2	0.1600	1201	96.0800
19	3	0.2400	1204	96.3200
20	3	0.2400	1207	96.5600
21	2	0.1600	1209	96.7200
22	3	0.2400	1212	96.9600
23	2	0.1600	1214	97.1200

Continued on next page

Degree value	Frequency	Frequency %	CumFreq	CumFreq%
24	2	0.1600	1216	97.2800
25	5	0.4000	1221	97.6800
26	1	0.0800	1222	97.7600
27	1	0.0800	1223	97.8400
28	3	0.2400	1226	98.0800
29	1	0.0800	1227	98.1600
31	1	0.0800	1228	98.2400
32	1	0.0800	1229	98.3200
33	1	0.0800	1230	98.4000
35	2	0.1600	1232	98.5600
36	2	0.1600	1234	98.7200
37	1	0.0800	1235	98.8000
38	1	0.0800	1236	98.8800
39	1	0.0800	1237	98.9600
40	1	0.0800	1238	99.0400
42	1	0.0800	1239	99.1200
51	1	0.0800	1240	99.2000
52	1	0.0800	1241	99.2800
56	2	0.1600	1243	99.4400
60	1	0.0800	1244	99.5200
71	1	0.0800	1245	99.6000
86	1	0.0800	1246	99.6800
90	2	0.1600	1248	99.8400
103	1	0.0800	1249	99.9200
209	1	0.0800	1250	100.0000

The symmetrized graph contains 36 components of which the largest includes 1,181 vertices, the same as the directed (i.e. unsymmetrized) graph. However, if the network is not symmetrized, the directed graph contains 705 *strong* components of which the largest contains 533 vertices or about 43%. In other words, the largest component in which any two vertices are connected by a path contains approximately 43% of vertices. The highest k-core in the symmetrized graph is an 8-core containing 43 vertices or approximately 3% of all vertices. The largest k-core is a 1-core containing about 51% of the whole community.

In order to understand communication patterns within the QA team, the graph was was divided into four clusters as follows:

- Cluster 1—Members of the community who were listed as code contributor and were active on other mailing lists.

- Cluster 2—Members of the community who were listed as code contributor and were not active on other mailing lists.
- Cluster 3—Members of the community who were not listed as code contributor and were active on other mailing lists.
- Cluster 4—Members of the community who were not listed as code contributor and were not active on other mailing lists. In addition, members who could not be categorised were added to this cluster.

Table 5.12 on page 114 provides more details regarding cluster distribution within the QA team graph. It is interesting to note that, of a total 1,564 QA contributors, only one belongs to cluster 2 and only 30 to cluster 1.

Table 5.12: Clusters in the Ubuntu QA team

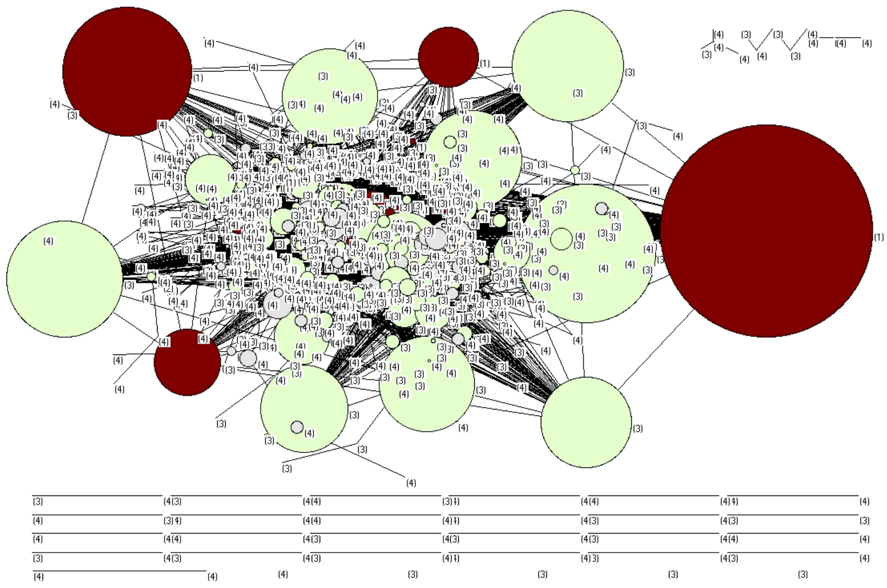
Cluster	Frequency	Frequency %	CumFreq	CumFreq%
1	30	2.4000	30	2.4000
2	1	0.0800	31	2.4800
3	499	39.9200	530	42.4000
4	720	57.6000	1250	100.0000

The betweenness centralisation value of the QA team graph is 0.1798; later in this section we will use this value to compare the network's information diffusion with temporal analysis results. Vertices with a betweenness centrality equal to 0 represent approximately 62% of the vertices. Vertices with betweenness centrality values between 0.060 and 0.181 represent 0.24% of the vertices. Figure 5.11 on page 115 shows clusters within the QA team where the vertex size shows the centrality betweenness score.

Until now we have carried out social network analysis of the Ubuntu community in general and the QA contributors in particular using aggregated data taking place over several years. In order to capture the dynamic nature of the community, we next analysed the Ubuntu subnetwork formed of members active on the QA mailing lists using six-month time frames. The subgraph associated with each time frame is further described as follows:

- SN_1—67 vertices connected by 147 arcs where the average degree is 4.38. The network betweenness centralisation score is 0.2646 where

Figure 5.11: Ubuntu QA team graph clusters—the size of each vertex is given by its betweenness centrality value.



about 3% of vertices had a betweenness centrality score higher than average (i.e. 0.092). After symmetrising the network it contained 112 edges. The average degree drops to 3.34.

- SN_2—88 vertices connected by 229 arcs where the average degree is 5.20. The network betweenness centralisation score is 0.1594 where about 3% of vertices had a betweenness centrality score higher than average (i.e. 0.056). After symmetrising the network it contained 177 edges. The average degree drops to 4.02.

- SN_3—112 vertices connected by 180 arcs where the average degree is 3.21. The network betweenness centralisation score is 0.1512 where about 5% of vertices had a betweenness centrality score higher than average (i.e. 0.054). After symmetrising the network it contained 147 edges. The average degree drops to 2.62.

- SN_4—86 vertices connected by 118 arcs where the average degree is 2.74. The network betweenness centralisation score is 0.0886 where about 7% of vertices had a betweenness centrality score higher than average (i.e. 0.031). After symmetrising the network it contained 93 edges. The average degree drops to 2.16.

- SN_5—64 vertices connected by 97 arcs where the network betweenness centralisation score is 0.0972 where about 9% of vertices had a betweenness centrality score higher than average (i.e. 0.036). After symmetrising the network it contained 81 edges. The average degree drops to 2.53.

- SN_6—86 vertices connected by 178 arcs where the average degree is 4.13. The network betweenness centralisation score is 0.1659 where about 9% of vertices had a betweenness centrality score higher than average (i.e. 0.060). After symmetrising the network it contained 144 edges. The average degree drops to 3.34.

- SN_7—107 vertices connected by 187 arcs where the average degree is 3.49. The network betweenness centralisation score is 0.2022 where about 2% of vertices had a betweenness centrality score higher than average (i.e. 0.069). After symmetrising the network it contained 154 edges. The average degree drops to 2.87.

- SN_8—102 vertices connected by 169 arcs where the average degree is 3.31. The network betweenness centralisation score is 0.1775 where about 3% of vertices had a betweenness centrality score higher than average (i.e. 0.061). After symmetrising the network it contained 141 edges. The average degree drops to 2.76.

- SN_9—120 vertices connected by 171 arcs where the average degree is 2.85. The network betweenness centralisation score is 0.1556 where about 2% of vertices had a betweenness centrality score higher than average (i.e. 0.053). After symmetrising the network it contained 140 edges. The average degree drops to 2.33.

- SN_{10}—272 vertices connected by 454 arcs where the average degree is 3.33. The network betweenness centralisation score is 0.1315 where about 1% of vertices had a betweenness centrality score higher than average (i.e. 0.045). After symmetrising the network it contained 389 edges. The average degree drops to 2.86.

- SN_{11}—172 vertices connected by 307 arcs where the average degree is 3.56. The network betweenness centralisation score is 0.137 where about 3% of vertices had a betweenness centrality score higher than average (i.e. 0.047). After symmetrising the network it contained 248 edges. The average degree drops to 2.88.

- SN_{12}—136 vertices connected by 213 arcs where the average degree is 3.13. The network betweenness centralisation score is 0.1547 where about 1% of vertices had a betweenness centrality score higher than average (i.e. 0.053). After symmetrising the network it contained 181 edges. The average degree drops to 2.66.

- SN_{13}—109 vertices connected by 177 arcs where the average degree is 3.24. The network betweenness centralisation score is 0.1822 where about 5% of vertices had a betweenness centrality score higher than average (i.e. 0.063). After symmetrising the network it contained 135 edges. The average degree drops to 2.47.

- SN_{14}—124 vertices connected by 172 arcs where the average degree is 2.77. The network betweenness centralisation score is 0.1979 where about 3% of vertices had a betweenness centrality score higher than average (i.e. 0.068). After symmetrising the network it contained 144 edges. The average degree drops to 2.32.

- SN_{15}—136 vertices connected by 361 arcs where the average degree is 5.3. The network betweenness centralisation score is 0.2896 where about 2% of vertices had a betweenness centrality score higher than average (i.e. 0.099). After symmetrising the network it contained 270 edges. The average degree drops to 3.97.

- SN_{16}—139 vertices connected by 362 arcs where the average degree is 5.2. The network betweenness centralisation score is 0.2387 where about 4% of vertices had a betweenness centrality score higher than average (i.e. 0.082). After symmetrising the network it contained 269 edges. The average degree drops to 3.87.

The Ubuntu QA team was active in all the time frames considered. The subnetworks associated with each time frame vary in size from 64 to 272 vertices[10]. The average degree after symmetrising the networks varies between 2.16 and 4.02. The findings, as predicted, seem to point to much smaller networks and lower degrees compared to the aggregated QA team graph. However, as the degree values are not exceptionally high in a given time frame compared to the aggregated graph it is safe to assume that a number of people made a sustained communication effort within the group and created connections between themselves as well as with members of the periphery or occasional contributors. These findings are consistent with the initial assumption that a small group of individuals within the QA team is

[10]As the dataset for SN_{16} was not complete, it was not taken into consideration

communicating more thus performing most of the tasks. The value of the betweenness centrality value for the aggregated graph is 0.1798 where only 0.24% of participants have a higher than average score. The subnetworks' betweenness centralisation scores vary between 0.0886 and 0.2896 where the percentage of participants having a higher than average betweenness score varied between 1% and 9%. These findings confirm that subnetworks will display higher centrality values than the aggregated form of the graph.

5.1.5 Summary of findings for this case study

Ubuntu-related communication activity rose and fell over the period studied in all communication channels and did not show consistent growth.

QA seems to be a separate layer in the Ubuntu community considering that 609 members sent more than one e-mail on the QA mailing lists and were not active on other venues.

Social network analysis suggests that in the Ubuntu community most members form a large group spanning mailing lists and issue tracker. However, most connections are created by a single unrepeated act. Furthermore, a small group of individuals is highly connected and highly engaged in communication activities. Fewer than 1% of community members have a higher than average betweenness centrality value, and those values are not particularly high, which suggests that information flow in the network is not particularly vulnerable. The QA cluster analysis showed that QA members communicate both among themselves, in a somewhat decentralised manner, as well as directly with people contributing to other project activities.

The subgraph formed of members active on the QA mailing lists displays similarities to the whole Ubuntu network in the sense that most participants are included in a large group, within which a small number of members are highly connected and engaged in communication.

5.2 Plone

5.2.1 The Plone project

Plone, an enterprise content management system written in Python, was first released in 2003. It was selected as a case study based on the fact that it has a mature community [4], its product is in use by many organisations around the world, and it has had a dedicated QA team since 2011. As opposed to Ubuntu and Mozilla, Plone has only one QA dedicated mailing list. The QA team has a webpage where one can find basic information such as activity description, communication channels and team leaders[11]. QA activities include triaging new bugs, validating submitted patches, ensuring that new releases are usable and generally help in the release process.

5.2.2 General analysis of the Plone dataset

Issue tracker data as well as mailing list data were taken into account. Data was retrieved between December 2012 and July 2013[12] and stored locally. The issue tracker data contains a total of 13,026 bugs with 55,883 associated comments, and was downloaded using a web crawler. The mailing lists contain a total of 176,144 e-mails. 28,588 e-mails were downloaded using MailingListStats[13], an open source tool[14]. However, two heavy traffic lists (Plone Users and Plone Developers) were archived in a format not compatible with MailingListStats and were downloaded using a Python script written specifically for this purpose. The two mailing lists contain 147,601 e-mails[15] of which 147,556 e-mails were processed as part of this research. In addition, a list of contributors' usernames, nicknames and (where available) real names was downloaded from Ohloh[16]. This list was used to perform data cleaning similarly to Ubuntu case.

Of all the e-mails exchanged 4,100 (2.32%) were sent by authors who had sent only one e-mail throughout the period of the study while of all the QA e-mails 38 (22.35%) were sent by authors who had sent only one e-mail. On the other hand 1,526 bugs (11.71%) were posted by members who had posted only one bug throughout the period of the study while 693 (1.24%) comments were posted by members who posted only one comment. However, it would seem that spam is a major problem on the Plone QA mailing list. For example, in the three month period starting 1 April 2013, ten of the 26 e-mails sent to the list were spam. The activity on mailing lists and the bug tracker on a yearly basis is shown in Figure 5.12 on page 120 while more detailed information is given in Table C.10 on page 274. The figure does not show a steady increase in the number of bug comments or bugs. Trends in the mailing lists activities are more difficult to discern from the figure as the numbers are much smaller than those of bug postings and comments but

[11]Plone quality assurance: http://plone.org/community/teams/qa-team

[12]See Table 3.3 on page 56 for more details.

[13]A total of 52,907 e-mails were analysed, however due to double postings, bad html and various other reasons 28,588 e-mails were stored. For a complete list of e-mails downloaded with MailingListsStats please see Table B.6 on page 222.

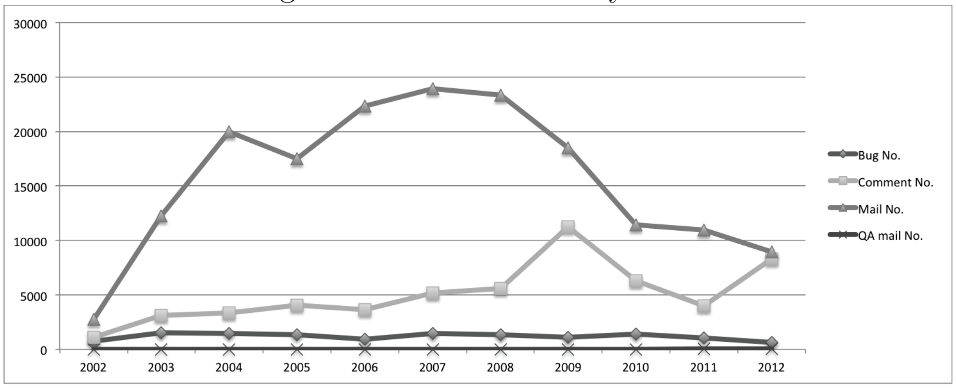
[14]MailingListStats: https://github.com/MetricsGrimoire/MailingListStats

[15]45 e-mails were not taken into consideration as they contained bad html that could not be parsed.

[16]Ohloh: https://www.ohloh.net/

they display the same irregular pattern. Furthermore, there seems to be no correlation between the number of bugs opened and the number of comments which leads to to question what causes these spikes in activity.

Figure 5.12: Plone Activity Chart



The QA mailing list activity started in 2011 and includes 57 members of whom 66.6% (38 participants) sent only one e-mail. The average number of e-mails sent is 2.98 (sd = 7.91). Only 19.3% of QA mailing lists participants sent more than three e-mails. A description of top QA e-mail participants' activity is provided in Table 5.13 on page 121[17]. Of the total number of individuals active on the QA mailing list who sent more than one e-mail, only five participants are not active on other mailing lists and do not contribute code. The rest of the participants contribute to the project by submitting code or by being active on other mailing which suggests that in the case of Plone, QA cannot be considered as a separate layer in the community.

[17]A table containing all QA mailing list participants and their activity can be found in Table C.9 on page 272

Table 5.13: Plone QA mailing list participants' activity levels on other channels

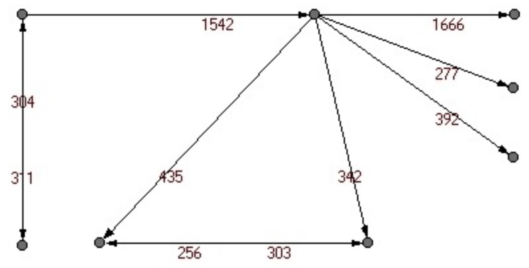
Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{13}	59	520	7	76	886	Yes
X_{57}	15	0	0	0	0	No
X_{11}	8	1,590	7	0	0	Yes
X_{47}	8	924	9	0	0	Yes
X_{6}	6	0	0	0	0	No
X_{32}	4	888	10	0	0	Yes
X_{38}	4	71	3	0	0	No
X_{14}	3	1,063	10	51	834	Yes
X_{22}	3	0	0	0	0	No
X_{23}	3	974	9	40	170	Yes
X_{39}	3	276	4	0	0	Yes
X_{10}	2	221	2	0	0	No
X_{20}	2	1	1	0	0	No
X_{24}	2	331	6	0	0	No
X_{29}	2	0	0	0	0	No
X_{31}	2	81	4	0	0	Yes
X_{41}	2	1,110	2	0	0	No
X_{45}	2	0	0	0	0	No
X_{55}	2	78	5	0	0	No

An interesting finding after analysing activity levels of QA list members is that the person listed as QA lead on the Plone webpage has only sent one e-mail using her name on the mailing list. This may suggest that QA communication is conducted on other channels as well and should be further investigated in future studies.

5.2.3 Social network analysis of the whole dataset

The next stage was to carry out a social network analysis of the Plone community. The dataset was exported to a Pajek-readable file that contained community members, the links between them and the periods in which they were active in the project. Members who did not form any connections with other members were eliminated from the dataset. As in the previous case studies, after importing this data into Pajek all loops were eliminated. The resulting network contains 9,480 vertices connected by a total of 61,686 arcs of which 18,514 have a value that is greater than 1 and 43,172 have a value equal to 1. This means that 43,172 connections (69.98%) are created by only one interaction (message). One could draw the conclusion that these members are occasional contributors or part of the periphery. The average degree is 13.01, which means that on average a person interacts with approximately 13 other people. The network's density is 0.00068. The "heaviest" ten arcs connected eight people and had values between 256 and 1666; see Figure 5.13 on page 122.

Figure 5.13: The heaviest ten arcs in the Plone graph connecting ten members—numbers represent e-mails and bug comments in each direction



The network contains 99 components of which the biggest component contains 9,362 vertices which or 98.7553% of the community. The rest of the components contain between one and two vertices.

To get the precise values of vertex degrees within the Plone community, the directed network was transformed into an undirected network by transforming all arcs into edges. The resulting network had 46,924 edges of which 19,474 have a value greater than 1 and 27,450 have a value equal to 1. The network's density is 0.00104. The average degree drops to 9.89 compared to 13.01 for the directed network. About 15% of individuals had connections with more than ten (i.e. the average degree) other individuals. Looking at the vertices with degree value from 0 to 10, we find that they account for approximately 85% of the community. The remaining approximately 15% of vertices with the highest degrees are connected to—that is communicate directly with— between 11 and 2,134 other members. Details of the 85% of vertices in the Plone graph with degree value between 0 and 10 are given in Table 5.14 on page 123. The largest degree cluster is formed by vertices with a degree value of 1 and represents approximately one third of the community.

Both the indegree and outdegree values were computed. In the directed Plone graph vertices with values between 0 and 10 make up about 88% of the community—see Table 5.15 on page 123. The remaining vertices have indegree values of between 11 and 1,102.

Table 5.14: Vertices in the symmetrized Plone graph clustered by degree value

Degree value	Frequency	Frequency %	CumFreq	CumFreq%
0	78	0.8228	78	0.8228
1	2,954	31.1603	3,032	31.9831
2	1,830	19.3038	4,862	51.2869
3	1,000	10.5485	5,862	61.8354
4	624	6.5823	6,486	68.4177
5	404	4.2616	6,890	72.6793
6	341	3.5970	7,231	76.2764
7	276	2.9114	7,507	79.1878
8	209	2.2046	7,716	81.3924
9	176	1.8565	7,892	83.2489
10	127	1.3397	8,019	84.5886

Table 5.15: Indegree clusters in the Plone graph

Indegree value	Frequency	Frequency %	CumFreq	CumFreq%
0	798	8.4177	798	8.4177
1	3,175	33.4916	3,973	41.9093
2	1,572	16.5823	5,545	58.4916
3	903	9.5253	6,448	68.0169
4	558	5.8861	7,006	73.9030
5	407	4.2932	7,413	78.1962
6	308	3.2489	7,721	81.4451
7	224	2.3629	7,945	83.8080
8	168	1.7722	8,113	85.5802
9	149	1.5717	8,262	87.1519
10	111	1.1709	8,373	88.3228

In the same directed Plone graph approximately 9% of vertices have outdegree values between 11 and 1,990; see Table 5.16 on page 124.

Table 5.16: Outdegree clusters in the Plone graph

Outdegree value	Frequency	Frequency %	CumFreq	CumFreq%
0	3329	35.1160	3329	35.1160
1	2605	27.4789	5934	62.5949
2	963	10.1582	6897	72.7532
3	535	5.6435	7432	78.3966
4	358	3.7764	7790	82.1730
5	245	2.5844	8035	84.7574
6	168	1.7722	8203	86.5295
7	109	1.1498	8312	87.6793
8	111	1.1709	8423	88.8502
9	104	1.0970	8527	89.9473
10	65	0.6857	8592	90.6329

The Plone directed graph contains 4,236 strong components. The largest strong component contains 5,228 vertices or about 55% of the whole network. The symmetrized i.e. undirected network contains 99 components where the largest components contains 9,362 vertices or approximately 99% of the whole community. In the same symmetrized Plone graph k-core values range between 0 and 37, in other words the most connected group of members contains individuals who interacted with a minimum of 37 other people. This group contains 104 community members. Table 5.17 on page 125 gives details of the members who have k-core values between 0 and 10, who make up approximately 87% of the graph.

Betweenness centrality values were then computed. The Plone directed graph vertices have betweenness centralities of between 0 and 0.108 where the network's betweenness centralisation value is equal to 0.1076. Table 5.18 on page 125 details the distribution of betweenness centrality vectors within the network. These findings suggest that approximately 58% of the Plone community do not represent a "step" in the information flow. Furthermore, fewer than 1% of vertices have a higher than average betweenness score and the maximum betweenness centrality is 0.108, which is higher than the maximum value for the Mozilla community which was 0.063.

To get a better understanding of communication patterns between QA team members and the rest of the community the graph was divided into

Table 5.17: K-core clusters in the Plone graph

K-core value	Frequency	Frequency %	CumFreq	CumFreq%
0	78	0.8228	78	0.8228
1	3,028	31.9409	3,106	32.7637
2	1,875	19.7785	4,981	52.5422
3	1,023	10.7911	6,004	63.3333
4	626	6.6034	6,630	69.9367
5	430	4.5359	7,060	74.4726
6	352	3.7131	7,412	78.1857
7	305	3.2173	7,717	81.4030
8	211	2.2257	7,928	83.6287
9	176	1.8565	8,104	85.4852
10	152	1.6034	8,256	87.0886

Table 5.18: Betweenness centrality vectors in in the Plone graph

Value intervals	Frequency	Frequency %	CumFreq	CumFreq%
0.000	5,506	58.0802	5,506	58.0802
0.000 – 0.036	3,968	41.8565	9,474	99.9367
0.036 – 0.072	4	0.0422	9,478	99.9789
0.072 – 0.108	2	0.0211	9,480	100.0000

four clusters as follows:

1. Cluster 1—Community participants who contribute code and are members of the QA mailing list.
2. Cluster 2—Community participants who contribute code but are not members of the QA mailing list.
3. Cluster 3—Community participants who do not contribute code but are members of the QA mailing list.
4. Cluster 4—Community participants who do not contribute code and are not members of the QA mailing list. Members who could not be categorised were also added to this cluster.

Table 5.19: Plone cluster frequency

Cluster	Frequency	Frequency %	CumFreq	CumFreq%
1	11	0.1160	11	0.1160
2	267	2.8165	278	2.9325
3	20	0.2110	298	3.1435
4	9,182	96.8565	9,480	100.0000

To categorise participants into clusters a list containing Plone code contributors was retrieved from Ohloh. A Python script was then written in order to compare all community members whose names contained more than four characters to the code contributor list and categorise them into clusters based on participation on the QA mailing list. Table 5.19 on page 126 provides more details regarding cluster distribution within the community.

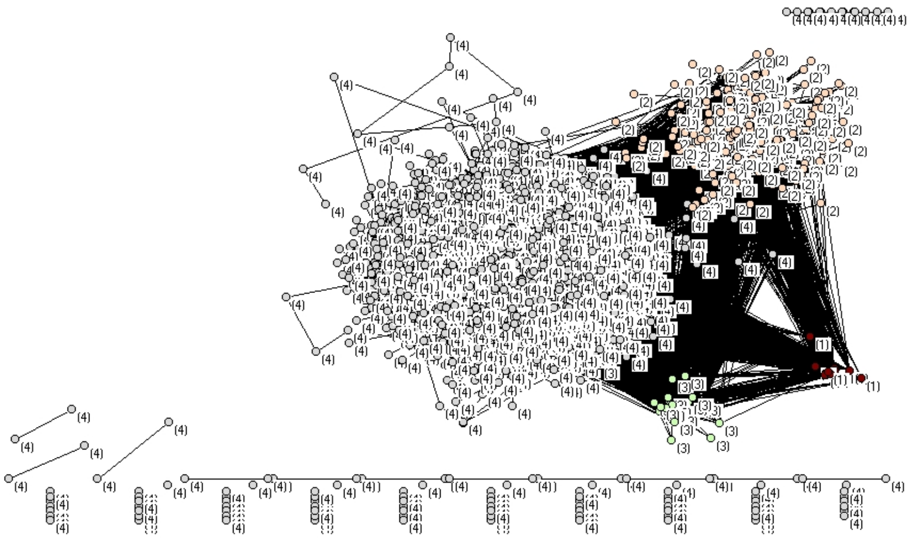
It will be observed that the cluster containing members who are neither code contributors nor QA contributors is the largest, representing approximately 97% of the whole community. On the other hand people performing QA tasks appear to add up to only about 0.3% of the whole community. Figure 5.14 on page 128 represents an aggregated graph of the Plone network where each cluster is represented by a different colour.

If each of the clusters is shrunk to one vertex then the resulting network has four vertices and is a complete network with a density value of 1. The graph contains a total of four loops and twelve lines and is depicted in Figure

Table 5.20: Shrunk Plone network communications

Rank	Line	Value
1	1.1	56,560
2	3.1	22,997
3	1.3	15,859
4	2.1	7,236
5	1.2	5,959
6	3.3	5,798
7	2.3	4,926
8	4.1	3,459
9	3.2	2,781
10	1.4	2,518
11	2.2	1,064
12	4.3	962
13	3.4	942
14	4.2	457
15	2.4	435
16	4.4	171

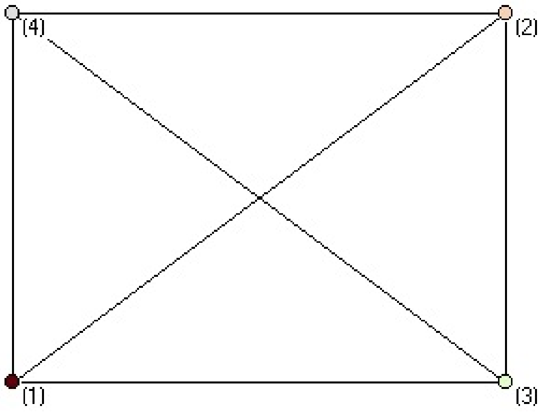
Figure 5.14: Plone community graph



5.15 on page 128. The values of the lines of the graph are given in Table 5.20 on page 127.

If each of the clusters is shrunk to one vertex then the resulting network has four vertices and is a complete network with a density value of 1. The graph contains a total of four loops and twelve lines and is depicted in Figure 5.15 on page 128. The values of the lines of the graph are given in Table 5.20 on page 127.

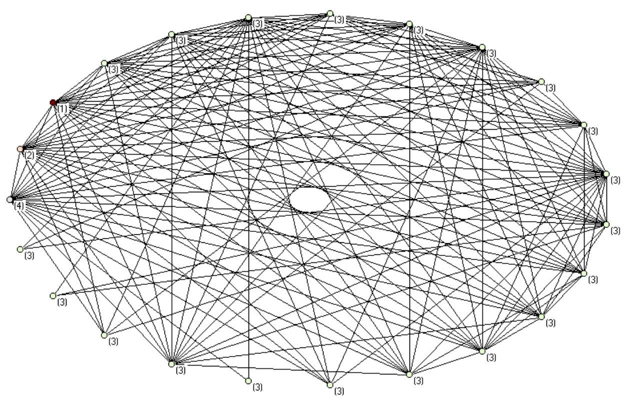
Figure 5.15: Plone reduced graph—each cluster was reduced to one vertex in order to illustrate connections between various groups within the community



The next step was to reduce all clusters to one vertex except for cluster

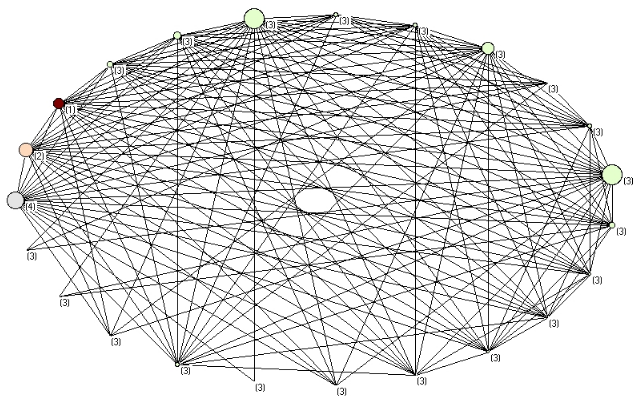
3, the QA-only cluster. The resulting network contains 23 vertices of which three represent the other clusters. The graph contains a total of 273 arcs of which 205 have a value greater than 1 and 68 have a value equal to 1. The network's density is 0.516 while the average degree is 23.73. The network is depicted in Figure 5.16 on page 129. This figure shows dense communication both between members of cluster 3 as well as with members of other clusters (but less dense than was the case for Ubuntu). In Figure 5.17 on page 130 the variety of betweenness centralities produces a variety of vertex sizes while a dense communication can be observed suggesting that members of cluster 3 communicate both within the cluster as well as directly with members of ofther clusters. If loops are eliminated the density value is 0.5335 and the degree drops to 23.47.

Figure 5.16: Plone graph graph shrunk except for those who are active on the QA mailing lists but do not submit code (cluster 3)—depicts relations between members of cluster 3 as well as between these members and other clusters; the red, orange and grey vertices represent shrunk clusters.



The same operations were applied to the Plone graph in order to reduce all clusters except for cluster 1, i.e. those who both submit code and are active on the QA mailing list. The resulting network contains 14 vertices of which three represent the other clusters. The graph contains a total of 130 arcs of which 103 have a value greater than 1 and 27 have a value equal to 1. The network's density is 0.6632 while the average degree is 18.57. The network is depicted in Figure 5.18 on page 131. This figure shows communication both between members of cluster 1 as well as with members of other clusters (but less dense than was the case for Ubuntu). In Figure 5.19 on page 133 the variety of betweenness centralities produces a variety of vertex sizes while a dense communication can be observed suggesting that members of cluster

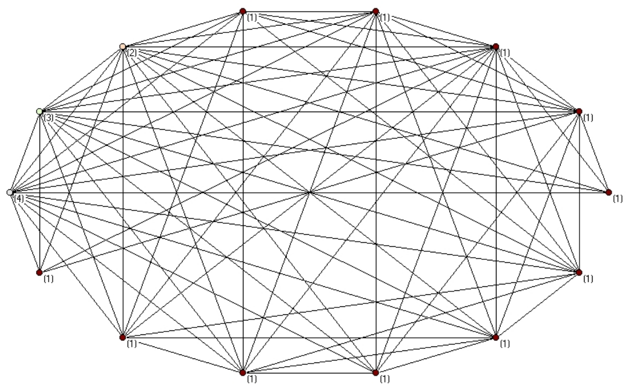
Figure 5.17: Plone graph shrunk except for those who are active on the QA mailing lists but do not submit code (cluster 3)—the size of each vertex is given by its betweenness centrality value; the red, orange and grey vertices represent shrunk clusters.



1 communicate both within the cluster as well as directly with members of ofther clusters. If loops are eliminated the density value is 0.6978 and the degree drops to 18.14.

In order to visualise the communication patterns between the QA team as a whole and the rest of the community, cluster 1 was merged with cluster 3 and clusters 2 and 4 were reduced. The resulting network contains 33 vertices of which two represent the other clusters (i.e. 2 and 4). The graph contains a total of 516 arcs of which 371 have a value greater than 1 and 145 arcs have a value equal to 1. The network's density is 0.4738 while the average degree is 31.27. The resulting network is depicted in Figure 5.20 on page 134. This figure shows dense communication both between members of cluster 1 as well as with members of other clusters (but less dense than was the case for Ubuntu). In Figure 5.19 on page 133 the variety of betweenness centralities produces a variety of vertex sizes while a dense communication can be observed suggesting that members of cluster 1 communicate both within the cluster as well as directly with members of ofther clusters. If loops are eliminated the density value is 0.4867 and the degree drops to 31.15.

Figure 5.18: Plone graph shrunk except for those who both code and are active on the QA mailing lists (cluster 1)—depicts relations between members of cluster 1 as well as between these members and other clusters; the green, orange and grey vertices represent shrunk clusters.



5.2.4 Social network analysis of QA-specific communications

So far we have analysed communications carried out on both issue tracker and mailing lists. Now, however, we want to focus on QA activities within the Plone community, so we restrict our dataset to QA-related mailing lists. After eliminating loops the network contains 31 vertices connected by 49 arcs of which 40 have a value equal to 1 and 9 have a value greater than 1 (the highest value an arc had is 5). The average degree is 3.1612 while the network's density is 0.0526. The graph is shown in Figure 5.22 on page 136.

The QA team graph contains only one component that contains all 31 vertices. In the directed QA graph the indegree values vary between 0 and 24. Table 5.21 on page 132 details the distribution of indegree values.

In the directed QA graph the outdegree values vary between 0 and 8. Table 5.22 on page 132 details the distribution of outdegree values.

With the notable exception of one individual who received messages from 24 other participants, around 97% of individuals received messages from four or less other individuals while 100% of individuals sent messages to eight or fewer other individuals. However, in order to portray the number of connections a member has regardless of direction of communication the graph was symmetrized. In this symmetrized graph i.e. undirected, the degree varies between 1 and 24. Table 5.23 on page 133 details the distribution of degree values. The result suggests no significant increase in the number

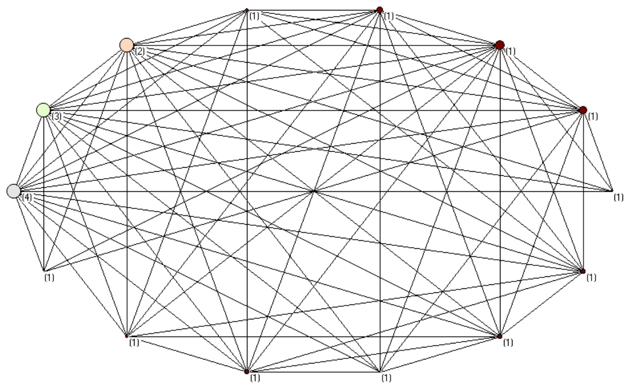
Table 5.21: Vertices in the directed Plone QA graph clustered by indegree value

Cluster	Frequency	Frequency %	CumFreq	CumFreq%
0	10	32.2581	10	32.2581
1	17	54.8387	27	87.0968
2	2	6.4516	29	93.5484
4	1	3.2258	30	96.7742
24	1	3.2258	31	100.0000

Table 5.22: Vertices in the directed Plone QA graph clustered by outdegree value

Cluster	Frequency	Frequency %	CumFreq	CumFreq%
0	1	3.2258	1	3.2258
1	21	67.7419	22	70.9677
2	6	19.3548	28	90.3226
3	1	3.2258	29	93.5484
5	1	3.2258	30	96.7742
8	1	3.2258	31	100.0000

Figure 5.19: Plone graph shrunk except for those who both code and are active on the QA mailing lists (cluster 1)—the size of each vertex is given by its betweenness centrality value; the green, orange and grey vertices represent shrunk clusters.



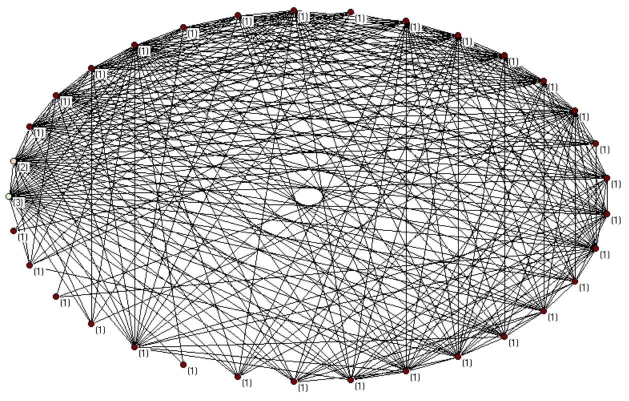
connections an individual has with other individuals considering that about 97% of individuals have connections with six or fewer other individuals (only one individual has connections with 24 other individuals).

Table 5.23: Vertices in the symmetrized Plone QA graph clustered by degree value

Cluster	Frequency	Frequency %	CumFreq	CumFreq%
1	16	51.6129	16	51.6129
2	11	35.4839	27	87.0968
3	2	6.4516	29	93.5484
6	1	3.2258	30	96.7742
24	1	3.2258	31	100.0000

The symmetrized graph contains one component that contains all 31 vertices. However, if the network is not symmetrized, the directed graph contains 17 *strong* components of which the largest contains 13 vertices or about 42% of all vertices. In other words, the largest component in which any two vertices are connected by a path contains 13 vertices. The highest k-core in the symmetrized graph is a 2-core containing 11 vertices or approximately 35% of all vertices. The largest k-core is a 1-core containing about 65% of

Figure 5.20: Plone graph graph shrunk except for cluster 1 and 3—depicts relations between members of cluster 1 and 3 as well as between these members and other clusters; the orange and green vertices represent reduced clusters.



the whole community.

In order to understand communication patterns within the QA team, the graph was was divided into four clusters as follows:

- Cluster 1—Members of the community who were listed as code contributor and were active on other mailing lists.
- Cluster 2—Members of the community who were listed as code contributor and were not active on other mailing lists.
- Cluster 3—Members of the community who were not listed as code contributor and were active on other mailing lists.
- Cluster 4—Members of the community who were not listed as code contributor and were not active on other mailing lists. In addition, members who could not be categorised were added to this cluster.

Table 5.24 on page 135 provides more details regarding cluster distribution within the QA team graph. It is interesting to note that, of a total of 31 QA contributors, none belong to cluster 2 and only one to cluster 4.

The betweenness centralisation value of the QA team graph is 0.3413. Vertices with a betweenness centrality equal to 0 represent approximately 71% of the vertices. Vertices with betweenness centrality values between 0.116 and 0.348 represent 3.22% of the vertices. Figure 5.23 on page 137 shows clusters within the QA team where the vertex shows the centrality betweenness score.

Figure 5.21: Plone graph shrunk except for cluster 1 and 3—the size of each vertex is given by its betweenness centrality value; the orange and green vertices represent reduced clusters.

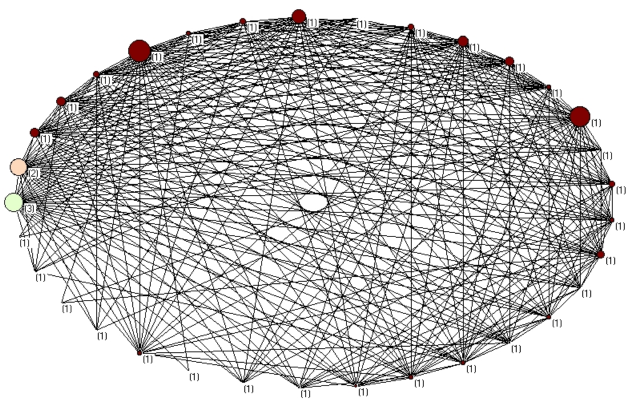


Table 5.24: Clusters in the Plone QA team

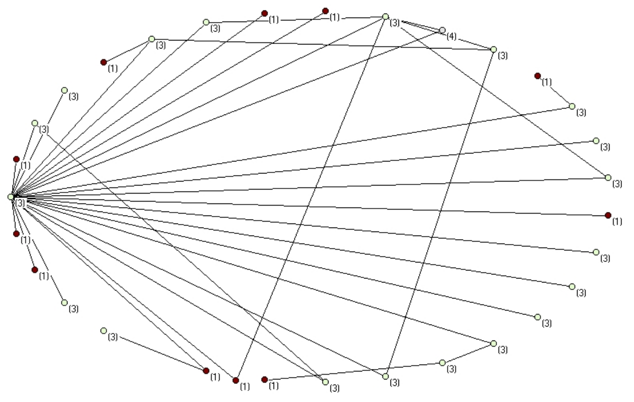
Cluster	Frequency	Frequency %	CumFreq	CumFreq%
1	11	35.4839	11	35.4839
3	19	61.2903	30	96.7742
4	1	3.2258	31	100.0000

Until now we have carried out social network analysis of the Ubuntu community in general and the QA contributors in particular using aggregated data taking place over several years. In order to capture the dynamic nature of the community, we next analysed the Ubuntu subnetwork formed of members active on the QA mailing lists using six-month time frames. The subgraph associated with each time frame is further described as follows:

- SN_1—16 vertices connected by 21 arcs where the average degree is 2.62. The network betweenness centralisation score is 0.2415 where about 12% of vertices had a betweenness centrality score higher than average (i.e. 0.086). After symmetrising the network it contained 17 edges. The average degree drops to 2.12.

- SN_3—13 vertices connected by 17 arcs where the average degree is 2.61. The network betweenness centralisation score is 0.2007 where

Figure 5.22: Plone QA team graph clusters—each colour in the graph represents a cluster and is labeled as such.



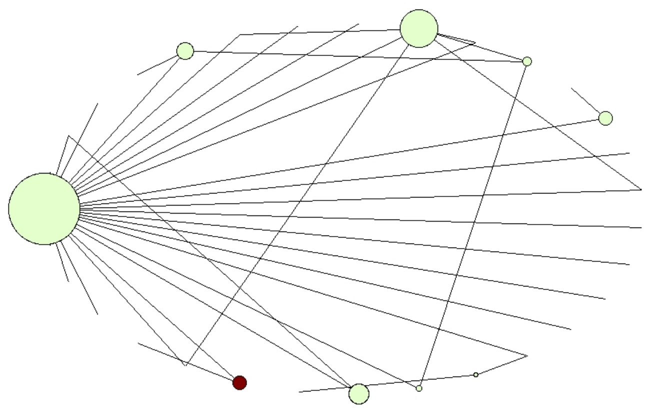
about 23% of vertices had a betweenness centrality score higher than average (i.e. 0.073). After symmetrising the network it contained 14 edges. The average degree drops to 2.15.

- SN_5—12 vertices connected by 18 arcs where the average degree is 3. The network betweenness centralisation score is 0.5223 where about 8% of vertices had a betweenness centrality score higher than average (i.e. 0.179). After symmetrising the network it contained 12 edges. The average degree drops to 2.

The Plone QA team was active in only three of the five time frames considered. The subnetworks associated with each time frame varied in size from 12 to 16 vertices[18]. The average degree after symmetrising the networks varies between 2 and 2.15. The findings, as predicted, seem to point to much smaller networks and lower degrees compared to the aggregated QA team graph. However, as the degree values are not exceptionally high in a given time frame compared to the aggregated graph it is safe to assume that a number of people made a sustained communication effort within the group and created connections between themselves as well as with members of the periphery or occasional contributors. These findings are consistent with the initial assumption that a small group of individuals within the QA team is communicating more thus performing most of the tasks. The value of the betweenness centrality value for the aggregated graph is 0.3413 where only

[18]As the dataset is too small SN_5 was also included in this analysis.

Figure 5.23: Plone QA team graph clusters—the size of each vertex is given by its betweenness centrality value.



3.22% of participants have a higher than average score. The subnetworks' betweenness centralisation scores vary between 0.2007 and 0.5223 where the percentage of participants having a higher than average betweenness score varied between 8% and 23%. These findings confirm that subnetworks will display higher centrality values than the aggregated form of the graph.

5.2.5 Summary of findings for this case study

Plone-related communication activity did not show constant growth but displayed a more irregular pattern. QA does not seem to be a separate layer in the Plone community considering that only five participants active on the QA mailing did not submit code and were not active on other venues. These findings suggest that in the case of Plone, community members have multiple roles.

Social network analysis suggests that in the Plone community most members form a large group spanning the mailing lists and issue tracker. However, most connections are created by a single unrepeated act. Furthermore, a small group of individuals is highly connected and highly engaged in communication activities. Fewer than 1% of community members have a higher than average betweenness centrality value, and those values are not particularly high, which suggests that information flow in the network is not particularly vulnerable. The QA cluster analysis showed that QA members communicate both among themselves in a somewhat decentralised manner as well as directly with people contributing to other project activities. The subgraph formed of members active on the QA mailing lists displays similarities to the whole Plone network in the sense that all participants are included in a large group, within which a small number of members are highly connected and engaged in communication.

5.3 KDE

5.3.1 The KDE project

KDE "is an international team co-operating on development and distribution of Free, Open Source Software"[19]. In other words, KDE is a community that produces a variety of FLOSS products among which the most well known is "Plasma Desktop"—a desktop environment. The KDE community was founded in October 1996 by Matthias Ettrich[20] and is another example of a successful FLOSS project. In the early 2000s as a part of the KDE community, a team of people were in charge of the "task of user case studies, writing articles, documentation, creating missing artwork for consistency, and other miscellanea"[21]. The team's activity was interrupted, but it then revived in 2012 as an official QA team. In addition to the QA team, the KDE community also has a Bugsquad team[22]. Both teams use dedicated mailing lists, webpages and wikis through which they offer information to potential contributors and/or carry out discussions. The KDE community was chosen as a case study due to its maturity and size as well as its dedicated QA team.

5.3.2 General analysis of the KDE dataset

Issue tracker data as well as mailing list data were taken into account. Data was retrieved between April and June 2013 and stored locally. The issue tracker data contained a total of 312,847 bugs with 1,364,871 associated comments, and was downloaded using a web crawler. The mailing lists contained a total of 529,488 e-mails that were downloaded using MailingListStats, an open source tool[23]. Considering the definition of QA activities given for the scope of this research, both QA mailing list and Bugsquad mailing lists were considered as QA related; together they contained 954 e-mails. In addition, a list of contributors' usernames, nicknames and (where available) real names

[19]KDE: http://www.kde.org/

[20]KDE history: http://www.kde.org/community/history/

[21]KDE quality: http://community.kde.org/Getinvolved/Quality, http://techbase.kde.org/Contribute/Quality_Team

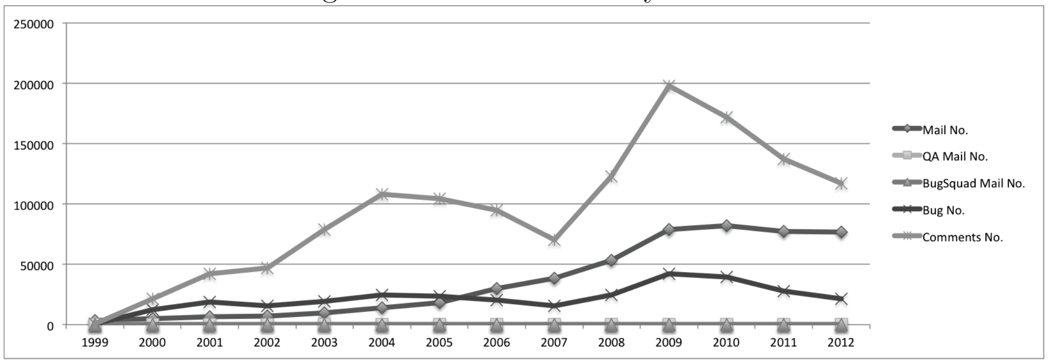
[22]KDE Bugsquad: http://techbase.kde.org/Contribute/Bugsquad

[23]MailingListStats: https://github.com/MetricsGrimoire/MailingListStats

was downloaded from Ohloh[24]. This list was used to perform data cleaning similarly to the previous cases.

Of all the e-mails exchanged 15,218 (about 3%) were sent by authors who had sent only one e-mail throughout the period of the study while of all the QA e-mails 45 (about 5%) were sent by authors who had sent only one e-mail. On the other hand 39,826 bugs (about 13%) were posted by members who had posted only one bug throughout the period of the study while 34,247 (about 3%) comments were posted by members who posted only one comment. The activity on mailing lists and the bug tracker on a yearly basis is shown in Figure 5.24 on page 139 while more detailed information is given in Table C.11 on page 279. The figure does not show a steady increase in the number of bug comments, bugs or e-mails; instead they display a somewhat irregular pattern. Bug comment activity seems to be correlated to bug activity as they follow the same general trajectory. Activity conducted on the mailing lists seems to be unrelated to issue tracker activity. The recent start of QA activity makes it difficult to observe trends in activity levels on the QA mailing lists over time.

Figure 5.24: KDE Activity Chart



The QA mailing list activity started in 2012 and includes 121 members of whom about 45 (37%) sent only one e-mail. The average number of e-mails sent is 7.88 (sd = 16.39). Only about 20% of QA mailing list participants sent more than eight e-mails. A detailed description of top QA e-mail participants' activity is given in Table 5.25[25] on page 140.

[24]Ohloh: https://www.ohloh.net/

[25]Table C.11 on page 275 contains all QA mailing list participants and their activity

Table 5.25: KDE QA mailing list participants' activity levels on other channels

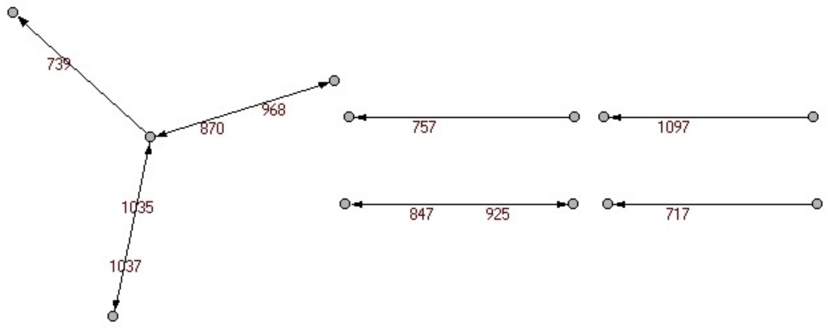
Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{21}	115	509	13	0	0	No
X_{100}	76	2491	28	119	23887	No
X_{2}	65	2267	17	329	12642	Yes
X_{32}	64	844	16	141	1316	No
X_{26}	45	3610	13	251	1863	Yes
X_{64}	42	543	26	121	3224	Yes
X_{89}	36	657	13	0	0	No
X_{9}	35	126	9	30	1676	No
X_{43}	31	806	14	216	2109	Yes
X_{121}	30	567	25	47	869	Yes
X_{45}	28	6248	22	78	34075	No
X_{105}	25	57	5	23	169	No
X_{71}	22	4718	51	342	5525	Yes
X_{90}	19	58	8	90	614	No
X_{40}	18	402	21	0	0	No
X_{8}	17	1071	15	956	3391	No
X_{35}	12	190	13	36	6173	Yes
X_{29}	10	310	20	73	6206	Yes
X_{72}	10	27	3	0	0	No
X_{25}	9	143	9	12	30	No

Of the total number of individuals active on the QA mailing list only eight (7%) participants were not active on other mailing lists , and of these one participant was listed as a code contributor. Of the seven members who were active exclusively active on the QA mailing list, three sent only one e-mail; the other four members sent between two and four e-mails. These findings suggest that in the KDE case, QA cannot be considered as a separate layer in the community.

5.3.3 Social network analysis of the whole dataset

The next stage was to carry out a social network analysis of the KDE community. The dataset was exported to a Pajek-readable file that contained community members, the links between them and the periods in which they were active in the project. Members who did not form any connections with other members were eliminated from the dataset. As in the previous case studies, after importing this data into Pajek all loops were eliminated. The resulting network contains 89,317 vertices connected by a total of 506,090 arcs of which 149,909 have a value that is greater than 1 and 356,181 have a value equal to 1. This means that 356,181 connections (about 70%) are created by only one interaction (message). One could draw the conclusion that these members are occasional contributors or part of the periphery. The average degree is 11.33, which means that on average a person interacts with approximately eleven other people. The network's density is 0.000063. The "heaviest" ten arcs in the network connected twelve people and had values between 717 and 1097; each of these represents one person sending many hundreds of communications to another person; see Figure 5.25 on page 141.

Figure 5.25: The heaviest ten arcs in the KDE graph connecting ten members—numbers represent e-mails and bug comments in each direction



The network contains 716 components of which the biggest component contains 88,502 vertices or about 99% of the community. The rest of the components contain between one and three vertices.

To get the precise values of vertex degrees within the KDE community, the directed network was transformed into an undirected network by transforming all arcs into edges. The resulting network has 396,437 edges of which 151,967 have a value greater than 1 and 244,470 have a value equal to 1. The network's density is 0.000099. The average degree drops to 8.87 compared to 11.33 for the directed network. About 11% of individuals had connections with more than nine other individuals (i.e. the average degree). Looking at the vertices with degree value from 0 to 10, we find that they account for approximately 90% of the community. The remaining approximately 10% of vertices with the highest degrees are connected to—that is, communicate directly with—between 11 and 9,972 other members. Details of the 90% of vertices in the KDE graph with degree value between 0 and 10 are given in Table 5.26 on page 142. The largest degree cluster is formed by vertices with a degree value of 1 and represents over one third (i.e. 43%) of the community.

Table 5.26: Vertices in the symmetrized KDE graph clustered by degree value

Degree value	Frequency	Frequency %	CumFreq	CumFreq%
0	625	0.6998	625	0.6998
1	38,747	43.3814	39,372	44.0812
2	18,390	20.5896	57,762	64.6708
3	7,773	8.7027	65,535	73.3735
4	4,624	5.1771	70,159	78.5506
5	3,108	3.4797	73,267	82.0303
6	2,198	2.4609	75,465	84.4912
7	1,621	1.8149	77,086	86.3061
8	1,335	1.4947	78,421	87.8008
9	1,008	1.1286	79,429	88.9293
10	843	0.9438	80,272	89.8731

Both the indegree and outdegree values were computed. In the directed KDE graph vertices with values between 0 and 10 make up about 92% of the community—see Table 5.27 on page 143. The remaining vertices have indegree values of between 11 and 3,679.

In the same directed KDE graph approximately 5% of vertices have outdegree values between 11 and 9,430; see Table 5.28 on page 143.

Table 5.27: Vertices in the symmetrized KDE graph clustered by indegree value

Indegree value	Frequency	Frequency %	CumFreq	CumFreq%
0	3,851	4.3116	3,851	4.3116
1	44,936	50.3107	48,787	54.6223
2	14,549	16.2892	63,336	70.9115
3	6,754	7.5618	70,090	78.4733
4	3,948	4.4202	74,038	82.8935
5	2,595	2.9054	76,633	85.7989
6	1,824	2.0422	78,457	87.8411
7	1,362	1.5249	79,819	89.3660
8	1,062	1.1890	80,881	90.5550
9	829	0.9282	81,710	91.4831
10	719	0.8050	82,429	92.2881

Table 5.28: Vertices in the symmetrized KDE graph clustered by outdegree value

Outdegree value	Frequency	Frequency %	CumFreq	CumFreq%
0	39,124	43.8035	39,124	43.8035
1	25,411	28.4504	64,535	72.2539
2	8,369	9.3700	72,904	81.6239
3	3,913	4.3810	76,817	86.0049
4	2,372	2.6557	79,189	88.6606
5	1,545	1.7298	80,734	90.3904
6	1,097	1.2282	81,831	91.6186
7	870	0.9741	82,701	92.5927
8	672	0.7524	83,373	93.3451
9	583	0.6527	83,956	93.9978
10	470	0.5262	84,426	94.5240

The KDE directed graph contains 43,388 strong components. The largest strong component contains 45,859 vertices which amounts to approximately 51% of the whole network. The symmetrized i.e. undirected network contains 716 components where the largest component contains 88,502 vertices or about 99% of the whole community. In the same symmetrized KDE graph, k-core values range between 0 and 81, in other words the most connected group of members contains individuals who interacted with a minimum of 81 other people. This group contains 274 community members. Table 5.29 on page 144 gives details of the members who have k-core values between 0 and 10, who make up approximately 91% of the graph.

Table 5.29: Vertices in the symmetrized KDE graph clustered by k-core value

K-core value	Frequency	Frequency %	CumFreq	CumFreq%
0	625	0.6998	625	0.6998
1	39,378	44.0879	40,003	44.7877
2	18,760	21.0038	58,763	65.7915
3	7,887	8.8303	66,650	74.6219
4	4,728	5.2935	71,378	79.9154
5	3,113	3.4853	74,491	83.4007
6	2,209	2.4732	76,700	85.8739
7	1,651	1.8485	78,351	87.7224
8	1,287	1.4409	79,638	89.1633
9	933	1.0446	80,571	90.2079
10	834	0.9338	81,405	91.1417

Betweenness centrality values were then computed. The KDE directed graph vertices have betweenness centralities of between 0 and 0.053 where network's betweenness centralisation value is equal to 0.0526. Table 5.30 on page 145 details the distribution of betweenness centrality vectors within the network. Findings suggest that approximately 64% of the KDE community do not represent a "step" in the information flow. Furthermore, much fewer than 1% (i.e. 0.01%) of vertices have a higher than average betweenness score and the maximum betweenness centrality is 0.053, which is lower than the maximum value for the Mozilla community which was 0.063.

To get a better understanding of communication patterns between QA team members and the rest of the community the graph was divided into four clusters as follows:

Table 5.30: Betweenness centrality clusters in the KDE community

Value intervals	Frequency	Frequency %	CumFreq	CumFreq%
0.000	57,137	63.9710	57,137	63.9710
0.000–0.018	32,171	36.0189	89,308	99.9899
0.018–0.035	7	0.0078	89,315	99.9978
0.035–0.053	2	0.0022	89,317	100.0000

1. Cluster 1—Community participants who contribute code and are members of the QA mailing lists.
2. Cluster 2—Community participants who contribute code but are not members of the QA mailing lists.
3. Cluster 3—Community participants who do not contribute code but are members of the QA mailing lists.
4. Cluster 4—Community participants who do not contribute code and are not members of the QA mailing lists. Members who could not be categorised were also added to this cluster.

Table 5.31: Clusters in the KDE community

Cluster	Frequency	Frequency %	CumFreq	CumFreq%
1	53	0.0593	53	0.0593
2	1,412	1.5809	1,465	1.6402
3	64	0.0717	1,529	1.7119
4	87,788	98.2881	89,317	100.0000

To categorise participants into clusters, a list containing KDE code contributors was retrieved from Ohloh. A Python script was then written in order to compare all community members whose names contained more than four characters to the code contributor list and categorise them into clusters based on participation on the QA mailing list. Table 5.31 on page 145 provides more details regarding cluster distribution within the community. It will be observed that the cluster containing members who are neither code contributors nor QA contributors is the largest, representing approximately 98% of the whole community. On the other hand people performing QA tasks appear to add up to only about 0.131% of the community.

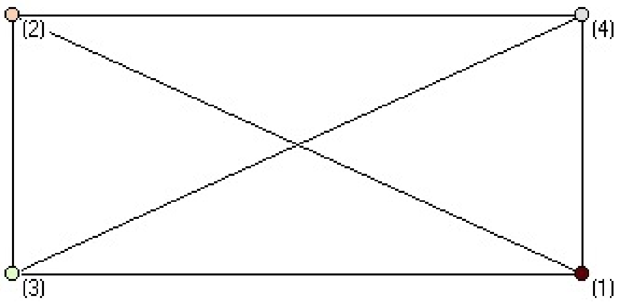
If each of these clusters is shrunk to one vertex then the resulting network has four vertices and is a complete network with a density value of 1. The graph contains a total of four loops and twelve lines and is depicted in Figure 5.26 on page 147. The values of the lines of the graph are described in Table 5.32 on page 146.

Table 5.32: Shrunk KDE network communication

Rank	Line	Value
1	2.2	271,866
2	1.2	235,070
3	2.1	120,460
4	3.2	109,354
5	4.2	77,192
6	1.1	69,536
7	2.3	44,258
8	2.4	30,108
9	3.1	26,433
10	1.3	21,115
11	4.1	16,185
12	1.4	14,645
13	3.4	10,766
14	3.3	9,906
15	4.3	9,369
16	4.4	5,237

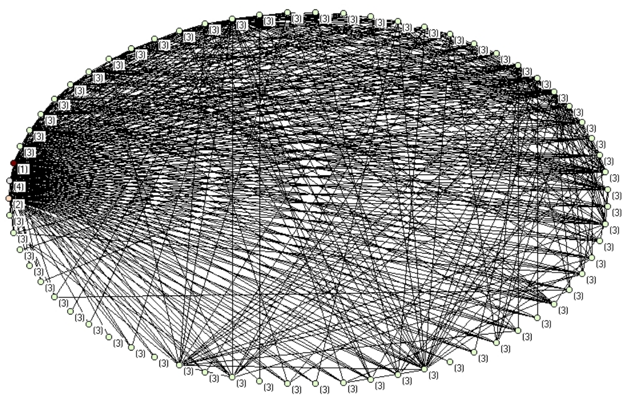
The next step was to reduce all clusters to one vertex except for cluster 3, the QA-only cluster. The resulting network contains 67 vertices of which three represent the other clusters. The graph contains a total of 855 arcs of which 647 have a value greater than 1 and 208 have a value equal to 1. The network's density is 0.1904 while the average degree is 25.52. The network is depicted in Figure 5.27 on page 147. This figure shows dense communication both between members of cluster 3 as well as with members of other clusters. In Figure 5.28 on page 148 the variety of betweenness centrality produces a variety of vertex sizes while a dense communication can be observed suggesting that members of cluster 3 communicate both

Figure 5.26: KDE reduced graph—each cluster was reduced in order to illustrate connections between various groups within the community



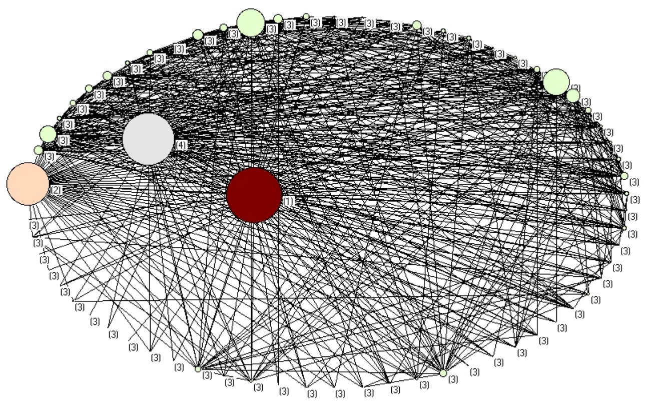
within the cluster as well as directly with members of other clusters. If loops are eliminated the density value becomes 0.1926 and the degree drops to 25.43.

Figure 5.27: KDE graph shrunk except for those who are active on the QA mailing lists but do not submit code (cluster 3)—depicts relations between members of cluster 3 as well as between these members and other clusters; the red, orange and grey vertices represent shrunk clusters.



The same operations were applied to the KDE graph in order to reduce all clusters except for cluster 1, i.e. those who both submit code and are active on the QA mailing lists. The resulting network contains 56 vertices of which three represent the other clusters. The graph contains a total of 1,204 arcs out of which 934 have a value greater than 1 and 270 have a value equal to 1. The network's density is 0.3839 while the average degree is 43. The resulting network is depicted in Figure 5.29 on page 149. This figure shows a very dense communication both between members of cluster 1 as well

Figure 5.28: KDE graph shrunk except for those who are active on the QA mailing lists but do not submit code (cluster 3)—the size of each vertex is given by its betweenness centrality value; the red, orange and grey vertices represent shrunk clusters.

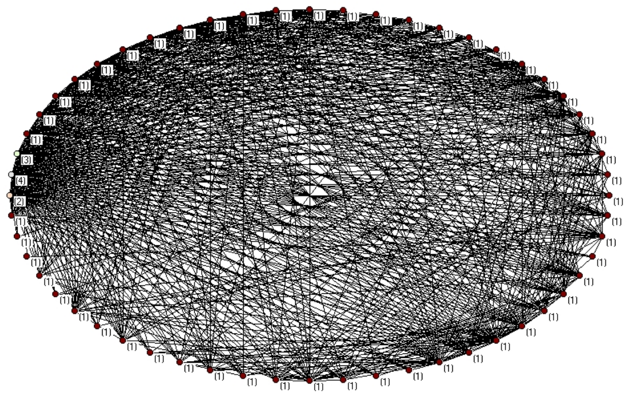


as with members of other clusters. In Figure 5.30 on page 152 the variety of betweenness centrality produces a variety of vertex sizes while a dense communication can be observed. This suggests that members of cluster 1 communicate both within the cluster as well as directly with members of other clusters. If loops are eliminated the density value is 0.3899 and the degree drops to 42.89.

In order to visualise the communication patterns between the QA team as a whole and the rest of the community, cluster 1 was merged with cluster 3 and clusters 2 and 4 were reduced. The resulting network contains 119 vertices of which two represent the other clusters (i.e. 2 and 4). The graph contains a total of 3,213 arcs of which 2,320 have a value greater than 1 and 893 have a value equal to 1. The network's density is 0.2268 while the average degree is 54. The resulting network is depicted in Figure 5.31 on page 154. This figure shows a very dense communication both between members of cluster 1 as well as with members of other clusters. In Figure 5.32 on page 155 the variety of betweenness centrality produces a variety of vertex sizes while a dense communication can be observed. This suggests that members of cluster 1 communicate both within the cluster as well as directly with members of other clusters. If loops are eliminated the density value is 0.2286 and the degree drops to 53.96.

The analysis of communication patterns between members of the community performed by reducing clusters suggests that members active on the QA mailing lists communicate among themselves and do not seem to be exclu-

Figure 5.29: KDE graph shrunk except for those who both code and are active on the QA mailing lists (cluster 1)—depicts relations between members of cluster 1 as well as between these members and other clusters; the green, orange and grey vertices represent shrunk clusters.



sively oriented towards other groups within the community. Communication seems to be somewhat decentralised in the sense that there no individual or few individuals are central in the network. The fact that QA members communicate directly with members of other layers in the community and the lack of few central individuals also suggests that there are no information brokers between layers.

5.3.4 Social network analysis of QA-specific communications

So far we have analysed communications carried out on both issue tracker and mailing lists. Now, however, we want to focus on QA activities within the KDE community, so we restrict our dataset to QA-related mailing list. After eliminating loops the network contains 116 vertices connected by 350 arcs of which 245 have a value equal to 1 and 105 have a value greater than 1 (the highest value an arc had is 22). The average degree is 6.03 while the network's density is 0.0262. The graph is shown in Figure 5.33 on page 156.

The QA team graph contains only one component that contains all 116 vertices. In the directed QA graph the indegree values vary between 0 and 25. Table 5.33 on page 150 details the distribution of indegree values.

In the directed QA graph the outdegree values vary between 0 and 21. Table 5.34 on 151 details details the distribution of outdegree values.

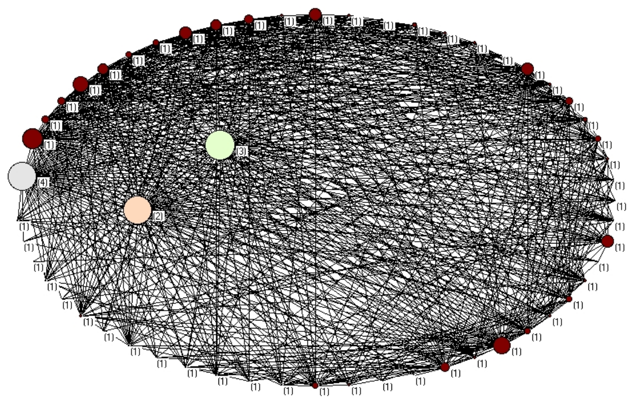
Table 5.33: Vertices in the directed KDE QA graph clustered by indegree value

Indegree value	Frequency	Frequency %	CumFreq	CumFreq%
0	24	20.6897	24	20.6897
1	36	31.0345	60	51.7241
2	21	18.1034	81	69.8276
3	8	6.8966	89	76.7241
4	6	5.1724	95	81.8966
5	5	4.3103	100	86.2069
6	3	2.5862	103	88.7931
7	2	1.7241	105	90.5172
9	1	0.8621	106	91.3793
10	1	0.8621	107	92.2414
11	1	0.8621	108	93.1034
12	2	1.7241	110	94.8276
13	1	0.8621	111	95.6897
17	1	0.8621	112	96.5517
19	2	1.7241	114	98.2759
20	1	0.8621	115	99.1379
25	1	0.8621	116	100.0000

Table 5.34: Vertices in the directed KDE QA graph clustered by outdegree value

Outdegree values	Frequency	Frequency %	CumFreq	CumFreq%
0	16	13.7931	16	13.7931
1	45	38.7931	61	52.5862
2	19	16.3793	80	68.9655
3	9	7.7586	89	76.7241
4	9	7.7586	98	84.4828
5	3	2.5862	101	87.0690
6	1	0.8621	102	87.9310
7	1	0.8621	103	88.7931
8	1	0.8621	104	89.6552
9	2	1.7241	106	91.3793
10	1	0.8621	107	92.2414
11	2	1.7241	109	93.9655
12	1	0.8621	110	94.8276
14	1	0.8621	111	95.6897
15	1	0.8621	112	96.5517
17	1	0.8621	113	97.4138
18	1	0.8621	114	98.2759
21	2	1.7241	116	100.0000

Figure 5.30: KDE graph shrunk except for those who both code and are active on the QA mailing lists (cluster 1)—the size of each vertex is given by its betweenness centrality value; the green, orange and grey vertices represent shrunk clusters.



Around 92% of individuals receive and send from/to ten or fewer individuals, in other words the majority of members active on the QA mailing lists have similar values of indegree and outdegree. However, in order to portray the number of connections a member has regardless of direction of communication the graph was symmetrized. In this symmetrized graph i.e. undirected, the degree varies between 1 and 30. Table 5.35 on page 153 details the distribution of degree values. The results suggest that the number of individuals with more connections has increased after symmetrizing the graph. One explanation for this could be that some members send messages to one set of individuals but receive messages from a different set of individuals.

The symmetrized graph contains one component that contains all 116 vertices, the same as the directed graph (not symmetrized). However, if the network is not symmetrized the directed graph contains 41 *strong* components of which the largest contains 76 vertices or about 66%. In other words, the largest component in which any two vertices are connected by a path contains approximately 66% of vertices. The highest k-core in the symmetrized graph is a 4-core containing 30 vertices or approximately 26% of all vertices. The largest k-core is a 1-core containing about 40% of the whole community.

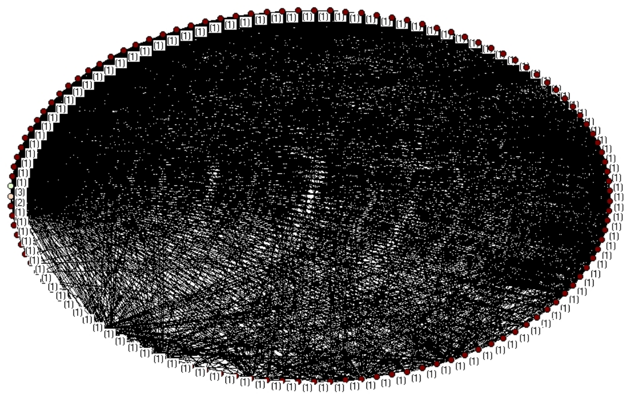
In order to understand communication patterns within the QA team, the graph was was divided into four clusters as follows:

- Cluster 1—Members of the community who were listed as code con-

Table 5.35: Vertices in the symmetrized KDE QA graph clustered by degree value

Degree value	Frequency	Frequency %	CumFreq	CumFreq%
1	46	39.6552	46	39.6552
2	19	16.3793	65	56.0345
3	13	11.2069	78	67.2414
4	10	8.6207	88	75.8621
5	8	6.8966	96	82.7586
6	2	1.7241	98	84.4828
7	4	3.4483	102	87.9310
10	1	0.8621	103	88.7931
11	2	1.7241	105	90.5172
13	1	0.8621	106	91.3793
14	1	0.8621	107	92.2414
16	2	1.7241	109	93.9655
19	1	0.8621	110	94.8276
20	1	0.8621	111	95.6897
21	1	0.8621	112	96.5517
22	1	0.8621	113	97.4138
24	1	0.8621	114	98.2759
26	1	0.8621	115	99.1379
30	1	0.8621	116	100.0000

Figure 5.31: KDE graph reduced except for cluster 1 and 3—depicts relations between members of cluster 1 and 3 as well as between these members and other clusters; the orange and green vertices represent reduced clusters.



tributor and were active on other mailing lists.

- Cluster 2—Members of the community who were listed as code contributor and were not active on other mailing lists.
- Cluster 3—Members of the community who were not listed as code contributor and were active on other mailing lists.
- Cluster 4—Members of the community who were not listed as code contributor and were not active on other mailing lists. In addition, members who could not be categorised were added to this cluster.

Table 5.36 on page 155 provides more details regarding cluster distribution within the QA team graph. It is interesting to note that, of a total 116 QA contributors, only one member belongs to cluster 2 and only eight members belong to cluster 4.

The betweenness centralisation value of the QA team graph is 0.2148. Vertices with a betweenness centrality equal to 0 represent approximately 58% of the vertices. Vertices with betweenness centrality values between 0.075 and 0.226 represent about 6.9% of the vertices. Figure 5.34 on page 157 shows clusters within the QA team where the vertex size shows the centrality betweenness score.

Until now we have carried out social network analysis of the Ubuntu community in general and the QA contributors in particular using aggregated data taking place over several years. In order to capture the dynamic

Figure 5.32: KDE graph reduced except for cluster 1 and 3—the size of each vertex is given by its betweenness centrality value; the orange and green vertices represent reduced clusters.

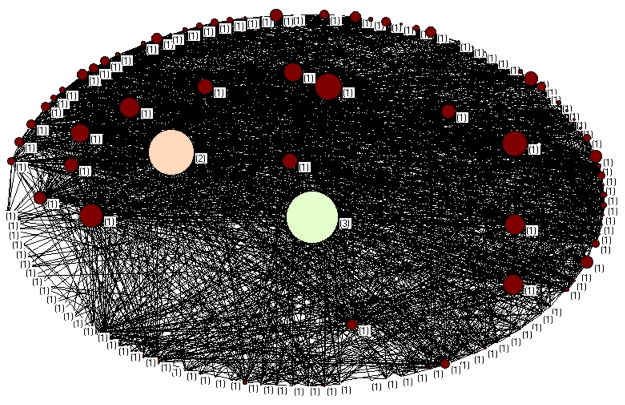


Table 5.36: Clusters in the KDE QA team

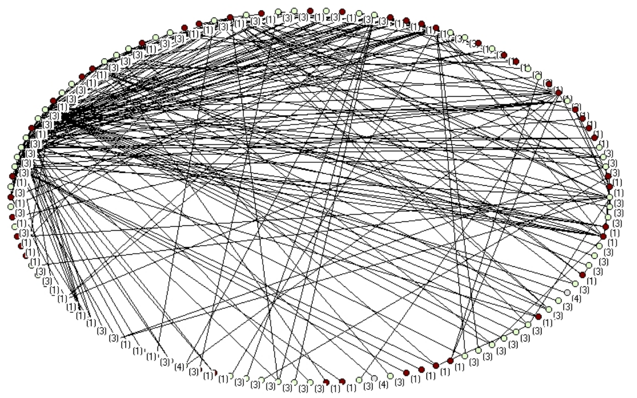
Cluster	Frequency	Frequency %	CumFreq	CumFreq%
1	51	43.9655	51	43.9655
2	1	0.8621	52	44.8276
3	56	48.2759	108	93.1034
4	8	6.8966	116	100.0000

nature of the community, we next analysed the Ubuntu subnetwork formed of members active on the QA mailing lists using six-month time frames. The subgraph associated with each time frame is further described as follows:

- SN_1—17 vertices connected by 41 arcs where the average degree is 4.82. The network betweenness centralisation score is 0.3858 where about 6% of vertices had a betweenness centrality score higher than average (i.e. 0.133). After symmetrising the network it contained 30 edges. The average degree drops to 3.52.

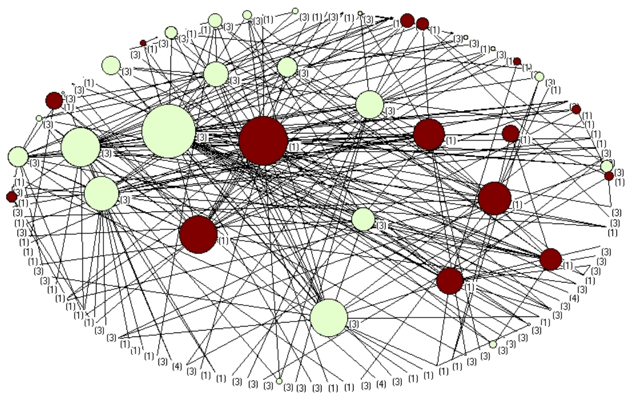
- SN_2—26 vertices connected by 73 arcs where the average degree is 5.61. The network betweenness centralisation score is 0.3793 where about 12% of vertices had a betweenness centrality score higher than average (i.e. 0.134). After symmetrising the network it contained 53 edges. The average degree drops to 4.07.

Figure 5.33: KDE QA team graph clusters—each colour in the graph represents a cluster and is labeled as such.



- SN_3—14 vertices connected by 21 arcs where the average degree is 3. The network betweenness centralisation score is 0.1124 where about 21% of vertices had a betweenness centrality score higher than average (i.e. 0.043). After symmetrising the network it contained 18 edges. The average degree drops to 2.57.
- SN_4—20 vertices connected by 38 arcs where the average degree is 3.8. The network betweenness centralisation score is 0.3689 where about 15% of vertices had a betweenness centrality score higher than average (i.e. 0.132). After symmetrising the network it contained 28 edges. The average degree drops to 2.8.
- SN_5—9 vertices connected by 11 arcs where the average degree is 2.44. The network betweenness centralisation score is 0.0535 where about 44% of vertices had a betweenness centrality score higher than average (i.e. 0.024). After symmetrising the network it contained 9 edges. The average degree drops to 2.
- SN_6—15 vertices connected by 31 arcs where the average degree is 4.13. The network betweenness centralisation score is 0.4462 where about 13% of vertices had a betweenness centrality score higher than average (i.e. 0.161). After symmetrising the network it contained 22 edges. The average degree drops to 2.93.
- SN_7—11 vertices connected by 13 arcs where the average degree is 2.36. The network betweenness centralisation score is 0.2655 where

Figure 5.34: KDE QA team graph clusters—the size of each vertex is given by its betweenness centrality value.



about 18% of vertices had a betweenness centrality score higher than average (i.e. 0.096). After symmetrising the network it contained 9 edges. The average degree drops to 1.63.

- SN_8—9 vertices connected by 8 arcs where the average degree is 1.77. The network betweenness centralisation score is 0.0848 where about 22% of vertices had a betweenness centrality score higher than average (i.e. 0.030). After symmetrising the network it contained 6 edges. The average degree drops to 1.33.

- SN_9—35 vertices connected by 75 arcs where the average degree is 4.28. The network betweenness centralisation score is 0.2933 where about 9% of vertices had a betweenness centrality score higher than average (i.e. 0.103). After symmetrising the network it contained 52 edges. The average degree drops to 2.97.

- SN_{10}—31 vertices connected by 54 arcs where the average degree is 3.48. The network betweenness centralisation score is 0.2271 where about 16% of vertices had a betweenness centrality score higher than average (i.e. 0.083). After symmetrising the network it contained 38 edges. The average degree drops to 2.45.

- SN_{11}—23 vertices connected by 44 arcs where the average degree is 3.82. The network betweenness centralisation score is 0.4574 where about 13% of vertices had a betweenness centrality score higher than average (i.e. 0.163). After symmetrising the network it contained 28 edges. The average degree drops to 2.43.

The KDE QA team were active in all time frames considered. The subnetworks associated with each time frame vary in size from 9 to 35 vertices[26]. The average degree after symmetrising the networks varies between 1.33 and 4.07. The findings, as predicted, seem to point to much smaller networks and lower degrees compared to the aggregated QA team graph. However, as the degree values are not exceptionally high in a given time frame compared to the aggregated graph it is safe to assume that a number of people made a sustained communication effort within the group and created connections between themselves as well as with members of the periphery or occasional contributors. These findings are consistent with the initial assumption that a small group of individuals within the QA team is communicating more thus performing most of the tasks. The value of the betweenness centrality value for the aggregated graph is 0.2148 where only 6.9% of participants have a higher than average score. The subnetworks' betweenness centralisation scores vary between 0.0535 and 0.4462 where the percentage of participants having a higher than average betweenness score varied between 6% and 44%. These findings confirm that subnetworks will display higher centrality values than the aggregated form of the graph.

5.3.5 Summary of findings for this case study

KDE-related communication activity rose and fell over the period studied in all communication channels and did not show consistent growth.

QA seems not to be a separate layer in the KDE community considering that only seven participants were not active on other venues and were not listed as code contributors.

Social network analysis suggests that in the KDE community most members form a large group spanning mailing lists and issue tracker. However, most connections are created by a single unrepeated act. Furthermore, a small group of individuals is highly connected and highly engaged in communication activities. Much fewer than 1% (i.e. 0.1%) of community members had a higher than average betweenness centrality value, and those values were not particularly high, which suggests that information flow in the network is not particularly vulnerable. The QA cluster analysis showed that QA members communicate both among themselves, in a somewhat decentralised manner, as well as directly with people contributing to other project activities.

[26]As the dataset for SN_{11} was not complete, it was not taken into consideration

The subgraph formed of members active on the QA mailing lists displays similarities to the whole KDE network in the sense that most participants are included in a large group, within which a small number of members are highly connected and engaged in communication.

5.4 LibreOffice

5.4.1 The LibreOffice project

LibreOffice "is a comprehensive, professional-quality productivity suite"[27]. The community is powered by the Document Foundation, which was established in 2010[28] after a fork with OpenOffice, another FLOSS product. LibreOffice has a QA team in charge of bug triage, manual testing, automatic testing and localisation QA activities[29]. The QA team uses a dedicated mailing list, an IRC channel as well as wikis for QA or bug triaging.

The wide adoption of LibreOffice by end users, many of whom are less technically knowledgeable than those who use products of the other case studies such as Plone or KDE, as well as its dedicated QA teams, justify its selection as the next case study in this research.

5.4.2 General analysis of the LibreOffice dataset

Issue tracker data as well as mailing list data were taken into account. Data was retrieved between May and June 2013 and stored locally. The issue tracker data contained a total of 19,432 bugs with 127,208 associated comments, and was downloaded using a web crawler. The mailing lists contained a total of 192,709 e-mails on all LibreOffice topics that were downloaded using MailingListStats, an open source tool[30]. The QA mailing list contained a total of 3,381 e-mails. In addition, a list of contributors' usernames, nicknames and (where available) real names was downloaded from Ohloh[31]. This list was used to perform data cleaning similarly to the previous cases.

Of all the e-mails exchanged 1,071 (0.55%) were sent by authors who had sent only one e-mail throughout the period of the study while of all the QA e-mails 187 (5.53%) were sent by authors who had sent only one e-mail. On the other hand 5,250 bugs (27.01%) were posted by members who had posted only

[27]Libreoffice—features. http://www.libreoffice.org/features/

[28]Document foundation—history: http://www.documentfoundation.org/foundation/history/

[29]Libreoffice—quality assurance: http://www.libreoffice.org/ get-involved/qa-testers

[30]MailingListStats: https://github.com/MetricsGrimoire/MailingListStats

[31]Ohloh: https://www.ohloh.net/

one bug throughout the period of the study while 3,494 (2.74%) comments were posted by members who posted only one comment. The activity on mailing lists and the bug tracker on a yearly basis is shown in Figure 5.35 on page 160 while more detailed information is given in Table C.14 on page 289. While the figure shows a steady increase over the three years, the dataset covers a period of time too small to draw significant conclusions.

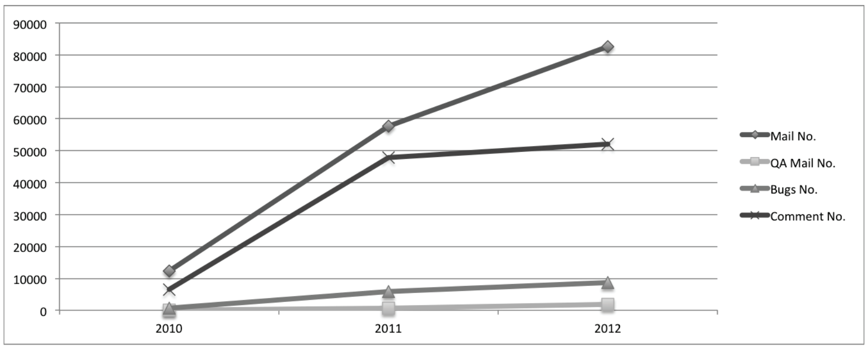


Figure 5.35: LibreOffice Activity Chart

The QA mailing list activity started in 2011 and includes 317 members of whom about 187 (59%) sent only one e-mail. The average number of e-mails sent was 10.67 (sd = 37.16). Only about 16% of QA mailing lists participants sent more than eleven e-mails. A detailed description of top QA e-mail participants' activity is given in Table 5.37 on page 161[32].

[32]A table containing all QA mailing list participants and their activity can be found in Table C.13 on page 280

Table 5.37: LibreOffice QA mailing list participants' activity levels on other channels

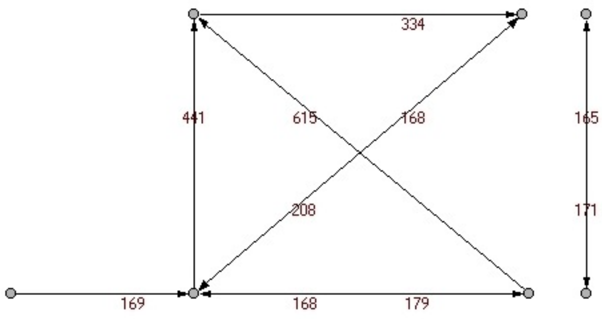
Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{153}	443	176	2	0	0	No
X_{293}	254	134	1	0	0	No
X_{8}	237	2462	4	42	1042	Yes
X_{80}	192	1009	3	0	0	Yes
X_{19}	136	663	2	152	1535	Yes
X_{140}	130	320	2	20	2620	Yes
X_{100}	124	4067	3	126	1876	Yes
X_{71}	101	5	1	58	4327	Yes
X_{58}	100	18	2	53	383	Yes
X_{217}	70	108	1	0	0	Yes
X_{82}	69	38	1	0	0	No
X_{135}	65	0	0	6	46	No
X_{284}	59	491	2	0	0	Yes
X_{301}	56	5	2	0	0	No
X_{22}	54	9	1	0	0	No
X_{29}	45	4	1	38	2779	Yes
X_{230}	42	109	1	34	257	Yes
X_{283}	42	66	2	0	0	No
X_{179}	39	36	1	0	0	No
X_{269}	36	930	2	25	841	Yes

Of the total 317 participants on the QA mailing list, 181 (57%) were not active on other mailing lists, and of these 43 sent more than one message and one was listed as a code contributor. In addition, 267 participants (84%) were not listed as code contributors. These facts suggest that QA contributors form a a somewhat separate layer in the LibreOffice community.

5.4.3 Social network analysis of the whole dataset

The next stage was to carry out a social network analysis of the LibreOffice community. The dataset was exported to a Pajek-readable file that contained community members, the links between them and the periods in which they were active in the project. Members who did not form any connections with other members were eliminated from the dataset. As in the previous case studies, after importing this data into Pajek all loops were eliminated. The resulting network contains 9,971 vertices connected by a total of 44,682 arcs of which 14,035 have a value that is greater than 1 and 30,647 have a value equal to 1. This means that 30,647 connections (68.58%) are created by only one interaction (message). One could draw the conclusion that these members are occasional contributors or part of the periphery. The average degree is 8.96, which means that on average a person interacts with approximately nine other people. The network's density is 0.00044. The "heaviest" ten arcs connected seven people and had values between 165 and 615; see Figure 5.36 on page 162.

Figure 5.36: The heaviest ten arcs in the LibreOffice graph connecting ten members—numbers represent e-mails and bug comments in each direction



The network contains 111 components of which the biggest component contains 9,842 vertices or about 99% of the community. The rest of the components contain between one and three vertices.

To get the precise values of vertex degrees within the LibreOffice community, the directed network was transformed into an undirected network by transforming all arcs into edges. The resulting network has 32,556 edges of which 14,019 have a value greater than 1 and 18,537 with a value equal to 1. The network's density is 0.00065. The average degree drops to 6.53 compared to 8.96 for the directed network. About 10% of individuals had connections with more than seven other individuals (i.e. the average degree). Looking at the vertices with degree value from 0 to 10, we find that they account for approximately 93% of the community. The remaining approximately 7% of vertices with the highest degree are connected to—that is, communicate directly with— between 11 and 2,026 other members. Details of the 93% of vertices in the LibreOffice graph with value between 0 and 10 are given in Table 5.38 on page 163. The largest degree cluster is formed by vertices with a degree value of 1 and represents more than one third (i.e. 41.8%) of the community.

Table 5.38: Vertices in the symmetrized LibreOffice graph clustered by degree value

Degree value	Frequency	Frequency %	CumFreq	CumFreq%
0	94	0.9427	94	0.9427
1	4,167	41.7912	4,261	42.7339
2	2,386	23.9294	6,647	66.6633
3	981	9.8385	7,628	76.5019
4	544	5.4558	8,172	81.9577
5	358	3.5904	8,530	85.5481
6	224	2.2465	8,754	87.7946
7	177	1.7751	8,931	89.5698
8	154	1.5445	9,085	91.1142
9	107	1.0731	9,192	92.1873
10	74	0.7422	9,266	92.9295

Both the indegree and outdegree values were computed. In the directed LibreOffice graph vertices with values between 0 and 10 make up about 95% of the community—see Table 5.39 on page 164. The remaining vertices have indegree values of between 11 and 1,230.

Table 5.39: Vertices in the directed LibreOffice graph clustered by indegree value

Indegree value	Frequency	Frequency %	CumFreq	CumFreq%
0	329	3.2996	329	3.2996
1	5,166	51.8102	5,495	55.1098
2	1,859	18.6441	7,354	73.7539
3	810	8.1236	8,164	81.8774
4	429	4.3025	8,593	86.1799
5	297	2.9786	8,890	89.1586
6	170	1.7049	9,060	90.8635
7	146	1.4642	9,206	92.3278
8	86	0.8625	9,292	93.1903
9	95	0.9528	9,387	94.1430
10	59	0.5917	9,446	94.7347

In the same directed LibreOffice graph approximately 4% of vertices have outdegree values between 11 and 1,804; see Table 5.40 on page 165.

The LibreOffice directed graph contains 3,911 strong components. The largest strong component contains 6,041 vertices which amounts to approximately 61% of the whole network. The symmetrized i.e. undirected network contains 111 components where the largest contains 9,842 vertices or approximately 99% of the whole community. In the same symmetrized LibreOffice graph, k-core values range between 0 and 38, in other words the most connected group of members contains individuals that interacted with a minimum of 38 other people. This group contains 69 community members. Table 5.41 on page 165 gives details of the members who have k-core values between 0 and 10, who make up approximately 94% of the graph.

Betweenness centrality values were then computed. The LibreOffice directed graph vertices have betweenness centralities of between 0 and 0.128 where the network's betweenness centralisation value is equal to 0.1277. Table 5.42 on page 166 details the distribution of betweenness centrality vectors within the network. Findings suggest that approximately 66% of the LibreOffice community do not represent a "step" in the information flow.

Table 5.40: Vertices in the directed LibreOffice graph clustered by outdegree value

Outdegree value	Frequency	Frequency %	CumFreq	CumFreq%
0	3,559	35.6935	3,559	35.6935
1	3,417	34.2694	6,976	69.9629
2	1,189	11.9246	8,165	81.8875
3	513	5.1449	8,678	87.0324
4	291	2.9185	8,969	89.9509
5	208	2.0860	9,177	92.0369
6	108	1.0831	9,285	93.1200
7	93	0.9327	9,378	94.0528
8	53	0.5315	9,431	94.5843
9	48	0.4814	9,479	95.0657
10	54	0.5416	9,533	95.6073

Table 5.41: Vertices in the symmetrized Ubuntu graph clustered by k-core value

K-core value	Frequency	Frequency %	CumFreq	CumFreq%
0	94	0.9427	94	0.9427
1	4,267	42.7941	4,361	43.7368
2	2,455	24.6214	6,816	68.3582
3	1,006	10.0893	7,822	78.4475
4	535	5.3656	8,357	83.8131
5	331	3.3196	8,688	87.1327
6	227	2.2766	8,915	89.4093
7	179	1.7952	9,094	91.2045
8	128	1.2837	9,222	92.4882
9	95	0.9528	9,317	93.4410
10	68	0.6820	9,385	94.1230

Furthermore, fewer than 1% of vertices (i.e. 0.0601%) have a higher than average betweenness score and the maximum betweenness centrality is 0.128, which is higher than the maximum value for the Mozilla community which was 0.063.

Table 5.42: Betweenness centrality clusters in the Ubuntu community

Value intervals	Frequency	Frequency %	CumFreq	CumFreq%
0.000	6,590	66.0917	6,590	66.0917
0.000–0.043	3,375	33.8482	9,965	99.9398
0.043–0.085	5	0.0501	9,970	99.9900
0.085–0.128	1	0.0100	9,971	100.0000

To get a better understanding of communication patterns between QA team members and the rest of the community the graph was divided into four clusters as follows:

1. Cluster 1—Community participants who contribute code and are members of the QA mailing list.
2. Cluster 2—Community participants who contribute code but are not members of the QA mailing list.
3. Cluster 3—Community participants who do not contribute code but are members of the QA mailing list.
4. Cluster 4—Community participants who do not contribute code and are not members of the QA mailing list. Members who could not be categorised were also added to this cluster.

To categorise participants into clusters a list containing LibreOffice code contributors was retrieved from Ohloh. A Python script was then written in order to compare all community members whose names contained more than four characters to the code contributor list and categorise them into clusters based on participation on the QA mailing list. Table 5.43 on page 167 provides more details regarding cluster distribution within the community. It will be observed that the cluster containing members who are neither code contributors nor QA contributors is the largest representing approximately 96% of the whole community. On the other hand people performing QA tasks appear to add up to only about 1.5% of the community.

Table 5.43: Clusters in the LibreOffice community

Cluster	Frequency	Frequency %	CumFreq	CumFreq%
1	33	0.3310	33	0.3310
2	269	2.6978	302	3.0288
3	114	1.1433	416	4.1721
4	9,555	95.8279	9,971	100.0000

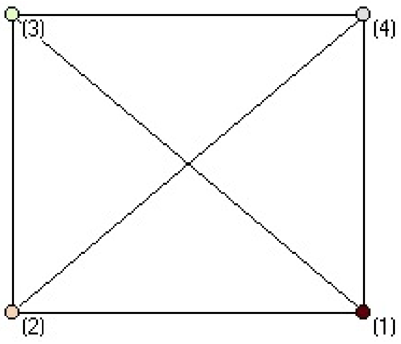
If each of these clusters is shrunk to one vertex then the resulting network has four vertices and is a complete network with a density value of 1. The graph contains a total of four loops and twelve lines and is depicted in Figure 5.37 on page 168. The values of the lines of the graph are described in Table 5.44 on page 167.

Table 5.44: Shrunk LibreOffice network communication

Rank	Line	Value
1	2.2	32,668
2	1.2	10,432
3	4.2	10,036
4	2.1	7,934
5	2.4	6,953
6	1.1	6,443
7	3.2	6,050
8	2.3	4,832
9	1.4	4,420
10	4.1	3,914
11	1.3	3,531
12	4.4	2,587
13	3.1	2,388
14	4.3	1,490
15	3.4	1,353
16	3.3	1,002

The next step was to reduce all clusters to one vertex except for cluster 3, the QA-only cluster. The resulting network contains 117 vertices of which

Figure 5.37: LibreOffice reduced graph—each cluster was reduced to one vertex in order to illustrate connections between various groups within the community

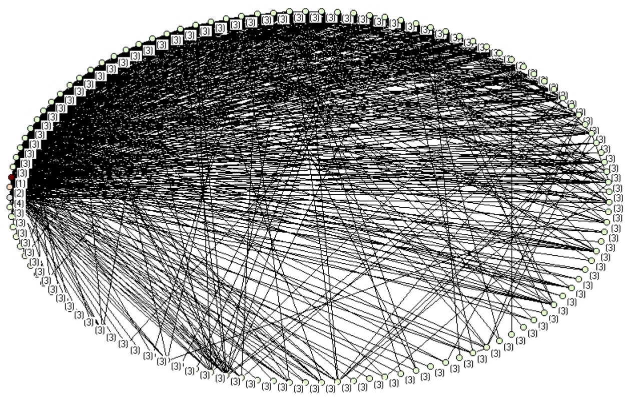


three represent the other clusters. The graph contains a total of 1,239 arcs of which 833 have a value greater than 1 and 406 have a value equal to 1. The network's density is 0.0905 while the average degree is 21.17. The network is depicted in Figure 5.38 on 169. This figure shows dense communication both between members of cluster 3 as well as with members of other clusters. In Figure 5.39 on page 170 the variety of betweenness centrality produces a variety of vertex sizes while a dense communication can be observed suggesting that members of cluster 3 communicate both within the cluster as well as directly with members of other clusters. If loops are eliminated the density value is 0.0910 and the degree drops to 21.12.

The same operations were applied to the LibreOffice graph in order to reduce all clusters except for cluster 1, i.e. those who both submit code and are active on the QA mailing lists. The resulting network contains 36 vertices of which three represent the other clusters. The graph contains a total of 630 arcs of which 499 have a value greater than 1 and 131 have a value equal to 1. The network's density is 0.4861 while the average degree is 35. The resulting network is depicted in Figure 5.40 on page 173. This figure shows dense communication both between members of cluster 1 as well as with members of other clusters. In Figure 5.41 on page 175 the variety of betweenness centrality produces a variety of vertex sizes while a dense communication can be observed. This suggests that members of cluster 1 communicate both within the cluster as well as directly with members of other clusters. If loops are eliminated the density value is 0.4976 and the degree drops to 34.83.

In order to visualise the communication patterns between the QA team as a whole and the rest of the community, cluster 1 was merged with cluster 3 and clusters 2 and 4 were reduced. The resulting network contains 149

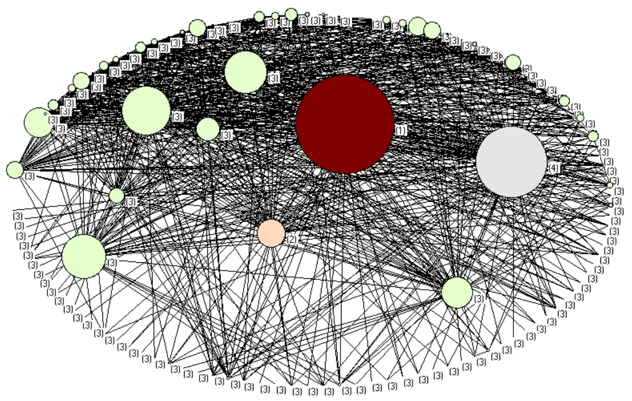
Figure 5.38: LibreOffice graph shrunk except for those who are active on the QA mailing lists but do not submit code (cluster 3)—depicts relations between members of cluster 3 as well as between these members and other clusters; the red, orange and grey vertices represent shrunk clusters.



vertices of which two represent the other clusters. The graph contains a total of 2,897 arcs out of which 2,003 have a value greater than 1 and 894 have a value equal to 1. The network's density is 0.1304 while the average degree is 38.88. The resulting network is depicted in Figure 5.42 on page 176. This figure shows a very dense communication both between members of cluster 1 as well as with members of other clusters. In Figure 5.43 on page 177 the variety of betweenness centrality produces a variety of vertex sizes while a dense communication can be observed suggesting that members of cluster 1 communicate both within the cluster as well as directly with members of other clusters. If loops are eliminated the density value is 0.1312 and the degree drops to 38.85.

The analysis of communication patterns between members of the community performed by reducing clusters suggests that members active on the QA mailing lists communicate among themselves and do not seem to be exclusively oriented towards other groups within the community. Communication seems to be somewhat decentralised in the sense that there is no individual or few individuals that are central in the network. The fact that QA members communicate directly with members of other layers in the community and the lack of few central individuals also suggests that there are no information brokers between layers.

Figure 5.39: LibreOffice graph shrunk except for those who are active on the QA mailing list but do not submit code (cluster 3)—the size of each vertex is given by its betweenness centrality values; the red, orange and grey vertices represent shrunk clusters.



5.4.4 Social network analysis of QA-specific communications

So far we have analysed communications carried out on both issue tracker and mailing lists. Now, however, we want to focus on QA activities within the LibreOffice community, so we restrict our dataset to the QA-related mailing list. After eliminating loops the network contains 153 vertices connected by 861 arcs of which 492 have a value equal to 1 and 369 have a value greater than 1 (the highest value an arc had is 30). The average degree is 11.25 while the network's density is 0.0370. The graph is shown in Figure 5.44 on page 178.

The QA team graph contains only one component that contains all 153 vertices. In the directed QA graph the indegree values vary between 0 and 51. Table 5.45 on page 171 details the distribution of indegree values.

In the directed QA graph the outdegree values vary between 0 and 59. Table 5.46 on page 172 details the distribution of outdegree values.

Around 84% of individuals receive messages from ten or fewer individuals while around 81% of individuals send messages to ten or fewer individuals. These findings suggest similar values between indegree and outdegree values. However,compared to other case studies, the number of individuals who have an indegree value that is less than 10 is slightly higher (i.e. about 3%) than the number of individuals who have an outdegree value that is less than 10.

However, in order to portray the number of connections a member has

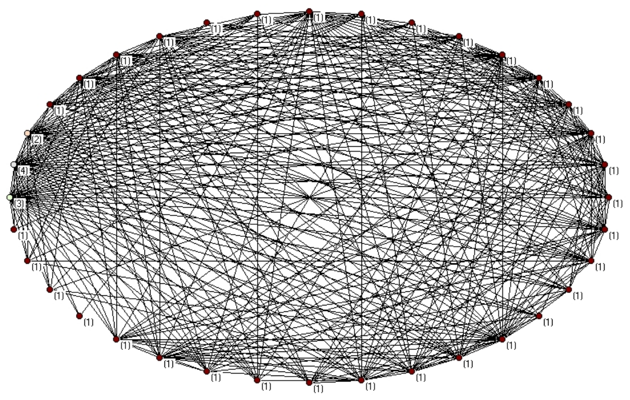
Table 5.45: Vertices in the directed LibreOffice QA graph clustered by indegree value

Indegree value	Frequency	Frequency %	CumFreq	CumFreq%
0	14	9.1503	14	9.1503
1	50	32.6797	64	41.8301
2	25	16.3399	89	58.1699
3	8	5.2288	97	63.3987
4	10	6.5359	107	69.9346
5	2	1.3072	109	71.2418
6	5	3.2680	114	74.5098
7	3	1.9608	117	76.4706
8	7	4.5752	124	81.0458
9	2	1.3072	126	82.3529
10	3	1.9608	129	84.3137
11	1	0.6536	130	84.9673
12	3	1.9608	133	86.9281
13	1	0.6536	134	87.5817
14	4	2.6144	138	90.1961
15	2	1.3072	140	91.5033
17	2	1.3072	142	92.8105
18	1	0.6536	143	93.4641
20	1	0.6536	144	94.1176
22	2	1.3072	146	95.4248
34	2	1.3072	148	96.7320
36	1	0.6536	149	97.3856
37	1	0.6536	150	98.0392
39	2	1.3072	152	99.3464
51	1	0.6536	153	100.0000

Table 5.46: Vertices in the directed LibreOffice QA graph clustered by outdegree value

Outdegree value	Frequency	Frequency %	CumFreq	CumFreq%
1	39	25.4902	77	50.3268
2	22	14.3791	99	64.7059
3	4	2.6144	103	67.3203
4	3	1.9608	106	69.2810
5	2	1.3072	108	70.5882
6	5	3.2680	113	73.8562
7	5	3.2680	118	77.1242
9	3	1.9608	121	79.0850
10	3	1.9608	124	81.0458
11	3	1.9608	127	83.0065
12	4	2.6144	131	85.6209
13	1	0.6536	132	86.2745
14	3	1.9608	135	88.2353
15	2	1.3072	137	89.5425
16	1	0.6536	138	90.1961
17	2	1.3072	140	91.5033
18	2	1.3072	142	92.8105
19	2	1.3072	144	94.1176
21	1	0.6536	145	94.7712
23	1	0.6536	146	95.4248
27	1	0.6536	147	96.0784
30	1	0.6536	148	96.7320
33	1	0.6536	149	97.3856
44	1	0.6536	150	98.0392
47	1	0.6536	151	98.6928
48	1	0.6536	152	99.3464
59	1	0.6536	153	100.0000

Figure 5.40: LibreOffice graph shrunk except for those who both code and are active on the QA mailing list (cluster 1)—depicts relations between members of cluster 1 as well as between these members and other clusters; the green, orange and grey vertices represent shrunk clusters.



regardless of direction of communication the graph was symmetrized. In this symmetrized graph i.e. undirected, the degree varies between 1 and 69. Table 5.47 on page 174 details the distribution of degree values. The results suggest that the number of individuals with more connections has increased after symmetrizing the graph. One explanation for this could be that some members send messages to one set of individuals but receive messages from a different set of individuals.

The symmetrized graph contains one component that contains all 31 vertices, the same as the directed graph (i.e. unsymmetrized). However, if the network is not symmetrized, the directed graph contains 55 *strong* components of which the largest contains 99 vertices or about 65%. In other words, the largest component in which any two vertices are connected by a path contains approximately 65% of vertices. The highest k-core in the symmetrized graph is an 11-core containing 26 vertices or approximately 17% of all vertices. The largest k-core is a 1-core containing about 29% of the whole community.

In order to understand communication patterns within the QA team, the graph was was divided into four clusters as follows:

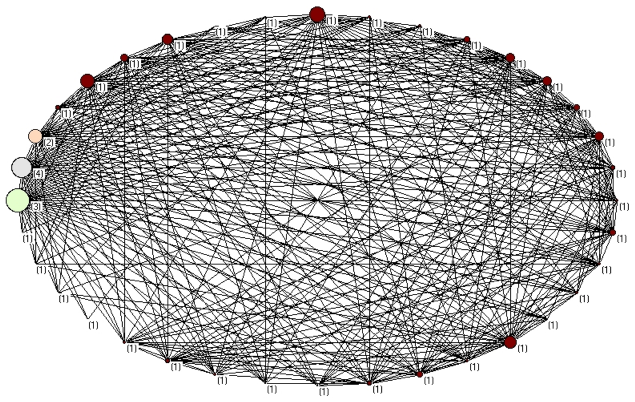
- Cluster 1—Members of the community who were listed as code contributor and were active on other mailing lists.

- Cluster 2—Members of the community who were listed as code contributor and were not active on other mailing lists.

Table 5.47: Vertices in the symmetrized LibreOffice QA graph clustered by degree value

Cluster	Frequency	Frequency %	CumFreq	CumFreq%
1	44	28.7582	44	28.7582
2	33	21.5686	77	50.3268
3	16	10.4575	93	60.7843
4	6	3.9216	99	64.7059
5	2	1.3072	101	66.0131
6	3	1.9608	104	67.9739
7	1	0.6536	105	68.6275
8	9	5.8824	114	74.5098
9	1	0.6536	115	75.1634
10	2	1.3072	117	76.4706
11	3	1.9608	120	78.4314
12	1	0.6536	121	79.0850
13	2	1.3072	123	80.3922
14	3	1.9608	126	82.3529
15	1	0.6536	127	83.0065
16	1	0.6536	128	83.6601
17	6	3.9216	134	87.5817
18	1	0.6536	135	88.2353
19	1	0.6536	136	88.8889
20	1	0.6536	137	89.5425
21	2	1.3072	139	90.8497
22	1	0.6536	140	91.5033
23	2	1.3072	142	92.8105
24	1	0.6536	143	93.4641
25	1	0.6536	144	94.1176
29	1	0.6536	145	94.7712
33	1	0.6536	146	95.4248
37	1	0.6536	147	96.0784
41	1	0.6536	148	96.7320
42	1	0.6536	149	97.3856
51	1	0.6536	150	98.0392
54	2	1.3072	152	99.3464
69	1	0.6536v153	100.0000	

Figure 5.41: LibreOffice graph shrunk except for those who both code and are active on the QA mailing list (cluster 1)—the size of each vertex is gibe by its betweenness centrality value; the green, orange and grey vertices represent shrunk clusters.



- Cluster 3—Members of the community who were not listed as code contributor and were active on other mailing lists.
- Cluster 4—Members of the community who were not listed as code contributor and were not active on other mailing lists. In addition, members who could not be categorised were added to this cluster.

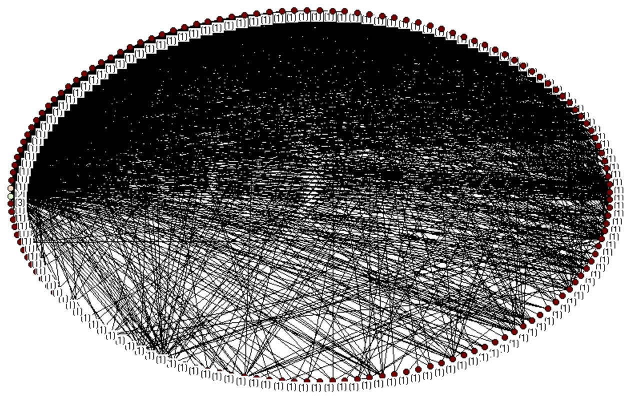
Table 5.48 on page 175 provides more details regarding cluster distribution within the QA team graph. It is interesting to note that, of a total 153 QA contributors, only one belongs to cluster 2. However, compared to other case studies, cluster 1 does not seem to contain a small percentage of participants.

Table 5.48: Clusters in the LibreOffice QA team

Cluster	Frequency	Frequency %	CumFreq	CumFreq%
1	31	20.2614	31	20.2614
2	1	0.6536	32	20.9150
3	66	43.1373	98	64.0523
4	55	35.9477	153	100.0000

The betweenness centralisation value of the QA team graph is 0.1667.

Figure 5.42: LibreOffice graph reduced except for cluster 1 and 3—depicts relations between members of cluster 1 and 3 as well as between these members and other ;the orange and green vertices represent reduced clusters.



Vertices with a betweenness centrality equal to 0 represent approximately 52% of the vertices. Vertices with betweenness centrality values between 0.058 and 0.173 represent 3.9% of the vertices. Figure 5.45 on page 179 shows clusters within the QA team where the vertex size is given by the centrality betweenness score.

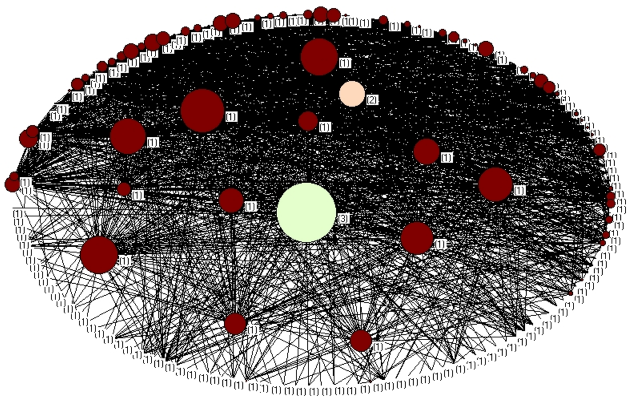
Until now we have carried out social network analysis of the Ubuntu community in general and the QA contributors in particular using aggregated data taking place over several years. In order to capture the dynamic nature of the community, we next analysed the Ubuntu subnetwork formed of members active on the QA mailing lists using six-month time frames. The subgraph associated with each time frame is further described as follows:

- SN_1—51 vertices connected by 167 arcs where the average degree is 6.54. The network betweenness centralisation score is 0.3176 where about 8% of vertices had a betweenness centrality score higher than average (i.e. 0.111). After symmetrising the network it contained 113 edges. The average degree drops to 4.43.

- SN_2—58 vertices connected by 271 arcs where the average degree is 9.34. The network betweenness centralisation score is 0.1811 where about 10% of vertices had a betweenness centrality score higher than average (i.e. 0.066). After symmetrising the network it contained 183 edges. The average degree drops to 6.31.

- SN_3—84 vertices connected by 354 arcs where the average degree is

Figure 5.43: LibreOffice graph reduced except for cluster 1 and 3—the size of each vertex if given by its betweenness centrality value; the orange and green vertices represent reduced clusters.



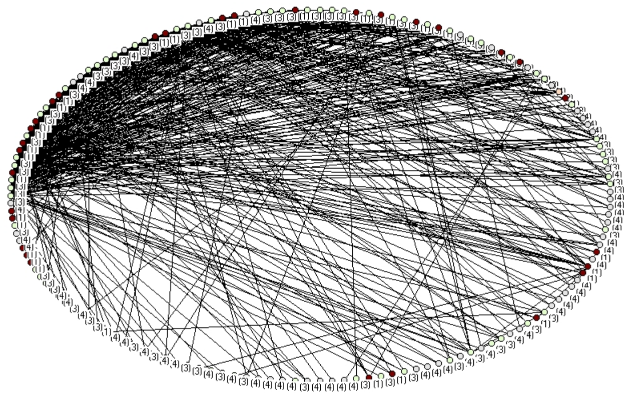
8.42. The network betweenness centralisation score is 0.198 where about 6% of vertices had a betweenness centrality score higher than average (i.e. 0.070). After symmetrising the network it contained 257 edges. The average degree drops to 6.11.

- SN_4—83 vertices connected by 322 arcs where the average degree is 7.75. The network betweenness centralisation score is 0.2115 where about 6% of vertices had a betweenness centrality score higher than average (i.e. 0.075). After symmetrising the network it contained 235 edges. The average degree drops to 5.66.

The LibreOffice QA team was active in all time frames considered. The subnetworks associated with each time frame varied in size from 51 to 84 vertices[33]. The average degree after symmetrising the networks varied between 4.43 and 6.11. The findings, as predicted, seem to point to much smaller networks and lower degrees compared to the aggregated QA team graph. However, as the degree values are not exceptionally high in a given time frame compared to the aggregated graph it is safe to assume that a number of people made a sustained communication effort within the group and created connections between themselves as well as with members of the periphery or occasional contributors. These findings are consistent with the initial assumption that a small group of individuals within the QA team is communicating more thus performing most of the tasks. The value of the

[33]As the dataset is too small, SN_4 was also included in this analysis.

Figure 5.44: LibreOffice QA team graph clusters—each colour in the graph represents a cluster and is labeled as such.



betweenness centrality value for the aggregated graph is 0.1667 where only 3.9% of participants have a higher than average score. The subnetworks' betweenness centralisation scores varied between 0.1811 and 0.3176. The percentage of participants having a higher than average betweenness score varied between 6% and 10%. These findings confirm that subnetworks will display higher centrality values than the aggregated form of the graph.

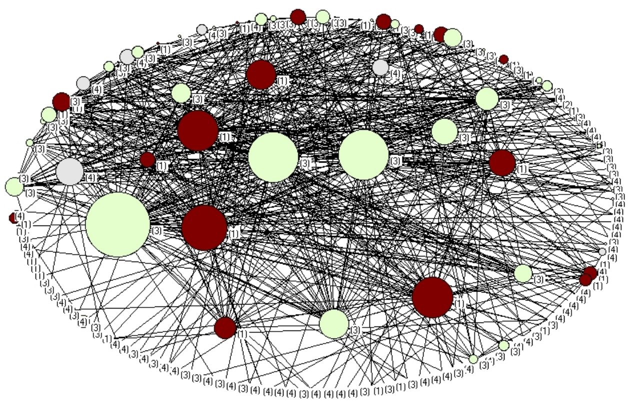
5.4.5 Summary of findings for this case study

LibreOffice-related communication activity rose constantly over the period of time. However, the dataset is too small for us to draw definitive conclusions with respect to a possible correlation between time progression and activity levels.

QA seems to be a somewhat separate layer in the LibreOffice community considering that 42 members were not listed as code contributors and were active exclusively on the QA mailing list.

Social network analysis suggests that in the LibreOffice community most members form a large group spanning mailing lists and issue tracker. Most connections are created by a single unrepeated act but such unrepeated acts are a smaller percentage of all communications compared with previous case studies. Furthermore, a small group of individuals is highly connected and highly engaged in communication activities. Much fewer than 1% (i.e. 0.0601%) of community members have a higher than average betweenness centrality value, and those values are not particularly high, which suggests

Figure 5.45: LibreOffice QA team graph clusters—the size of each vertex is given by its betweenness centrality value.



that information flow in the network is not particularly vulnerable. The QA cluster analysis showed that QA members communicate both among themselves, in a somewhat decentralised manner, as well as directly with people contributing to other project activities.

The subgraph formed of members active on the QA mailing lists displays similarities to the whole LibreOffice network in the sense that most participants are included in a large group, within which a small number of members are highly connected and engaged in communication.

Chapter 6

Conclusions and Discussion

In this chapter we will conduct a comparative analysis of the case studies in order to test and refine the set of hypotheses proposed after the analysis of the pilot case study. Furthermore, general conclusions will be discussed as well as possible threats to validity and topics for future research.

6.1 Comparative Analysis

The main goal of this research is to investigate the relation between formal QA adoption and the structure of FLOSS communities. In order to investigate this, a study was conducted in two phases:

Phase I: A pilot case study was chosen in order to refine the research methodology and propose a set of hypotheses describing QA activities within a FLOSS project. The Mozilla community was chosen as the pilot case study.

Phase II: The hypotheses proposed in Phase I were then tested in the context of four other case studies: Ubuntu, Plone, KDE and LibreOffice.

Having carried out the analysis of each case study in the second phase of this research, it is now time to revisit each hypothesis and either confirm it or modify it in light of the case study findings.

The first hypothesis states that a smaller percentage of peripheral members are engaged in conversation than in posting bugs. In the Mozilla community about 4% e-mails and about 1% of comments, as opposed to about 20% of bugs, were sent by individuals who contributed with only one action on each channel (i.e. sent only one e-mail, posted only one comment or opened only one bug). In the Ubuntu community about 2% of all e-mails, about 10% of QA e-mails, about 3% of comments and about 12% of bugs were contributed by such occasional members. In the Plone community about 2% of all e-mails, about 22% of QA e-mails, about 1% of comments and about 12% of bugs were contributed by such occasional members. In the KDE community about 3% of all e-mails, about 5% of QA e-mails, about 3% of comments, and about 13% of bugs were contributed by such occasional members. In the LibreOffice community about 1% of all e-mails, about 6% of QA e-mails, about 3% of comments, and about 27% of bugs were contributed by such occasional members.

These findings suggest that a very small percentage of community participants contribute to the project by sending only one e-mail or posting only one comment. This hypothesis is not validated by our findings for the Plone QA mailing list, but we suspect a high volume of spam may be the reason for that. The contribution of members who post only one bug seems to be over 10% for all case studies but higher then average in the LibreOffice project. One reason for this might be that LibreOffice is a software program that targets a wider range of end-users who are not necessarily technically skilled. In conclusion, a smaller percentage of peripheral members tends to be engaged in conversation than in posting bugs.

Hypothesis 1 (supported): A smaller percentage of peripheral members are engaged in conversation than in posting bugs.

The second hypothesis states that activity on the mailing list is independent from activity on the other communication channels and that traffic on the QA mailing list displays an irregular pattern of spikes and troughs. In the Mozilla projects e-mail activity peaked in 2009 while comment and bug activity showed constant growth over the period. In the Ubuntu project QA e-mail activity displayed multiple spikes that were not correlated with activity on all Ubuntu mailing lists. Bug and comment activity showed less pronounced variation. In the KDE project, activity on the QA mailing list displayed slight variation with spikes throughout the period of adoption. Bug and comment activity also displayed an irregular pattern as opposed to activity on all mailing lists which showed a consistent growth until 2009 after which almost no change was found. In the Plone and LibreOffice projects QA activities were only formally adopted in 2011 providing insufficient data for such a comparison. Thus the findings seem to support Hypothesis 2a and 2b.

Hypothesis 2a (somewhat supported): Activity levels on the QA mailing lists are independent of activity levels on other communication channels.

Hypothesis 2b (somewhat supported): Traffic on the QA mailing lists does not show a consistent upward trend.

The third and fourth hypotheses characterise contributors active on the QA mailing lists. In the Mozilla project the average number of sent e-mails was 11.28 (sd = 37.23) where only about 17% participants sent more than eleven e-mails. In the Ubuntu project the average number of sent e-mails was 5.63 (sd = 22.5) where only about 15% of participants sent more than six e-mails. In the Plone project the average number of sent e-mails was 2.98 (sd = 7.91) where only about 19% participants sent more than three e-mails. In the KDE project the average number of sent e-mails was 7.88 (sd = 16.39) where only about 20% participants sent more than eight e-mails. In the LibreOffice project the average number of sent e-mails was 10.67 (sd = 37.16) where only about 16% participants sent more than eleven e-mails. These findings suggest that only a small percentage (less than 20%) of the community is engaged higher than average activity, thus supporting Hypotheses 3a and 3b.

Hypothesis 3a (supported): A small group of people is highly active on the QA mailing lists. These groups tend to represent less that 20% of the mailing list participants.

Hypothesis 3b (supported): A large percentage of QA mailing list participants send e-mails only very occasionally.

The fourth hypothesis describes the structure of the communities. In the Mozilla community of 1,132,413 arcs, 779,313 (about 69%) had a value of 1; in other words, most of the connections were single, unrepeated acts of communication from one participant to another. In the Ubuntu community of 1,530,150 arcs, 1,206,837 (about 79%) had a value of 1. In the Plone community of 61,686 arcs, 43,172 (about 70%) had a value of 1. In the KDE community of 506,090 arcs 356,181 (about 70%) had a value of 1. In the LibreOffice community of 44,682 arcs 30,647 (about 69%) had a value of 1. These findings support the hypothesis that almost two thirds of connections within the communities' graph are created by single acts of communication from one participant to another.

Hypothesis 4 (revised): At least two thirds of connections within a project's graph are created by single acts of communication from one participant to another.

Hypothesis 5 states that the community consists of large groups of people who span both mailing lists and issue tracker. The Mozilla community contains 786 components, of which the biggest component contains 148,180 vertices or approximately 99% of the community. The Ubuntu community contains 2,796 components of which the biggest component contains 291,357 vertices or about 99% of the community. The Plone community contains 99 components of which the biggest component contains 9,362 vertices which or about 99% of the community. The KDE contains 716 components of which the biggest component contains 88,502 vertices or about 99% of the community. The LibreOffice community 111 components of which the biggest component contains 9,842 vertices or about 99% of the community. Most communities consisted of a large component containing approximately 99% of vertices and a number of small components containing between one and three vertices. However, a particularity was observed in the case of Ubuntu where the smaller components contained up to 19 vertices which could mean that groups of individuals are working separately from the rest of the community. Studying this particularity in future research could provide important insights regarding various FLOSS community structures. These findings support Hypothesis 5.

Hypothesis 5 (supported): The community consists of a large group of people that spans both issue tracker and mailing lists.

Hypotheses 6a and 6b state that less than one tenth of community participants are highly connected while more than a third have connections with only one other participant. In the symmetrized graph of Mozilla approximately 90% of the community had a degree value between 0 to 10 while about 40% had a degree value of 1. About 9% of individuals had connections with more than eleven (i.e. the average degree) other individuals. In the Ubuntu symmetrized graph approximately 89% of the community had a degree value between 0 to 10 while about 34% had a degree value of 1. About 13% of individuals had connections with more than eight (i.e. the average degree) other individuals. In the Plone symmetrized graph approximately 85% of the community had a degree value between 0 to 10 while about 31% had a degree value of 1. About 15% of individuals had connections with more than ten (i.e. the average degree) other individuals. In the KDE symmetrized graph approximately 90% of the community had a degree value between 0 to 10 while about 43% had a degree value of 1. About 11% of individuals had connections with more than nine (i.e. the average degree) other individuals. In the LibreOffice symmetrized graph approximately 93% of the community had a degree value between 0 to 10 while about 42% had a degree value of 1. About 10% of individuals had connections with more than seven (i.e. the average degree) other individuals. In conclusion the percentage of individuals connected to a higher than average number of other individuals varies between 9 and 15%. As a result, we modify Hypothesis 6a to state that fewer than 15% of community members are connected to a higher than average number of individuals. Similarly, our findings support Hypothesis 6b only partially as in the case of Plone about 31% of vertices had a degree value of 1, so we modify Hypothesis 6b to reflect this.

Hypothesis 6a (Revised): Fewer than 15% of the community members are connected to a higher than average number of other individuals.

Hypothesis 6b (Revised): More than 30% of community participants have connections with only one other participant.

Hypothesis 7 states that more than half of community participants are connected to only one other member from whom they receive messages. The approximate percentages of vertices with an indegree value of 1 in the directed graphs of the projects are as follows: Mozilla 52%, Ubuntu 50%, Plone 33%, KDE 50%, LibreOffice 52%. These findings support Hypothesis 7 with the exception of Plone, where only about a third of community participants are connected to only one other community member from whom they received messages.

Hypothesis 7 (Revised): More than one third of community participants are connected to only one other community member from whom they received one or more messages.

Hypothesis 8 states that almost one third of community participants are connected to only one other individual to whom they had sent messages. The approximate percentages of vertices with an outdegree value of 1 in the directed graphs of the projects are as follows: Mozilla 32%, Ubuntu 38%, Plone 27%, KDE 28%, LibreOffice 34%. These findings require us to revise our hypothesis.

Hypothesis 8 (Revised): More than one quarter of community participants are connected to only one other community member to whom they had sent one or more messages.

Hypothesis 9 states that more than half of community participants form a strongly connected subnetwork. The directed Mozilla graph contained 57,781 strong components of which the largest contained about 61% of vertices. The corresponding numbers for the other projects were: Ubuntu 94,586 and 68%, Plone 4,236 and 55%, KDE 43,388 and 51%, and LibreOffice 3,911 and 61%. These findings seem to support Hypothesis 9 which states that more than half of community participants form a strongly connected subnetwork.

Hypothesis 9 (Supported): More than half of community participants form a strongly connected subnetwork.

Hypothesis 10 states that less than 1% of vertices have a higher than average betweenness centrality score. In all the case studies less than 1% of vertices had a higher than average betweenness centrality score, which supports the hypothesis.

Hypothesis 10 (Supported): Less than 1% of vertices have a higher than average betweenness centrality score.

Hypothesis 11 states that the community graph does not contain a small number of people brokering information between QA team members and the rest of the community. For each case study included in this research, social network analysis techniques included shrinking non-QA related clusters and analysing the communication patterns between members of the QA teams and the shrunk clusters. The findings suggest that communication is somewhat decentralised in the sense that participants communicate directly with members of other layers of the community, thus supporting Hypothesis 11.

Hypothesis 11 (supported): The community graph does not contain a small number of people brokering information between the QA team and the rest of the community.

Hypothesis 12 states that QA team members form a large group of people working together. In all the case studies, the QA team displayed similar characteristics to the whole network but at a smaller scale. The Mozilla and Ubuntu QA teams were formed of a large component containing over 96% and 94% of vertices. In the case of Ubuntu, the largest remaining components contained 19, 13 and ten vertices while the remainder contained between one and seven vertices. In the Mozilla case all the remaining components contained fewer than four vertices. One possible reason for these small components could be that some messages are actually spam messages. Another reason could be that the messages received replies after the dataset was retrieved or that the messages were posted by members not active previously as they might be peripheral or new members who did not yet have the chance to create relations with more active users. In the Plone, KDE and LibreOffice communities the QA teams were formed of a large component containing all vertices. These findings thus support Hypothesis 12.

Hypothesis 12 (supported): The QA team forms a large group of people working together.

Hypothesis 13 and 14 state that more than a third of QA team members create only one connection by receiving and sending messages respectively. In the Mozilla QA team about 37% of individuals created only one connection by receiving messages and about 34% created only one connection by sending messages. The corresponding percentages for the other projects were: Ubuntu 47% and 31%; Plone 55% and 68%; KDE 31% and 39%; and LibreOffice 33% and 25%. In view of the wide variation in the percentages Hypotheses 13 and 14 are not supported. Hence, further research should investigate the factors influencing these high values for the Ubuntu and Plone communities.

Hypothesis 13 (Not supported): About one third of the QA team members create only one connection by receiving messages.

Hypothesis 14 (Not supported): About one third of the QA team members create only one connection by sending messages.

In the Mozilla community about 59% of community members were not assigned as bug fixers and about 9% of the 59% were active on the more technical Mozmill mailing list. However the dataset did not allow the inspection of members' activity on other mailing lists.

The following analysis should produce different percentages compared to the analysis of QA list participants' activity on other channels as in this case, only members who created at least one connection were included. In the Ubuntu community members who were not listed as code contributors and were not active on other mailing lists represented about 58% of individuals while code contributors represented about 2%. For Plone the corresponding figures were 3% and 35%; for KDE 7% and 45%; and for LibreOffice 36% and 21%. These findings suggest that in the Plone and KDE communities a very small percentage of people are performing exclusively QA related activities. On the other hand, in the Ubuntu and LibreOffice communities the percentage of individuals performing exclusively QA tasks seems to be considerably higher. A reason for these variations might be the target user of the software products in the sense that KDE and Plone are aimed at a more technically "savvy" end-user while Ubuntu and LibreOffice are dedicated to a more technically diverse user-base. Another factor that could influence this aspect could be the size or the history of the QA team itself. In other words, it could be the case that when implemented in a FLOSS project, QA is performed by individuals who initially have various roles within the community but as QA practices mature they slowly shift to an exclusively QA role. These suppositions are not confirmed by this study, however, but could provide useful insights if investigated in future research. In conclusion, Hypotheses 15 is not supported.

The percentage of individuals listed as code contributors who are active on the QA mailing lists seems to be somewhat inversely proportional with the number of individuals who are performing exclusively QA activities. In other words, in the Plone and KDE communities the number of code contributors active on the QA mailing list is very high compared to the Ubuntu and LibreOffice communities. Thus Hypothesis 16 is also not supported.

Hypothesis 15 (Not supported): More than half of the individuals active on the QA mailing lists are not active on other mailing lists and are not code contributors.

Hypothesis 16 (Not supported): More than one third of the individuals active on the QA mailing lists are listed as code contributors.

Hypothesis 17 states that size, degree and betweenness values of temporal subgraphs do not display a consistent growth over time but display a more irregular pattern. For each case study the sub-networks did not display a consistent growth for neither of the metrics (i.e. size, degree and betweenness values). However, the Plone and LibreOffice QA teams were formally adopted in 2011 and so did not provide enough data for a precise assessment. Furthermore, the Plone QA team was only active in three six-month time frames out of the five frames for which data was retrieved. Hence, Hypothesis 17 is only supported for the Ubuntu and KDE communities.

Hypothesis 17 (Somewhat supported): Size, degree and betweenness values of temporal subgraphs do not display a consistent growth over time but display a more irregular pattern.

6.2 Answers to the Research Questions

Q1: Is QA a separate layer in FLOSS communities?

This research has investigated whether QA has become a separate category of contributors in FLOSS communities. The five communities studied have dedicated communication channels, wikis and other resources for providing QA related information. However, only the Ubuntu and LibreOffice communities displayed a somewhat separate QA layer where a large percentage of its members do not appear to be contributing code or communicating on other mailing lists. In the other analysed communities a much smaller percentage of users were performing exclusively QA related activities; instead, they had multiple roles within the project. However, it is possible that non-QA activities were performed in different time frames than the ones in which the contributors were part of the QA team. Further study is required to clarify this point.

Q2: What are the communication patterns between QA members as well as with other project participants?

All communities' graphs displayed a large group of people spanning both mailing lists and issue tracker. Previous research [8] has suggested that FLOSS networks contain a small number of individuals with significantly higher connections than the network's average (called hubs). Our case studies supported this argument, as we found a very small number of individuals with a high degree compared to the network's average. Similarly, in the QA teams studied a small number of vertices had a higher degree than average. Within these teams, participants did not direct all communication efforts to one or a few members who then conveyed that information to members of other groups. Instead, QA team members seemed to communicate not only among themselves but also directly with members of other groups. Furthermore, fewer than 1% of community members had a higher than average betweenness centrality value, and those values were not particularly high, which suggests that information flow in the networks was not particularly vulnerable to a small number of individuals leaving the community.

6.3 Discussion and Limitations

6.3.1 Limitations

The main goal of this thesis is to start investigating the impact of QA adoption on the structure of FLOSS communities by proposing a set of hypotheses generated from the findings of a preliminary case study. These hypotheses were then tested on four other cases. Considering the diversity of FLOSS projects, some might argue that this data sample is not sufficient for drawing conclusions applicable to all FLOSS communities. This research is structured according to the Mockus model [59] in which hypotheses are formulated after the analysis of a preliminary case study and then validated by analysing another case study. Like the Mockus study, this thesis does not aim to create a theory that can be applied to all FLOSS communities. Furthermore, the purpose of this research is not to create a fail proof method for FLOSS communities looking to adopt formal QA practices. Assessing whether the quality of FLOSS projects has improved or not after implementing QA is also not within the scope of this research; such assessment is difficult, although previous studies have attempted to create a methodology for measuring FLOSS quality [74].

The case studies taken into consideration for this study do not cover all possible configurations of FLOSS projects with respect to size, programming language, targeted end users and so on. However, the case studies included projects of various size, programming language, history and target user. The five case studies should thus offer enough basis for proposing a set of hypotheses that can form the foundation for future studies. Furthermore, Plone and LibreOffice cases did not provide enough data in order to determine activity tendencies over the years, as LibreOffice is a relatively new FLOSS project and Plone only started implementing QA formally in 2011. It might be a useful exercise to note differences between projects that formally adopted QA recently with projects that have a longer experience of implementing QA practices.

The datasets used for this study consist of mailing lists, issue trackers and contributor lists downloaded from Ohloh. Mailing list data was downloaded both using an open source tool called MailingListStats as well as Python scripts written for that purpose. However, the mailing list datasets were not complete due to the fact that some mailing lists were private, some were archived in a format not compatible with the tools used for retrieval and some e-mails contained bad html that could not be parsed, double postings and so on. For a full list of mailing lists analysed and retrieved please consult Appendix B on page 212. In addition, the list of code contributors was retrieved from a third party (i.e. a well known and widely used online repository of FLOSS related information) and it may contain some inconsistencies.

Another issue that should be addressed is the discrepancies between the number of unique QA contributors considered in the cluster analysis of the whole community and the number of individuals active on the QA mailing lists. The number of QA contributors was calculated considering only members with names containing more than four characters and that had at least one connection in the community (i.e. at least one individual replied to their message or they replied to at least one individual). Names containing less that four characters (i.e. Tom, John, Tim, etc.) are far too common and the chance of multiple individuals posting with the same name was rather high and as a consequence these individuals were added to cluster 4 (i.e. the cluster containing participants who are neither code contributors nor QA contributors or who could not be categorised). Furthermore, it is possible for multiple individuals with names containing more than four characters to be contributing using the same name. Therefore, future studies should consider more efficient algorithms for identifying unique contributors.

Replies were used for creating the graph associated with each FLOSS community. Fields such as "<Replied-to>" were used for mailing lists while for issue trackers an individual was considered as replying to the individual who had previously posted under the same thread. However, in the case of mailing lists it is possible for an individual to reply to one e-mail and quote another. Another issue is that sometimes an individual would reply to more than one member in the same e-mail, in which case for the purposes of this study only the first addressee' e-mail address was taken into account. The same situation might occur on issue trackers in the sense that one member could be in fact replying to a member that had posted something much earlier in the thread history.

Data cleaning was performed manually in the pilot case study by removing all double postings and spam from the QA mailing lists. The names of contributors active on the QA mailing lists were also unified manually after going through the data. Data cleaning for the other four case studies was automated and consisted in unifying contributor names using a list of code contributors retrieved from Ohloh as follows: names contained in the issue tracker and the mailing list were compared with names from code repositories and unified. A potential thread to validity is the fact the individuals may use various aliases to contribute to the projects based on communication venue or simply change their alias over a period of time.

Spam messages were not included in the social network analysis, as the dataset used for social network analysis consisted only from individuals who created at least a connection within the network and it is safe to assume that people do not reply to spam messages. Removal of double postings was not performed. However each e-mail has a unique id and double postings should only occur when the same e-mail is posted to more than one list and has multiple ids. For the issue tracker data it is possible that comments were posted more than once due to personal mistakes but the number should not be large enough to influence the results of this study.

The QA definition used for the scope of this research was proposed after reviewing literature describing QA activities and QA contributors' responsibilities within proprietary software development, and comparing the results with QA activities described on the QA website associated with each case study. We must therefore acknowledge the possibility that some QA activities performed by QA contributors were not listed on the appropriate websites or that QA contributors do not actually perform some of the activities that were in fact listed in the appropriate websites. As this may be the case, future studies should include triangulation with regard to QA practices and conduct qualitative research (i.e. interviews) in order to validate this definition.

Another limitation of this study is that community members may be using private communication channels such as instant messaging, private IRC channels, private e-mails or even live meetings or phone calls. They may also be performing QA activities on public but non-archived channels such as IRC. The existence of such undisclosed or unarchived communication may bias the conclusions of this study. However, the premise of FLOSS communities is that information should be available for anyone and as such, we believe that private communication is limited.

With respect to social network analysis techniques applied on each data set temporal analysis was performed using six-month windows. However, considering the dynamic nature of communities, these windows may be too large in order to describe an accurate evolution of the networks state. Perhaps the greatest impact that temporal analysis has is on centrality values, which can be highly skewed in aggregated forms of a graph. However, considering the small sizes of sub-networks associated with each time frame and the centrality values for the aggregated form, it was possible to observe general trends.

6.3.2 Future Research

The goal of this research was to investigate the impact of formal QA adoption on FLOSS communities' structure. The findings seem to suggest that within some communities, a consequence of QA adoption is the emergence of a new layer of contributors. The members of this layer seem to communicate both among themselves as well as with members of other layers of contributors. However, the topics of communication between various participants were beyond the scope of this study and thus message contents were not analysed. A possible direction for future studies would be applying text mining techniques in order to deepen the understanding of tasks performed by QA contributors as well as the extent to which developers participate in the QA process. For example, it would be useful to find out what are developers discussing on the QA dedicated mailing lists, what are the QA contributors discussing on the issue trackers, and which contributors are asking questions and which are answering. Furthermore, it would be interesting to compare topics of communication among various FLOSS communities in order to reveal whether each community has specific QA related topics or all communities share similar topics. This study also revealed that the majority of connections within communities are created by a single unrepeated act of communication, in other words a large number of individuals sent only one message or posted only one comment and could be regarded as occasional contributors or members of the so called “periphery”. Considering the large contribution of these members to projects it would be worth investigating the contents of their messages in order to investigate the nature of their contribution (i.e. are they asking questions related to the project or are they offering more details regarding a bug?).

Another interesting finding revealed by this research is that most FLOSS activity did not show a consistent growth over time but displayed a somewhat more irregular pattern with spikes and troughs. Future research should investigate the reasons behind these irregularities. Identifying the factors behind these irregularities could prove valuable for FLOSS practitioners as they influence activity and thus participation. These factors could be simple, such as the release of a new version, or more complex such as the release of a competing product or even a marketing campaign.

One of the threats to the validity of this research is the fact that community members might be communicating on private channels (i.e. instant messaging services, IRC, Skype meetings, etc.) or conducting activities which are not tracked/documented. As a result, a direction for future studies would be using qualitative research in order to confirm the findings of this thesis by conducting interviews with key QA contributors or surveys. Furthermore, as previous studies have focused on the profiles of FLOSS code contributors, future research could investigate the profiles of QA contributors in order to investigate whether QA adoption has created new contribution opportunities for non-technical community members. Another limitation is the relatively small sample of FLOSS communities used for this research. As a result, future studies could confirm the findings by repeat this investigation on a larger or more varied dataset.

Future research can also focus on applying a more varied range of SNA techniques. The time frames chosen for the temporal analysis conducted in this research consisted of six-month windows. Potentially, six months can be a too wide of a time frame and thus future studies could consider smaller windows. Furthermore, in this research, only betweenness centrality values have been computed in order to determine the importance of various vertices in the diffusion of information within the network. Future research could attempt to compute for example, outdegree and indegree centrality values in order to compare central figures based on the number of people they connect to by sending messages with central figures based on the number of people they connect to by receiving messages. In other words, outdegree centrality might determine individuals who are looking to communicate with as many other individuals as possible as opposed to indegree centrality which might determine "popular" individuals who receive messages from a high number of community members. In this research, the information flows have been investigated from the perspective of betweenness centrality, however, future research should undertake a more detailed analysis of this aspect by identifying potential risks and weaknesses. Another aspect that should be investigated is the actual nature and particularities of FLOSS networks, for example determining if they are complex networks as they display some common characteristics such as structural complexity, network evolution, connection diversity, dynamical complexity and node diversity [81].

Appendix A

A.1 Preliminary Analysis

Table A.1: Top 100 FLOSS projects (www.ohloh.net) — general statistics

Project	Current contributors	Total contributors	Lines of code	Project Start
Mozilla Firefox	867	2,096	8,207,780	April, 2002
MySQL	101	1,180	1,423,109	July, 2000
PHP	166	873	3,782,756	November, 1997
Linux Kernel	2,973	10,665	15,382,092	February, 2002
Ubuntu	57	506	974,209	September, 1996
Apache OpenOffice	56	397	19,603,114	September, 2000
Thunderbird	171	726	1,179,797	March, 1998
GNOME	1,035	4,917	7,837,564	January, 1997
Debian GNU/Linux	1,060	4,501	67,996,161	April, 1996
PostgreSQL Database Server	13	61	659,766	July, 1996
WordPress	15	46	176,755	April, 2003
Inkscape	26	145	537,694	January, 2006
Perl	124	1,120	4,552,672	December, 1987
KDE	776	4,204	24,421,852	April, 1997
NetBeans IDE	97	672	9,816,195	January, 1999
VirtualBox OSE	2	8	3,367,817	October, 2002

Continued on next page

Project	Current contributors	Total contributors	Lines of code	Project Start
phpBB	45	133	423,481	February, 2001
MediaWiki	154	344	931,141	April, 2003
Drupal (core)	88	146	833,347	May, 2000
Amarok	49	385	264,833	September, 2003
GNU Compiler Collection	181	486	6,340,668	November, 1988
Python programming language	63	186	867,730	August, 1990
X.Org	180	1,055	2,238,940	August, 1999
Chromium (Google Chrome)	914	1,304	6,557,593	July, 2008
GDB	102	313	2,892,581	April, 1999
Wine	143	1,338	2,445,402	June, 1993
GTK+	211	947	737,749	November, 1997
Funambol Client for Mozilla Thunderbird	0	9	8,089	August, 2008
Apache HTTP Server	35	112	1,634,476	July, 1996
Subversion	39	169	491,569	March, 2000
Firebug	17	33	474,801	August, 2007
Bash	1	2	188,589	August, 1996
PuTTY	4	8	87,870	January, 1999
GIMP	92	474	726,078	January, 1997
phpMyAdmin	250	510	278,267	May, 2001
Vim	1	14	1,716,310	December, 1999
TortoiseSVN	8	163	252,739	April, 2003
Git	201	1,045	390,837	April, 2005
GNU grep	9	33	9,272	November, 1998
VLC media player	124	546	610,068	August, 1999

Continued on next page

Project	Current contributors	Total contributors	Lines of code	Project Start
sudo	1	4	95,772	February, 1993
GNU tar	4	18	20,460	November, 1990
Eclipse Platform Project	108	333	2,729,367	May, 2001
jQuery	81	214	25,926	March, 2006
OpenSSH	10	81	110,260	September, 1999
GNU Make	2	22	33,351	April, 1988
7-Zip	–	–	–	–
GNU Core Utilities	33	118	78,076	October, 1992
Wget	20	80	36,421	December, 1999
Pidgin	34	699	354,435	March, 2000
GNU GRUB	39	79	246,329	December, 2002
FileZilla	2	15	250,305	June, 2001
CakePHP	99	216	467,203	May, 2005
OpenSSL	5	17	423,088	December, 1998
GNU Screen	6	30	43,231	April, 2003
rsync	1	13	47,403	June, 1996
Notepad++	1	4	251,840	April, 2009
Apache Tomcat	14	46	4,874,751	May, 2001
Trac	11	36	76,182	August, 2003
man	–	–	–	–
Subclipse	4	25	141,904	June, 2003
GNU findutils	5	20	17,899	February, 1996
Junit	37	63	27,762	December, 2000
SQLite	0	25	135,231	May, 2000
MPlayer	18	134	451,705	February, 2001
bzip2	–	–	–	–
Apache Ant	5	48	262,186	January, 2000
ImageMagick	4	8	752,227	September, 2009
Samba	70	263	1,438,986	December, 2006
Wireshark	25	64	2,510,572	September, 1998
GnuPG	8	19	196,000	November, 1997

Continued on next page

Project	Current contributors	Total contributors	Lines of code	Project Start
Ruby	44	85	905,440	January, 1998
Web Developer (Browser Add-on)	1	3	56,683	January, 2004
GNU sed	3	11	23,538	October, 2004
Eclipse Java Development Tools (JDT)	37	112	2,087,369	May, 2001
Apache Maven 2	40	128	40,657,564	September, 2003
Hibernate	53	140	1,041,113	January, 2003
log4j	3	33	79,986	November, 2000
Common Unix Printing System (CUPS)	1	8	550,886	May, 2000
GNU Emacs	211	529	1,679,862	April, 1985
LaTeX	6	20	50,247	August, 1997
Audacity	10	59	161,441	November, 2001
Spring Framework	74	164	1,153,214	March, 2004
cURL	70	185	158,647	December, 1999
Django	178	221	154,263	July, 2005
Ruby on Rails	796	2,058	161,272	November, 2004
WinSCP	2	4	384,712	July, 2003
Mercurial	115	498	77,856	May, 2005
Nmap Security Scanner	17	45	13,428,827	December, 2006
Ffmpeg	260	908	646,540	December, 2000
Adblock Plus	3	11	15,309	January, 2006
Scripting Layer for Android	0	2	1,670,314	March, 2009

Continued on next page

Project	Current contributors	Total contributors	Lines of code	Project Start
GNU C Library	92	227	1,171,616	April, 1989
GNU Autoconf	12	100	38,351	March, 1992
GNU binutils	73	227	2,687,113	May, 1999
Postfix	–	–	–	–
Facebook Plugin for Pidgin	2	6	6,037	May, 2008
Cygwin	8	43	1,397,863	February, 2000
LAME (Lame Ain't an MP3 Encoder)	3	33	116,710	November, 1999
GNU Diff Utilities	6	16	9,668	July, 1988

Table A.2: Top 100 FLOSS projects (www.ohloh.net) — QA adoption

Project	QA	Description
Mozilla Firefox	Yes	QA dedicated mailing lists found.
MySQL	Yes	QA dedicated forum found. Job listings from 2006 for QA engineer positions.
PHP	Yes	QA dedicated mailing list found.
Linux Kernel	Yes	Kernel-testers mailing list found; posts are related to bugs. Relies on community of testers to encounter bugs.
Ubuntu	Yes	QA dedicated webpage and three mailing lists found.
Apache OpenOffice	Yes	QA dedicated mailing lists found (mailing lists could not be retrieved as link was not working).
Thunderbird	Yes	QA dedicated wiki, irc channel and team found.
GNOME	Yes	Bug squad dedicated communication channels found (listed as QA).
Debian GNU/Linux	Yes	Dedicated QA website and wiki describing specific activities found.

Continued on next page

Project	QA	Description
PostgreSQL Database Server	Yes	Testing dedicated mailing list for posting test results and potential issues.
WordPress	Yes	Testing dedicated mailing list found. Testers install nightly builds and perform testing tasks.
Inkscape	Yes	Testing dedicated mailing list found.
Perl	Yes	CPAN tester community contributes by testing. CPAN is comprehensive perl archive network (modules and extensions).
KDE	Yes	QA team revived in april 2012. Active in the early 2000s; later becoming inactive.
NetBeans IDE	Yes	NetCat community handles beta testing.
VirtualBox OSE	Yes	A large test laboratory at Oracle's facilities and a dedicated test team in 24/7 operation ensures that code quality remains excellent: dozens of test machines perform automated tests to spot regressions and monitor performance.
phpBB	Yes	QA mailing list, sub forum found. Not much activity on topic. Area 51 development board, phpBB's testing ground of bleeding edge developmental code, to discuss development and code changes, RFCs, future versions of phpBB, and also sneak peeks of currently available development versions of phpBB.
MediaWiki	Yes	QA dedicated wiki found. QA practices employed. QA dedicated mailing list was not found.
Drupal (core)	Yes	QA mailing list not found. QA has a dedicated website, group, etc.
Amarok	Yes	QA is recognised as a separate form of contribution. QA dedicated channels were not found.
GNU Compiler Collection	Yes	No dedicated QA mailing list found. Two mailing lists for submitting test results used by developers found.

Continued on next page

Project	QA	Description
Python programming language	Yes	No dedicated QA mailing list found. Testing list used by developers found.
X.Org	Yes	No dedicated QA mailing list found. Webpage describing testing methodologies found. Xtest is an extension.
Chromium (Google Chrome)	Yes	No dedicated QA mailing list found. Specific instructions for testers found http://www.chromium.org/getting-involved.
GDB	Yes	Test result reporting mailing list found. Used mainly by developers.
Wine	Yes	Test dedicated mailing list used mainly by developers to post test results. QA dedicated page found http://www.kegel.com/wine/qa/. No QA dedicated communication channel found.
GTK+	Yes	No dedicated QA mailing list found. 2–4 individuals are responsible for bug testing, triaging and patch testing: https://live.gnome.org/GtkTasks.
Funambol Client for Mozilla Thunderbird	–	Website not working.
Apache HTTP Server	No	All httpd release candidates are announced to a mailing list prior to acceptance. Subscribers are encouraged to test drive the candidates and submit feedback.
Subversion	No	No dedicated QA mailing list found. Users are urged to submit bug reports and help with triaging bugs.
Firebug	No	No dedicated QA mailing list found.
Bash	No	No dedicated QA mailing list found. Development, translating and maintenance groups are listed.
PuTTY	No	Website lists four individuals as developers. No dedicated QA mailing list found.

Continued on next page

Project	QA	Description
GIMP	No	No dedicated QA mailing list found. Documentation list found.
phpMyAdmin	No	No dedicated QA mailing list found. Community members are urged to test and open bug reports on the issue tracker.
Vim	No	No dedicated QA mailing list found.
TortoiseSVN	No	No dedicated QA mailing list found.
Git	No	No dedicated QA mailing list found.
GNU grep	No	No dedicated QA mailing list found.
VLC media player	No	No dedicated QA mailing list found. An attempt to implement QA was found—2006 (http://wiki.videolan.org/Quality , http://www.jbkempf.com/blog/post/2006/11/15 /VLC-QA-team). Forum threads related to testing were found (http://forum.videolan.org/).
sudo	No	No dedicated QA mailing list found.
GNU tar	No	No dedicated QA mailing list found.
Eclipse Platform Project	No	No dedicated QA mailing list found.
jQuery	No	No dedicated QA mailing list found.
OpenSSH	No	No dedicated QA mailing list found.
GNU Make	No	No dedicated QA mailing list found.
7-Zip	No	No dedicated QA mailing list found.
GNU Core Utilities	No	No dedicated QA mailing list found.
Wget	No	No dedicated QA mailing list found.
Pidgin	No	No dedicated QA mailing list found.
GNU GRUB	No	No dedicated QA mailing list found.
FileZilla	No	No dedicated QA mailing list found.
CakePHP	No	No dedicated QA mailing list found.
OpenSSL	No	No dedicated QA mailing list found.
GNU Screen	No	No dedicated QA mailing list found.
rsync	No	No dedicated QA mailing list found.
Notepad++	No	No dedicated QA mailing list found.
Apache Tomcat	No	No dedicated QA mailing list found.

Continued on next page

Project	QA	Description
Trac	No	No dedicated QA mailing list found.
man	No	No dedicated QA mailing list found.
Subclipse	No	No dedicated QA mailing list found.
GNU findutils	No	No dedicated QA mailing list found.
Junit	No	No dedicated QA mailing list found.
SQLite	No	No dedicated QA mailing list found.
MPlayer	No	No dedicated QA mailing list found.
bzip2	No	No dedicated QA mailing list found.
Apache Ant	No	No dedicated QA mailing list found.
ImageMagick	No	No dedicated QA mailing list found.
Samba	No	No dedicated QA mailing list found.
Wireshark	No	No dedicated QA mailing list found.
GnuPG	No	No dedicated QA mailing list found.
Ruby	No	No dedicated QA mailing list found.
Web Developer (Browser Add-on)	No	No dedicated QA mailing list found.
GNU sed	No	No dedicated QA mailing list found. Website does not provide a list of mailing lists.
Eclipse Java Development Tools (JDT)	No	No dedicated QA mailing list found.
Apache Maven 2	No	No dedicated QA mailing list found. Testing procedures are described. Community members are urged to join testing activities.
Hibernate	No	No dedicated QA mailing list found.
log4j	No	No dedicated QA mailing list found.
Common Unix Printing System (CUPS)	No	No dedicated QA mailing list found.
GNU Emacs	No	No dedicated QA mailing list found.
LaTeX	No	No dedicated QA mailing list found.
Audacity	No	No dedicated QA mailing list found.

Continued on next page

Project	QA	Description
Spring Framework	No	Dedicated QA team; No dedicated communication channel found.
cURL	No	No dedicated QA mailing list found.
Django	No	No dedicated QA mailing list found. Detailed info regarding bug triaging process (https://docs.djangoproject.com/en/dev/internals /contributing/triaging-tickets/#how-can-i-help-with-triaging). Unit tests are written when committing a new feature: https://docs.djangoproject.com/en/dev/internals /contributing/writing-code/unit-tests/
Ruby on Rails	No	No dedicated QA mailing list found. Community is encouraged to contribute also by testing patches.
WinSCP	No	No dedicated QA mailing list found. Project seems to be developed by one person with the exception of minor contributions, documentation and transalations (http://winscp.net/eng/docs/project_history).
Mercurial	No	No dedicated QA mailing list found. Patch contributors are encouraged to add test scripts.
Nmap Security Scanner	No	No dedicated QA mailing list found. Beta versions are released to the development list for QA purposes.
Ffmpeg	No	No dedicated QA mailing list found.
Adblock Plus	No	No dedicated QA mailing list found. Community is encouraged to test beta versions.
Scripting Layer for Android	No	No dedicated QA mailing list found.
GNU C Library	No	No dedicated QA mailing list found.
GNU Autoconf	No	No dedicated QA mailing list found.
GNU binutils	No	No dedicated QA mailing list found.

Continued on next page

Project	QA	Description
Postfix	No	No dedicated QA mailing list found.
Facebook Plugin for Pidgin	No	No dedicated QA mailing list found.
Cygwin	No	No dedicated QA mailing list found.
LAME (Lame Ain't an MP3 Encoder)	No	No dedicated QA mailing list found.
GNU Diff Utilities	No	No dedicated QA mailing list found. Community is urged to test releases.

Appendix B

B.1 Data Models

The Mozilla database has the simplest structure as it contains only three tables as follows:

- bugs—bugs with the associated metadata are saved with the appropriate metadata. The bug_id field is a primary key (the bug id is unique to each bug retrieved);
- bug_comments—comments posted for each bugs are saved with the appropriate metadata. The bug_id field references one instance (bug_id) from the bugs table; and
- mozdbqa—e-mails from QA mailing lists are saved with associated metadata. This table is not referenced and does not reference other tables.

The databases associated with the KDE, Ubuntu, LibreOffice and Plone projects are structured as follows:

- a bugs table—bugs with the associated metadata are saved with the appropriate metadata. The bug_id field is a primary key (the bug id is unique to each bug retrieved);
- a bug comments table—comments posted for each bugs are saved with the appropriate metadata. The bug_id field references one instance (bug_id) from the bugs table;
- a mailing lists table—e-mails from retrieved mailing lists are saved with associated metadata. This table is not referenced and does not reference other tables; and
- a contributors table—code contributor associated metadata such as nicknames and commits is saved. This table is not referenced and does not reference other tables.

Figure B.1: Mozilla Database Diagram

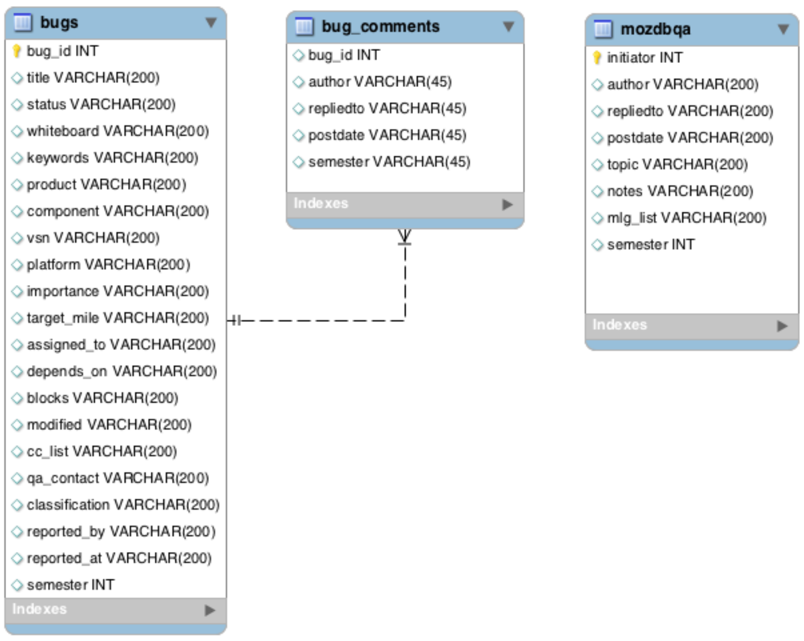


Figure B.2: KDE Database Diagram

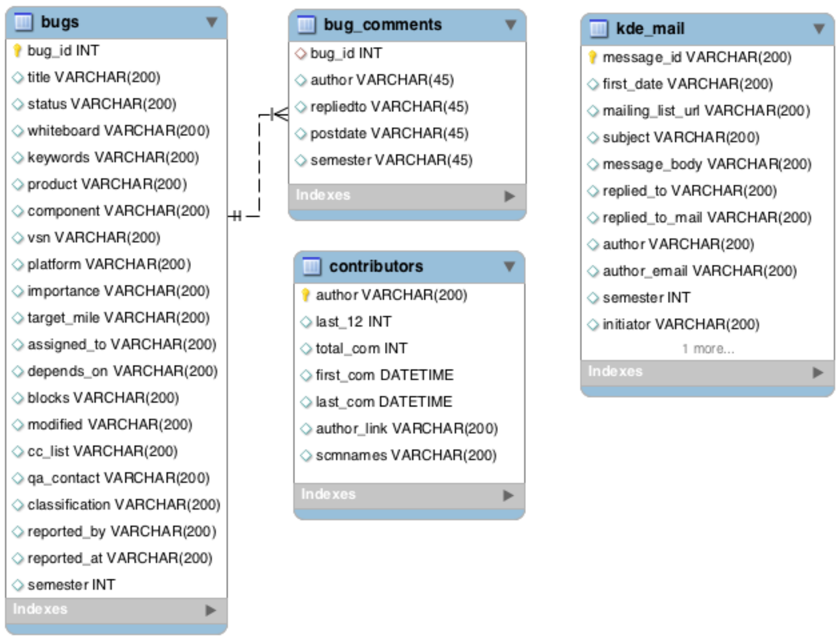


Figure B.3: LibreOffice Database Diagram

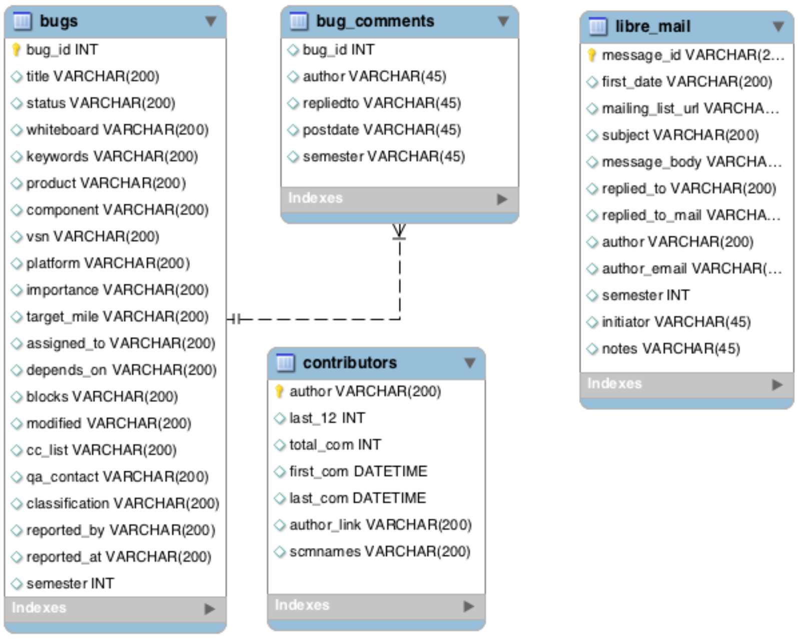


Figure B.4: Plone Database Diagram

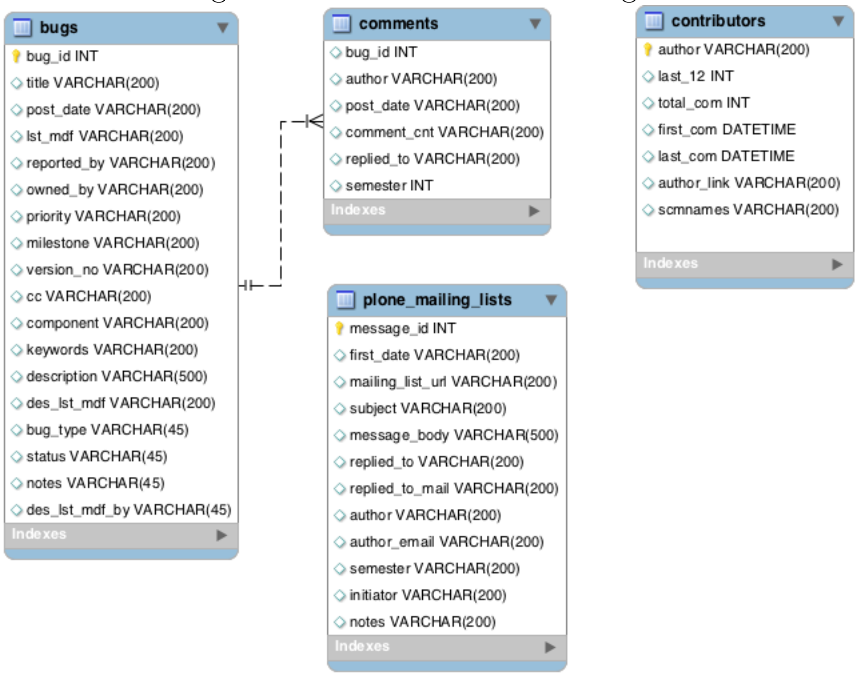
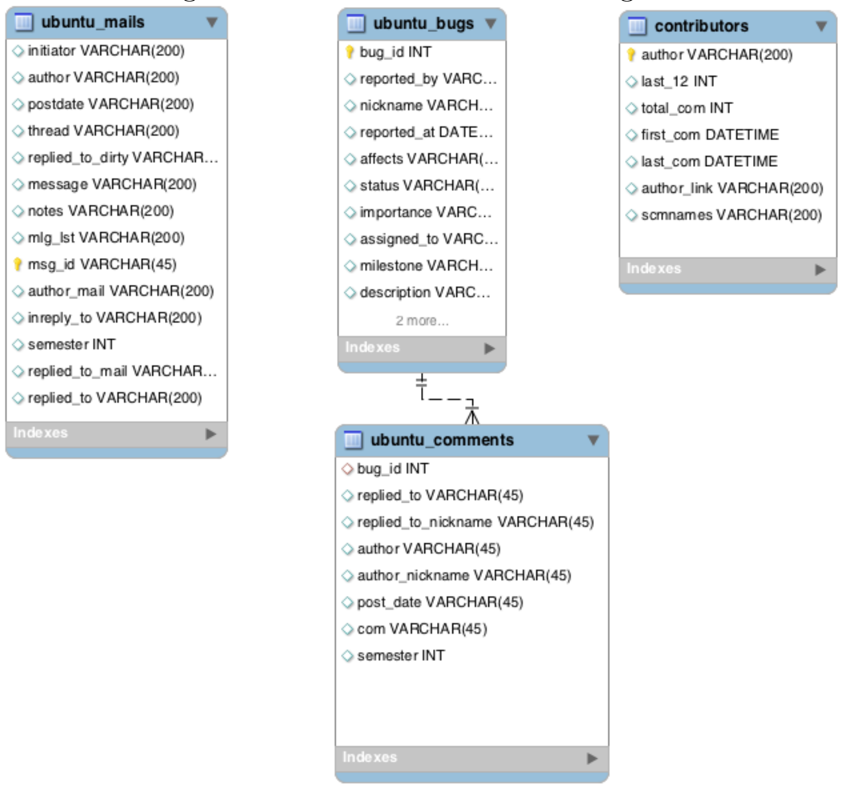


Figure B.5: Ubuntu Database Diagram



B.2 Datasets

Table B.3: Retrieved LibreOffice Mailing Lists

List Name	Analysed	Stored	Issues Encountered
Libre-graphics-meeting	474	474	–
LibreOffice	52,601	52,293	–
Libreoffice-bugs	122,307	122,294	–
Libreoffice-commits	Nan	12,603	Unknown error
Libreoffice-qa	4,608	3,381	–
Libreoffice-release	62	57	–
Libreoffice-ux-advise	2,072	1,607	–
Totals	182,124	192,709	*Nan*

Table B.4: Retrieved KDE Mailing Lists

List Name	Analysed	Stored	Issues Encountered
Active	6,164	6,157	–
Akademy-br	0	0	Unknown error
akunambol	30	30	–
Amarok	33,042	11,357	–
Amarok-bugs-dist	0	0	Private list
Amarok-private	0	0	Private list
Amarok-promo	1,418	1245	–
Bugsquad	637	635	–
Bugsquad-triage	0	0	Empty list
Calligra-author	0	0	Empty list
calligra-devel	9,377	9,343	–
Campkde-participants	6	6	–
Contour	0	0	Empty list
Digest	803	803	–
Digikam-devel	68,258	67,846	–
digikam-soc-devel	0	0	Private list
Digikam-users	0	4,850	Unknown error (data retrieved 01/2008)
FreeNX-kNX	9,882	9,836	–
Gluon	439	439	–
Gwenview-devel	0	0	Empty list
Inqlude	7	7	–
k3b	282	282	–
Kalzium	1,395	1,394	–
kbabel	881	862	–
kde	0	0	Tool not compatible with this format
kde-accessibility	0	0	Tool not compatible with this format
kde-announce	0	0	Tool not compatible with this format
Kde-announce-apps	5,009	4,854	–
kde-artists	0	0	Tool not compatible with this format

Continued on next page

List Name	Analysed	Stored	Issues Encountered
Kde-bindings	0	0	Tool not compatible with this format
Kde-br	0	0	Private list
Kde-bretzn	51	51	–
kde-bugs-dist	0	0	Tool not compatible with this format
Kde-buildsystem	9,254	9,220	–
kde-china	5,651	5,640	–
Kde-cl	267	267	–
kde-commits	0	0	Tool not compatible with this format
kde-community	32	32	–
kde-core-devel	0	0	Tool not compatible with this format
Kde-cvs-announce	110	110	–
Kde-dashboard	0	0	Empty list
kde-de	0	0	Tool not compatible with this format
kde-devel	0	0	Tool not compatible with this format
Kde-devel-es	1,739	1,723	–
kde-doc-english	0	0	Tool not compatible with this format
kde-docbook	0	0	Tool not compatible with this format
kde-edu	0	0	Tool not compatible with this format
Kde-edu-pt_br	0	0	Empty list
Kde-el	166	166	–
kde-embedded	19	19	–
Kde-ev-board	0	0	Private list
kde-ev-marketing	0	0	–
Kde-ev-patrons	0	0	Private list
Kde-ev-research	0	0	Private list
Kde-ev-sprint	0	0	Private list
Kde-ev-supporters	0	0	Private list

Continued on next page

List Name	Analysed	Stored	Issues Encountered
Kde-events	0	0	Tool not compatible with this format
Kde-events-fr	231	231	–
Kde-events-in	0	0	Private list
Kde-extra-gear	0	0	Tool not compatible with this format
Kde-finance-apps	668	667	–
Kde-frameworks-devel	2,625	2,489	–
kde-francophone	0	0	Tool not compatible with this format
kde-freebsd	0	13,921	Unknown error
kde-games-bugs	0	0	Empty list
kde-games-devel	0	0	–
Kde-graphics-devel	383	366	–
kde-guidelines	156	155	–
Kde-hardware-devel	2,213	2,179	–
kde-i18n-ca	0	0	Tool not compatible with this format
kde-i18n-de	0	0	Tool not compatible with this format
kde-i18n-doc	0	0	Tool not compatible with this format
kde-i18n-el	0	0	Empty list
Kde-i18n-eo	0	0	Tool not compatible with this format
Kde-i18n-fa	0	64	Unknown error
Kde-i18n-fr	0	0	Empty list
Kde-i18n-fry	229	228	–
Kde-i18n-it	0	0	Tool not compatible with this format
kde-i18n-lt	410	409	–
Kde-i18n-nds	292	286	–
Kde-i18n-nl	0	0	Tool not compatible with this format

Continued on next page

List Name	Analysed	Stored	Issues Encountered
Kde-i18n-no	73	72	–
Kde-i18n-pa	11	10	–
kde-i18n-pt	0	0	Unknown error
Kde-i18n-pt_br	0	0	Unknown error
kde-i18n-ro	608	574	–
Kde-i18n-sr	2	2	Unknown error
Kde-i18n-uk	56	56	Unknown error
Kde-i18n-vi	119	119	Unknown error
Kde-imaging	15,353	14,791	–
KDE-india	1,164	1,155	–
KDE-Italia	152	148	Unknown error
Kde-jp	836	621	–
KDE-Kamoso	36	36	–
kde-kiosk	0	0	Tool not compatible with this format
Kde-l10n-es	2,087	2,077	–
Kde-l10n-he	263	263	Unknown error
Kde-l10n-hu	45	45	Unknown error
Kde-l10n-ia	58	58	–
Kde-l10n-in	3	3	–
Kde-l10n-kn	237	237	–
Kde-l10n-si	3	3	–
Kde-l10n-sw	0	0	Empty list
kde-l10n-tr	202	202	–
Kde-l10n-vi	171	164	–
Kde-latam	44	44	–
Kde-licensing	0	0	Tool not compatible with this format
kde-linux	0	0	Tool not compatible with this format
KDE-Look	0	0	Tool not compatible with this format
kde-mac	847	798	–
Kde-metrics	0	0	Private list
kde-mexico	478	477	–
Kde-mobile	605	575	–

Continued on next page

List Name	Analysed	Stored	Issues Encountered
Kde-mobile-users	263	232	–
kde-multimedia	0	0	Tool not compatible with this format
kde-networkmanager	1,132	1,083	–
kde-nonlinux	0	0	Tool not compatible with this format
Kde-oldies	0	0	Private list
kde-openserver	1	0	–
Kde-partnership	0	0	Empty list
Kde-perl	1,668	1,663	–
kde-pim	0	0	Tool not compatible with this format
Kde-policies	0	0	Tool not compatible with this format
KDE-Press-Announce-PL	0	0	Private list
Kde-print-devel	2,973	2,970	–
kde-promo	0	0	Tool not compatible with this format
Kde-science	53	52	–
Kde-scm-interest	2,296	2,271	–
kde-sdk-devel	116	58	–
kde-services-devel	91	89	–
Kde-silk	78	78	–
Kde-soc	876	859	–
kde-solaris	0	0	Tool not compatible with this format
kde-sonnet	74	74	–
Kde-teaching	2	2	–
KDE-Telepathy	9,358	9,332	–
Kde-telepathy-bugs	3,727	3,727	–
Kde-testing	360	319	–
kde-usa	79	79	–
kde-usability	0	0	Tool not compatible with this format

Continued on next page

List Name	Analysed	Stored	Issues Encountered
Kde-usability-devel	630	619	–
Kde-utils-devel	982	978	–
Kde-uwg	10	10	–
Kde-ux-meeting	47	47	–
Kde-winbuild	6,982	6,935	–
Kde-windows	7,643	7,396	–
kde-women	0	0	Tool not compatible with this format
kde-www	0	0	Private list
Kdeev-books	0	0	Tool not compatible with this format
Kdelibs-bugs	3,524	3,524	–
Kdepim-bugs	85,384	85,377	–
Kdepim-builds	0	0	Empty list
Kdepim-maintainers	3	3	–
kdepim-users	0	0	Private list
KDevelop	17,996	17,382	–
KDevelop-devel	0	20,851	Unknown error
Kexi	985	963	–
Kexi-devel	125	112	–
Kexi-pl	0	0	Unknown error
kfm-devel	0	0	Tool not compatible with this format
Kget	4,798	4,794	–
kgraphviewer-devel	340	340	–
Khtml-cvs	0	0	Private list
kimageshop	11,897	11,697	–
Klink	138	138	–
KMyMoney	1,392	1,390	–
KMyMoney-devel	10,357	10,312	–
koffice-devel	0	0	Tool not compatible with this format
Kompare-devel	1,296	1,294	–
konq-e	0	0	Tool not compatible with this format

Continued on next page

List Name	Analysed	Stored	Issues Encountered
konsole-devel	0	0	Tool not compatible with this format
Konversation-devel	5,344	5,325	–
kopete-devel	0	0	Private list
Korganizer-devel	11,563	11,561	–
kpovmodeler-devel	0	0	Tool not compatible with this format
Ksecretservice-devel	148	138	–
Kst	21,303	21,288	–
Kstars-devel	0	0	Tool not compatible with this format
kwin	0	0	Tool not compatible with this format
KWrite-Devel	0	0	Tool not compatible with this format
Lokalize	6	6	–
Marble	53	53	–
Marble-bugs	2,772	2,772	–
Marble-commits	2,365	2,365	–
Marble-devel	4,280	4,233	–
Massif-visualizer	67	66	–
Necessitas-devel	1,417	1,416	–
Nepomuk	4,399	4,315	–
Nepomuk-bugs	1,649	1,649	–
Noatun-bugs	9	9	–
Okular-devel	15,095	15,073	–
Open-collaboration-services	73	73	–
Owncloud	9,285	9,268	–
Parley-devel	631	630	–
Phonon-backends	733	712	–
Plasma-bugs	0	0	private
Plasma-devel	25,407	24,778	–
Qtscript-bindings	137	125	–
Quanta	0	0	Tool not compatible with this format

Continued on next page

List Name	Analysed	Stored	Issues Encountered
quanta-devel	0	0	Tool not compatible with this format
Raptor	119	119	–
rekonq	4,219	4,201	–
release-team	6,988	6,534	–
Rocs-devel	0	0	Empty list
Social-Desktop	41	41	–
Solidkreator	6	5	–
Sysadmin	0	0	Private list
taglib-devel	2,489	2,479	–
tellico-users	898	897	–
Unassigned-bugs	29,231	29,223	–
WebKit-devel	2,395	2,385	–
www-de	0	0	Private list
www-pl	0	0	Private list
Zanshin-devel	0	0	Empty list
Totals	516,377	529,488	Error[1]

[1] *The difference in totals results from unknown errors that prevented from estimating the number of analysed e-mails for certain lists.*

Table B.5: Retrieved Ubuntu Mailing Lists

Mailing List Name	Number in DB	Analysed	Stored in DB
ubuntu-quality	1	3,679	3,679
ubuntu-bugsquad	2	4,001	3,796
laptop-testing-team	3	1,323	1,323
ubuntu-users	4	267,156	267,155
edubuntu-users	5	7,242	7,242
kubuntu-users	6	57,280	57,280
lubuntu-users	7	3,964	3,964
ubuntu-studio-users	8	8,932	8,923
xubuntu-users	9	5,326	5,324
ubuntu-devel	10	36,093	36,081
ubuntu-devel-discuss	11	13,946	13,945
edubuntu-devel	12	3,463	3,321

Continued on next page

Mailing List Name	Number in DB	Analysed	Stored in DB
edubuntu-devel-es	13	133	133
fwts-devel	14	3,119	3,119
kernel-team	15	25,033	24,359
kubuntu-devel	16	6,674	6,402
laptop-devel	17	40	40
mir-devel	18	128	128
ubuntu-accessibility-devel	19	890	866
ubuntu-app-devel	20	250	250
ubuntu-archive	21	46,089	1,168
ubuntu-desktop	22	3,729	3,571
ubuntu-distributed-devel	23	1,167	1,167
ubuntu-motu	24	6,536	6,534
ubuntu-mozillateam	25	1,257	1,257
ubuntu-release	26	2,140	2,140
ubuntu-server	27	6,010	6,010
ubuntu-studio-devel	28	4,687	4,366
ubuntu-utah-devel	29	93	93
upstart-devel	30	2,246	2,246
xubuntu-devel	31	8,576	7,811
community-web-projects	32	50	50
universe-bugs	33	191,307	37
desktop-bugs	34	88,777	22
kubuntu-bugs	35	128,056	13
kernel-bugs	36	153,143	8
ubuntu-accessibility-bugs	37	5,594	30
ubuntu-mozillateam-bugs	38	135,751	508
foundations-bugs	39	148,775	0
ubuntu-server-bugs	40	85,459	3,333
Totals		146,8114	487,694

Table B.6: Retrieved Plone Mailing Lists

List Name	Analysed	Stored	Issues Encountered
Plone-a11y	1	1	–
Plone-Board	0	0	Private list
Plone-com	277	277	–
Communications	0	0	Private list
Plone-conference	53	53	–
Diversity	0	0	Empty list
Doc-Editors	1,794	962	–
Educational	362	201	–
Enterprise	207	105	–
Environmental	72	36	–
Evangelism	2,455	1,296	–
Framework-Team	7,025	3,553	–
Framework41	0	0	Private list
Gsoc-mentors	0	0	Private list
GSOC-Students	150	75	–
Mailman	0	0	Empty list
Membership	0	0	Private list
Membership-Committee	0	0	Private list
NGO	1,555	734	–
P4u-webadmin	0	0	Private list
PLIP-Advisories	2,951	1,628	–
Plone-AsiaPacific	184	95	–
Plone-cat	87	48	–
Plone-ConoSur	3,731	1,969	–
Plone-cz	0	0	Error
Plone-EasternEurope	22	9	–
Plone-FR	2,087	1,363	–
Plone-Hungary	174	87	–
Plone-IT	0	0	Error
Plone-NL	111	59	–
Plone-pl	48	24	–
Plone-Scandinavia	42	21	–
Product-Developers	12,282	7,082	–

Continued on next page

List Name	Analysed	Stored	Issues Encountered
QA-Team	170	170	–
Plone-Roadmap	89	81	–
Scientific	78	32	–
Setup	11,455	5,614	–
Sprints	707	300	–
Testbot	0	0	Error
Tester	0	0	Empty list
Plone-testing-team	70	69	–
UI	2,859	1,565	–
Usergroups	99	45	–
Usuarios-Plone	1,387	833	–
ZopeSkel	323	201	–
Totals	52,907	28,588	*Nan*

Appendix C

C.3 Ubuntu

Table C.7: Ubuntu QA mailing list participants activity levels on other channels

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_1	2	227	3	358	0	No
X_2	21	127	7	0	0	No
X_3	5	0	0	0	0	No
X_4	1	0	0	0	0	No
X_5	4	2,360	12	1,137	37,689	Yes
X_6	4	0	0	0	0	No
X_7	1	0	0	0	0	No
X_8	1	0	0	0	0	No
X_9	2	2	1	1	0	No
X_{10}	2	3	2	0	0	No
X_{11}	1	131	10	540	0	No
X_{12}	1	1	1	146	0	No
X_{13}	1	0	0	0	0	No
X_{14}	1	0	0	0	0	No
X_{15}	1	4	1	59	0	No
X_{16}	37	41	3	232	0	No
X_{17}	3	9	2	0	0	No
X_{18}	1	0	0	4	0	No
X_{19}	3	158	7	49	0	No
X_{20}	1	0	0	0	0	No
X_{21}	1	0	0	0	0	No
X_{22}	1	1	1	12	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{23}	2	0	0	5	0	No
X_{24}	1	10	2	0	0	No
X_{25}	6	5	2	121	1,272	Yes
X_{26}	3	36	3	18	0	No
X_{27}	1	0	0	3	0	No
X_{28}	2	0	0	7	0	No
X_{29}	1	0	0	0	0	No
X_{30}	1	9	5	0	0	No
X_{31}	1	0	0	0	0	No
X_{32}	1	0	0	0	0	No
X_{33}	1	0	0	1	0	No
X_{34}	1	0	0	3	0	No
X_{35}	1	0	0	0	0	No
X_{36}	1	0	0	0	0	No
X_{37}	1	35	1	148	0	No
X_{38}	1	1	1	0	0	No
X_{39}	1	0	0	0	0	No
X_{40}	1	4,091	15	774	0	Yes
X_{41}	1	0	0	3	0	No
X_{42}	1	1	1	0	0	No
X_{43}	1	0	0	12	0	No
X_{44}	3	0	0	20	0	No
X_{45}	4	113	1	12	0	No
X_{46}	1	0	0	0	0	No
X_{47}	72	737	6	114	0	No
X_{48}	1	0	0	31	0	No
X_{49}	1	0	0	0	0	No
X_{50}	4	0	0	4	0	No
X_{51}	1	0	0	0	0	No
X_{52}	6	91	2	64	0	No
X_{53}	1	0	0	0	0	No
X_{54}	1	6	2	0	0	No
X_{55}	1	11	2	31	0	No
X_{56}	1	0	0	0	0	No
X_{57}	1	0	0	224	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{58}	3	0	0	0	0	No
X_{59}	2	83	3	14	0	No
X_{60}	1	147	10	233	733	Yes
X_{61}	3	2	1	67	0	No
X_{62}	1	10	5	63	0	No
X_{63}	1	0	0	0	0	No
X_{64}	1	0	0	0	0	No
X_{65}	35	0	0	151	0	No
X_{66}	1	1	1	22	0	No
X_{67}	1	0	0	0	0	No
X_{68}	7	4	2	314	0	No
X_{69}	40	928	7	220	0	No
X_{70}	3	128	2	0	0	No
X_{71}	3	28	4	296	0	No
X_{72}	1	0	0	0	0	No
X_{73}	1	0	0	0	0	No
X_{74}	1	0	0	0	0	No
X_{75}	7	0	0	0	0	No
X_{76}	3	0	0	15	0	No
X_{77}	3	32	4	773	0	No
X_{78}	1	0	0	0	0	No
X_{79}	1	105	7	362	2,258	Yes
X_{80}	2	0	0	0	0	No
X_{81}	2	0	0	4	0	No
X_{82}	1	0	0	0	0	No
X_{83}	1	227	5	495	0	No
X_{84}	1	0	0	6	0	No
X_{85}	2	115	2	88	0	No
X_{86}	1	0	0	0	0	No
X_{87}	2	25	2	0	0	No
X_{88}	7	0	0	0	0	No
X_{89}	1	0	0	38	0	No
X_{90}	1	2	1	432	0	No
X_{91}	3	0	0	47	0	No
X_{92}	2	0	0	3	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{93}	6	0	0	0	0	No
X_{94}	1	0	0	0	0	No
X_{95}	3	5	1	0	0	No
X_{96}	1	0	0	0	0	No
X_{97}	1	0	0	0	0	No
X_{98}	2	0	0	2	0	No
X_{99}	1	15	2	0	0	No
X_{100}	1	1	1	9	0	No
X_{101}	2	0	0	0	0	No
X_{102}	1	13	1	92	0	No
X_{103}	2	51	3	174	0	No
X_{104}	3	3	2	12	0	No
X_{105}	23	443	6	508	2,340	Yes
X_{106}	10	0	0	54	0	No
X_{107}	10	1	1	0	0	No
X_{108}	1	145	4	0	0	No
X_{109}	13	856	3	12	0	No
X_{110}	1	0	0	0	0	No
X_{111}	2	0	0	2	0	No
X_{112}	13	354	7	461	4,894	Yes
X_{113}	11	28	5	74	0	No
X_{114}	1	0	0	0	0	No
X_{115}	2	168	6	2,003	0	No
X_{116}	1	0	0	1	0	No
X_{117}	1	0	0	5	0	No
X_{118}	2	0	0	71	0	No
X_{119}	1	0	0	4	0	No
X_{120}	16	7	1	189	0	No
X_{121}	3	0	0	0	0	No
X_{122}	24	1	1	23	0	No
X_{123}	1	0	0	0	0	No
X_{124}	6	27	4	342	5,903	Yes
X_{125}	1	2	1	0	0	No
X_{126}	2	163	1	0	0	No
X_{127}	2	0	0	194	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{128}	1	3	1	24	0	No
X_{129}	3	0	0	0	0	No
X_{130}	1	0	0	13	0	No
X_{131}	1	0	0	0	0	No
X_{132}	1	0	0	0	0	No
X_{133}	1	120	2	15	0	No
X_{134}	1	0	0	1	0	No
X_{135}	1	0	0	0	0	No
X_{136}	2	0	0	0	0	No
X_{137}	1	0	0	0	0	No
X_{138}	1	3	1	0	0	No
X_{139}	1	0	0	0	0	No
X_{140}	1	1	1	0	0	No
X_{141}	1	0	0	0	0	No
X_{142}	1	246	5	0	0	No
X_{143}	2	11	3	8	0	No
X_{144}	1	33	6	0	0	No
X_{145}	1	0	0	10	0	No
X_{146}	1	31	2	0	0	No
X_{147}	3	27	6	39	0	No
X_{148}	29	10	1	121	0	No
X_{149}	1	3	1	0	0	No
X_{150}	7	0	0	0	0	No
X_{151}	1	0	0	0	0	No
X_{152}	5	0	0	0	0	No
X_{153}	1	0	0	0	0	No
X_{154}	1	1	1	2	0	No
X_{155}	16	52	2	37	0	No
X_{156}	2	0	0	0	0	No
X_{157}	1	0	0	0	0	No
X_{158}	1	0	0	18	0	No
X_{159}	4	0	0	0	0	No
X_{160}	27	13	5	259	1,742	Yes
X_{161}	1	0	0	0	0	No
X_{162}	1	0	0	0	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{163}	1	0	0	0	0	No
X_{164}	2	0	0	0	0	No
X_{165}	4	18	2	230	801	Yes
X_{166}	23	0	0	30	0	No
X_{167}	1	0	0	0	0	No
X_{168}	25	64	2	0	0	No
X_{169}	8	0	0	0	0	No
X_{170}	1	0	0	0	0	No
X_{171}	10	7	3	217	0	No
X_{172}	39	333	8	0	0	Yes
X_{173}	2	337	7	305	0	No
X_{174}	32	53	4	209	0	No
X_{175}	1	0	0	0	0	No
X_{176}	12	0	0	21	0	No
X_{177}	4	1	1	9	0	No
X_{178}	1	0	0	0	0	No
X_{179}	81	110	8	0	0	No
X_{180}	1	0	0	70	0	No
X_{181}	2	0	0	0	0	No
X_{182}	5	0	0	0	0	No
X_{183}	1	0	0	0	0	No
X_{184}	8	0	0	14	0	No
X_{185}	26	119	4	218	0	No
X_{186}	6	68	4	104	0	No
X_{187}	1	0	0	0	0	No
X_{188}	1	0	0	0	0	No
X_{189}	1	0	0	0	0	No
X_{190}	2	0	0	0	0	No
X_{191}	1	0	0	0	0	No
X_{192}	1	0	0	2	0	No
X_{193}	2	0	0	0	0	No
X_{194}	1	0	0	0	0	No
X_{195}	2	0	0	1	0	No
X_{196}	1	0	0	7	0	No
X_{197}	17	27	3	46	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{198}	2	0	0	0	0	No
X_{199}	1	0	0	3	0	No
X_{200}	2	0	0	0	0	No
X_{201}	2	0	0	0	0	No
X_{202}	1	0	0	0	0	No
X_{203}	2	1	1	181	0	No
X_{204}	1	305	1	0	0	No
X_{205}	1	43	4	65	0	No
X_{206}	1	0	0	0	0	No
X_{207}	13	0	0	139	0	No
X_{208}	1	0	0	0	0	No
X_{209}	1	0	0	0	0	No
X_{210}	1	8	3	153	0	No
X_{211}	1	0	0	0	0	No
X_{212}	6	195	2	0	0	No
X_{213}	39	4	2	34	0	No
X_{214}	1	0	0	0	0	No
X_{215}	1	0	0	0	0	No
X_{216}	3	0	0	0	0	No
X_{217}	1	0	0	0	0	No
X_{218}	1	0	0	0	0	No
X_{219}	4	0	0	0	0	No
X_{220}	17	24	2	55	0	No
X_{221}	1	0	0	0	0	No
X_{222}	5	0	0	0	0	No
X_{223}	1	320	2	24	0	No
X_{224}	1	0	0	0	0	No
X_{225}	2	0	0	0	0	No
X_{226}	14	45	7	561	0	No
X_{227}	1	46	3	213	0	No
X_{228}	1	0	0	0	0	No
X_{229}	3	0	0	0	0	No
X_{230}	2	0	0	34	0	No
X_{231}	1	1	1	0	0	No
X_{232}	1	0	0	0	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{233}	1	0	0	0	0	No
X_{234}	1	0	0	0	0	No
X_{235}	1	0	0	0	0	No
X_{236}	12	4	1	2	0	No
X_{237}	1	0	0	1	0	No
X_{238}	1	114	5	170	0	No
X_{239}	6	0	0	0	0	No
X_{240}	1	5	1	6	0	No
X_{241}	1	0	0	0	0	No
X_{242}	2	2	1	46	0	No
X_{243}	2	0	0	0	0	No
X_{244}	1	0	0	0	0	No
X_{245}	1	0	0	0	0	No
X_{246}	2	133	11	150	1,797	Yes
X_{247}	1	507	5	269	0	No
X_{248}	2	4	2	10	0	No
X_{249}	1	0	0	0	0	No
X_{250}	1	0	0	2	0	No
X_{251}	1	0	0	0	0	No
X_{252}	1	0	0	0	0	No
X_{253}	2	0	0	0	0	No
X_{254}	1	0	0	5	0	No
X_{255}	1	0	0	6	0	No
X_{256}	2	0	0	0	0	No
X_{257}	2	0	0	0	0	No
X_{258}	2	0	0	151	0	No
X_{259}	3	0	0	0	0	No
X_{260}	1	12	3	117	0	No
X_{261}	10	2	1	14	0	No
X_{262}	2	14	5	70	0	No
X_{263}	2	0	0	5	0	No
X_{264}	1	85	1	14	0	No
X_{265}	1	0	0	32	0	No
X_{266}	1	0	0	0	0	No
X_{267}	2	2	1	10	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{268}	1	0	0	1	0	No
X_{269}	1	0	0	0	0	No
X_{270}	8	83	3	0	0	No
X_{271}	2	1	1	7	0	No
X_{272}	2	0	0	0	0	No
X_{273}	1	0	0	0	0	No
X_{274}	3	51	2	149	0	No
X_{275}	3	87	2	0	0	No
X_{276}	2	0	0	3	0	No
X_{277}	1	0	0	0	0	No
X_{278}	2	0	0	0	0	No
X_{279}	2	0	0	0	0	No
X_{280}	2	0	0	1	0	No
X_{281}	5	41	3	4	0	No
X_{282}	3	4	1	107	0	No
X_{283}	3	0	0	0	0	No
X_{284}	9	0	0	25	0	No
X_{285}	1	1	1	1	0	No
X_{286}	1	0	0	160	0	No
X_{287}	1	1	1	0	0	No
X_{288}	2	0	0	11	0	No
X_{289}	2	19	1	0	0	No
X_{290}	1	0	0	0	0	No
X_{291}	1	3	1	0	0	No
X_{292}	1	1	1	0	0	No
X_{293}	1	0	0	0	0	No
X_{294}	2	0	0	7	0	No
X_{295}	1	0	0	0	0	No
X_{296}	1	20	7	3	0	No
X_{297}	1	0	0	0	0	No
X_{298}	5	107	3	83	0	No
X_{299}	2	0	0	4	0	No
X_{300}	1	0	0	0	0	No
X_{301}	2	10	4	0	0	No
X_{302}	1	0	0	0	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{303}	1	0	0	0	0	No
X_{304}	2	297	11	890	12,030	Yes
X_{305}	1	0	0	6	0	No
X_{306}	1	0	0	0	0	No
X_{307}	1	0	0	0	0	No
X_{308}	1	0	0	0	0	No
X_{309}	3	0	0	0	0	No
X_{310}	1	1	1	0	0	No
X_{311}	1	0	0	0	0	No
X_{312}	3	214	7	1,284	0	No
X_{313}	2	3	2	17	0	No
X_{314}	3	194	4	170	0	No
X_{315}	3	0	0	0	0	No
X_{316}	1	0	0	0	0	No
X_{317}	1	0	0	0	0	No
X_{318}	41	0	0	37	0	No
X_{319}	20	0	0	0	0	No
X_{320}	1	26	1	3	0	No
X_{321}	1	0	0	0	0	No
X_{322}	1	0	0	3	0	No
X_{323}	1	0	0	0	0	No
X_{324}	1	0	0	0	0	No
X_{325}	5	14	2	1	0	No
X_{326}	1	2	1	104	0	No
X_{327}	2	0	0	4	0	No
X_{328}	14	398	2	79	0	No
X_{329}	6	0	0	51	0	No
X_{330}	1	0	0	0	0	No
X_{331}	1	0	0	1	0	No
X_{332}	1	0	0	0	0	No
X_{333}	2	0	0	0	0	No
X_{334}	36	25	5	829	0	No
X_{335}	1	0	0	0	0	No
X_{336}	5	0	0	0	0	No
X_{337}	3	0	0	0	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{338}	1	0	0	2	0	No
X_{339}	1	0	0	0	0	No
X_{340}	1	0	0	0	0	No
X_{341}	6	9	3	79	0	No
X_{342}	1	1	1	0	0	No
X_{343}	4	0	0	15	0	No
X_{344}	1	14	1	4	0	No
X_{345}	1	0	0	0	0	No
X_{346}	1	0	0	5	0	No
X_{347}	3	0	0	0	0	No
X_{348}	1	0	0	0	0	No
X_{349}	4	38	5	153	0	No
X_{350}	2	0	0	7	0	No
X_{351}	1	0	0	0	0	No
X_{352}	456	167	9	845	24,231	Yes
X_{353}	1	0	0	1	0	No
X_{354}	1	0	0	0	0	No
X_{355}	1	0	0	0	0	No
X_{356}	1	0	0	0	0	No
X_{357}	2	0	0	0	0	No
X_{358}	2	141	4	0	0	No
X_{359}	1	0	0	3	0	No
X_{360}	1	2	2	1	0	No
X_{361}	15	1	1	0	0	No
X_{362}	1	0	0	0	0	No
X_{363}	2	0	0	21	0	No
X_{364}	1	0	0	0	0	No
X_{365}	1	347	2	0	0	No
X_{366}	1	0	0	0	0	No
X_{367}	7	0	0	0	0	No
X_{368}	2	0	0	0	0	No
X_{369}	1	0	0	0	0	No
X_{370}	1	0	0	0	0	No
X_{371}	2	0	0	5	0	No
X_{372}	1	0	0	0	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{373}	1	0	0	0	0	No
X_{374}	1	0	0	0	0	No
X_{375}	16	12	3	55	0	No
X_{376}	3	0	0	0	0	No
X_{377}	4	828	10	395	0	No
X_{378}	1	0	0	0	0	No
X_{379}	1	2	1	0	0	No
X_{380}	1	0	0	2	0	No
X_{381}	1	0	0	0	0	No
X_{382}	1	0	0	0	0	No
X_{383}	1	0	0	0	0	No
X_{384}	2	0	0	0	0	No
X_{385}	1	0	0	6	0	No
X_{386}	1	3	2	1	0	No
X_{387}	1	28	1	6	0	No
X_{388}	1	0	0	0	0	No
X_{389}	1	0	0	0	0	No
X_{390}	17	27	3	1	0	No
X_{391}	2	7	2	0	0	No
X_{392}	1	0	0	0	0	No
X_{393}	1	0	0	0	0	No
X_{394}	1	0	0	0	0	No
X_{395}	3	6	3	0	0	No
X_{396}	1	0	0	0	0	No
X_{397}	2	0	0	0	0	No
X_{398}	1	4	1	0	0	No
X_{399}	1	0	0	0	0	No
X_{400}	1	401	6	172	0	No
X_{401}	1	0	0	0	0	No
X_{402}	3	62	4	60	0	No
X_{403}	4	0	0	0	0	No
X_{404}	3	74	3	41	0	No
X_{405}	4	0	0	7	0	No
X_{406}	1	0	0	0	0	No
X_{407}	1	0	0	0	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{408}	1	0	0	0	0	No
X_{409}	1	0	0	16	0	No
X_{410}	2	0	0	0	0	No
X_{411}	7	170	10	524	0	No
X_{412}	1	2	1	4	0	No
X_{413}	2	0	0	4	0	No
X_{414}	1	44	3	0	0	No
X_{415}	1	0	0	0	0	No
X_{416}	3	0	0	0	0	No
X_{417}	1	0	0	6	0	No
X_{418}	1	0	0	0	0	No
X_{419}	3	4	1	25	0	No
X_{420}	2	415	4	54	0	No
X_{421}	22	410	12	265	0	No
X_{422}	1	0	0	0	0	No
X_{423}	3	0	0	17	0	No
X_{424}	1	0	0	0	0	No
X_{425}	1	0	0	0	0	No
X_{426}	36	1,199	7	182	0	No
X_{427}	5	2	1	0	0	No
X_{428}	1	0	0	0	0	No
X_{429}	6	1	1	155	0	No
X_{430}	1	2	2	1	0	No
X_{431}	3	7	3	22	0	No
X_{432}	3	0	0	0	0	No
X_{433}	1	0	0	3	0	No
X_{434}	1	36	3	0	2,015	No
X_{435}	1	0	0	0	0	No
X_{436}	1	44	3	0	0	No
X_{437}	2	1,440	9	89	0	No
X_{438}	1	6	2	35	0	No
X_{439}	2	0	0	6	0	No
X_{440}	20	0	0	84	0	No
X_{441}	2	0	0	11	0	No
X_{442}	1	2	1	32	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{443}	1	0	0	0	0	No
X_{444}	7	0	0	0	0	No
X_{445}	4	3	1	13	0	No
X_{446}	4	575	11	719	14,224	Yes
X_{447}	1	0	0	0	0	No
X_{448}	4	0	0	0	0	No
X_{449}	55	260	12	1,075	0	No
X_{450}	3	0	0	0	0	No
X_{451}	1	1	1	0	0	No
X_{452}	1	1	1	0	0	No
X_{453}	2	191	6	181	0	No
X_{454}	1	0	0	0	0	No
X_{455}	4	4	1	0	0	No
X_{456}	1	16	1	2	0	No
X_{457}	5	2	1	0	0	No
X_{458}	2	92	3	0	0	No
X_{459}	1	1	1	0	0	No
X_{460}	11	0	0	0	0	No
X_{461}	1	92	5	297	0	No
X_{462}	7	1	1	0	0	No
X_{463}	2	0	0	0	0	No
X_{464}	1	0	0	0	0	No
X_{465}	10	2	1	0	0	No
X_{466}	2	0	0	0	0	No
X_{467}	1	0	0	0	0	No
X_{468}	1	229	4	0	0	No
X_{469}	5	68	2	95	0	No
X_{470}	1	131	5	154	0	No
X_{471}	1	0	0	0	0	No
X_{472}	3	3	1	178	0	No
X_{473}	2	0	0	0	0	No
X_{474}	1	0	0	2	0	No
X_{475}	1	0	0	0	0	No
X_{476}	2	0	0	0	0	No
X_{477}	2	1	1	0	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{478}	25	0	0	49	0	No
X_{479}	249	12	1	0	0	No
X_{480}	2	0	0	0	0	No
X_{481}	3	160	6	632	0	No
X_{482}	1	0	0	0	0	No
X_{483}	1	12	1	14	0	No
X_{484}	4	8	2	0	0	No
X_{485}	2	7	2	0	0	No
X_{486}	4	5	2	9	0	No
X_{487}	1	0	0	0	0	No
X_{488}	10	1	1	0	0	No
X_{489}	1	0	0	0	0	No
X_{490}	6	0	0	0	0	No
X_{491}	1	0	0	0	0	No
X_{492}	1	164	1	1	0	No
X_{493}	1	0	0	0	0	No
X_{494}	3	0	0	0	0	No
X_{495}	1	1	1	0	0	No
X_{496}	1	119	5	144	0	No
X_{497}	3	39	6	0	0	No
X_{498}	2	57	6	10	0	No
X_{499}	2	1	1	68	0	No
X_{500}	1	0	0	2	0	No
X_{501}	1	0	0	0	0	No
X_{502}	2	0	0	5	0	No
X_{503}	1	0	0	0	0	No
X_{504}	2	0	0	0	0	No
X_{505}	5	12	3	91	0	No
X_{506}	12	0	0	0	0	No
X_{507}	1	0	0	0	0	No
X_{508}	1	0	0	0	0	No
X_{509}	1	13	1	0	0	No
X_{510}	1	0	0	0	0	No
X_{511}	1	111	2	9	0	No
X_{512}	1	4	3	2	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{513}	1	0	0	0	0	No
X_{514}	1	1	1	0	0	No
X_{515}	1	0	0	0	0	No
X_{516}	3	0	0	25	0	No
X_{517}	22	26	1	0	0	No
X_{518}	1	0	0	0	0	No
X_{519}	7	1	1	13	0	No
X_{520}	2	0	0	0	0	No
X_{521}	1	0	0	2	0	No
X_{522}	13	0	0	0	0	No
X_{523}	1	45	5	119	0	No
X_{524}	7	3	2	0	0	No
X_{525}	4	1	1	171	0	No
X_{526}	1	0	0	0	0	No
X_{527}	1	0	0	0	0	No
X_{528}	1	0	0	0	0	No
X_{529}	1	0	0	0	0	No
X_{530}	3	122	4	63	0	No
X_{531}	1	5	2	0	0	No
X_{532}	2	0	0	0	0	No
X_{533}	2	0	0	0	0	No
X_{534}	3	0	0	0	0	No
X_{535}	1	0	0	18	0	No
X_{536}	35	7	2	28	0	No
X_{537}	2	3	2	0	0	No
X_{538}	1	0	0	0	0	No
X_{539}	3	0	0	0	0	No
X_{540}	1	43	4	791	3,783	Yes
X_{541}	2	0	0	0	0	No
X_{542}	1	0	0	0	0	No
X_{543}	3	0	0	0	0	No
X_{544}	2	0	0	0	0	No
X_{545}	1	0	0	13	0	No
X_{546}	2	0	0	0	0	No
X_{547}	1	19	2	4	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{548}	1	0	0	0	0	No
X_{549}	1	0	0	5	0	No
X_{550}	1	0	0	0	0	No
X_{551}	3	0	0	0	0	No
X_{552}	2	68	2	29	0	No
X_{553}	1	0	0	0	0	No
X_{554}	9	44	4	31	0	No
X_{555}	19	21	3	53	0	No
X_{556}	6	1	1	316	0	No
X_{557}	1	0	0	0	0	No
X_{558}	21	0	0	0	0	No
X_{559}	1	0	0	0	0	No
X_{560}	1	0	0	0	0	No
X_{561}	11	0	0	0	0	No
X_{562}	2	0	0	6	0	No
X_{563}	2	0	0	8	0	No
X_{564}	1	0	0	6	0	No
X_{565}	38	226	2	0	0	No
X_{566}	2	0	0	0	0	No
X_{567}	1	0	0	0	0	No
X_{568}	1	2	1	202	0	No
X_{569}	5	606	3	0	0	No
X_{570}	2	0	0	0	0	No
X_{571}	6	5	2	44	0	No
X_{572}	1	0	0	6	0	No
X_{573}	1	0	0	0	0	No
X_{574}	3	0	0	0	0	No
X_{575}	2	0	0	0	0	No
X_{576}	1	0	0	0	0	No
X_{577}	1	0	0	0	0	No
X_{578}	1	6	1	12	0	No
X_{579}	1	9	3	35	0	No
X_{580}	2	3	1	0	0	No
X_{581}	2	0	0	0	0	No
X_{582}	1	0	0	0	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{583}	11	2	1	201	0	No
X_{584}	1	1	1	0	0	No
X_{585}	1	0	0	0	0	No
X_{586}	53	2,720	12	1,540	9,290	Yes
X_{587}	1	0	0	0	0	No
X_{588}	1	41	3	0	0	No
X_{589}	10	100	6	426	8,001	Yes
X_{590}	2	35	1	491	0	No
X_{591}	2	93	5	181	0	No
X_{592}	3	1	1	73	0	No
X_{593}	3	0	0	0	0	No
X_{594}	2	0	0	0	0	No
X_{595}	1	0	0	0	0	No
X_{596}	3	0	0	3	0	No
X_{597}	1	88	1	0	0	No
X_{598}	1	1	1	0	0	No
X_{599}	1	168	4	36	0	No
X_{600}	1	0	0	0	0	No
X_{601}	1	31	2	0	0	No
X_{602}	1	0	0	0	0	No
X_{603}	2	0	0	4	0	No
X_{604}	1	0	0	2	0	No
X_{605}	1	0	0	0	0	No
X_{606}	7	3	2	0	0	No
X_{607}	2	153	3	25	0	No
X_{608}	1	0	0	3	0	No
X_{609}	1	0	0	0	0	No
X_{610}	1	0	0	0	0	No
X_{611}	3	3	1	0	0	No
X_{612}	8	2	2	25	0	No
X_{613}	1	0	0	0	0	No
X_{614}	1	0	0	0	0	No
X_{615}	1	0	0	0	0	No
X_{616}	1	0	0	0	0	No
X_{617}	1	4	2	1	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{618}	7	0	0	0	0	No
X_{619}	15	0	0	6	0	No
X_{620}	10	79	5	55	2,597	Yes
X_{621}	14	0	0	0	0	No
X_{622}	2	0	0	0	0	No
X_{623}	1	0	0	0	0	No
X_{624}	201	24	7	401	0	No
X_{625}	1	0	0	58	0	No
X_{626}	1	0	0	0	0	No
X_{627}	1	0	0	0	0	No
X_{628}	10	7	4	0	0	No
X_{629}	7	0	0	0	0	No
X_{630}	1	6	2	6	0	No
X_{631}	2	0	0	0	0	No
X_{632}	1	16	1	1	0	No
X_{633}	3	0	0	7	0	No
X_{634}	16	5	1	0	0	No
X_{635}	2	0	0	0	0	No
X_{636}	2	0	0	9	0	No
X_{637}	1	0	0	0	0	No
X_{638}	1	0	0	0	0	No
X_{639}	3	0	0	0	0	No
X_{640}	1	0	0	0	0	No
X_{641}	3	0	0	0	0	No
X_{642}	1	0	0	0	0	No
X_{643}	1	0	0	0	0	No
X_{644}	3	0	0	0	0	No
X_{645}	3	7	4	0	0	No
X_{646}	1	0	0	1	0	No
X_{647}	1	6	1	7	0	No
X_{648}	2	0	0	0	0	No
X_{649}	1	28	5	247	0	No
X_{650}	42	0	0	0	0	No
X_{651}	1	0	0	4	0	No
X_{652}	5	3	1	0	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{653}	1	0	0	0	0	No
X_{654}	1	0	0	0	0	No
X_{655}	4	0	0	0	0	No
X_{656}	1	0	0	12	0	No
X_{657}	18	6	4	2	0	No
X_{658}	1	354	5	32	0	No
X_{659}	1	28	2	0	0	No
X_{660}	2	750	3	502	0	No
X_{661}	2	0	0	2	0	No
X_{662}	1	48	3	59	0	No
X_{663}	1	0	0	0	0	No
X_{664}	1	0	0	0	0	No
X_{665}	1	0	0	0	0	No
X_{666}	1	0	0	0	0	No
X_{667}	1	0	0	0	0	No
X_{668}	1	0	0	0	0	No
X_{669}	1	0	0	0	0	No
X_{670}	2	0	0	0	0	No
X_{671}	8	0	0	0	0	No
X_{672}	1	0	0	0	0	No
X_{673}	2	0	0	0	0	No
X_{674}	1	0	0	0	0	No
X_{675}	1	0	0	0	0	No
X_{676}	3	0	0	26	0	No
X_{677}	5	0	0	0	0	No
X_{678}	1	0	0	0	0	No
X_{679}	1	0	0	13	0	No
X_{680}	1	0	0	0	0	No
X_{681}	2	220	5	487	0	No
X_{682}	1	0	0	0	0	No
X_{683}	1	473	9	589	0	No
X_{684}	1	94	6	191	0	No
X_{685}	1	86	5	112	0	No
X_{686}	2	0	0	0	0	No
X_{687}	3	36	2	1	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{688}	18	208	8	0	0	No
X_{689}	3	49	5	0	0	No
X_{690}	2	50	4	0	0	No
X_{691}	76	18	1	0	0	No
X_{692}	4	0	0	4	0	No
X_{693}	1	0	0	0	0	No
X_{694}	1	0	0	0	0	No
X_{695}	1	0	0	0	0	No
X_{696}	1	0	0	0	0	No
X_{697}	1	0	0	0	0	No
X_{698}	1	0	0	0	0	No
X_{699}	2	0	0	3	0	No
X_{700}	7	4	2	12	0	No
X_{701}	3	0	0	9	0	No
X_{702}	1	0	0	0	0	No
X_{703}	2	0	0	0	0	No
X_{704}	2	0	0	0	0	No
X_{705}	1	4	1	0	0	No
X_{706}	1	1	1	5	0	No
X_{707}	1	0	0	0	0	No
X_{708}	1	0	0	2	0	No
X_{709}	4	9	2	30	0	No
X_{710}	5	0	0	5	0	No
X_{711}	1	11	3	0	0	No
X_{712}	1	2,674	18	833	22,959	Yes
X_{713}	1	0	0	0	0	No
X_{714}	1	0	0	4	0	No
X_{715}	1	0	0	7	0	No
X_{716}	1	0	0	0	0	No
X_{717}	1	0	0	1	0	No
X_{718}	1	0	0	0	0	No
X_{719}	1	0	0	0	0	No
X_{720}	1	32	3	0	0	No
X_{721}	34	607	11	507	3,868	Yes
X_{722}	1	0	0	1	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{723}	126	149	5	0	0	No
X_{724}	1	0	0	0	0	No
X_{725}	2	1	1	84	0	No
X_{726}	33	40	1	0	0	No
X_{727}	2	2	1	3	0	No
X_{728}	2	0	0	4	0	No
X_{729}	1	0	0	0	0	No
X_{730}	1	2	2	120	0	No
X_{731}	1	0	0	0	0	No
X_{732}	1	0	0	0	0	No
X_{733}	1	0	0	0	0	No
X_{734}	2	0	0	0	0	No
X_{735}	1	0	0	0	0	No
X_{736}	4	0	0	0	0	No
X_{737}	13	74	2	8	0	No
X_{738}	9	14	1	179	0	No
X_{739}	5	0	0	0	0	No
X_{740}	3	0	0	0	0	No
X_{741}	2	4	1	0	0	No
X_{742}	2	6	6	0	0	No
X_{743}	2	0	0	0	0	No
X_{744}	2	22	2	0	0	No
X_{745}	1	0	0	0	0	No
X_{746}	1	0	0	0	0	No
X_{747}	1	0	0	0	0	No
X_{748}	1	0	0	1	0	No
X_{749}	1	0	0	0	0	No
X_{750}	2	34	2	12	0	No
X_{751}	1	0	0	0	0	No
X_{752}	7	0	0	161	0	No
X_{753}	3	1	1	0	0	No
X_{754}	1	0	0	0	0	No
X_{755}	12	276	2	28	0	No
X_{756}	1	0	0	0	0	No
X_{757}	16	0	0	0	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{758}	1	13	5	29	0	No
X_{759}	2	0	0	10	0	No
X_{760}	1	0	0	11	0	No
X_{761}	1	0	0	0	0	No
X_{762}	11	0	0	34	0	No
X_{763}	2	11	1	1	0	No
X_{764}	1	5	2	14	0	No
X_{765}	1	2	1	0	0	No
X_{766}	2	57	3	261	0	No
X_{767}	5	26	5	115	0	No
X_{768}	1	0	0	1	0	No
X_{769}	5	140	4	38	0	No
X_{770}	1	0	0	0	0	No
X_{771}	1	0	0	0	0	No
X_{772}	66	30	4	45	0	No
X_{773}	4	8	2	0	0	No
X_{774}	1	0	0	0	0	No
X_{775}	2	0	0	0	0	No
X_{776}	1	0	0	0	0	No
X_{777}	1	0	0	2	0	No
X_{778}	1	0	0	1	0	No
X_{779}	6	1	1	0	0	No
X_{780}	1	1	1	5	0	No
X_{781}	15	27	4	131	0	No
X_{782}	19	46	2	75	0	No
X_{783}	1	0	0	0	0	No
X_{784}	4	0	0	0	0	No
X_{785}	1	0	0	0	0	No
X_{786}	1	0	0	1	0	No
X_{787}	1	0	0	0	0	No
X_{788}	2	35	2	6	0	No
X_{789}	1	0	0	0	0	No
X_{790}	1	0	0	1	0	No
X_{791}	6	0	0	0	0	No
X_{792}	1	0	0	0	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{793}	1	1	1	0	0	No
X_{794}	1	0	0	0	0	No
X_{795}	7	17	2	0	0	No
X_{796}	3	0	0	0	0	No
X_{797}	1	0	0	0	0	No
X_{798}	1	266	11	96	0	No
X_{799}	1	0	0	3	0	No
X_{800}	1	0	0	0	0	No
X_{801}	3	385	1	64	0	No
X_{802}	4	16	1	0	0	No
X_{803}	2	0	0	0	0	No
X_{804}	2	75	7	40	0	No
X_{805}	3	0	0	0	0	No
X_{806}	1	0	0	0	0	No
X_{807}	1	0	0	51	0	No
X_{808}	4	255	7	273	0	No
X_{809}	1	0	0	0	0	No
X_{810}	4	15	3	97	0	No
X_{811}	2	0	0	0	0	No
X_{812}	1	0	0	0	0	No
X_{813}	1	85	2	20	0	No
X_{814}	1	0	0	1	0	No
X_{815}	3	71	3	378	0	No
X_{816}	1	0	0	13	0	No
X_{817}	1	0	0	0	0	No
X_{818}	20	10	3	70	0	No
X_{819}	1	0	0	0	0	No
X_{820}	1	0	0	0	0	No
X_{821}	1	1	1	28	0	No
X_{822}	8	557	5	0	0	No
X_{823}	1	0	0	0	0	No
X_{824}	2	0	0	0	0	No
X_{825}	1	60	2	69	0	No
X_{826}	1	4	2	12	0	No
X_{827}	1	0	0	0	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{828}	1	338	5	0	0	No
X_{829}	1	0	0	0	0	No
X_{830}	1	0	0	0	0	No
X_{831}	1	7	2	0	0	No
X_{832}	1	3	1	0	0	No
X_{833}	1	2	1	27	0	No
X_{834}	4	0	0	6	0	No
X_{835}	1	0	0	0	0	No
X_{836}	5	46	3	183	0	No
X_{837}	1	0	0	9	0	No
X_{838}	1	0	0	0	0	No
X_{839}	1	174	11	244	0	No
X_{840}	1	0	0	6	0	No
X_{841}	1	1	1	8	0	No
X_{842}	4	31	7	214	0	No
X_{843}	1	0	0	0	0	No
X_{844}	23	942	8	826	82,719	Yes
X_{845}	3	24	4	1	0	No
X_{846}	1	0	0	0	0	No
X_{847}	1	5	1	14	0	No
X_{848}	2	11	3	172	0	No
X_{849}	2	0	0	0	0	No
X_{850}	1	0	0	0	0	No
X_{851}	1	0	0	16	0	No
X_{852}	2	0	0	0	0	No
X_{853}	1	31	4	58	0	No
X_{854}	1	0	0	0	0	No
X_{855}	4	0	0	5	0	No
X_{856}	16	405	2	0	0	No
X_{857}	1	2	2	0	0	No
X_{858}	1	0	0	85	0	No
X_{859}	5	145	7	384	0	No
X_{860}	1	0	0	0	0	No
X_{861}	1	0	0	0	0	No
X_{862}	1	0	0	0	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{863}	1	0	0	0	0	No
X_{864}	1	0	0	0	0	No
X_{865}	1	0	0	0	0	No
X_{866}	1	0	0	0	0	No
X_{867}	3	2	1	15	0	No
X_{868}	1	0	0	1	0	No
X_{869}	1	0	0	4	0	No
X_{870}	1	0	0	1	0	No
X_{871}	1	0	0	0	0	No
X_{872}	1	715	4	58	0	No
X_{873}	1	28	5	588	0	No
X_{874}	12	1	1	32	0	No
X_{875}	1	0	0	0	0	No
X_{876}	1	127	6	46	0	No
X_{877}	1	0	0	0	0	No
X_{878}	35	101	4	0	0	No
X_{879}	2	5	1	3	0	No
X_{880}	4	20	6	0	0	No
X_{881}	1	8	1	532	0	No
X_{882}	5	29	5	0	0	No
X_{883}	1	0	0	0	0	No
X_{884}	1	0	0	0	0	No
X_{885}	10	154	4	99	0	No
X_{886}	4	3	1	0	0	No
X_{887}	2	0	0	0	0	No
X_{888}	1	0	0	0	0	No
X_{889}	1	0	0	0	0	No
X_{890}	1	3	1	0	0	No
X_{891}	1	6	2	26	0	No
X_{892}	8	89	8	646	0	No
X_{893}	1	0	0	0	0	No
X_{894}	1	0	0	0	0	No
X_{895}	1	0	0	1	0	No
X_{896}	1	0	0	0	0	No
X_{897}	1	0	0	0	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{898}	1	0	0	0	0	No
X_{899}	1	0	0	0	0	No
X_{900}	6	2	1	0	0	No
X_{901}	1	0	0	0	0	No
X_{902}	1	0	0	0	0	No
X_{903}	1	0	0	1	0	No
X_{904}	1	0	0	47	0	No
X_{905}	2	0	0	0	0	No
X_{906}	4	6	3	46	0	No
X_{907}	343	0	0	0	0	No
X_{908}	3	7	2	0	0	No
X_{909}	8	0	0	0	0	No
X_{910}	1	0	0	1	0	No
X_{911}	11	215	8	39	0	No
X_{912}	1	0	0	1	0	No
X_{913}	1	0	0	0	0	No
X_{914}	17	24	2	0	0	No
X_{915}	1	0	0	3	0	No
X_{916}	119	1,256	14	1,218	15,392	Yes
X_{917}	8	8	2	79	0	No
X_{918}	1	0	0	0	0	No
X_{919}	12	32	4	75	0	No
X_{920}	2	0	0	0	0	No
X_{921}	1	0	0	1	0	No
X_{922}	1	5	1	2	0	No
X_{923}	1	0	0	0	0	No
X_{924}	1	0	0	0	0	No
X_{925}	1	8	2	0	0	No
X_{926}	1	0	0	0	0	No
X_{927}	12	6	2	0	0	No
X_{928}	6	0	0	4	0	No
X_{929}	4	5	3	1	0	No
X_{930}	1	0	0	0	0	No
X_{931}	3	80	4	0	0	No
X_{932}	1	0	0	0	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{933}	1	0	0	0	0	No
X_{934}	1	8	1	252	0	No
X_{935}	1	0	0	0	0	No
X_{936}	2	118	7	334	1,021	Yes
X_{937}	28	0	0	0	0	No
X_{938}	1	0	0	0	0	No
X_{939}	1	1	1	47	0	No
X_{940}	1	4	1	250	0	No
X_{941}	1	0	0	11	0	No
X_{942}	4	11	2	0	0	No
X_{943}	2	0	0	0	0	No
X_{944}	1	0	0	22	0	No
X_{945}	2	0	0	0	0	No
X_{946}	1	60	1	0	0	No
X_{947}	10	47	5	0	0	No
X_{948}	2	37	4	55	0	No
X_{949}	3	0	0	0	0	No
X_{950}	1	0	0	0	0	No
X_{951}	8	27	2	60	0	No
X_{952}	9	181	5	0	0	No
X_{953}	1	0	0	0	0	No
X_{954}	4	18	2	11	0	No
X_{955}	1	0	0	0	0	No
X_{956}	1	0	0	0	0	No
X_{957}	1	44	7	0	0	No
X_{958}	397	42	5	133	0	No
X_{959}	1	368	12	384	0	No
X_{960}	1	0	0	14	0	No
X_{961}	1	1	1	0	0	No
X_{962}	1	0	0	0	0	No
X_{963}	1	0	0	0	0	No
X_{964}	3	0	0	0	0	No
X_{965}	29	246	9	0	0	No
X_{966}	2	2	1	0	0	No
X_{967}	3	0	0	1	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{968}	1	107	2	10	0	No
X_{969}	1	2	2	0	0	No
X_{970}	1	0	0	0	0	No
X_{971}	35	0	0	211	0	No
X_{972}	10	122	7	143	0	No
X_{973}	1	2	1	0	0	No
X_{974}	16	0	0	127	0	No
X_{975}	1	3	1	0	0	No
X_{976}	1	0	0	142	0	No
X_{977}	1	0	0	0	0	No
X_{978}	2	15	1	0	0	No
X_{979}	2	0	0	0	0	No
X_{980}	1	4	3	20	0	No
X_{981}	1	0	0	0	0	No
X_{982}	3	0	0	8	0	No
X_{983}	1	8	1	0	0	No
X_{984}	12	0	0	0	0	No
X_{985}	5	22	1	23	0	No
X_{986}	3	1	1	382	0	No
X_{987}	1	0	0	0	0	No
X_{988}	23	21	3	726	0	No
X_{989}	19	447	6	0	0	Yes
X_{990}	23	6	2	46	0	No
X_{991}	1	3,991	3	272	0	No
X_{992}	1	0	0	0	0	No
X_{993}	1	13	2	2	0	No
X_{994}	1	1	1	127	0	No
X_{995}	3	0	0	2	0	No
X_{996}	1	0	0	12	0	No
X_{997}	2	0	0	0	0	No
X_{998}	1	0	0	0	0	No
X_{999}	2	9	1	16	0	No
X_{1000}	1	0	0	0	0	No
X_{1001}	1	0	0	0	0	No
X_{1002}	40	32	3	110	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{1003}	2	2	1	0	0	No
X_{1004}	3	0	0	2	0	No
X_{1005}	1	0	0	0	0	No
X_{1006}	1	0	0	0	0	No
X_{1007}	2	0	0	12	0	No
X_{1008}	1	0	0	1	0	No
X_{1009}	4	0	0	10	0	No
X_{1010}	14	143	9	0	0	No
X_{1011}	1	30	2	47	0	No
X_{1012}	4	120	5	0	0	No
X_{1013}	1	5	1	0	0	No
X_{1014}	3	0	0	0	0	No
X_{1015}	1	0	0	0	0	No
X_{1016}	9	1	1	48	0	No
X_{1017}	1	58	3	31	0	No
X_{1018}	14	37	2	64	0	No
X_{1019}	1	1	1	0	0	No
X_{1020}	1	0	0	0	0	No
X_{1021}	2	0	0	0	0	No
X_{1022}	1	1	1	0	0	No
X_{1023}	2	0	0	0	0	No
X_{1024}	2	0	0	0	0	No
X_{1025}	1	0	0	9	0	No
X_{1026}	1	0	0	1	0	No
X_{1027}	3	0	0	0	0	No
X_{1028}	9	0	0	0	0	No
X_{1029}	87	42	3	0	0	No
X_{1030}	3	0	0	0	0	No
X_{1031}	1	1	1	0	0	No
X_{1032}	1	4	2	0	0	No
X_{1033}	1	0	0	0	0	No
X_{1034}	2	0	0	0	0	No
X_{1035}	41	7	1	483	0	No
X_{1036}	2	0	0	0	0	No
X_{1037}	2	2	1	7	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{1038}	2	0	0	0	0	No
X_{1039}	1	0	0	0	0	No
X_{1040}	3	0	0	8	0	No
X_{1041}	30	470	6	136	0	No
X_{1042}	3	5	1	46	0	No
X_{1043}	4	0	0	0	0	No
X_{1044}	2	0	0	0	0	No
X_{1045}	1	0	0	0	0	No
X_{1046}	1	0	0	1	0	No
X_{1047}	17	115	3	123	0	No
X_{1048}	1	0	0	12	0	No
X_{1049}	3	0	0	0	0	No
X_{1050}	1	1	1	0	0	No
X_{1051}	1	0	0	0	0	No
X_{1052}	9	22	5	92	0	No
X_{1053}	4	13	1	23	0	No
X_{1054}	4	0	0	51	0	No
X_{1055}	4	433	5	157	0	No
X_{1056}	110	258	9	185	0	No
X_{1057}	1	0	0	1	0	No
X_{1058}	1	0	0	1	0	No
X_{1059}	1	0	0	0	0	No
X_{1060}	1	0	0	0	0	No
X_{1061}	3	0	0	0	0	No
X_{1062}	2	0	0	0	0	No
X_{1063}	7	1	1	0	0	No
X_{1064}	1	0	0	0	0	No
X_{1065}	1	3	1	0	0	No
X_{1066}	1	4	1	19	0	No
X_{1067}	2	0	0	9	0	No
X_{1068}	1	0	0	0	0	No
X_{1069}	1	0	0	2	0	No
X_{1070}	2	0	0	0	0	No
X_{1071}	1	8	3	37	0	No
X_{1072}	2	0	0	0	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{1073}	1	0	0	0	0	No
X_{1074}	3	22	2	0	0	No
X_{1075}	1	36	2	89	0	No
X_{1076}	2	2	1	5	0	No
X_{1077}	4	0	0	21	0	No
X_{1078}	2	0	0	0	0	No
X_{1079}	11	271	3	28	0	No
X_{1080}	1	0	0	1	0	No
X_{1081}	1	0	0	3	0	No
X_{1082}	1	0	0	0	0	No
X_{1083}	1	0	0	0	0	No
X_{1084}	1	0	0	0	0	No
X_{1085}	1	66	3	0	0	No
X_{1086}	1	0	0	1	0	No
X_{1087}	2	0	0	1	0	No
X_{1088}	2	0	0	0	0	No
X_{1089}	3	0	0	0	0	No
X_{1090}	4	12	2	52	0	No
X_{1091}	1	5	1	59	0	No
X_{1092}	5	57	2	0	0	No
X_{1093}	2	0	0	0	0	No
X_{1094}	1	0	0	0	0	No
X_{1095}	2	11	1	0	0	No
X_{1096}	3	0	0	1	0	No
X_{1097}	2	0	0	0	0	No
X_{1098}	1	1	1	13	0	No
X_{1099}	1	0	0	0	0	No
X_{1100}	7	94	2	26	0	No
X_{1101}	5	0	0	26	0	No
X_{1102}	5	724	7	0	0	No
X_{1103}	1	0	0	23	0	No
X_{1104}	2	0	0	13	0	No
X_{1105}	1	1	1	0	0	No
X_{1106}	1	0	0	0	0	No
X_{1107}	4	1	1	0	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{1108}	86	62	2	44	0	No
X_{1109}	1	0	0	0	0	No
X_{1110}	1	0	0	70	0	No
X_{1111}	1	0	0	0	0	No
X_{1112}	6	5	2	344	0	No
X_{1113}	1	0	0	0	0	No
X_{1114}	2	0	0	0	0	No
X_{1115}	2	0	0	1	0	No
X_{1116}	3	3	1	4	0	No
X_{1117}	1	0	0	9	0	No
X_{1118}	1	0	0	0	0	No
X_{1119}	1	0	0	0	0	No
X_{1120}	2	0	0	0	0	No
X_{1121}	1	0	0	0	0	No
X_{1122}	1	0	0	0	0	No
X_{1123}	1	0	0	0	0	No
X_{1124}	1	0	0	0	0	No
X_{1125}	5	26	1	4	0	No
X_{1126}	1	3	2	10	0	No
X_{1127}	1	0	0	0	0	No
X_{1128}	1	0	0	11	0	No
X_{1129}	2	27	2	0	0	No
X_{1130}	3	0	0	18	0	No
X_{1131}	2	0	0	0	0	No
X_{1132}	1	0	0	0	0	No
X_{1133}	5	8	1	119	0	No
X_{1134}	22	69	4	139	0	No
X_{1135}	8	23	2	25	0	No
X_{1136}	1	5	1	22	0	No
X_{1137}	2	7	3	110	0	No
X_{1138}	1	0	0	0	0	No
X_{1139}	3	494	6	0	0	No
X_{1140}	1	5	1	0	0	No
X_{1141}	13	100	6	20	0	No
X_{1142}	1	24	5	180	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{1143}	1	0	0	0	0	No
X_{1144}	1	0	0	0	0	No
X_{1145}	33	0	0	0	0	No
X_{1146}	1	0	0	0	0	No
X_{1147}	1	0	0	0	0	No
X_{1148}	3	9	1	3	0	No
X_{1149}	1	0	0	37	0	No
X_{1150}	2	0	0	1	0	No
X_{1151}	5	9	3	60	0	No
X_{1152}	1	13	4	203	0	No
X_{1153}	1	0	0	0	0	No
X_{1154}	1	0	0	0	0	No
X_{1155}	15	0	0	10	0	No
X_{1156}	11	0	0	33	0	No
X_{1157}	1	1	1	21	0	No
X_{1158}	2	0	0	0	0	No
X_{1159}	1	156	5	16	0	No
X_{1160}	1	3	1	7	0	No
X_{1161}	5	253	1	22	0	No
X_{1162}	1	0	0	0	0	No
X_{1163}	1	178	4	0	0	No
X_{1164}	2	2	1	0	0	No
X_{1165}	1	0	0	0	0	No
X_{1166}	1	213	8	0	0	No
X_{1167}	6	26	6	107	0	No
X_{1168}	5	142	6	488	4,751	Yes
X_{1169}	2	0	0	0	0	No
X_{1170}	1	77	3	0	0	No
X_{1171}	1	0	0	0	0	No
X_{1172}	2	0	0	0	0	No
X_{1173}	8	0	0	0	0	No
X_{1174}	1	0	0	0	0	No
X_{1175}	13	9	4	51	0	No
X_{1176}	5	0	0	0	0	No
X_{1177}	1	0	0	12	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{1178}	1	0	0	0	0	No
X_{1179}	5	330	8	43	0	No
X_{1180}	3	178	5	759	0	No
X_{1181}	2	22	7	220	0	No
X_{1182}	2	17	1	0	0	No
X_{1183}	120	1,046	5	169	0	No
X_{1184}	1	0	0	0	0	No
X_{1185}	25	0	0	411	0	No
X_{1186}	1	33	2	0	0	No
X_{1187}	1	0	0	0	0	No
X_{1188}	1	0	0	1	0	No
X_{1189}	1	0	0	0	0	No
X_{1190}	2	0	0	0	0	No
X_{1191}	2	0	0	0	0	No
X_{1192}	1	0	0	0	0	No
X_{1193}	10	7	2	0	0	No
X_{1194}	1	0	0	8	0	No
X_{1195}	1	0	0	2	0	No
X_{1196}	1	0	0	0	0	No
X_{1197}	1	7	1	0	0	No
X_{1198}	1	0	0	70	0	No
X_{1199}	1	0	0	0	0	No
X_{1200}	1	0	0	0	0	No
X_{1201}	2	104	1	0	0	No
X_{1202}	3	0	0	0	0	No
X_{1203}	1	0	0	0	0	No
X_{1204}	3	33	1	0	0	No
X_{1205}	1	0	0	0	0	No
X_{1206}	4	0	0	0	0	No
X_{1207}	5	0	0	2	0	No
X_{1208}	6	152	5	10	0	No
X_{1209}	3	0	0	0	0	No
X_{1210}	1	0	0	1	0	No
X_{1211}	1	0	0	0	0	No
X_{1212}	2	0	0	0	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{1213}	1	8,640	4	53	0	No
X_{1214}	1	2	1	0	0	No
X_{1215}	1	0	0	32	0	No
X_{1216}	4	0	0	0	0	No
X_{1217}	9	414	7	256	2,082	Yes
X_{1218}	1	0	0	49	0	No
X_{1219}	1	0	0	1	0	No
X_{1220}	1	0	0	0	0	No
X_{1221}	5	7	3	0	0	No
X_{1222}	2	60	3	0	0	No
X_{1223}	82	580	6	64	0	No
X_{1224}	1	2	1	0	0	No
X_{1225}	3	2	1	0	0	No
X_{1226}	4	0	0	6	0	No
X_{1227}	2	0	0	3	0	No
X_{1228}	5	0	0	0	0	No
X_{1229}	1	0	0	0	0	No
X_{1230}	1	0	0	0	0	No
X_{1231}	2	0	0	56	0	No
X_{1232}	1	0	0	0	0	No
X_{1233}	2	0	0	0	0	No
X_{1234}	1	0	0	0	0	No
X_{1235}	1	0	0	0	0	No
X_{1236}	1	0	0	0	0	No
X_{1237}	1	0	0	3	0	No
X_{1238}	2	34	6	0	0	No
X_{1239}	1	0	0	0	0	No
X_{1240}	1	0	0	0	0	No
X_{1241}	1	0	0	1	0	No
X_{1242}	6	1,054	10	276	0	No
X_{1243}	1	0	0	0	0	No
X_{1244}	19	219	7	0	0	Yes
X_{1245}	1	0	0	0	0	No
X_{1246}	2	29	1	0	0	No
X_{1247}	1	0	0	0	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{1248}	10	75	6	524	0	No
X_{1249}	18	75	3	27	0	No
X_{1250}	9	0	0	0	0	No
X_{1251}	3	5	1	8	0	No
X_{1252}	1	0	0	0	0	No
X_{1253}	3	0	0	0	0	No
X_{1254}	1	273	2	3	0	No
X_{1255}	1	0	0	0	0	No
X_{1256}	7	32	5	1,371	0	No
X_{1257}	1	134	7	42	0	No
X_{1258}	1	0	0	0	0	No
X_{1259}	1	0	0	0	0	No
X_{1260}	2	0	0	0	0	No
X_{1261}	1	0	0	0	0	No
X_{1262}	1	0	0	0	0	No
X_{1263}	2	0	0	0	0	No
X_{1264}	2	0	0	1	0	No
X_{1265}	7	0	0	16	0	No
X_{1266}	5	0	0	54	0	No
X_{1267}	2	110	1	0	0	No
X_{1268}	1	0	0	0	0	No
X_{1269}	2	0	0	17	0	No
X_{1270}	1	0	0	10	0	No
X_{1271}	2	0	0	0	0	No
X_{1272}	5	0	0	0	0	No
X_{1273}	2	0	0	76	0	No
X_{1274}	1	0	0	45	0	No
X_{1275}	1	0	0	0	0	No
X_{1276}	1	0	0	0	0	No
X_{1277}	1	0	0	0	0	No
X_{1278}	19	0	0	0	0	No
X_{1279}	7	0	0	0	0	No
X_{1280}	1	0	0	0	0	No
X_{1281}	1	5	2	363	0	No
X_{1282}	2	0	0	0	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{1283}	1	0	0	0	0	No
X_{1284}	1	0	0	0	0	No
X_{1285}	16	0	0	0	0	No
X_{1286}	2	1	1	0	0	No
X_{1287}	7	304	10	0	0	No
X_{1288}	2	0	0	0	0	No
X_{1289}	2	34	3	0	0	No
X_{1290}	61	0	0	148	0	No
X_{1291}	4	0	0	0	0	No
X_{1292}	1	0	0	0	0	No
X_{1293}	1	0	0	1	0	No
X_{1294}	2	251	2	3	0	No
X_{1295}	8	12	1	26	0	No
X_{1296}	16	23	4	0	0	No
X_{1297}	63	32	2	0	0	No
X_{1298}	1	0	0	1	0	No
X_{1299}	1	0	0	0	729	Yes
X_{1300}	1	0	0	3	0	No
X_{1301}	1	0	0	0	0	No
X_{1302}	2	41	2	0	0	No
X_{1303}	5	424	6	52	3,687	Yes
X_{1304}	2	0	0	4	0	No
X_{1305}	3	0	0	0	0	No
X_{1306}	5	0	0	0	0	No
X_{1307}	3	6	2	70	0	No
X_{1308}	1	1	1	16	0	No
X_{1309}	1	0	0	0	0	No
X_{1310}	1	0	0	1	0	No
X_{1311}	5	0	0	0	0	No
X_{1312}	9	9	2	377	0	No
X_{1313}	1	0	0	0	0	No
X_{1314}	2	0	0	3	0	No
X_{1315}	2	0	0	0	0	No
X_{1316}	1	248	5	485	0	No
X_{1317}	1	3	1	2	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{1318}	1	0	0	0	0	No
X_{1319}	1	0	0	0	0	No
X_{1320}	1	8	3	254	0	No
X_{1321}	1	20	3	83	0	No
X_{1322}	6	110	4	0	0	No
X_{1323}	1	0	0	0	0	No
X_{1324}	1	1,124	9	90	0	No
X_{1325}	25	37	4	0	0	No
X_{1326}	1	0	0	0	0	No
X_{1327}	1	0	0	10	0	No
X_{1328}	1	1	1	0	0	No
X_{1329}	1	0	0	0	0	No
X_{1330}	1	1	1	97	0	No
X_{1331}	5	13	2	462	0	No
X_{1332}	18	134	2	69	0	No
X_{1333}	11	254	7	250	0	No
X_{1334}	1	0	0	0	0	No
X_{1335}	1	0	0	26	0	No
X_{1336}	1	0	0	0	0	No
X_{1337}	1	20	4	174	0	No
X_{1338}	3	5	1	3	0	No
X_{1339}	1	0	0	10	0	No
X_{1340}	1	0	0	0	0	No
X_{1341}	3	7	4	61	0	No
X_{1342}	1	232	9	718	8,387	Yes
X_{1343}	1	0	0	0	0	No
X_{1344}	1	0	0	0	0	No
X_{1345}	7	0	0	39	0	No
X_{1346}	4	51	1	0	0	No
X_{1347}	2	0	0	0	0	No
X_{1348}	4	7	2	659	0	No
X_{1349}	2	1	1	1	0	No
X_{1350}	2	0	0	0	0	No
X_{1351}	8	10	4	122	0	No
X_{1352}	2	0	0	0	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{1353}	1	0	0	0	0	No
X_{1354}	1	0	0	0	0	No
X_{1355}	1	12	1	7	0	No
X_{1356}	1	0	0	0	0	No
X_{1357}	2	0	0	0	0	No
X_{1358}	1	0	0	211	0	No
X_{1359}	1	0	0	0	0	No
X_{1360}	1	0	0	0	0	No
X_{1361}	4	1	1	0	0	No
X_{1362}	1	0	0	0	0	No
X_{1363}	2	99	6	1,540	4,381	Yes
X_{1364}	2	0	0	0	0	No
X_{1365}	2	0	0	0	0	No
X_{1366}	5	7	1	2	0	No
X_{1367}	2	0	0	64	0	No
X_{1368}	1	111	6	227	0	No
X_{1369}	6	33	5	142	0	No
X_{1370}	1	1	1	0	0	No
X_{1371}	1	0	0	0	0	No
X_{1372}	1	0	0	5	0	No
X_{1373}	1	0	0	0	0	No
X_{1374}	1	0	0	0	0	No
X_{1375}	2	0	0	0	0	No
X_{1376}	2	0	0	0	0	No
X_{1377}	1	1	1	0	0	No
X_{1378}	10	476	8	3,329	0	No
X_{1379}	1	0	0	0	0	No
X_{1380}	1	0	0	0	0	No
X_{1381}	1	0	0	0	0	No
X_{1382}	1	0	0	0	0	No
X_{1383}	3	85	6	407	0	No
X_{1384}	5	17	2	26	0	No
X_{1385}	1	0	0	0	0	No
X_{1386}	1	0	0	0	0	No
X_{1387}	2	0	0	0	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{1388}	1	79	6	0	0	No
X_{1389}	13	0	0	13	0	No
X_{1390}	2	0	0	0	0	No
X_{1391}	1	0	0	0	0	No
X_{1392}	5	2	1	52	0	No
X_{1393}	7	0	0	31	0	No
X_{1394}	1	0	0	0	0	No
X_{1395}	1	12	1	0	0	No
X_{1396}	21	117	5	160	0	No
X_{1397}	1	1	1	63	0	No
X_{1398}	1	0	0	4	0	No
X_{1399}	1	1	1	0	0	No
X_{1400}	1	5	2	54	0	No
X_{1401}	1	0	0	0	0	No
X_{1402}	1	0	0	5	0	No
X_{1403}	1	13	1	0	0	No
X_{1404}	1	0	0	0	0	No
X_{1405}	1	0	0	0	0	No
X_{1406}	69	6	1	36	0	No
X_{1407}	10	0	0	18	0	No
X_{1408}	1	0	0	0	0	No
X_{1409}	1	0	0	3	0	No
X_{1410}	4	0	0	1	0	No
X_{1411}	1	0	0	17	0	No
X_{1412}	1	0	0	2	0	No
X_{1413}	5	446	11	654	4,719	Yes
X_{1414}	1	0	0	0	0	No
X_{1415}	3	0	0	3	0	No
X_{1416}	1	0	0	0	0	No
X_{1417}	5	491	6	826	26,259	Yes
X_{1418}	3	0	0	0	0	No
X_{1419}	8	0	0	0	0	No
X_{1420}	1	0	0	0	0	No
X_{1421}	119	11	2	0	0	No
X_{1422}	1	0	0	0	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{1423}	12	31	1	0	0	No
X_{1424}	4	3	1	0	0	No
X_{1425}	3	0	0	0	0	No
X_{1426}	42	288	8	309	0	No
X_{1427}	3	0	0	0	0	No
X_{1428}	1	0	0	6	0	No
X_{1429}	1	11	2	319	0	No
X_{1430}	1	0	0	1	0	No
X_{1431}	1	0	0	3	0	No
X_{1432}	4	0	0	45	0	No
X_{1433}	1	0	0	0	0	No
X_{1434}	3	0	0	0	0	No
X_{1435}	1	0	0	0	0	No
X_{1436}	2	74	5	85	943	Yes
X_{1437}	11	34	5	0	0	No
X_{1438}	4	0	0	0	0	No
X_{1439}	1	0	0	6	0	No
X_{1440}	2	0	0	215	0	No
X_{1441}	2	0	0	275	0	No
X_{1442}	9	0	0	0	0	No
X_{1443}	2	14	3	46	0	No
X_{1444}	2	0	0	0	0	No
X_{1445}	2	0	0	0	0	No
X_{1446}	1	0	0	0	0	No
X_{1447}	7	20	2	7	0	No
X_{1448}	2	0	0	0	0	No
X_{1449}	1	0	0	0	0	No
X_{1450}	3	0	0	0	0	No
X_{1451}	5	5	2	48	0	No
X_{1452}	1	18	2	172	0	No
X_{1453}	1	4	1	0	0	No
X_{1454}	1	0	0	0	0	No
X_{1455}	1	2	2	28	0	No
X_{1456}	2	0	0	0	0	No
X_{1457}	34	0	0	42	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{1458}	2	22	1	0	0	No
X_{1459}	1	0	0	2	0	No
X_{1460}	2	0	0	0	0	No
X_{1461}	1	24	2	0	0	No
X_{1462}	1	0	0	0	0	No
X_{1463}	1	92	5	0	0	No
X_{1464}	1	34	4	0	0	No
X_{1465}	5	2	1	0	0	No
X_{1466}	1	104	3	0	0	No
X_{1467}	7	812	7	0	0	No
X_{1468}	207	436	5	23	0	No
X_{1469}	2	0	0	0	0	No
X_{1470}	1	4	1	0	0	No
X_{1471}	2	0	0	0	0	No
X_{1472}	1	0	0	0	0	No
X_{1473}	1	0	0	5	0	No
X_{1474}	1	0	0	0	0	No
X_{1475}	1	0	0	2	0	No
X_{1476}	16	0	0	0	0	No
X_{1477}	5	35	5	39	0	No
X_{1478}	3	0	0	7	0	No
X_{1479}	40	14	1	0	0	No
X_{1480}	6	0	0	26	0	No
X_{1481}	1	0	0	4	0	No
X_{1482}	1	0	0	0	0	No
X_{1483}	1	0	0	4	0	No
X_{1484}	21	189	6	235	0	No
X_{1485}	2	0	0	0	0	No
X_{1486}	1	0	0	1	0	No
X_{1487}	1	0	0	0	0	No
X_{1488}	1	0	0	0	0	No
X_{1489}	4	37	6	5	0	No
X_{1490}	55	426	4	1,232	4,014	Yes
X_{1491}	8	0	0	0	0	No
X_{1492}	2	0	0	2	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{1493}	3	0	0	2	0	No
X_{1494}	1	0	0	0	0	No
X_{1495}	1	0	0	0	0	No
X_{1496}	1	0	0	3	0	No
X_{1497}	3	7	1	0	0	No
X_{1498}	1	0	0	0	0	No
X_{1499}	1	0	0	0	0	No
X_{1500}	20	480	11	979	5,411	Yes
X_{1501}	2	0	0	0	0	No
X_{1502}	1	0	0	14	0	No
X_{1503}	22	43	2	1	0	No
X_{1504}	1	0	0	0	0	No
X_{1505}	1	0	0	0	0	No
X_{1506}	5	26	2	0	0	No
X_{1507}	5	0	0	78	0	No
X_{1508}	67	669	5	352	0	No
X_{1509}	5	11	3	169	456	Yes
X_{1510}	2	1	1	0	0	No
X_{1511}	5	185	3	51	0	No
X_{1512}	1	0	0	0	0	No
X_{1513}	1	0	0	0	0	No
X_{1514}	8	96	5	110	0	No
X_{1515}	2	0	0	0	0	No
X_{1516}	1	34	2	0	0	No
X_{1517}	1	0	0	7	0	No
X_{1518}	1	0	0	0	0	No
X_{1519}	1	145	3	155	0	No
X_{1520}	1	3	1	96	0	No
X_{1521}	1	6	1	0	0	No
X_{1522}	1	4	1	24	0	No
X_{1523}	1	0	0	0	0	No
X_{1524}	3	1	1	0	0	No
X_{1525}	12	7	1	20	0	No
X_{1526}	1	3	1	0	0	No
X_{1527}	1	0	0	4	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{1528}	1	0	0	0	0	No
X_{1529}	1	2	1	2	0	No
X_{1530}	1	0	0	0	0	No
X_{1531}	1	0	0	0	0	No
X_{1532}	4	110	5	171	0	No
X_{1533}	1	0	0	0	0	No
X_{1534}	1	2	1	1	0	No
X_{1535}	1	0	0	0	0	No
X_{1536}	2	1,145	10	518	0	No
X_{1537}	1	0	0	0	0	No
X_{1538}	2	0	0	73	0	No
X_{1539}	1	0	0	0	0	No
X_{1540}	1	0	0	0	0	No
X_{1541}	2	0	0	177	0	No
X_{1542}	15	1	1	16	0	No
X_{1543}	1	0	0	0	0	No
X_{1544}	3	0	0	0	0	No
X_{1545}	2	0	0	0	0	No
X_{1546}	5	94	8	0	0	No
X_{1547}	1	74	2	0	0	No
X_{1548}	1	0	0	0	0	No
X_{1549}	36	1	1	652	0	No
X_{1550}	1	0	0	5	0	No
X_{1551}	1	0	0	137	0	No
X_{1552}	3	0	0	0	0	No
X_{1553}	2	0	0	0	0	No
X_{1554}	4	7	1	2	0	No
X_{1555}	4	67	1	0	0	No
X_{1556}	2	0	0	0	0	No
X_{1557}	3	0	0	6	0	No
X_{1558}	2	0	0	113	0	No
X_{1559}	1	0	0	0	0	No
X_{1560}	1	0	0	1	0	No
X_{1561}	21	0	0	3	0	No
X_{1562}	1	0	0	0	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{1563}	3	1,109	8	168	0	No
X_{1564}	1	3	1	0	0	No

Table C.8: Ubuntu activity levels on a yearly basis

Channel	2000	2001	2002	2003	2004	2005	2006
E-mails	0	0	0	0	20,142	61,100	67,227
QA e-mails	0	0	0	0	0	338	921
Bugs	327	392	1,459	2,155	8,124	25,707	42,000
Bug comments	655	1,065	4,394	7,258	42,516	98,446	178,512

Channel	2007	2008	2009	2010	2011	2012	2013
E-mails	59,443	69,147	68,502	54,387	37,058	37,595	13,086
QA e-mails	498	846	785	1,899	1,123	1,526	862
Bugs	88,443	121,861	169,975	164,447	184,826	149,616	34,029
Bug comments	361,402	568,934	705,424	726,786	664,882	612,333	141,725

C.4 Plone

Table C.9: Plone QA mailing list participants' activity levels on other channels

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_1	1	1	1	0	0	No
X_2	1	0	0	0	0	No
X_3	1	266	6	0	0	Yes
X_4	1	0	0	0	0	No
X_5	1	0	0	0	0	No
X_6	6	0	0	0	0	No
X_7	1	17	2	0	0	No
X_8	1	0	0	0	0	No
X_9	1	0	0	0	0	No
X_{10}	2	221	2	0	0	No
X_{11}	8	1,590	7	0	0	Yes
X_{12}	1	0	0	0	0	No
X_{13}	59	520	7	76	886	Yes
X_{14}	3	1,063	10	51	834	Yes
X_{15}	1	0	0	0	0	No
X_{16}	1	0	0	0	0	No
X_{17}	1	0	0	0	0	No
X_{18}	1	0	0	0	0	No
X_{19}	1	573	10	31	127	Yes
X_{20}	2	1	1	0	0	No
X_{21}	1	1,126	10	113	10,848	Yes
X_{22}	3	0	0	0	0	No
X_{23}	3	974	9	40	170	Yes
X_{24}	2	331	6	0	0	No
X_{25}	1	0	0	0	0	No
X_{26}	1	81	6	4	3	No
X_{27}	1	1,659	13	0	0	Yes
X_{28}	1	256	6	37	96	Yes
X_{29}	2	0	0	0	0	No
X_{30}	1	0	0	0	0	No
X_{31}	2	81	4	0	0	Yes

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{32}	4	888	10	0	0	Yes
X_{33}	1	0	0	0	0	No
X_{34}	1	1,153	8	0	0	Yes
X_{35}	1	0	0	0	0	No
X_{36}	1	27	3	0	0	Yes
X_{37}	1	0	0	0	0	No
X_{38}	4	71	3	0	0	No
X_{39}	3	276	4	0	0	Yes
X_{40}	1	0	0	0	0	No
X_{41}	2	1,110	2	0	0	No
X_{42}	1	34	3	0	0	Yes
X_{43}	1	65	3	0	0	No
X_{44}	1	0	0	0	0	No
X_{45}	2	0	0	0	0	No
X_{46}	1	24	1	0	0	No
X_{47}	8	924	9	0	0	Yes
X_{48}	1	217	6	0	0	Yes
X_{49}	1	0	0	0	0	No
X_{50}	1	0	0	0	0	No
X_{51}	1	162	5	0	0	No
X_{52}	1	1	1	0	0	No
X_{53}	1	0	0	0	0	No
X_{54}	1	201	6	0	0	Yes
X_{55}	2	78	5	0	0	No
X_{56}	1	190	7	15	233	Yes
X_{57}	15	0	0	0	0	No

Table C.10: Plone activity levels on a yearly basis

Channel	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	Total
Bug No.	748	1,503	1,461	1,349	944	1,482	1,336	1,132	1,391	1,060	620	–	13,026
Comment No.	1,102	3,106	3,322	4,057	3,636	5,193	5,592	1,1251	6,297	4,015	8,312	–	55,883
No.	2,742	12,228	19,951	17,486	22,312	23,959	23,323	18,508	11,403	10,930	8,973	4,177	175,992
QA mail No.	–	–	–	–	–	–	–	–	–	47	64	59	170

C.5 KDE

Table C.11: KDE QA mailing list participants' activity levels on other channels

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_1	2	5,128	22	366	5,332	Yes
X_2	65	2,267	17	329	12,642	Yes
X_3	1	81	9	0	0	Yes
X_4	3	236	17	96	319	Yes
X_5	2	29	4	0	0	No
X_6	1	114	10	17	75	No
X_7	4	1,317	13	14	1,219	Yes
X_8	17	1,071	15	956	3,391	No
X_9	35	126	9	30	1,676	No
X_{10}	2	11	3	2	60	No
X_{11}	4	0	0	0	0	No
X_{12}	1	537	16	19	273	Yes
X_{13}	1	0	0	0	0	No
X_{14}	4	40	4	0	0	No
X_{15}	1	155	30	186	1,166	Yes
X_{16}	1	301	20	8	264	Yes
X_{17}	3	626	33	87	1,570	Yes
X_{18}	4	1,580	24	122	2736	Yes
X_{19}	1	1,049	10	116	397	No
X_{20}	1	76	7	46	178	No
X_{21}	115	509	13	0	0	No
X_{22}	2	0	0	0	0	No
X_{23}	3	88	10	142	1,171	Yes
X_{24}	1	0	0	0	0	No
X_{25}	9	143	9	12	30	No
X_{26}	45	3,610	13	251	1,863	Yes
X_{27}	1	1,648	4	533	1,397	No
X_{28}	1	3	3	6	80	No
X_{29}	10	310	20	73	6,206	Yes
X_{30}	4	2,369	16	100	1,164	Yes
X_{31}	1	1,854	36	80	13,494	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{32}	64	844	16	141	1,316	No
X_{33}	3	7	3	0	0	No
X_{34}	4	5,388	17	452	5,588	No
X_{35}	12	190	13	36	6,173	Yes
X_{36}	1	6,362	37	85	11,737	Yes
X_{37}	3	46	9	13	35	Yes
X_{38}	2	759	20	17	1,320	Yes
X_{39}	6	34	2	11	310	No
X_{40}	18	402	21	0	0	No
X_{41}	8	430	8	0	0	Yes
X_{42}	1	26	1	2	11	Yes
X_{43}	31	806	14	216	2,109	Yes
X_{44}	2	0	0	0	0	No
X_{45}	28	6,248	22	78	34,075	No
X_{46}	2	269	13	0	0	No
X_{47}	1	7	3	119	674	No
X_{48}	1	1	1	0	1	No
X_{49}	2	8	3	0	3	No
X_{50}	2	217	17	75	343	No
X_{51}	1	5	2	1	50	Yes
X_{52}	3	690	17	5	952	Yes
X_{53}	6	144	6	17	545	No
X_{54}	2	71	5	21	49	Yes
X_{55}	1	44	1	0	0	No
X_{56}	1	0	0	0	0	Yes
X_{57}	1	3	1	0	0	No
X_{58}	8	638	11	21	402	No
X_{59}	3	18	2	0	0	No
X_{60}	3	187	17	0	0	Yes
X_{61}	1	6	2	0	0	No
X_{62}	6	5	2	1	28	No
X_{63}	4	4,824	37	387	17,054	Yes
X_{64}	42	543	26	121	3,224	Yes
X_{65}	1	178	12	153	348	Yes
X_{66}	2	0	0	0	34	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{67}	1	2	2	32	68	No
X_{68}	6	9,812	23	220	11,350	Yes
X_{69}	3	112	9	8	325	Yes
X_{70}	2	271	16	19	6,124	Yes
X_{71}	22	4,718	51	342	5,525	Yes
X_{72}	10	27	3	0	0	No
X_{73}	3	514	10	154	1,352	Yes
X_{74}	1	335	18	103	12,168	Yes
X_{75}	2	2	2	1	31	Yes
X_{76}	1	131	14	0	0	No
X_{77}	2	583	14	0	0	No
X_{78}	3	2,472	24	60	1,911	Yes
X_{79}	7	2,257	27	0	0	No
X_{80}	1	6	2	6	592	Yes
X_{81}	2	127	18	99	1,319	Yes
X_{82}	6	212	6	7	110	Yes
X_{83}	1	154	7	0	0	No
X_{84}	1	98	8	162	652	Yes
X_{85}	8	607	16	97	1,065	No
X_{86}	2	566	27	84	1,567	Yes
X_{87}	1	503	19	809	3,591	No
X_{88}	1	2	2	84	1,337	Yes
X_{89}	36	657	13	0	0	No
X_{90}	19	58	8	90	614	No
X_{91}	1	1	1	0	0	No
X_{92}	2	32	7	0	0	No
X_{93}	1	44	12	0	0	Yes
X_{94}	1	3	1	2	7	No
X_{95}	1	208	13	38	1,469	Yes
X_{96}	8	316	6	51	244	Yes
X_{97}	1	88	13	215	608	Yes
X_{98}	1	7	4	9	14	No
X_{99}	1	0	0	0	0	No
X_{100}	76	2,491	28	119	23,887	No
X_{101}	1	2,610	19	43	406	Yes

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{102}	1	80	9	98	225	Yes
X_{103}	1	88	12	24	231	Yes
X_{104}	1	459	20	100	1,404	No
X_{105}	25	57	5	23	169	No
X_{106}	4	15	4	0	22	Yes
X_{107}	2	360	22	0	0	No
X_{108}	2	4,988	37	38	7,469	Yes
X_{109}	2	7,660	64	0	0	No
X_{110}	6	1	1	1	121	No
X_{111}	4	4,256	19	44	4,566	Yes
X_{112}	1	2	2	0	0	No
X_{113}	4	1,046	26	47	484	Yes
X_{114}	1	2,753	27	123	11,010	Yes
X_{115}	2	2	2	0	0	No
X_{116}	1	55	10	34	1,139	Yes
X_{117}	6	513	19	129	3,926	No
X_{118}	2	282	16	81	268	Yes
X_{119}	2	317	13	0	0	No
X_{120}	1	3	2	0	1	No
X_{121}	30	567	25	47	869	Yes

Table C.12: KDE activity levels on a yearly basis

Channel	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013
Mail No.	3,472	5,081	6,647	7,089	9,910	13,928	18,264	29,860	38,319	53,051	78,574	81,781	77,262	76,478	29,751
QA Mail No.	0	0	0	0	0	0	0	0	0	0	0	0	0	236	83
BugSquad Mail No.	0	0	0	0	0	0	0	0	0	312	128	88	40	52	15
Bug No.	389	12,107	18,618	15,741	19,115	24,531	23,253	20,161	15,475	24,347	41,852	39,224	27,752	21,395	8,887
Comments No.	415	21,112	41,899	46,936	79,065	108,157	104,285	94,593	70,118	123,141	197,682	171,779	137,500	117,210	50,143

C.6 LibreOffice

Table C.13: LibreOffice QA mailing list participants activity levels on other channels

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_1	33	3,086	4	10	320	Yes
X_2	1	4	1	0	0	No
X_3	1	0	0	0	0	No
X_4	17	2,185	3	50	1,757	Yes
X_5	1	0	0	0	0	No
X_6	1	0	0	0	0	No
X_7	1	2	1	5	49	No
X_8	237	2,462	4	42	1,042	Yes
X_9	1	7	1	0	0	No
X_{10}	3	0	0	0	0	No
X_{11}	1	115	2	5	19	Yes
X_{12}	1	0	0	0	0	No
X_{13}	1	2	2	0	0	No
X_{14}	1	0	0	0	0	No
X_{15}	2	887	4	3	182	No
X_{16}	1	1	1	0	0	No
X_{17}	1	3	2	0	0	No
X_{18}	1	0	0	0	0	No
X_{19}	136	663	2	152	1,535	Yes
X_{20}	1	0	0	0	0	No
X_{21}	3	0	0	0	0	No
X_{22}	54	9	1	0	0	No
X_{23}	2	0	0	0	0	No
X_{24}	1	1	1	0	0	No
X_{25}	3	2	1	0	0	No
X_{26}	1	1	1	0	0	No
X_{27}	1	0	0	0	0	No
X_{28}	2	0	0	0	0	No
X_{29}	45	4	1	38	2,779	Yes
X_{30}	33	41	2	12	245	No
X_{31}	1	0	0	0	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{32}	1	0	0	0	0	No
X_{33}	3	25	2	7	31	Yes
X_{34}	1	0	0	0	0	No
X_{35}	2	0	0	0	0	No
X_{36}	2	2	1	0	0	No
X_{37}	1	0	0	0	0	No
X_{38}	2	0	0	0	0	No
X_{39}	1	0	0	1	1	No
X_{40}	4	0	0	0	0	No
X_{41}	1	0	0	0	0	No
X_{42}	2	93	1	0	18	No
X_{43}	1	0	0	0	0	No
X_{44}	1	2	2	0	0	No
X_{45}	1	0	0	0	0	No
X_{46}	1	0	0	0	0	No
X_{47}	1	0	0	0	0	No
X_{48}	1	0	0	0	0	No
X_{49}	1	0	0	0	0	No
X_{50}	1	0	0	0	0	No
X_{51}	2	0	0	0	0	No
X_{52}	1	0	0	0	0	No
X_{53}	1	121	1	26	86	No
X_{54}	2	0	0	0	0	No
X_{55}	1	2	2	0	0	No
X_{56}	6	0	0	0	0	No
X_{57}	1	0	0	0	0	No
X_{58}	100	18	2	53	383	Yes
X_{59}	1	0	0	0	0	No
X_{60}	2	0	0	0	0	No
X_{61}	1	13	1	0	5	No
X_{62}	5	592	2	130	1,088	Yes
X_{63}	1	0	0	0	0	No
X_{64}	1	0	0	0	0	No
X_{65}	1	0	0	0	0	No
X_{66}	1	0	0	0	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{67}	7	56	1	0	0	No
X_{68}	1	0	0	0	0	No
X_{69}	11	0	0	0	0	No
X_{70}	3	208	3	0	0	No
X_{71}	101	5	1	58	4,327	Yes
X_{72}	2	2	1	3	10	No
X_{73}	1	1	1	92	717	No
X_{74}	1	0	0	0	0	No
X_{75}	1	1	1	83	950	Yes
X_{76}	3	0	0	1	2	No
X_{77}	1	0	0	0	0	No
X_{78}	19	73	1	0	0	No
X_{79}	3	1	1	5	20	No
X_{80}	192	1,009	3	0	0	Yes
X_{81}	1	1	1	0	181	No
X_{82}	69	38	1	0	0	No
X_{83}	23	1	1	0	0	No
X_{84}	1	23	1	0	12	No
X_{85}	1	1	1	0	0	Yes
X_{86}	1	0	0	0	0	No
X_{87}	14	2	1	0	0	No
X_{88}	2	0	0	0	9	No
X_{89}	23	212	2	39	570	Yes
X_{90}	1	1	1	0	0	No
X_{91}	1	0	0	0	0	No
X_{92}	7	278	3	0	0	No
X_{93}	1	0	0	0	0	No
X_{94}	1	0	0	0	0	No
X_{95}	1	0	0	0	0	No
X_{96}	1	9	1	0	0	No
X_{97}	1	0	0	0	0	No
X_{98}	1	0	0	0	0	No
X_{99}	1	0	0	0	0	No
X_{100}	124	4,067	3	126	1,876	Yes
X_{101}	29	440	2	9	153	Yes

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{102}	1	0	0	0	0	No
X_{103}	1	0	0	0	0	No
X_{104}	1	0	0	0	0	No
X_{105}	1	0	0	0	0	No
X_{106}	1	0	0	0	0	No
X_{107}	1	0	0	0	0	No
X_{108}	1	141	3	33	369	Yes
X_{109}	1	0	0	0	0	No
X_{110}	1	0	0	0	0	No
X_{111}	13	0	0	0	0	No
X_{112}	1	1	1	0	0	No
X_{113}	3	45	2	4	18	Yes
X_{114}	1	0	0	0	0	No
X_{115}	2	0	0	0	0	No
X_{116}	4	290	2	68	398	No
X_{117}	1	0	0	0	0	No
X_{118}	4	44	1	0	0	Yes
X_{119}	1	0	0	0	0	No
X_{120}	23	1,650	2	34	616	Yes
X_{121}	1	0	0	0	0	No
X_{122}	1	0	0	0	0	No
X_{123}	1	0	0	0	0	No
X_{124}	1	0	0	0	0	No
X_{125}	1	0	0	0	0	No
X_{126}	1	20	1	0	0	No
X_{127}	1	0	0	0	0	No
X_{128}	3	102	2	11	39	Yes
X_{129}	1	0	0	0	0	No
X_{130}	3	0	0	0	0	No
X_{131}	1	0	0	0	0	No
X_{132}	1	0	0	0	0	No
X_{133}	1	1	1	0	0	No
X_{134}	1	3	2	0	0	No
X_{135}	65	0	0	6	46	No
X_{136}	2	0	0	0	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{137}	1	0	0	0	0	No
X_{138}	1	0	0	0	0	No
X_{139}	7	8	1	0	0	No
X_{140}	130	320	2	20	2,620	Yes
X_{141}	28	6	2	38	594	No
X_{142}	1	0	0	0	0	No
X_{143}	3	111	2	36	250	Yes
X_{144}	1	0	0	1	1	No
X_{145}	4	0	0	1	1	No
X_{146}	1	0	0	0	0	No
X_{147}	2	2	1	0	0	No
X_{148}	1	0	0	0	0	No
X_{149}	1	0	0	0	0	No
X_{150}	1	0	0	0	0	No
X_{151}	1	0	0	0	0	No
X_{152}	17	3	1	0	0	No
X_{153}	443	176	2	0	0	No
X_{154}	1	0	0	0	0	No
X_{155}	5	839	1	0	0	Yes
X_{156}	2	2	1	0	0	No
X_{157}	1	0	0	2	148	No
X_{158}	3	0	0	0	0	No
X_{159}	4	56	1	0	0	No
X_{160}	1	1	1	0	0	No
X_{161}	1	0	0	0	0	No
X_{162}	1	0	0	0	0	No
X_{163}	1	0	0	0	0	No
X_{164}	8	0	0	8	30	Yes
X_{165}	1	0	0	0	0	No
X_{166}	2	0	0	0	0	No
X_{167}	25	130	1	32	436	Yes
X_{168}	1	0	0	0	0	No
X_{169}	1	0	0	0	0	No
X_{170}	13	5	1	2	35	No
X_{171}	3	36	1	39	263	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{172}	2	0	0	0	0	No
X_{173}	1	0	0	0	0	No
X_{174}	1	0	0	0	0	No
X_{175}	18	3	1	0	0	No
X_{176}	5	17	1	0	0	Yes
X_{177}	6	1	1	0	0	No
X_{178}	1	0	0	2	7	No
X_{179}	39	36	1	0	0	No
X_{180}	2	0	0	0	0	No
X_{181}	2	3	2	0	0	No
X_{182}	1	2	2	0	0	No
X_{183}	4	333	1	3	7	Yes
X_{184}	5	1	1	34	139	No
X_{185}	1	0	0	0	0	No
X_{186}	1	0	0	0	0	No
X_{187}	1	0	0	0	0	No
X_{188}	3	0	0	0	0	No
X_{189}	30	13	1	6	15	No
X_{190}	25	148	3	23	153	No
X_{191}	1	0	0	0	0	No
X_{192}	1	0	0	0	0	No
X_{193}	2	0	0	0	0	No
X_{194}	11	0	0	0	0	No
X_{195}	1	0	0	0	0	No
X_{196}	1	1	1	4	16	No
X_{197}	1	0	0	0	0	No
X_{198}	1	0	0	0	0	No
X_{199}	1	0	0	0	0	No
X_{200}	1	0	0	0	0	No
X_{201}	1	0	0	0	0	No
X_{202}	1	0	0	0	0	No
X_{203}	1	0	0	0	0	No
X_{204}	1	0	0	0	0	No
X_{205}	1	30	1	0	0	No
X_{206}	1	0	0	0	0	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{207}	1	0	0	0	0	No
X_{208}	1	72	1	0	0	No
X_{209}	3	39	1	0	0	Yes
X_{210}	11	0	0	0	0	No
X_{211}	1	0	0	0	0	No
X_{212}	1	3	1	31	58	No
X_{213}	1	0	0	0	0	No
X_{214}	1	0	0	0	0	No
X_{215}	4	0	0	0	0	No
X_{216}	1	0	0	0	0	No
X_{217}	70	108	1	0	0	Yes
X_{218}	1	0	0	0	0	No
X_{219}	1	0	0	0	0	No
X_{220}	30	245	2	78	1,365	Yes
X_{221}	1	0	0	0	0	No
X_{222}	1	0	0	0	0	No
X_{223}	5	1	1	0	0	No
X_{224}	1	0	0	0	0	No
X_{225}	1	265	1	0	0	Yes
X_{226}	2	8	1	26	69	No
X_{227}	1	0	0	0	0	No
X_{228}	1	0	0	0	0	No
X_{229}	4	0	0	0	79	No
X_{230}	42	109	1	34	257	Yes
X_{231}	5	2	1	0	0	No
X_{232}	30	1,349	4	0	0	Yes
X_{233}	1	0	0	0	0	No
X_{234}	2	0	0	0	0	No
X_{235}	2	0	0	0	0	No
X_{236}	1	0	0	0	0	No
X_{237}	1	0	0	0	0	No
X_{238}	2	0	0	0	0	No
X_{239}	1	1	1	0	0	No
X_{240}	11	355	2	91	344	Yes
X_{241}	20	832	3	90	1,346	Yes

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{242}	2	55	2	0	0	No
X_{243}	26	0	0	19	378	No
X_{244}	1	12	1	0	0	No
X_{245}	1	0	0	1	1	No
X_{246}	3	80	2	0	0	Yes
X_{247}	1	1	1	0	0	No
X_{248}	7	1	1	2	60	No
X_{249}	1	0	0	0	0	No
X_{250}	1	0	0	0	0	No
X_{251}	1	251	2	0	0	Yes
X_{252}	4	216	1	13	122	Yes
X_{253}	1	11	1	0	0	No
X_{254}	1	0	0	0	0	No
X_{255}	3	6	2	1	1	No
X_{256}	9	1,008	2	8	279	Yes
X_{257}	1	0	0	0	0	No
X_{258}	1	0	0	0	0	No
X_{259}	1	0	0	0	0	No
X_{260}	1	1	1	0	0	No
X_{261}	1	0	0	0	0	No
X_{262}	11	28	2	0	0	No
X_{263}	1	1	1	0	0	No
X_{264}	3	0	0	0	0	No
X_{265}	15	221	2	0	0	Yes
X_{266}	1	0	0	0	0	No
X_{267}	1	0	0	0	0	No
X_{268}	2	0	0	0	0	No
X_{269}	36	930	2	25	841	Yes
X_{270}	1	5	2	0	0	No
X_{271}	1	0	0	0	0	No
X_{272}	1	0	0	0	0	No
X_{273}	1	0	0	0	0	No
X_{274}	2	1	1	0	0	No
X_{275}	1	12	1	0	0	Yes
X_{276}	3	0	0	0	5	No

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{277}	1	0	0	0	0	No
X_{278}	1	60	1	3	13	Yes
X_{279}	3	804	2	8	586	Yes
X_{280}	7	0	0	0	0	No
X_{281}	1	0	0	0	0	No
X_{282}	1	0	0	0	0	No
X_{283}	42	66	2	0	0	No
X_{284}	59	491	2	0	0	Yes
X_{285}	21	635	3	46	1,681	Yes
X_{286}	1	1	1	0	0	No
X_{287}	1	0	0	0	0	No
X_{288}	1	1	1	0	0	No
X_{289}	3	4	2	0	0	No
X_{290}	1	0	0	0	0	No
X_{291}	1	0	0	0	0	No
X_{292}	1	0	0	0	0	No
X_{293}	254	134	1	0	0	No
X_{294}	1	0	0	0	0	No
X_{295}	1	2	2	0	0	No
X_{296}	2	0	0	0	0	No
X_{297}	1	0	0	0	0	No
X_{298}	1	0	0	0	0	No
X_{299}	1	1	1	0	0	No
X_{300}	5	1	1	0	0	No
X_{301}	56	5	2	0	0	No
X_{302}	2	0	0	0	0	No
X_{303}	1	0	0	0	0	No
X_{304}	1	0	0	0	0	No
X_{305}	1	5	2	0	0	No
X_{306}	1	0	0	0	0	No
X_{307}	1	4	1	1	1	Yes
X_{308}	21	48	1	11	69	No
X_{309}	25	3	1	4	20	No
X_{310}	1	2	2	0	0	No
X_{311}	7	1,592	3	4	64	Yes

Continued on next page

Author	QA e-mails	Other e-mails	Lists	Bugs	Comments	Code Contributor
X_{312}	3	698	2	25	561	Yes
X_{313}	1	0	0	0	0	No
X_{314}	4	0	0	0	0	No
X_{315}	2	0	0	0	0	No
X_{316}	1	0	0	0	0	No
X_{317}	1	0	0	3	10	No

Table C.14: LibreOffice activity levels on a yearly basis

Venue	2010	2011	2012	2013
Mail No.	12,344	57,745	82,658	39,962
QA Mail No.	0	585	1,848	948
Bugs No.	691	5,959	8,603	4,179
Comment No.	6,565	47,927	52,041	20,675

Bibliography

[1] Mark Aberdour. Achieving quality in open source software. *IEEE Software*, 24(1):58–64, January 2007.

[2] Philippe Aigrain. Positive intellectual rights and information exchanges. In Rishab Aiyer Ghosh, editor, *CODE: Collaborative Ownership and the Digital Economy*, pages 287–315. MIT Press, 2005.

[3] Sinan Aral, Erik Brynjolfsson, and Marshall W. van Alstyne. Productivity effects of information diffusion in e-mail networks. In *Proceedings of the International Conference on Information Systems, ICIS 2007, Montreal, Quebec, Canada, December 9-12, 2007*, page 17. Association for Information Systems, 2007.

[4] Martin Aspeli. Plone: A model of a mature open source project. Master's thesis, London School of Economics, 2004.

[5] Yoris A. Au, Darrell Carpenter, Xiaogang Chen, and Jan Guynes Clark. Virtual organizational learning in open source software development projects. *Information and Management*, 46(1):9–15, 2009.

[6] Jono Bacon. *The Art of Community - Building the New Age of Participation, 2nd Edition.* O'Reilly, 2012.

[7] Albert-Laszlo Barabasi. *Linked: How Everything Is Connected to Everything Else and What It Means for Business, Science, and Everyday Life.* Plume Books, 2003.

[8] Donato Barbagallo, Chlara Francalenei, and Francesco Merlo. The impact of social networking on software design quality and development effort in open source projects. In *Proceedings of the International Conference on Information Systems, ICIS 2008, Paris, France, December 14-17, 2008*, page 201. Association for Information Systems, 2008.

[9] Flore Barcellini, Françoise Détienne, Jean-Marie Burkhardt, and Warren Sack. Thematic coherence and quotation practices in OSS design-oriented online discussions. In *Proceedings of the 2005 International ACM SIGGROUP Conference on Supporting Group Work*, GROUP '05, pages 177–186. ACM, 2005.

[10] Adina Barham. The emergence of quality assurance in open source software development. In *Proceedings of the OSS 2011 Doctoral Consortium*, volume 20, pages 1–19. Tampere University of Technology, 2011.

[11] Adina Barham. The impact of QA practices on FLOSS communities. In *Proceedings of the OSS 2013 Doctoral Consortium*, volume 22. Skövde University Studies in Informatics (SUSI), 2013.

[12] Andrea Bonaccorsi, Silvia Giannangeli, and Cristina Rossi. Entry strategies under competing standards: Hybrid business models in the open source software industry. *Management Science - Management*, 52(7):1085–1098, July 2006.

[13] Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering.* Addison-Wesley Professional, 2nd edition, 1995.

[14] Andrea Capiluppi and Karl Beecher. Structural complexity and decay in FLOSS systems: An inter-repository study. *13th European Conference on Software Maintenance and Reengineering (CSMR 2009)*, pages 169–178, 2009.

[15] Kevin Carillo and Chitu Okoli. The open source movement: a revolution in software development. *Journal of Computer Information Systems*, 49(7):1–9, July 2008.

[16] Paul E Ceruzzi. *A History of Modern Computing.* MIT Press, 2000.

[17] InduShobha N. Chengalur-Smith, Anna Sidorova, and Sherae L. Daniel. Sustainability of free/libre open source projects: A longitudinal study. *Journal of the Association for Information Systems*, 11(11), 2010.

[18] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994.

[19] Scott Christley and Greg Madey. Global and temporal analysis of social positions at SourceForge.net. In *Proceedings of the Third International Conference on Open Source Systems (OSS)*, Limerick, Ireland, 2007. IFIP WG 2.13.

[20] J. Daniel Couger and Robert A. Zawacki. *Motivating and Managing Computer Personnel.* John Wiley & Sons, Inc., 1st edition, 1980.

[21] Alan Cox. Cathedrals, Bazaars and the Town Council. *Slashdot*, October 1998.

[22] Philip B Crosby. *Quality Is Free.* McGraw-Hill, 1979.

[23] Kevin Crowston, Hala Annabi, and James Howison. Defining open source software project success. In *Proceedings of the 24th International Conference on Information Systems (ICIS 2003*, pages 327–340, 2003.

[24] Kevin Crowston and James Howison. The social structure of free and open source software development. *First Monday*, 10(2), 2005.

[25] Kevin Crowston, Qing Li, Kangning Wei, U. Yeliz Eseryel, and James Howison. Self-organization of teams for free/libre open source software development. *Information & Software Technology*, 49(6):564–575, 2007.

[26] B. Curtis and et al. People management capability maturity model. Technical report, Software Engineering Institute, 1994.

[27] Jean-Michel Dalle, Matthijs den Besten, and Héla Masmoudi. Channeling Firefox developers: Mom and Dad aren't happy yet. In *Open Source Development, Communities and Quality, IFIP 20th World Computer Congress, Working Group 2.3 on Open Source Software, OSS 2008, September 7-10, 2008, Milano, Italy*, volume 275 of *IFIP*, pages 265–271. Springer, 2008.

[28] Tom DeMarco and Timothy Lister. *Peopleware: Productive Projects and Teams.* Dorset House, 2nd edition, 1999.

[29] Hla Masmoudi & Matthijs L. den Besten & Claude De Loupy& Jean-Michel Dalle. "Peeling the onion": The words and actions that distinguish core from periphery in bug reports and how core and periphery interact together. In *Fifth International Conference on Open Source Systems - OSS 2009, Sweden, June 2009*, FLOSS '10, 2009.

[30] Chris DiBona, Mark Stone, and Danese Cooper. *Open sources 2.0.* O'Reilly, 1st edition, 2005.

[31] Edsger W. Dijkstra. Notes on structured programming. In O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, editors, *Structured programming*, pages 1–82. Academic Press Ltd., 1972.

[32] Michael English, Chris Exton, Irene Rigon, and Brendan Cleary. Fault detection and prediction in an open-source software project. In Thomas J. Ostrand, editor, *PROMISE*, page 17. ACM, 2009.

[33] Joseph Feller and Brian Fitzgerald. *Understanding Open Source Software Development.* Addison-Wesley, 2002.

[34] Brian Fitzgerald. The transformation of open source software. *MIS Quarterly*, 30(3):587–598, September 2006.

[35] Bent Flyvbjerg. Five misunderstandings about case-study research. *Qualitative Inquiry*, 12(2):219–245, 2006.

[36] Karl Fogel. *Producing Open Source Software : How to Run a Successful Free Software Project.* O'Reilly, October 2005.

[37] Cristina Gacek, Tony Lawrie, and Budi Arief. The Many Meanings of Open Source. *IEEE Software*, 21(1):34–40, January 2004.

[38] Daniel M. German and Jesús M. González-Barahona. An empirical study of the reuse of software licensed under the gnu general public license. In *Proceedings of the 5th IFIP WG 2.13 International Conference on Open Source Systems, OSS 2009, Skövde, Sweden, June 3-6, 2009*, volume 299, pages 185–198. Springer, 2009.

[39] Rishab Aiyer Ghosh, Rüdiger Glott, Bernhard Krieger, and Gregorio Robles. Free/libre and open source software: Survey and study (FLOSS) part 4: Survey of developers. Technical report, International Institute of Infonomics, University of Maastricht, 2002.

[40] Mark S. Granovetter. The Strength of Weak Ties. *The American Journal of Sociology*, 78(6):1360–1380, 1973.

[41] A. Güneş Koru and Hongfang Liu. Identifying and characterizing change-prone classes in two large-scale open-source products. *Journal of Systems and Software*, 80(1):63–73, January 2007.

[42] Jungpil Hahn, Jae Y. Moon, and Chen Zhang. Emergence of new project teams from open source software developer networks: Impact of prior collaboration ties. *Information Systems Research*, 19(3):369–391, 2008.

[43] T. Halloran and W. Scherlis. High quality and open source software practices. In *Proceedings of the 2nd ICSE Workshop on Open Source*, 2002.

[44] Alexander Hars and Shaosong Ou. Working for free? Motivations for participating in open-source projects. *International Journal of Electronic Commerce*, 6(3):25–39, April 2002.

[45] Henrik Hedberg, Netta Iivari, Mikko Rajanen, and Lasse Harjumaa. Assuring quality and usability in open source software development. In *Proceedings of the 29th International Conference on Software Engineering Workshops, Washington, DC, USA*, ICSEW '07, pages 122–127. IEEE Computer Society, 2007.

[46] Pieter Hooimeijer and Westley Weimer. Modeling bug report quality. In R. E. Kurt Stirewalt, Alexander Egyed, and Bernd Fischer, editors, *22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007), November 5-9, 2007, Atlanta, Georgia, USA*, pages 34–43. ACM, 2007.

[47] Janet Hope. *Biobazaar the open source revolution and biotechnology.* Harvard University Press, 2008.

[48] Liaquat Hossain and David Zhu. Social networks and coordination performance of distributed software development teams. *The Journal of High Technology Management Research*, 20(1):52–61, 2009.

[49] James Howison, Kevin Crowston, and Andrea Wiggins. Validity issues in the use of social network analysis with digital trace data. *Journal of the Association for Information Systems*, 12, 2011.

[50] Chris Jensen and Walt Scacchi. Role migration and advancement processes in OSSD projects: A comparative case study. In *Proceedings of the 29th International Conference on Software Engineering*, ICSE '07, pages 364–374. IEEE Computer Society, 2007.

[51] Christopher Kelty. *Two Bits: The Cultural Significance of Free Software.* Duke University Press, 2008.

[52] Terhi Kilamo. Essential properties of open development communities. In *Proceedings of the OSS 2011 Doctoral Consortium*, volume 20, pages 20–27. Tampere University of Technology, 2011.

[53] Terhi Kilamo, Timo Aaltonen, and Teemu J. Heinimäki. BULB: Onion-based measuring of OSS communities. In *OSS*, pages 342–347, 2010.

[54] Terhi Kilamo, Tommi Mikkonen, and Santtu Mikkonen. My summer as a mole: Evaluating an open source community via participation. In

proceedings of Open Source Workshop 2009 in conjunction with the 4th IEEE Systems and Software Week, 2009.

[55] J. Lakhani, B. Wolf, J. Bates, and C. DiBona. The Boston Consulting Group Hacker Survey. Technical report, Bosting Consulting Group and Open Source Developers Network, 2002.

[56] Timothy C. Lethbridge, Susan Elliott Sim, and Janice Singer. Studying software engineers: Data collection techniques for software field studies. *Empirical Software Engineering*, 10(3):311–341, July 2005.

[57] Martin Michlmayr, Francis Hunt, and David Probert. Quality practices and problems in free software projects. In Marco Scotto and Giancarlo Succi, editors, *Proceedings of the First International Conference on Open Source Systems*, pages 24–28, Genova, Italy, 2005.

[58] Audris Mockus, Roy T. Fielding, and James Herbsleb. A case study of open source software development: the Apache server. In *Proceedings of the 22nd International Conference on Software Engineering*, ICSE '00, pages 263–272. ACM, 2000.

[59] Audris Mockus, Roy T. Fielding, and James D. Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):309–346, July 2002.

[60] Glyn Moody. *Rebel Code: Linux and the Open Source Revolution.* Perseus Publishing, 2002.

[61] Kumiyo Nakakoji, Yasuhiro Yamamoto, Yoshiyuki Nishinaka, Kouichi Kishida, and Yunwen Ye. Evolution patterns of open-source software systems and communities. *Proceedings of the International Workshop on Principles of Software Evolution*, pages 76–85, 2002.

[62] Tetsuo Noda, Terutaka Tansho, and Shane Coughlan. The effect of open source licensing on the evolution of business strategy. In *Proceedings of the 8th IFIP WG 2.13 International Conference on Open Source Systems, OSS 2012, Hammamet, Tunisia, September 10-13, 2012*, pages 344–349, 2012.

[63] Wouter De Nooy, Andrej Mrvar, and Vladimir Batagelj. *Exploratory Social Network Analysis with Pajek.* Cambridge University Press, 2011.

[64] Christopher Oezbek, Lutz Prechelt, and Florian Thiel. The onion has cancer: some social network analysis visualizations of open source project communication. In *Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*, FLOSS '10, pages 5–10. ACM, 2010.

[65] Tim O'Reilly. Lessons from open-source software development. *Communications of the ACM*, 42(4):32–37, 1999.

[66] Dewayne E. Perry, Susan Elliott Sim, and Steve M. Easterbrook. Case studies for software engineers. In *Software Engineering Workshop – Tutorial Notes. 29th Annual IEEE/NASA*, pages 96–159, 2005.

[67] Roger S. Pressman. *Software Engineering: A Practitioner's Approach.* McGraw-Hill Higher Education, 5th edition, 2001.

[68] Eric S. Raymond. *The Cathedral and the Bazaar.* O'Reilly, 1st edition, 1999.

[69] Howard Rheingold. *The virtual community : homesteading on the electronic frontier.* MIT Press, 2000.

[70] C. Robson. *Real World Research - A Resource for Social Scientists and Practitioner-Researchers.* Blackwell Publishing, 2nd edition, 2002.

[71] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009.

[72] Douglas C. Schmidt and Adam Porter. Leveraging open-source communities to improve the quality performance of open-source software. In *Paper presented at the First Workshop on Open-Source Software Engineering at 23rd International Conference on Software Engineering, Toronto, Canada*, 2001.

[73] Sonali K. Shah. Motivation, governance, and the viability of hybrid forms in open source software development. *Management Science - Management*, 52(7):1000–1014, 2006.

[74] Siraj Ahmed Shaikh and Antonio Cerone. Towards a metric for open source software quality. *Electronic Communications of the EASST*, 20, 2009.

[75] Srinarayan Sharma, Vijayan Sugumaran, and Balaji Rajagopalan. A framework for creating hybrid-open source software communities. *Information Systems Journal*, 12(1):7–26, 2002.

[76] Ian Sommerville. *Software engineering (5th ed.)*. Addison Wesley Longman, 1995.

[77] Sulayman K. Sowe, Ioannis Samoladas, Ioannis Stamelos, and Lefteris Angelis. Are FLOSS developers committing to CVS/SVN as much as they are talking in mailing lists? Challenges for integrating data from multiple repositories. In *3rd Workshop on Public Data about Software Development (WoPDaSD 2008)*, pages 49–54, 2008.

[78] Diomidis Spinellis, Georgios Gousios, Vassilios Karakoidas, Panagiotis Louridas, Paul J. Adams, Ioannis Samoladas, and Ioannis Stamelos. Evaluating the quality of open source software. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 233:5–28, 2009.

[79] Ioannis Stamelos, Lefteris Angelis, Apostolos Oikonomou, and Georgios L. Bleris. Code quality analysis in open source software development. *Information Systems Journal*, 12:43–60, 2002.

[80] Klaas Stol and Brian Fitzgerald. Uncovering theories in software engineering. In *2nd Workshop on a General Theory of Software Engineering (GTSE) collocated with ICSE 2013*, 2013.

[81] Steven H. Strogatz. Exploring complex networks. *Nature*, 410(6825):268–276, 2001.

[82] F. Tönnies and J. Harris. *Tönnies: Community and Civil Society*. Cambridge Texts in the History of Political Thought. Cambridge University Press, 2001.

[83] Sergi Valverde, Guy Theraulaz, Jacques Gautrais, Vincent Fourcassi, and Ricard V. Sol. Self-organization patterns in wasp and open source communities. *IEEE Intelligent Systems*, 21(2):36–40, 2006.

[84] Stanley Wasserman and Stephen D Berkowitz. *Advances in social network analysis: research in the social and behavioural sciences*. Sage Publications, 1994.

[85] Duncan J. Watts. *Six Degrees: The Science of a Connected Age*. W. W. Norton, 2004.

[86] Duncan J. Watts. A twenty-first century science. *Nature*, 445(7127):489–489, 2007.

[87] Steven Weber. *The Success of Open Source*. Harvard University Press, April 2004.

[88] Barry Wellman and Joseph Galaskiewicz. *Social structures: a network approach*. Cambridge University Press, 1988.

[89] Ken Whitaker. *Managing software maniacs: finding, managing, and rewarding a winning development team*. Wiley, 1994.

[90] Andrea Wiggins, James Howison, and Kevin Crowston. Social dynamics of FLOSS team communication across channels. In *Proceedings of the IFIP 2.13 Working Conference on Open Source Software (OSS)*, pages 131–142. Springer, 2008.

[91] John Willinsky. The unacknowledged convergence of open source, open access, and open science. *First Monday*, 10(8), 2005.

[92] Robert K. Yin. *Case Study Research: Design and Methods (Applied Social Research Methods)*. Sage Publications, 4th edition, 2008.

[93] Luyin Zhao and Sebastian Elbaum. Quality assurance under the open source development model. *Journal of Systems and Software*, 66(1):65–75, 2003.