Politecnico di Milano
Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingegneria

POLITECNICO
MILANO 1863

An approach to measure Community Smells in software development communities

Relatore: Prof.ssa Elisabetta Di Nitto
Correlatore: Dr. Damian Andrew Tamburri

Tesi di laurea di:
Simone Magnoni Matr. 816316

Anno Accademico 2015–2016

To my Family,
my Friends
and A.

Acknowledgments

Above all, I would like to express my deepest gratitude to professor Elisabetta Di Nitto and Damian Andrew Tamburri, who supported me in the elaboration of this master thesis with their precious guidance and advice.

I would like to thank all Open Source Software developers for their remarkable work, whoever participated to the survey and every single person who contacted me with a positive or negative comment about the questionnaire.

In conclusion, I would like to thank everybody who supported me, in any possible way, during these last years.

Simone

Abstract

Software development and software engineering are now more than ever a community effort because their success often depends on people and their socio-technical characteristics. Therefore, it becomes fundamental balancing delicate forces, such as global distance or organisational barriers, with ad-hoc global software engineering practices. In this complex community scenario, it is likely that the arise of unforeseen socio-technical circumstances requires extra attention from community leaders in order to reduce any additional socio-technical cost, known as Social Debt.

To offer support in these situations and study the causality around Social Debt within Open Source projects, we conducted an empirical research in order to define, operationalise and evaluate a Socio-technical Quality Framework for software development communities. Community Smells are synonyms of negative organisational and social patterns that represent a potential risk related to the presence of Social Debt. The proposed framework provides the identification and quantification of Community Smells and it is also constituted by a set of fundamental factors capable of tracking and quantifying organisational and socio-technical key qualities, constituting a tool that can be used for continuous Social Debt management and improvement, much like code analysis and verification are used to improve software products.

We evaluated our framework on 60 Open Source development communities and made several key findings concerning organisational and socio-technical quality factors correlated to the occurrence of Community Smells, thus capable of influencing the wellbeing of software projects. Moreover, we determined several socio-technical quality thresholds and identified some developer perceptions capable of acting as qualitative indicators of the presence of Community Smells.

Sommario

L'ingegneria e lo sviluppo del software sono ora più che mai uno sforzo comunitario, dal momento che il loro successo dipende dalle persone e dalle loro caratteristiche socio-tecniche. E' dunque diventato fondamentale equilibrare forze delicate, come la distanza globale e le barriere organizzative, con pratiche ad-hoc di ingegneria del software. In questo complesso scenario è possibile che l'insorgere di circostanze socio-tecniche impreviste richieda una maggiore attenzione da parte dei leader di una comunità al fine di ridurre ogni costo socio-tecnico aggiuntivo, noto come Social Debt.

Per offrire un supporto in queste situazioni e studiare le caratteristiche del Social Debt in progetti Open Source, è stata condotta una ricerca empirica al fine di definire, operazionalizzare e valutare un Socio-technical Quality Framework per comunità di sviluppo software. I Community Smells sono modelli sociali e organizzativi negativi che rappresentano un potenziale rischio relativo alla presenza di Social Debt. Il framework proposto fornisce l'identificazione e la quantificazione di Community Smells ed è inoltre costituito da un insieme di fattori capaci di tracciare e quantificare importanti qualità socio-tecniche e organizzative, costituendo uno strumento che può essere utilizzato per migliorare e gestire in modo continuo il Social Debt, così come l'analisi e la verifica del codice sono usate per migliorare i prodotti software.

Il framework è stato valutato in 60 comunità di sviluppo Open Source e sono stati individuati molti fattori socio-tecnici ed organizzativi correlati alla presenza di Community Smells, quindi in grado di influenzare il benessere dei progetti software. Inoltre, sono state determinate diverse soglie di qualità ed identificate alcune percezioni degli sviluppatori in grado di fungere da indicatori qualitativi della presenza di Community Smells.

Contents

Chapter 1

Introduction

In the last decade, software became predominantly engineered by large and globally-distributed communities and consequently, now more than ever, it is of vital importance knowing more on the quality of these communities to ensure the success of a software project [1]. Socio-technical decisions, like changing the organisational structure of a software development community or its internal development processes (e.g., adopting agile methods), modify how people work and interact with each other and, as a side effect, they influence the well-being and success of the software project [2].

Previous researches revealed that software development communities can develop ills that collectively contribute to a form of additional project cost that was defined *Social Debt* [3], which is similar but parallel to Technical Debt [4], because it represents additional project costs not necessarily related to the source code itself but rather to its "social" nature and thus it is correlated to sub-optimal organisational structure and socio-technical characteristics of a software community.

This master thesis elaborates, validates and discusses a *Socio-technical Quality Framework* for software development communities, constituted by *quality factors*, which reflect projects' organisational and socio-technical characteristics (e.g., socio-technical congruence [5]), and *Community Smells*, which identify sub-optimal organisational and socio-technical characteristics that lead to issues in communities' organisational and social structures [6]. To the best of our knowledge, our Socio-technical Quality Framework is the first of its kind and may well inspire further research in the intriguing social software engineering field of managing Social Debt, through the identification and quantification of Community Smells.

In order to incorporate within our Socio-technical Quality Framework the most relevant software development community quality factors, we designed it considering the organisational and socio-technical literature, metrics elicited from Social Debt researches [7] and by means of a survey to FLOSS developers featuring almost 60 respondents, executed with the goal of isolating critical success and failure factors within software development communities in three large and widely known Open Source communities: Firefox, LibreOffice and FFmpeg. The resulting framework is constituted by a total of 40 quality factors, classified in five different categories of metrics: Developer Social Network, socio-technical, core community members, Turnover and Social Network Analysis. Furthermore, we defined identification patterns of several Community Smells and operationalised them within the tool we developed.

We proceeded by formulating several hypotheses on potential correlations between specific quality factors belonging to the Socio-technical Quality Framework and the occurrence of Community Smells defined within the model and then we evaluated our hypotheses against our corpus of data, consisting of community quality factors and occurrences of Community Smells for 60 analysed Open Source Software development communities.

As a result of our evaluation, we found several valuable insights to assess the quality of software development communities. For example, considering the literature [8] we conjectured that a higher number of developers sponsored by commercial companies would lead to higher community attractiveness and health, which in turn would lead to a lower number of Community Smells. Conversely we found that Community Smells increase quadratically with the linear growth of the number of developers sponsored by commercial companies, while for software communities below 50 trimestral participants the number of sponsored developers becomes irrelevant. Furthermore, we conjectured and later verified that socio-technical congruence leads to a lesser number of Community Smells and thus to a higher quality of organisational and social structures. Within the various proposed hypotheses, we conjectured that the number of time-zones, representing the geographic and temporal dispersion of a software development community, would weigh heavily on the creation of Community Smells, but we observed that the number of time-zones involved in the development activity do not mediate in any way the emergence of any Community Smell currently detected by our model.

Moreover, we executed a questionnaire in three large and widely known FLOSS development communities (Firefox, LibreOffice and FFmpeg), aimed at investigating if perceptions of FLOSS developers can be used as indicators of the presence of Community Smells within a development community and we achieved many interesting findings. For example, we discovered that software development communities with higher perceived documentation quality are characterised by less Community Smells.

Chapter 2 discusses the state of the art of important aspects that were fundamental in the definition and elaboration of this master thesis. Chapter 3 provides an overview of the problem analysis and research questions that are at the foundation of this work. Identification patterns of several Community Smells are defined in

Chapter 4, while the complete list of identified key quality factors composing our Socio-technical Quality Framework are presented in Chapter 5. Chapter 6 discusses the survey proposed to Firefox, LibreOffice and FFmpeg development communities and provides several findings related to Community Smells and Social Debt. Our implementation of Community Smells' identification patterns and quality factors belonging to our Socio-technical Quality Framework is proposed and explained in Chapter 7. Further on, Chapter 8 provides the evaluation of our work with the relative findings related to the occurrence of Community Smells within FLOSS development communities and Chapter 9 concludes this master thesis.

Chapter 2

State of the art

This chapter introduces background information and related work which were fundamental in the formulation and execution of this master thesis. The state of the art of Conway's law research field, presented in Section 2.1, provided the theoretical concepts and hypothesis at the foundation of our research; moreover, concepts summarised in this section were important in the identification and definition of quality factors constituting our Socio-technical Quality Framework (e.g., socio-technical congruence). Sections related to Global Software Development and to its most particular case constituted by Free/Libre and Open Source Software, discussed respectively in Section 2.2 and Section 2.3, are introduced to provide the necessary background information to understand the context of our empirical research; furthermore, the provided concepts were fundamental to identify potential socio-technical issues that are intrinsic within the two typologies of development environments studied within the literature, in order to further comprise and define quality factors capable of capturing every aspect of a software development community and their associated side effects within our framework. Section 2.4 presents a set of researches, in order to demonstrate the validity, effectiveness and efficacy of using Developer Social Networks in empirical software engineering researches, build considering either mailing lists or Version Control Systems. Social Debt and its technical counterpart are introduced in Section 2.5, to ensure a deeper understanding of the main topics covered within this master thesis and to identify potential quality factors capable of impacting the health of a software development community (e.g., communicability). Finally, Section 2.6 summarises two important software engineering researches that provided the theoretical foundations and verified the effectiveness and validity of the applied empirical research approach of this master thesis.

2.1 Conway's law and beyond

In 1968 with his article titled "How do committees invent?" [9], Dr. Melvin Conway introduced for the first time the idea, now commonly called "*Conway's law*", that systems designed by an organisation are constrained to produce designs which are copies of the communication structure of the same organisation. Conway, through the use of linear-graph notation, demonstrated that there is a very close relationship between the structure of a system and the structure of the organisation which designed it. The consequence of this homomorphism is that if subsystems do have their own separate design group then the structure of each design group and the system's organisation will be identical, otherwise if the same group designed multiple subsystems, every subsystem's structure will have the same design group collapsed into one node representing that group. The phenomenon described by Conway's law is more evident as the organisation size increases, because its flexibility diminishes.

Software development is characterized by a technical and a social component. The technical component is composed by the processes, tasks and technologies used during the software development, while the social component is constituted by organisations and people involved in the development and their characteristics. Due to this dichotomy, *software development can be considered a social-technical activity*, in which the technical and the social components need to be aligned to succeed [5].

To design a computer program or any other type of artefact, the initial steps are more related to design activity rather then to the system itself since the design activity cannot proceed until its boundaries and the boundaries of the system to be defined are understood and until a preliminary notion of the system's organisation is achieved. As a consequence of this Conway concluded that "the very act of organizing a design team means that certain design decisions have already been made, explicitly or otherwise". The steps after the choice of such preliminary system concepts are [9]: organisation of the design activity and delegation of tasks according to that concept, coordination among delegated tasks and consolidation of sub-designs into a single design . A system is then structured from the interconnection of smaller subsystems, and so on, until a stage in which the subsystems are easy enough to be understood without further subdivisions is reached. Large systems naturally tend to disintegrate themselves more than small systems during the development activities and so a system management activity should be used to mitigate this dangerous characteristic. To achieve an effective coordination among teams, architecture is not the only dimension that should be considered but even plans, processes and coordination mechanisms are fundamental elements [10].

Fred Brooks in his book titled "The Mythical Man-Month" [11], in agreement with Melvin Conway's theory, verified that the product quality is strongly related to the organisational structure. In real world projects estimates can be inaccurate, processes may be executed imperfectly, requirements and technology changes and people leave. Even if mature companies should anticipate those events, empirical studies suggest that developers rely on informal communications to correct errors and problems raising from that kind of events [10].

Due to the homomorphic relation between components and the organisational structure, Conway [9] proposed the theory that a team can work on many components but that a single module must be assigned to a single team. This concept was introduced for the first time in the software engineering field in 1972 by Parnas [12], who argued that software modules should not only be considered a collection of sub-programs but instead as work items. Different modules can be developed in parallel and independently from each other and the development time should be shortened since separate teams work on different modules and, as a consequence, the communication need is reduced.

In single location organisations, informal communications and informal meetings are surprisingly important to keep the project coordinated and to resolve conflicts as soon as possible. Conway's law do not consider this type of communications because they are implicit and invisible in single site companies.

In 1999 Herbsleb and Grinter [10] analysed Conway's law validity and scalability in Global Software Development (GSD). In geographically distributed development environments different time zones, languages and cultures may complicate informal communications. This distributed development's side effect, in conjunction with the impossibility of frequent informal physical meetings, influences the project's coordination structure since participants may not be aware of a coordination need toward other members, without having informal communications with them. In their work, Herbsleb and Grinter, identified the following coordination activities that were influenced by the geographically distributed nature of Global Software Development: knowing whom to contact, difficulty of initial contact, ability to communicate effectively and lack of trust.

Since coordination mechanisms and informal communications are both very important in geographically distributed environments, methods to overcome and mitigate the effect of temporal, cultural and geographic diversity are vital. The following quality approaches can be implemented to reduce the cross-communication needs between different geographically distributed sites of a same company, and thus reducing one of the GSD's critical factor [10]:

- consider Conway's law and keep a good modular design and assign separated tasks to different sites;
- do not split the development across different sites, whenever it is possible, to avoid the creation of instabilities;
- maintain the documentation available and up to date;

- overcome informal communication barriers whenever it is possible, encouraging the establishment of informal cross-site relationships (e.g. front-load travels).

In 2002 Mockus [13] considered mailing lists, code repositories and issue tracking systems of two important FLOSS projects and analysed developers participation and community metrics. He concluded that a *higher organisational structure modularity may indicate a lower coordination* need and that a "communication-only" approach in a distributed software development environment, as FLOSS ecosystem, does not scale because communication channels will be overwhelmed as the size and the complexity of the project and community will grow.

Since the beginning of software development, metrics were defined to estimate the quality of developed software (e.g. LOC, code churn, code complexity, code dependencies) but they measured only the technical aspect of software and ignored the "social" factor of software development which is related to people and to the organisational structure. Using Brooks' theory as a starting point, Nagappan et al. [1] analysed the relation between organisational structure and software quality. They proposed eight measures to quantify organisational complexity from the code viewpoint and empirically evaluated their efficacy to identify failure-prone binaries in a commercial project. The failure-proneness prediction model based on the organisational metrics outperformed traditional technical metrics (e.g. code churn, code complexity, LOC).

Conway's law and Brooks' theory imply that changes within the organisation can influence the software quality because the organisational structure will be modified. Those theories were addressed and verified by Mockus [14], who stated that one of the main goal of an organisation is to increase its efficiency. To achieve this goal an organisation should define roles, processes, formal reporting relationship roles and reporting relationships to improve its internal functioning; therefore, any change in the organisational structure will affect the software product quality.

In his research Mockus [14] investigated the relationship between developer-centric measures of organisational change and the probability of customer-reported defects in the context of a large software project and found that factors negatively influencing the software quality are: proximity to an organisational change, distributed development and recent developers departure. Mockus was not able to identify at what extent the organisational volatility causes or is a cause of defects but from his results it was evident that larger size organisations were associated with higher chances of defects. This correlation can be caused by the increased coordination need and by the reduced decision making speed.

The concept of socio-technical congruence was introduced by Cataldo et al. as the "match between the coordination requirements established by the dependencies among tasks and the actual coordination activities carried out by the engineers" [5, 15]. Cataldo et al. discovered that *socio-technical congruence is highly correlated* *to the software development productivity*: a higher socio-technical congruence is proven to speed-up software development, reducing the amount of time needed to perform a task and they demonstrated that over time developers learn to use the available communication channels in such a way to reach a higher congruence, thus if the development base is stable then the socio-technical congruence should increase over time [15].

The concept of socio-technical congruence was later redefined in 2008 by Sarma et al. as "the state in which a software development organisation harbors sufficient coordination capabilities to meet the coordination demands of the technical products under development" [16] and the following socio-technical congruence characteristics were identified:

1. it represents a *state* because it captures a particular moment in time of the company's social and technical context;

2. it is *descriptive* of a certain state in which an organisation finds itself in a defined time because individuals that perform the work may take non-optimal decisions;

3. it is *dynamic* because the technical and social structures change and evolve over time;

4. it is *multi-dimensional* because it depends on every possible way to coordinate work;

5. it can be considered at *multiple levels*: individuals, sub-teams, teams or entire organisation;

6. it i*nvolves trade-offs* because congruence may differ in every considered level and achieving congruence in a level may create an incongruence in another one.

In 2010 Colfer and Baldwin [17] complemented Conway's law verifying the validity of their *mirroring hypothesis*, which assumed that the organisational patters of a development community (e.g. team co-membership and geographic distribution, communication links) mirror the technical dependency patterns of the software under development. Their contribution added the opposite causality relationship to Conway's law: *the technical structure mirrors the organisational structure*, essentially turning Conway's original argument into an *isomorphism*. In other words, a change to the communication structure will eventually trigger a change to the design structure to return the socio-technical system into a state of socio-technical congruence.

The relation between software development organisational changes (e.g. forks, company acquisition, open-sourcing) and software quality was addressed even by

Sato et al. [18], who demonstrated that when multiple organisations (concurrently or in temporal succession) modify the same file, the increased modification frequency and complexity will lead the file to be more faulty.

Summing up, socio-technical congruence states that if two people work on dependent tasks then they need to communicate with each other. Communication needs can be computed analysing code modules and their inter-dependences. For example, if two developers work on inter-dependent modules then they have to communicate to coordinate their work and if they do not communicate and this gap is detected, then it can suggests a coordination problem. Socio-technical congruence management consists in reducing the number of this kind of gaps. To minimize those gaps it is possible to promote coordination mechanisms or reducing the coordination needs (e.g. reducing modules inter-dependences [12]).

2.2 Global Software Development

One of the main innovations that is characterizing the 21th century is globalisation: "the process of international integration arising from the interchange of world views, products, ideas and mutual sharing, and other aspects of culture" [19]. Globalisation is influencing every aspect of our life, from politic to economy, from society to technological systems, through the connection and integration of companies, people and nations on a global scale.

Friedman in his book "The World Is Flat: A brief history of the twenty-first century" defined ten "flattener" events that allowed all commercial competitors to have the same opportunities by playing with the same set of rules, enabling the globalisation process. Those historical events are [20]:

1. *11 September 1989 – "Fall of Berlin wall"*: represents the end of the Cold War and the revolutionary possibility of creating personal software programs, contents and interconnections with other people around the world using Personal Computers;

2. *8 September 1995 – "Netscape"*: Internet is accessible to everyone;

3. *"Work-flow software"*: virtual applications are now able to cooperate without human assistance. This is considered by Friedman the "genesis" because it is the moment in time where the global platform enabling multiple forms of collaboration was born;

4. *"Open-sourcing"*: communities collaborate and upload their work on on-line projects;

5. *"Outsourcing"*: companies can split and externalise their activities in efficient and effective ways;

6. *"Off-shoring"*: international relocation of company's processes in countries where production costs are lower;

7. *"Supply-chaining"*: supply and demand management is integrated across companies;

8. *"In-sourcing"*: Commercial companies employees perform services for connected third party companies;

9. *"Informing"*: social and search engines and information-rich websites allow access to a massive amount of information;

10. *"The steroids"*: any analogical content can be digitised and telematically transmitted at high speed, in mobility, any time and by anyone.

As a consequence of outsourcing and off-shoring flatteners, commercial software started to be developed by different geographically distributed and cooperative commercial software companies. There two flatteners are the fundamental prerequisites to enable Global Software Development (GSD), that is defined as "the nature of globalisation which reduces temporal, geographic, social, and cultural distance across countries" [21].

Global Software Development differs from traditional software development because in addition to the customer that buys the software and the commercial company that sells it, there are suppliers whom develop software through the mechanism of outsourcing and off-shoring. Global Software Development can be attractive for commercial companies because it reduces production costs due to its off-shoring nature, it allows companies to hire the best developers from any country of the world, it creates the chance of constitute virtual corporations in very fast ways, it allows to benefit from proximity to the market and it enables a "round the clock" software development approach, through the exploitation of different time zones, improving the time-to-market.

In a Global Software Development environment, as previously seen, the lack of communications between developers and the assignment of the same task to two different geographical sites can compromise the development success. Considering the off-shoring characteristic of GSD, social and cultural differences between developers can impact on the overall trust and software quality. To achieve the best performance from a GSD approach, commercial companies should take some precautions to avoid potential side effects, that can be categorized largely as temporal, geographical, social and cultural barriers [21]. Some useful strategies to limit Global Software Development side effects are: communication and coordination executed through common processes, strategic sub-division of tasks [12], offer cultural education to employees, understand diversity and taking advantage from it [22].

Diversity arises from attributes that differentiate people as their demographic information (e.g., gender, nationality, age), their functional information (e.g., role, knowledge, expertise) or their subjective information (e.g., personality, ethic). Molleman et al. [23] considered team diversity by addressing demographic characteristics, personality traits, technical skills and knowledge characteristics and analysed their impact on team functioning and performance in industrial manufacturing and service environments. *The characteristics of a team can be considered at three different levels: global, shared and compositional.* Global characteristics can be measured at team level (e.g., time size), shared characteristics are related to individual team members perceptions that tend to be shared by all the other team members (e.g. mutual trust) and compositional characteristics are related to individual team members attributes (e.g., age, skills). Within global team characteristics Molleman et al. considered team size and verified the intuitive idea that the optimal team size depends on the team tasks and discovered that "if workers are independent or only have to share resources such as tools, a larger team will achieve a better performance" because team tasks will be simpler and require less coordination effort. This result is similar to the one obtained by Parnas [12] and other researches reported in Section 2.1. Molleman et al. concluded that demographical similarity (e.g., gender, age) facilitates team functioning and effectiveness, enhancing mutual linking and trust. On the opposite side demographic diversity can cause cliquishness, stereotyping and subgroups conflicts [23]. Earley et al. [24] discovered that even if team diversity negatively impacts team functioning and effectiveness in the short-medium term, its side-effect tends to be less relevant as time passes because a common identity will be created with the institution of ways to interact and communicate.

In conclusion, the increasing interest in Global Software Development created new generations of "software development processes, practices and trends such as ubiquitous computing, agile methodologies, project outsourcing, distributed software development, process improvement and standardization, mobile applications development, social networking, and process tailoring practices" [22]. These new typologies of software applications generated new software development trends and styles that should be implemented by commercial companies to improve their efficiency and effectiveness in software development (e.g., agile methods).

2.3 Free/Libre and Open Source Software

In February 1986 Richard Stallman, founder of the Free Software Foundation (FSF), defined "*Free Software*" as any software that respects user and community freedom, allowing users to be free to run, copy, study, change, improve and distribute the software. Free software is an ethical matter of liberty and freedom, it is not related to price. Free Software does not mean non-commercial and a free software program must be available for commercial use, development and distribution. To highlight the fundamental idea that it does not mean gratis, sometimes Free Software is called Free/Libre Software, adding the French or Spanish word that means free in the sense of freedom. Four fundamental freedoms were specified to define Free Software with the purpose of allowing users to control the program and what it can do for them. The four freedoms to classify a software as Free Software are [25]:

1. The freedom to run the program as you wish, for any purpose (*Freedom 0*);
2. The freedom to study how the program works and possibility to change it so it will compute as you wish (*Freedom 1*). Access to the source code is a precondition for this freedom;
3. The freedom to re-distribute copies, so you can help your neighbors (*Freedom 2*);
4. The freedom to distribute copies of your modified versions to others (*Freedom 3*). By doing this you can give to the whole community a chance to benefit from your changes. Access to the source code is a precondition for this freedom.

Free Software Foundation's social activism and the misunderstanding of the word "Free" were considered not appealing to commercial software companies by some developers and to promote the potential business deriving from the collaboration and the sharing of source code, the term "*Open Source*" was created and in February 1998 the Open Source Initiative was founded. A computer software is classified as Open Source Software (OSS) if its source code is available and it is licensed to provide the rights to study, change and distribute the software for any purpose. The Open Source Initiative states that Open Source does not just mean granting access to the source code but the software must obey to the following ten criteria [26]:

1. *Free re-distribution*: the license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale;
2. *Source code*: the program must include source code and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with its source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost, preferably downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed;

3. *Derived works*: the license must allow modifications and derived works and must allow them to be distributed under the same terms as the license of the original software;

4. *Integrity of the author's source code*: the license may restrict source-code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software;

5. *No discrimination against people or groups*: the license must not discriminate against any person or group of persons;

6. *No discrimination against fields of endeavor*: the license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research;

7. *Distribution of license*: the rights attached to the program must apply to all to whom the program is redistributed, without the need for execution of an additional license by those parties;

8. *License must not be specific to a product*: the rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution;

9. *License must not restrict other software*: the license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other software distributed on the same medium must be open-source software;

10. *License must be technology-neutral*: no provision of the license may be predicated on any individual technology or style of interface.

Even if legally Free Software is qualified as Open Source Software, the Free Software Foundation consider the term Open Source Software close but not identical to Free Software as the word "Open" never refers to freedom, which is one fundamental component of the Free Software definition [25]. Stallman said that the two terms describe almost the same category of software, but they stand for views based on fundamentally different values: *Free Software is a social movement and*

Open Source Software is a development methodology. He classifies the Free Software as an ethical imperative to respect the user freedom while Open Source concern is how improve software and increasing its popularity and success [27]. As the main difference between these definitions can be defined as political, in situations where the developer political views are not considered important it is possible to be neutral using the term *Free/Libre and Open Source Software* (FLOSS). The term Free and Open Source Software (FOSS) can also be used but the Free Software Foundation considers it misleading because it fails to explain that "free" refers to freedom [28]. In this master thesis we will use all the above definitions as synonyms.

An Open Source Community can be defined as a network platform in which the source code of the software is opened under an Open Source Software license agreement. Open Source Communities are fundamental for Open Source Software's promotion and development. In the last decade the interest in FLOSS grew widely and now many commercial companies develop, maintain and distribute their products through an Open Source Community (e.g. MySQL, Firefox). The pervasive diffusion and growing interest in FLOSS projects from both voluntary developers and commercial software companies constitute a precious asset since developers can extend, modify or reuse code from already existing projects. This possibility can increase developers productivity and reduce development costs.

FLOSS development can be driven by voluntary developers encouraging knowledge sharing rather than the protection of intellectual property, commercial companies and institutions from every part of the world. FLOSS software development is a perfect example of Global Software Development and it has all the GSD pros and cons. Group dynamics in FLOSS communities are substantially different from commercial off-line teams, for example [29]:

- *Geographic dispersion* and cultural differences are the norm, as community members rarely meet in person;
- Collaborators assemble in *on-line communities* and coordinate their activities through distributed communication channels (e.g., mail lists);
- *Teams are fluid*: they tend to form and dissolve organically around a specific task;
- *High turnover* since FLOSS contributors are often volunteers;
- FLOSS communities are generally constituted by a set of *core developers* and a more loosely coupled group of contributors that support the development by reporting issues, submitting patches or contributing with documentation (core-periphery structure).

Open Source communities have an implicit diverse nature as they are usually composed by a variety of contributors ranging from volunteers to developers sponsored by companies and all of them have different demographic characteristics, knowledge, personalities, skills, cultures and educations. Open Source Software projects can benefit from their intrinsic diversity since it stimulates creativity, diversity of ideas and problem solving skills coming from different background and knowledge, therefore increasing global projects productivity [29]. On the other hand if diversity increases but it is not managed, it may create conflicts within the development team negatively effecting the team's cohesiveness and its performance, due to greater perceived differences in values, ideas, norms and communication style. [23].

Software licenses define under which conditions software can be used, copied, modified or redistributed without incurring in legal problems. As proprietary licensing tend to restrict the possible ways in which a software will be used, Open Source licenses tend to limit the restrictions that can be associated to a software, to ensure development freedom and source code re-distribution. Usually the license under which a project is released can be explicated in three different ways:

1. Adding a licensing comment on top of each file of the project. This approach allows a fine-grained license definition;
2. A specific file specifies the license under which the software is released;
3. The software license is expressed within the project's official website.

Several FLOSS licenses exist, from highly restrictive (e.g., GPL) to more flexible (e.g., MIT), but their main goal is to promote and enable the right to fork, copy, modify and redistribute the program source code. As it evolves a project can change its license to better meet the requirements and needs of the development community or of external actors interested in the project. Vendome et al. [30] highlighted that the initial software license is influenced by the communities to which core developers are already contributing and that external actors do not have impact in this choice. As projects grow their current licenses are heavily affected by their need to be commercially reused and, to accomplish this purpose, Open Source projects tend to migrate toward less restrictive and more permissive licenses. Licenses do not only define how software code source can be reused but they might also affect other components of the projects where the FLOSS code will be used. For example GPL license requires that all the source code, in which a GPL licensed component is used, has to be released under the GPL license. More flexible licenses (e.g., MIT license) do not require this condition and allow the use of FLOSS code under any other license, including commercial use. This license dependency was found even by Vendome et al. [30] who stated that a license change of a sub-component might start a chain reaction that will influence the final project's license or will cause the drop of the sub-component usage due to incompatibilities between licenses.

The number of commercial driven Open Source Projects is increasing year after year, *FLOSS development has long become an important commercial activity* and the Open Source Software ecosystem is full of successful projects which are completely driven by commercial companies (e.g., Android) or which have developers sponsored by commercial companies (e.g., Linux). When a voluntary-based Open Source Software project become promising, commercial companies may be interested in participating in its development, to adequate the software to their needs. It is possible to consider the ratio of volunteer to paid work as an indicator for the health of FLOSS projects and it can aid project leaders in managing their community [8]. Riehle et al. [8] in 2014 discovered that even if Open Source Software has been growing near-exponentially, its global ratio of voluntary and paid development is almost constant. A possible explanation is that for every project which increases its economic significance receiving sponsored development, a new totally voluntary driven project is started. Voluntary FLOSS developers, in contrast to commercial software developers, usually experiment a high degree of development and organisational freedom with respect to the possible ways through which contribute to the project and how to organize themselves and their tasks.

2.4 Developer Social Networks

In a software development community every interaction and relationship between developers can be modelled through a self-organised network, which can be considered as a latent developer Social Network [31]. In such developer Social Network, considering the case of FLOSS projects, developers and their relationships are subjects to continuous variations and changes as the set of active developers and their activities change over time.

A Developer Social Networks (DSN) can be modelled through the use of nodes, that represent actors, and edges, that represent relationships between different actors (or groups). A Social Network and its actors have two fundamental properties: connection and distance [32]. *Connectivity* can be measured using density, size, centrality and reachability of the Social Network. As a member of the Social Network is more connected then he or she is exposed to more information, can be considered more influential in the community and may be easily influenced by others. *Distance* in a Social Network represents the closeness of two actors within the network and may be a useful indicator to identify macro-properties differences, like diffusion and homogeneity. Distance influences the information diffusion time across the community and it can be measured using walks and paths. Connections and distances are fundamental characteristics to enable the identification of sub-communities within a Social Network, which are defined as "subsets of actors among whom there are relatively strong, direct, intense, frequent, or positive ties" [32].

Since Free/Libre Open Source Software communities number increase daily, the amount of open and accessible information about FLOSS development grow exponentially. Issue tracking systems [33], mailing lists [34] and code repository histories [35] of FLOSS projects can be easily and freely mined by researchers to analyse defects, communication and distributed collaboration habits of Open Source Software developers. The existence of FLOSS project is enabled by Internet that allows communication and coordination (C&C) activities between developers. Such activities are typically *public and accessible to anyone* and this allows researchers to track and mine coordination and communication activities and study them through the usage of Developer Social Networks, in contrast to industrial closed-source projects where C&C activities are predominantly direct and informal [34].

FLOSS projects are extremely interesting for empirical software engineering studies because they imply a distributed development occurring at a global scale and all the information related to a project (communications, code modifications, bugs, etc.) are available on-line (mailing lists, code repositories, bug tracking systems, etc.), granting the possibility of mining them. To take advantage of this enormous quantity of available data and to be able to mine and make sense of the organisational, social, technical and communicational aspects of a FLOSS project, researchers should re-factor the retrieved information into a structured and analysable form. During last decade the main technique used to model technical and social aspects of software developers to enable the possibility of studying how people collaborate and organize their work in a global software development environment is the Social Network approach.

Public mailing lists are the classic channel used in FLOSS projects to perform communication and coordination activities and their archives (usually available online) in conjunction with VCS and other on-line development artefacts (e.g., bug tracking systems) allow researchers to create a developer Social Networks able to model and understand communication, coordination and collaboration practices and patterns in FLOSS projects.

During the last decade researchers have generated Developer Networks from every possible development source of information to enable the usage of Social Network Analysis methodologies and metrics. A Social Network can be created from code source history and mostly from any other kind of open and accessible data source used to support the software development (e.g., bug reporting [31], VCS [36] and mailing lists [34]). As an example, to conduct a knowledge-centric software engineering empirical study, the mailing list of a project should be considered: every member whom sent a message on the mailing list is considered a node and if a person A received a reply to one of his messages from another member B, then it exists an edge connecting A and B.

Conway law states that the project structure is strongly correlated to the organisational structure of the project, thus understanding the Developer Network is fundamental to estimate the quality and efficiency of software development activities. Social Network Analysis is based on individuals and how these individuals are related between them through relationships. In software engineering these relationships can be extracted mining software development artefacts, allowing to study all the possible ways in which people interact through all the available channels used to develop software. Empirical software engineering studies often apply SNA methodologies because they offer a solid systematic and quantitative framework.

FLOSS projects, due their intrinsic nature of being developed mainly by voluntary developers without a monetary retribution, can be affected by a sentiment of mistrust from companies that use them in their commercial activity. To avoid this phenomenon, in 2002, Madey et al. used a Social Networks approach to model FLOSS communities because "a better understanding of how the OSS community functions may help IT planners make more informed decisions and develop more effective strategies for using OSS software" [35].

One of the first attempt to use Social Network Analysis to analyse on-line communities was conducted in 2004 by Lin and Chen [32]. In their research study social ties, information flows, information and resource acquisition and coalitions creation were considered with the scope of accessing team collaborations, evaluate the performance of the system and enable the identification of relationships and interaction patterns within the community.

A socio-technical Developer Network can be created from socio-technical connections found exploiting the collaboration and communication channels and it can be analysed using SNA metrics. Social Network Analysis metrics calculated on socio-technical Developer Networks, created from connections observed in development artefacts, were proved to be representative of actual and real socio-technical relationships present within the software development communities [37]. Nia et al. [38] demonstrated that the effect of paths with broken information flow (consecutive edges which are out of temporal order) on the centrality measure of nodes within the network and the effect of missing links on such measures do not invalidate the Social Network Analysis metrics validity, but such metrics are stable with respect to such phenomenons. Betweenness centrality and clustering coefficient are stable in presence of a large number of missing links and this this essentially means that most of the activity in Developer Social Networks arise from few participants, thus it is sufficient to look at the 10% of developers [38].

In software engineering with the term *Version Control* (VC), it is considered any practice devoted to track and control changes to any possible project element: source code, documentation or configuration files. Since FLOSS development is intrinsically distributed and anybody can contribute modifying the source code, Version Control Systems (VCS) are extremely useful as they can track ownership of changes to the project source code. There are two main VCS typologies: centralized and distributed. *Centralized VCS* have a single central authoritative repository on which developers can synchronize their code-base; file locking and version merging are used to enable different developers to operate on the same file at the same time. Some famous Centralized VCS are: Concurrent Versions System (CVS) and Subversion (SVN). Opposed to the client-server approach of Centralized VCS, *Distributed VCS* implement a peer-to-peer approach as they don't have any central authoritative repository but the source code can be checkout and committed into any existing repository with a merge operation. Some famous Distributed VCS are: Git, Mercurial and Bazaar. Brindescu et al. [39] conducted an empirical software engineering study to compare the impact of Centralized VCS and Distributed VCS on software changes. They discovered that Distributed VCS have a smaller commit size in terms of lines of code and that hybrid repositories (repositories that migrated from a centralized to a distributed VCS) do not show any difference between the size of commits performed before and after the switch of paradigm due to commit policies formed in the team while using the centralized approach. In the past decade Distributed VCS saw an increase in popularity with respect to Centralized VCS and many popular FLOSS projects migrated from a centralized to a Distributed VCS. Distributed VCS have the following main differences respect to the centralized approach:

1. Only working copies exist because a reference copy of the code does not exist by default;
2. Every working copy is a remote backup of the change-history of the entire project;
3. It is possible to work without the need of being connected to a network;
4. Version Control operations are fast because no communication is needed;
5. Communications are necessary only when sharing a change between peers;
6. A web-of-trust approach can be used to merge changes coming from different repositories; this enables new work-flows that are impossible in Centralized VCS (e.g., intermediate roles can be responsible for integrating new changes proposed by developers);
7. Allow non-core developers (the ones who do not have write permissions on the repository) to contribute to the source code;
8. Authorship of changes of non-core developers is kept in historical records;
9. Individual changed lines of a file can be committed instead sending the whole file again;
10. Initial repository cloning is slower than Centralized check-out since all branches and change history are copied.

Version Control Systems (VCS) have been used to construct developer collaboration networks since the introduction of Social Networks and Social Network Analysis in empirical software engineering studies, due to their intrinsic capability of capturing inter-relationships among large software project components. Different VCS and VCS typologies will provide different grain level information to construct the collaboration network. For example Centralized VCS will provide only information about the committer, instead Distributed VCS will usually provide even information about the author of the commit. Since the information volume within a VCS can reach an incredible dimension, this can be considered a Big Data research area. Data retrieved from VCS is unusable without techniques to extract coherent information from this amount of data and highlight relevant trends and interesting aspects of a software project.

The first empirical software engineering research that considered VCS to generate a collaboration network was conducted by Lopez-Fernandez et al. in 2004 [36] and it proposed a set of Social Network Analysis methodologies to characterise the evolution and internal structure of FLOSS projects. They proposed to consider VCS committers or VCS directories (considered software modules) as nodes and the common commits as weighted edges between two nodes. In 2011 Jermakovics et al. [40] improved the methodology proposed by Lopez-Fernandez and allowed the generation of a more detailed and cleaner collaboration network, considering a file level grain instead of directories level to detect common commits (software modules). Developer networks generated using the methodologies proposed by Lopez-Fernandez or Jermakovics can be too dense and inefficient to obtain useful results during developer collaboration analysis. In 2015 Joblin et al. [41] addressed this problem and introduced a *collaboration network generation methodology that consider the code structure and detect when developers collaborated on the same function of a file, enabling a function level grain collaboration analysis*.

After 2004 every other possible development artefact was considered as a data source and used by researchers to create Developer Networks. Bird et al. [34] in 2006 were the first to exploit mailing list archives to construct a Developer Social Network of community members participating in a project. Always in 2006 Howison et al. [33] created for the first time in software engineering history a Developer Social Network from a bug reporting system. Both mailing lists and bug-tracking systems of FLOSS projects enable to explore communication and coordination activities of all the participants of a community and do not limiting the analysis just to software developers, as in the case of software code source (VCS) analysis, because mailing lists and bug-tracking systems contain many social interactions and bug reporting activities performed by users and people not necessarily directly involved in the software development (e.g., report bugs but do not provide patches).

Hong at al. [31] considered how and to what extreme Developer Social Networks can be analysed using General Social Networks (GSN) techniques, studied the evolution of Developer Social Networks in time and how the DSN topological structures can be influenced by project events (e.g., release, turnover). General Social Networks (e.g., Facebook, Twitter) as Developer Social Networks are founded on the freedom of participation but GSN offer more freedom of topics, while Developer Social Networks are mainly focused only on project development activities. Developer Social Networks are latent and not instantly usable, so they have to be extracted and constructed from information rich artefacts that support software development (e.g. VCS, mailing lists). Some other interesting aspects which characterise Developer Social Networks are [31]:

- DSN are usually characterised by a small portion of developers with high degree (core developers) and many developers with low degree, thus Developer Social Networks can be considered as *scale-free networks*;
- In DSN most pairs of developers can communicate or are connected between each other through an exiguous number of hops in the network ("*small world*");
- DSN are *highly modular*, thus they do have a significant community structure, and modularity tend to increase over time.

Social Network Analysis methodologies and metrics were used in many empirical software engineering studies to implement models capable of predicting faults [1], failures [42], and vulnerabilities [43]. Nan and Kumar [44] took advantage of Social Networks Analysis to examine the joint effect of developer team structure and software architecture in Open Source Software and discovered that they moderate each other's effect on software development performance. Valetto et al. [45] applied Social Networks theories to Developer Social Networks and defined a useful methodology to compute socio-technical congruence, which is based on the direct comparison of the structure of an organization with the project code source. In 2014 Jorge Colazo used Social Networks Analysis to analyse how collaboration DSNs change when collaborating teams become temporally dispersed and he discovered that "the collaboration structure networks of more temporally dispersed teams are sparser and more centralized, and these associations are stronger in those teams exhibiting higher relative performance" [46].

2.5 Technical and Social Debt

Since socio-technical decisions influence both the technical and social aspect of the software development environment, non-optimal or uninformed socio-technical decisions may generate additional costs to the technical or social area, or even both. Due to the nature of these additional costs, they can be considered as a debit because their resolution can be postponed in time since usually these non-optimal decisions are not easily detectable and visible.

Technical debt (TD) is a software engineering metaphor defined in 1992 by Cunningham [47] to describe the internal tasks that some decisions imply but that are not performed. If these tasks are not completed, the debt is not repaid and it will continue to accumulate interests, creating future problems and making it harder to implement changes in the future. A classical example of technical debt generated by a development team is when a decision that simplify a short term goal is taken but it has a great potential to negatively impact the development activity on the long term.

When a change in the source-code of a project is performed, it is often necessary to execute some other coordinated changes to other software components (e.g., other code modules, documentation) due to their inter-dependencies or due to development policies. Whenever this situation happen but the change associated to the software modification is delayed, technical debt arises and it must be paid off sooner or later in the future to avoid the failure of the software development.

Kruchten et al. redefined Technical Debt as “the invisible result of past decisions about software that negatively affect its future” [4], not limiting the concept to situations that imply a cost. Technical Debt is generated by invisible aspects of software aging and its evolution or it can be caused by external events. Some technical debt causes are: technological obsolescence, development environment changes, rapid commercial success and advent of new and better technologies.

During the past decade, technical debt was deeply studied and analysed along every software development life cycle process. In 2014 Alves et al. fathomed all the available literature related to Technical Debt and classified all its forms in the following ontology [48]:

- *Architecture debt*: issues in the project's architecture (e.g., violation of modularity) that affect some architectural requirements (e.g., performance, robustness). It usually cannot be repaid only through source code interventions but it implies more extensive corrective development activities;
- *Build debt*: issues that make task building more time and processing consuming and harder than the necessary (e.g., unnecessary code to the customer);
- *Code debt*: bad coding practices in the source code that impact on its maintainability (e.g., reducing its legibility);
- *Defect debt*: known software defects whose fix are deferred to the future due to different priorities or limited resources;
- *Design debt:* bad design practices that violate the principles of good object-oriented design;

- *Documentation debt*: missing, inadequate or incomplete project documentation;
- *Infrastructure debt*: software organization issues that can delay or hinter some development activities (e.g., infrastructure fix);
- *People debt*: people issues that can delay or hinder some development activities (e.g., new knowledge brokers);
- *Processes debt:* issues caused by inefficient processes;
- *Requirement debt*: trade-off between the requirements that a development team has to implement and how they implement them (e.g., requirements implement for a limited number of cases);
- *Service debt*: issues introduced by an inefficient web service substitution;
- *Test automation debt*: unnecessary work done by automated tests of previously developed functionality to support continuous integration and faster development cycles;
- *Test debt*: issues in testing activities that influence testing qualities (e.g. low code coverage).

Brown et al. defined the concept of "*anti-pattern*" as a "commonly occurring solution that will always generate negative consequences when it is applied to a recurring problem" [49], thus an anti-pattern is a pattern which implies negative connotations.

Within the Technical Debt research area, Fowler [50] defined the term "*Code Smell*" to refer to code patterns that can be symptoms of poor design and implementation choices. Code smells are usually considered as symptoms of the presence of anti-patterns and thus are mined to detect them. Since Code Smells can be characterised by suboptimal development choices or they can be associated to some poor recurring design and implementation decisions, they can diminish code comprehension and increase change and fault proneness of a project. *Code Smells can be used as indicators of the presence of accumulated Technical Debt* [51].

Tamburri et al. analysed another type of debt in which a software development may incur, generated by non-optimal socio-technical decisions. This "*Social Debt*" is correlated to the social components of an organisation and it was defined as the "*unforeseen project cost connected to a suboptimal development community*" [3]. Social Debt was later redefined in 2015 by the same authors as the "cumulative and increasing cost in the current state of things, connected to invisible and negative effects within a development community" [6].

While decisions in Technical Debt are about technologies and their applications, *decisions that cause Social Debt are about social interactions and people themselves.*

Social Debt shares many aspects with Technical Debt since they have many similarities and common points. Social Debt, as well as Technical Debt, can be used as an indicator of the development process quality, considered as the result of past accumulated decisions [6]. Tamburri et al. highlighted this relation between the two diametrically opposite typologies of debt paraphrasing the Cunningham's definition of technical debt and describing Social Debt as “not quite right development community - which we postpone making right” [6].

Global Software development is characterised by many socio-technical decisions (e.g., outsourcing, organisational structure, communications organization) that do not only influence the technical area but even the social one, influencing how people interact, communicate and organize themselves. Since socio-technical decisions can influence and modify people's social behaviours, in addition to Technical Debt, they may produce Social Debt due to non-optimal socio-technical decisions. Social and Technical Debt can generate delays and addictions costs within the development process or within the development community, that may increase over time and be invisible or intentionally delayed due to the intrinsic nature of social and technical debt.

De Farias Junior et al. [52] conducted a study on communication related risks in distributed software development that can be considered a Social Debt study because it analyses some organisational issues created by a non-optimal usage of communications within a software company. They considered as communication related risks: issues related to physical and temporal distance, trust, difference of cultural and linguistic orientations between different teams. To avoid or mitigate the listed communication related risks within a distributed software development, Farias Junior et al. proposed these recommendations:

- *encourage frequent communication*: it reduces misunderstandings created by cultural and linguistic differences and it increases distributed teams cohesion and trust, which can generate an increment informal communications between developers;

- *establish an appropriate communication infrastructure*: it addresses uncertainty and unpredictability of the communications and it reduces the negative effect of the absence of “face-to-face” meeting;

- *promote socialisation*: it increases cohesion, inter-personal relationships between different team members, communication effectiveness and informal communication;

- *encourage effective communication*;

- *promote visits among distributed sites*: it increases trust and it constitutes new interpersonal relationships with the creation of new informal communications;

- *promote informal communication*: it diminishes misunderstandings, it creates trust and facilitates knowledge sharing;

- *promote cultural awarenes*s and adopt group-ware applications.

In Social Debt studies, mirroring the Code Smell concept, it is possible to define "*Community Smells*" as social related anti-patterns useful to understand negative community characteristics and trends. *Community Smells are formally defined as "socio-technical anti-patterns that may appear normal but in fact reflect unlikeable community characteristics"* [6], thus Community Smells identify anti-social organisational behaviours within a community. An example of Social Debt is when developers refuse or delay information sharing for any reason. Community Smells are a set of social and organisational circumstances with implicit causal relations which do not constitute a problem if considered alone but that if repeated over time, they may cause Social Debt in the form of delays, mistrust, uninformed and miscommunicated architectural decision-making.

Social Debt, as its technical counterpart, can be paid back adopting specific socio-technical decisions with the purpose of mitigating a precise Social Debt aspect, possibly detected by a Community Smell. Tamburri et al. [6] found some "*mitigations*" that were proven to have a beneficial effect on Social Debt reduction and discovered that some socio-technical decisions made to extinguish contracted Social Debt eventually worsen the situation or did not yield positive outcomes (40% of socio-technical mitigations considered). Mitigations addressed to resolve Community Smells and to pay back the related Social Debt are called "deodorants".

Architectural decisions were considered in both Technical Debt [53] and Social Debt [7] studies and in both cases they are highlighted as one of the most important cause of debt generation in professional software environments. Ernst et al. [53] studied the relation between Technical Debt and architectural decisions and they concluded that architectural issues are the most relevant cause of technical debt generation and that to pay back such generated Technical Debt is hard because usually the incriminated architectural decisions were taken many years in the past. Tamburri et al. [7] further investigated architectural decisions with a Social Debt perspective, identifying architectural smells and proposing a possible metric to measure potential Social Debt contracted in software architecture processes. Their identification methodology, based of Social Network Analysis theories, computes the communicability of an architectural design decision to identify architectural smells with the purpose of avoiding or diminish the related side effects (e.g., architecture erosion, lack of vision, mistrust).

Referring to Conway's law and its related studies, it is possible to re-conduct Technical Debt to not-optimal development processes decisions and Social Debt can be re-conducted to not-optimal organisational processes. Since socio-technical decisions are indirectly correlated to Social Debt [6] and that socio-technical congruence can be considered an agreement indicator to Conway's law, *socio-technical congruence can be considered as a metric to identify possible Social Debt present within a community because it quantifies the similarity of social and technical processes whenever a communication need is present.*

Both Social or Technical Debt can depend from the context evolution because it is possible that the original decision which created it, was correct but as time passed the context changed and such decision was not positive in retrospect.

In analogy to the monetary debt, Technical or Social Debt in software engineering is not necessary a bad thing if it is known, accepted and controlled. For example sometimes Technical Debt is necessary to move forward the project development. Debt, similarly to congruence [16], may imply trade-offs because resolving the debt in a particular level may create another debt in another level. Potdat et al. [54] discovered that self-admitted Technical Debt in Open Software development is a common phenomenon (from 2.4% to 31% of project files is affected), that developers with higher experience usually tend to introduce most of the self-admitted Technical Debt and that in the optimal case only slightly more than half of the introduced debt is paid off.

In their Technical Debt ontology published in 2014, Alves et al. [48] identified a Technical Debt category called "people debt" that can be associated to the concept of Social Debt. They define people debt as the debt associated to people issues, in the context of software organization, which may delay or hinder development activities and they provide as an example the case of a concentration of expertise limited to few people as a consequence of delayed training and/or hiring.

Another study that can be considered belonging to the Social Debt research area is the one conducted in 2014 by Zhou et al. [55], concerning the quantification of global team performance and profitability, because it investigated the effects of some social and organisational structure properties (e.g., temporal dispersion, language difference, skilled workers turnover). Zhou et al. state that "the right organisational structure is required to achieve benefits of lower labour costs", acknowledging the fact that "combined effects of the [social and organisational] factors could lead to reduced profitability" and concluded, in complete accord to Social Debt definition, that "in some extreme cases the cumulative effects of external factors could outweigh the advantage of lower labour rates for globally outsourced work".

In "Why good developers write bad code", published in 2015, Lavallée et al [56] analysed the relationships between some organisational factors and their impact on developers' working conditions and performances. Lavallée et al. identified the following socio-technical organisational issues that may compromise the software quality and its success [56]:

- *Documenting complexity*: presence of large, old and poor documentation that causes the unwanted situation in which unimplemented requirements are discovered at the end of software development. This issue is related to knowledge management and support maintenance activity transmissions;

- *Internal dependencies*: presence of many inter-dependent modules that create conflicts between projects on the deployment schedule;
- *External dependencies*: changes to third-party modules result in costly delays;
- *Cloud storage*: third-parties do not support vulnerability testing;
- *Organically grown processes:* software defects are documented but developers are not aware of them because information exchange is hindered by frontiers between processes. This issue is caused by the creation of "islands of formality", which are zones where different processes have limited interactions between them;
- *Budget protection*: due to the external dependencies issue an "home-made patch" approach, through the creation of wrappers, is preferred to avoid additional third-party costs;
- *Scope protection*: teams try to deny every other team's change requests to protect their project's scope;
- *Organisational polities*: issues that arise when the wrong or uninformed person is contacted;
- *Human resource planning* (truck number [57]): development is performed in silos and there is a high possibility of project knowledge loss due to developer turnover or delays due to developer's unavailability;
- *Undue pressure*: developers are threatened to deliver in time by managers and senior developers.

Comparing this research results to Tamburri et al. researches about Social Debt, it evident that there are many common findings and shared results, especially considering the more "human-related" socio-technical issues: "organically grown processes", "scope protection" "organisational polities" and "human resource planning". Lavallée et al. study, conducted within a large commercial company, can be considered in every aspect a Social Debt related research because the authors conclude their work stating that software quality can be negatively affected by decisions taken under certain organisational conditions and assert that "the design flaws introduced because of the organisational issues presented here will no doubt come back to haunt" [56], which it is a paraphrase of the Social Debt definition proposed by Tamburri et al. [6].

2.6 Motivational research

The first fundamental empirical software engineering research paper that can be considered as a motivational research for this master thesis was conducted by Palomba et al. and it was named "Mining Version Histories for Detecting Code Smells" [51]. During the definition and execution of our research we were inspired by Palomba et al. work and the two empirical researches can be considered similar with respect to their empirical study structure, research question approach and research context. But they had diametrically opposed targets, because Palomba et al. research addressed Technical Debt while ours is founded on Social Debt. Within their research, Palomba and his collaborators, proposed an approach called Historical Information for Smell deTection (HIST) that is capable of exploiting a project's change historical information, mined from versioning systems, to detect the presence of Code Smells through heuristics computations. The main innovation of this empirical software engineering research was the introduction of the temporal factor within Code Smell identification processes, basing such process on revision history analysis rather then the exclusively consideration of structural information of a project's source code. Paloma et al. did not limit their research to Code Smells detection but further investigated if Code Smells, considered within their study, were actually perceived by Open Source Software developers as poor design and implementation choices. Some of the research decisions and methodologies that we adopted within this master thesis can find their motivation and empirical validity through their assimilations and parallelisms with their study. The following list of characteristics constitutes the minimum common denominator of the two researches:

1. Usage of historical information to detect debt related patterns;
2. Capture developer perceptions with respect to the importance and validity of identified patterns, through the execution of a survey;
3. Operationalization of identified hazard situations;
4. Patterns identification is executed at a fine-grained source code level (method level);
5. Patterns identification is considered in different temporal instances of a project's life cycle and it not limited to the last version of the source code;
6. Research's context is focused on Open Source Software projects.

An empirical software engineering research paper that constituted a fundamental contribution to the Social Debt research area and that can be considered as another motivational study for this master thesis was conducted by Tamburri et al. and it was named “Social Debt in software engineering: insights from industry” [6]. Tamburri and his collaborators laid the theoretical foundations on which this master thesis was built, since they improved the Social Debt definition, added background to its insurgence’s conditions, highlighted possible mitigations to Social Debt and provided the precious contribution of defining several Community Smells that were proven to be capable of detecting the presence of Social Debt. The operationalisation and characterisation of Community Smells within a FLOSS software development community, that constitute a fundamental basic building block of our study, were based on Community Smells identified by their software engineering research. Tamburri et al. analysed correlations between a set of socio-organisational circumstances and the raise of additional costs in software processes within a large industrial software case study. They summarized the identified Social Debt circumstances in a framework, relating them with their causes, consequences, conditions, contexts, covariances, contingents, anti-patterns (“Community Smells”) and they suggest some possible techniques useful to avoid the insurgency of such negative circumstances. In their research Tamburri et al. were able to capture some Social Debt characteristics [6]:

- Socio-technical decisions and Social Debt are indirectly connected;
- Social Debt is an emergent property of the development community itself and, despite Technical Debt, it cannot be ascribed to any particular software artefact or operation in the development process but, at the same time, Social Debt has a strong effect on many different software artefacts;
- Social anti-patterns (i.e., Community Smells) can be considered as indicators of the emergence of Social Debt within a community;
- Social Network Analysis methodologies can be used to identify Community Smells and quantify Social Debt costs;
- Social Debt can generate Technical Debt;
- Specific socio-technical decisions (“mitigations”) can be implemented to pay back totally or partially the detected Social Debt.

It is possible to classify Community Smells within three different classes: smells related to the community structure and its related properties (e.g, community formality), smells related to the community context (e.g., political boundaries) and smells related to the community members’ interactions (e.g., socio-technical relationships). In their industrial case study, Tamburri et al., identified and classified nine different Community Smells [6]:

1. **Organisational Silo Effect**: this Community Smell occurs when it is present a too high decoupling between developers and their related development tasks. This occurrence causes low mutual awareness, low socio-technical congruence and lack of communications and cooperation in checking task dependencies within the community. An organisational silo is present in the development community whenever an isolated subgroup of loosely dependent development partners waste resources (e.g., time) or duplicate resources over the development life-cycle. Another possible side effect of this Community Smell can be the establishment of a “tunnel vision” due to the lack of cooperation and collaboration, which may imply a lack of creativity within the development team and eventually developers will make architectural decisions without the necessary authority and knowledge. A mitigation to the Organisational Silo Effect Community Smell is the institution of a “social wiki” within the development community, combining developers profiles with the artefacts they are working on and the related documentation.

2. **Black-cloud Effect**: this Community Smell occurs when two concurrent circumstances take place together: lack of people able to cover the experience or knowledge gap between two software products and the lack of official and periodic knowledge sharing opportunities (e.g., daily stand-ups). Whenever those two circumstances are verified, every knowledge exchange initiative can create a confusing communication overload (“black-cloud”) with back-and-forth emails which obfuscate reality. This Community Smells is caused by the absence of officially defined sharing protocols, lack of boundary spanners (individuals whom link internal team network to other teams) and presence of not efficient information filtering protocols. The main side effects of this Community Smell are the creation of mistrust between developers, the possibility of information obfuscation and the rise of egoistic behaviours (e.g., developers take decisions even if they do not have decisional authority). As for the Organisational Silo Effect Community Smell, the adoption of a “social wiki” can mitigate the negative effects of the Black-cloud Effect Community Smell.

3. **Prima-donnas Effect**: this Community Smell occurs whenever a team of developers is unreceptive to change its internal processes and/or characteristics, or it is unwilling to be influenced by external team members through the forms of collaborations and/or communications. This selfish and condescending team behaviour can create serious isolation problems and tensions between the community and the “prima-donnas” team, which is unable to welcome support from other development partners. Prima-donnas Effect Community Smell can raise due to stagnant collaboration within the community, due to inefficient structural innovation or due to organisational inertia. The consequences

of the lack of collaboration and communication can be worsened by organisational changes because they create fear in the prima-donnas teams and increase their egoistical behaviours. A possible mitigation for the Prima-donnas Effect Community Smell is the institution of “culture conveyors”, which allow an harmonization process of different organisational cultures through the promotion of developers coming from different communities to the role of architects. Another mitigation technique is the adoption of a “community-based contingency planning” in which managers decide to make technical and socio-technical decisions together and use the learning community as a device to generate contingency plans, if some decisions lead to undesirable outcomes. As for the Organisational Silo Effect and Black-cloud Effect Community Smells, the adoption of a “social wiki” can mitigate the negative effects of the Prima-donnas Effect Community Smell.

4. **Leftover-techie Effect**: this Community Smell is caused by an increasing isolation of the maintenance and the help-desk operations from the operative people, with a related feeling of abandonment by the technicians. The main side effect of this Community Smell are mistrust and the emergence of a sharing villainy behaviour, related to knowledge and status awareness. A mitigation capable of reducing Social Debt connected to this Community Smell is the “full-circle” and it consists in the creation of a dedicated communication line (e.g., instant-messaging) between key developers, managers and operation technicians.

5. **Sharing Villainy**: this Community Smell is caused by the absence of experience sharing initiative and high-quality knowledge exchange activities (e.g., face-to-face meetings), in addition to a shared mindset that considers knowledge interactions between developers as wasting time activities rather than positive opportunities. The main side effect of the Sharing Villainy Community Smell is the limitation to developers’ propensity to share knowledge and meaningful experiences, to the extreme of sharing outdated, wrong or unconfirmed information. A possible mitigation technique for this Community Smell, as for the Prima-donnas Effect Community Smell, is the creation of “culture conveyors”. As for the Organisational Silo Effect, Black-cloud Effect and Prima-donnas Effect Community Smells, the adoption of a “social wiki” can mitigate the negative effects of the Sharing Villainy Community Smell.

6. **Organisational Skirmish**: this Community Smells occurs whenever operations and development units are misaligned in their organisational culture (e.g., organisational layout and properties), in their communication habits and in their expertise levels. These misalignments cause severe managerial issues and delays.

7. **Architecture Hood Effect**: this Community Smell occurs whenever decision-makers are not well integrated and geographically distant from other developers and operators of the community and their decisions are taken using a software architects board that makes decisions' responsibility and logic not easily discernible. The side effects of this Community Smell are the inability to identify decision-maker responsibilities and the unwillingness of developers to accept decisions with a related uncooperative behaviour within the development community. Architecture-hood Community Smell can be mitigated by the socio-technical decision of adopting "stand-up voting" in an anonymous form to accept decisions at the end of fixed daily stand-ups.

8. **Solution Defiance**: this Community Smell occurs when the development community divides itself into overly similar subgroups with different levels of cultural and experience backgrounds; those homophile subgroups divide themselves into smaller factions with opposite and conflicting opinions toward some socio-technical decisions that should be taken. The side effects of this Community Smell are delays, uncooperative behaviours, decisions ignoring and "organisational rebellion" due to the unwillingness of developers to take a shared decision within different factions until the very last possible moment. A socio-technical decision to mitigate the effect of solution defiance, as for the Prima-donnas Effect Community Smell, is the adoption of a "community-based contingency planning". As for the Organisational Silo Effect, Black-cloud Effect, Prima-donnas Effect and Sharing Villainy Community Smells, the adoption of a "social wiki" can mitigate the negative effects of the Solution Defiance Community Smell.

9. **Radio Silence**: this Community Smell occurs when the organisational structure is highly formal, complex and constituted by regular procedures which cause changes to be delayed and people time to be wasted due to required formal actions and filters hiding necessary information. The main side effect of this Community Smell is the massive delay in decision making processes due to people unavailability or due to further information needs. A mitigation able to reduce almost completely the negative effects of the Radio Silence Community Smell is the creation of a "learning community" that involves all the developers and operators. This socio-technical decision can reduce delays in a direct way with the creation of strong organisational and social relationships between developers and in an indirect way, enabling a passive knowledge sharing channel.

Chapter 3

Problem analysis

This chapter summarises the fundamental aspects of this master thesis and it provides an overview of how our empirical software engineering research was defined, characterised and executed.

Section 3.1 presents the definitions of a set of key terminologies that should be kept in mind while reading this work. The motivations at the root of this master thesis and the research questions that were addressed within the execution of this empirical software engineering study are described in Section 3.2. The original contributions constituting the basic building blocks of this master thesis are presented in Section 3.3, together with their characteristics, usefulness and purposes. Finally, Section 3.4 explores the context of the dataset considered within this research, motivating and providing informations about the set of Open Source Software development communities considered in the analysis.

3.1 Definitions

This section provides definitions of several important and fundamental terminologies used within this master thesis.

- **Factor**: element, circumstance or influence which contribute to produce a result;
- **Socio-technical factors**: elements, circumstances or influences which contribute to produce a result that has both social and technological aspects;
- **Quality factors**: elements, circumstances or influences which contribute to increase or decrease the well-being and quality of a software development community;
- **Social Debt**: Unforeseen project costs connected to a “suboptimal” development community [3];

- **Community Smells**: organisational and social patterns that produce negative effects on the long run and lead eventually to Social Debt [6]. Therefore, Community Smells represent a risk correlated to the potential presence of Social Debt;

- **Identification pattern**: model capable of identifying the occurrence of a specific phenomenon using Social Network methodologies;

- **Developer Social Network (DSN)**: social network model of a software development community, where developers are represented by the means of nodes and links between nodes represent developer relationships;

- **Communication DSN**: particular case of Developer Social Network, in which nodes represent community members who communicate on the project's development mailing list and two nodes are connected through a link if and only if, one of the two members replied to a previously sent e-mail of the other. The weight of each link is the total number of exchanged e-mails between the two considered community members, within a defined time window. Communication DSN represents the organisational structure of a software development community and its coordination activities;

- **Collaboration DSN**: particular case of Developer Social Network, in which nodes represent developers who contributed to the project's source code development (extracted from the project's VCS) and two nodes are connected through a link if and only if, the two developers collaborated at least once during software development. Two developers are considered collaborating if they modify similar portions of the project's source code. The weight of each link is the total number of collaborations between the two considered developers, within a defined time window. Collaboration DSN represents the technical structure of the project, its task dependencies and its communication needs;

- **Global DSN**: particular case of Developer Social Network, which is the fusion of communication and collaboration DSNs. Therefore, a node is the representation of a community member who participates to the mailing list and/or to the development activity and two community members are connected through a link if and only if, they exchanged at least one e-mail and/or collaborated at least once. The weight of each link is the sum of total number of collaborations and the total number of e-mails exchanged between the two considered community members, within a defined time window;

- **Core community member**: community member who exhibit a high global centrality within the considered DSN.

3.2 Research questions

The main goals of this master thesis are the *understanding of which quality factors of a software development community can influence to the emergence of Community Smells* and the *identification of meaningful quality thresholds* of such factors, useful to keep Community Smells under control. To pursue these goals it was of fundamental importance the identification and quantification of Community Smells with respect to their influence from the point of view of the Social Debit in the context of Open Source Software development communities.

In order to achieve our prefixed goals we performed an empirical *software engineering research*, aimed to discover and quantify Community Smells within development community, and a *survey*, necessary to understand if developer perceptions can act as indicators of the presence of Community Smells. Furthermore, executed survey had the purpose of accessing developer perceptions inherent to Social Debt related issues.

This master thesis aims at addressing the following three research questions:

- **RQ1.** *Are there quality factors that can influence the emergence of Community Smells?*

  Considering the literature, the following sub-questions were formulated:

  - **RQ1a.** *Do sponsored developers decrease Community Smells?*
  - **RQ1b.** *Do temporal and geographic dispersion increase Community Smells?*
  - **RQ1c.** *Do high socio-technical congruence decrease Community Smells?*
  - **RQ1d.** *Do high communicability decrease Community Smells?*
  - **RQ1e.** *Do high modularity decrease Community Smells?*
  - **RQ1f.** *Do low turnover decrease Community Smells?*

- **RQ2.** *If such quality factors exist, is it possible to identify quality thresholds?*

- **RQ3.** *Do developer perceptions indicate the presence of Community Smells?*

3.3 Contributions

This section briefly introduces the four main basic building blocks on which this master thesis is founded. Each of them can be considered a precious *original contribution* to the Social Debt and socio-technical research fields as they provide additional information, methodologies and tools.

Figure 3.1 provides a graphical representation of the four main contributions and specifies their relationships through a work-flow in order to describe how proposed components interacted during the execution of this study.

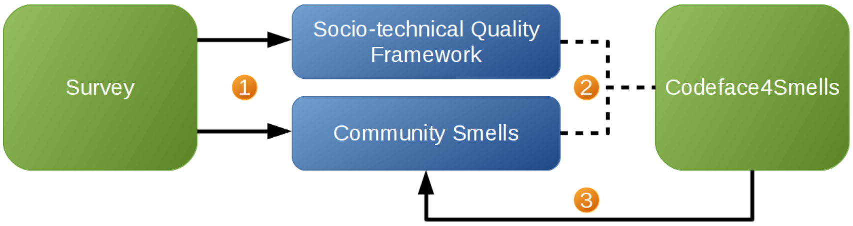


Figure 3.1: Work-flow of contributions

The four fundamental contributions of this master thesis are:

- **Survey.** The survey was aimed to identify important quality factors that developers perceive as important within a development community, confirm some assumptions made within this study, collect developer perceptions to achieve a deeper understanding of Social Debt and Community Smells and, moreover, verify if developer perceptions can act as indicators of the presence of Community Smells;

- **Socio-technical Quality Framework.** This framework is constituted by 40 quality factors, identified from the literature and from the results of the executed survey, with the purpose of capturing every possible socio-technical characteristic of a software development community to understand if the insurgence of Community Smells can be correlated to some specific quality factors;

- **Identification patterns of Community Smells.** Starting from the literature related to Community Smells, we defined identification patterns of five Community Smells that, if operationalised, are able to identify and quantify Community Smells within a software development community;

- **Codeface4Smells.** This is the actual tool that implements our Socio-technical Quality Framework and operationalise the proposed identification patterns of Community Smells. Furthermore, for every analysed software development community, it executes a correlation analysis between all quality factors and Community Smells and detects if and to what extend the quality factors influence the number of identified Community Smells within a community.

As suggested from the numbers reported in Figure 3.1, it is possible to identify the following work-flow between the four basic building blocks:

1. The survey results had a fundamental role in the definition and individuation of quality factors considered important by developers and, thus, that needed to be included in our Socio-technical Quality Framework. Moreover, obtained responses allowed us to discover further notions and characteristics of Social Debt and Community Smells within FLOSS development communities;

2. The proposed Socio-technical Quality Framework and the identification patterns of Community Smells were implemented together within the tool named Codeface4Smells;

3. The results of Codeface4Smells analysis allowed us to identify which quality factors were able to influence the occurrence of Community Smells and, in some particular cases, identify quality thresholds. Furthermore, the results of Codeface4Smells analysis of the three reference projects, in conjunction with the survey results, allowed us to identify which developer perceptions can be potentially used as indicators of the presence of Community Smells.

3.4 Dataset selection

The dataset considered in this master thesis consisted of 60 Open Source Software projects. The complete list of analysed projects can be consulted in Table 3.1, where for every considered project it is showed its name, its code repository address and the development mailing list considered as its primary development communication channel. As it is possible to deduce from the list of analysed projects, we considered FLOSS development communities of different dimensions, popularity, development habits, openness and application contexts.

Our choice of FLOSS development communities was not random but it was guided by specific and rigid requirements, dictated by the infrastructure of Codeface4Smells or by analysis requirements. More specifically, it was mandatory that every analysed project was characterised by the following list of requirements:

1. Source code was available on-line through git or SVN repositories;

2. It was possible to identify an «active» development mailing list and its archive was present on `www.gmane.com`;

3. Within every analysed window it was sent at least one e-mail to the considered development mailing list;

4. Within every analysed window it was committed at least one source code contribution;

5. Codeface's collaboration and communication analysis terminated without errors.

The list of analysed projects were partially retrieved from datasets used in previous empirical software engineering researches [58, 59], in which diversity was verified with respect to several factors. We further verified the diversity of our dataset defining three dimensional categories characterised by an equal number of software

#	Project	Source code repository	Development mailing list (Gmane)
1	LibreOffice	http://anongit.freedesktop.org/git/libreoffice/core.git	gmane.comp.documentfoundation.libreoffice.devel
2	Firefox	https://github.com/mozilla/gecko-dev.git	gmane.comp.mozilla.firefox.devel
3	FFmpeg	git://source.ffmpeg.org/ffmpeg.git	gmane.comp.video.ffmpeg.devel
4	Cassandra	http://git-wip-us.apache.org/repos/asf/cassandra.git	gmane.comp.db.cassandra.devel
5	Git	https://github.com/git/git.git	gmane.comp.version-control.git
6	OpenSSL	git://git.openssl.org/openssl.git	gmane.comp.encryption.openssl.devel
7	GRUB	git://git.savannah.gnu.org/grub.git	gmane.comp.boot-loaders.grub.devel
8	nginx	https://github.com/nginx/nginx.git	gmane.comp.web.nginx.devel
9	Audacity	https://github.com/audacity/audacity.git	gmane.comp.audio.audacity.devel
10	VLC	git://git.videolan.org/vlc.git	gmane.comp.video.videolan.vlc.devel
11	Tomcat	git://git.apache.org/tomcat.git	gmane.comp.jakarta.tomcat.devel
12	GIMP	git://git.gnome.org/gimp	gmane.comp.video.gimp.devel
13	Guix	git://git.savannah.gnu.org/guix/dhcp.git	gmane.comp.gnu.guix.devel
14	Mahout	git://git.apache.org/mahout.git	gmane.comp.apache.mahout.devel
15	CXF	git://git.apache.org/cxf.git	gmane.comp.apache.cxf.devel
16	Rails	https://github.com/rails/rails.git	gmane.comp.lang.ruby.rails.core
17	AngularJS	https://github.com/angular/angular.js.git	gmane.comp.lang.javascript.angularjs
18	libuv	https://github.com/libuv/libuv.git	gmane.comp.lang.javascript.nodejs.libuv
19	Bitcoin	https://github.com/bitcoin/bitcoin	gmane.comp.bitcoin.devel
20	Scala	https://github.com/scala/scala	gmane.comp.lang.scala
21	matplotlib	https://github.com/matplotlib/matplotlib	gmane.comp.python.matplotlib.devel
22	Qt	git://code.qt.io/qt/qtbase.git	gmane.comp.lib.qt.devel
23	NodeJS	https://github.com/nodejs/node.git	gmane.comp.lang.javascript.nodejs
24	GitLab	https://github.com/gitlabhq/gitlabhq.git	gmane.comp.version-control.gitlab
25	Tornado	https://github.com/tornadoweb/tornado.git	gmane.comp.python.tornado
26	Arduino	https://github.com/arduino/Arduino.git	gmane.comp.hardware.arduino.devel
27	ipython	https://github.com/ipython/ipython	gmane.comp.python.ipython.devel
28	Lucene	git://git.apache.org/lucene-solr.git	gmane.comp.jakarta.lucene.devel
29	Capistrano	https://github.com/capistrano/capistrano	gmane.comp.lang.ruby.capistrano.general
30	Django	https://github.com/django/django.git	gmane.comp.python.django.devel
31	Salt	https://github.com/saltstack/salt.git	gmane.comp.sysutils.salt.user
32	mongoose	https://github.com/Automattic/mongoose.git	gmane.comp.lang.javascript.mongoose
33	APR	git://git.apache.org/apr.git	gmane.comp.apache.apr.devel
34	Jackrabbit	git://git.apache.org/jackrabbit.git	gmane.comp.apache.jackrabbit.devel
35	Gnome-shell	git://git.gnome.org/gnome-shell	gmane.comp.gnome.shell
36	Krita	git://anongit.kde.org/krita.git	gmane.comp.kde.devel.krita
37	Blender	https://git.blender.org/blender.git	gmane.comp.video.blender.devel
38	Vagrant	https://github.com/mitchellh/vagrant.git	gmane.comp.tools.vagrant
39	NetworkManager	git://anongit.freedesktop.org/NetworkManager/NetworkManager.git	gmane.linux.network.networkmanager.devel
40	Eclipse CDT	https://git.eclipse.org/r/cdt/org.eclipse.cdt	gmane.comp.ide.eclipse.cdt.devel
41	Enlightenment	https://git.enlightenment.org/core/enlightenment.git	gmane.comp.window-managers.enlightenment.devel
42	libva	git://anongit.freedesktop.org/git/libva	gmane.comp.freedesktop.libva
43	JDO	http://svn.apache.org/repos/asf/db/jdo	gmane.comp.apache.db.jdo.devel
44	Jena	git://git.apache.org/jena.git	gmane.comp.apache.jena.devel
45	OpenNLP	git://git.apache.org/opennlp.git	gmane.comp.apache.opennlp.devel
46	Cayenne	git://git.apache.org/cayenne.git	gmane.comp.java.cayenne.devel
47	Pig	git://git.apache.org/pig.git	gmane.comp.java.hadoop.pig.devel
48	Calligra	git://anongit.kde.org/calligra.git	gmane.comp.kde.devel.calligra
49	Wine	git://source.winehq.org/git/wine.git	gmane.comp.emulators.wine.devel
50	Mallet	https://github.com/mimno/Mallet.git	gmane.comp.ai.mallet.devel
51	Gstreamer	git://anongit.freedesktop.org/gstreamer/gstreamer	gmane.comp.video.gstreamer.devel
52	U-boot	http://git.denx.de/u-boot.git	gmane.comp.boot-loaders.u-boot
53	LLVM	http://llvm.org/git/llvm	gmane.comp.compilers.llvm.devel
54	gPhoto	svn://svn.code.sf.net/p/gphoto/code/trunk	gmane.comp.multimedia.gphoto.devel
55	Emacs	git://git.savannah.gnu.org/emacs.git	gmane.emacs.devel
56	QEMU	git://git.qemu.org/qemu.git	gmane.comp.emulators.qemu
57	Python	https://github.com/python/cpython.git	gmane.comp.python.devel
58	Mesa	git://anongit.freedesktop.org/mesa/mesa	gmane.comp.video.mesa3d.devel
59	Sympy	git://github.com/sympy/sympy.git	gmane.comp.python.sympy
60	Okular	git://anongit.kde.org/okular	gmane.comp.kde.devel.okular

Table 3.1: List of analysed projects

development communities with respect to the mean number of trimestral community members present in the global DSN. In conclusion, the 60 considered software development communities were proven to be diversified with respect to:

- *code-base size*: medium, large (500-850) and very large (850>KLOC);
- *main programming language*: Java, C#, C, Python, YAML and more;
- *community size*: 20 medium projects (<50 members), 20 large projects (50>150 members) and 20 very-large projects (150>);
- *age*: young (<24 months), established (24>32 months) and popular (32> months).

Metric	Mean	Min	Max	St.deviation
Global DSN members	143.78	4	949	152.45
Time-zones	6.61	1	36	10.26
Core global DSN members	45.53	1	311	44.1
Core communication DSN members	39.27	1	285	37.42
Core collaboration DSN members	14.73	0	213	24.3
Sponsored developers	13.35	0	104	17.82
Sponsored core developers	1.36	0	21	2.43

Table 3.2: Trimestral variability of analysed projects

Table 3.2 reports an overview of the most important metrics which characterised the analyzed Open Source Software development communities within Codeface4Smells analysis and represents their respective trimestral mean, minimum, maximum and standard deviation values.

Every FLOSS project was analysed executing Codeface4Smells analysis using three-month analysis windows for the last three years, therefore every project's analysis was constituted by a total of 12 ranges. We used this temporal analysis window because it was previously used in another empirical software engineering research based on the usage of Codeface [60] and because it was demonstrated that a software development community does not change significantly after a consideration window of three months [37]. The total time lapse of three years was setted in order to achieve a relevant number of analysed ranges capable of capturing trends and correlations. Another motivation to the decision of limiting the analysis to three years is that before that temporal limit, the standard within FLOSS development was Centralized VCS, as explained in Section 2.4, and thus information about the author of the software contributions were not accessible.

Chapter 4

Identification patterns of Community Smells

This chapter presents one fundamental contribution of this master thesis, which is the specification of *identification patterns of five Community Smells* considered in the scope of this study, namely: Organisational Silo Effect, Missing Links, Black-cloud Effect, Prima-donnas Effect and Radio Silence.

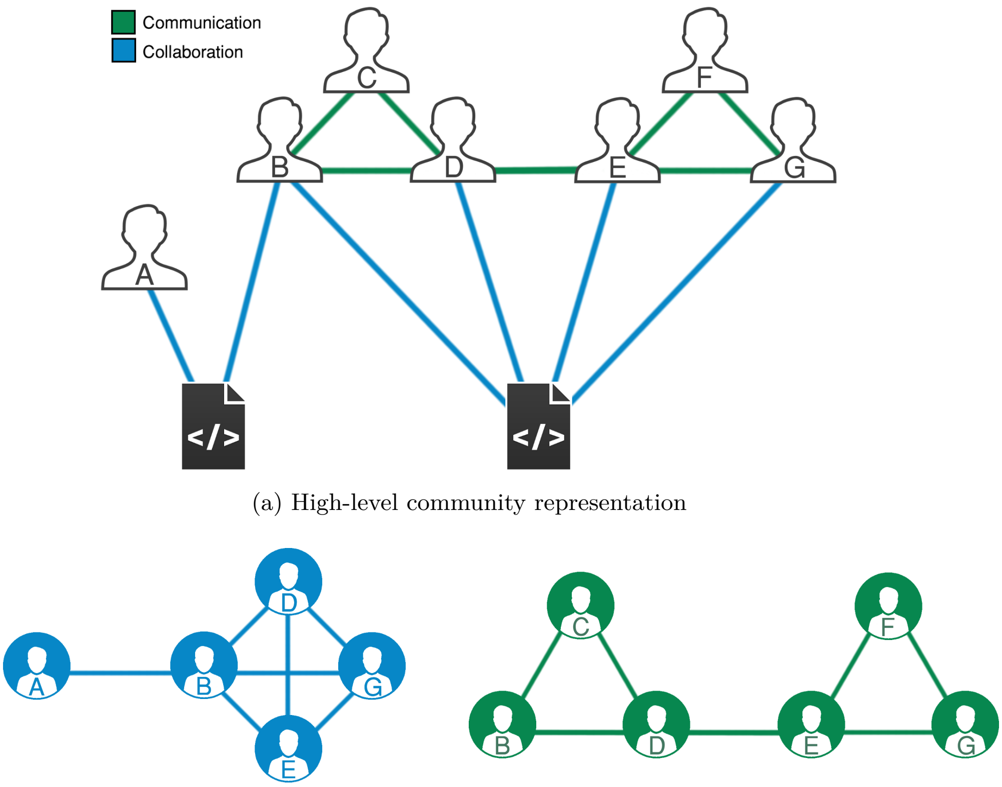
We considered the set of Community Smells identified within an industrial case study, conducted by Tamburri et al. [6], as a starting point and then we proposed identification patterns, capable of capturing possible social related problematic situations, for five different Community Smells. Proposed *identification patterns can be operationalised in order to provide an automated identification, quantification and recovery of Community Smells*, within a software development community, by means of tool-supported processes or software-repository mining techniques.

Every identification pattern of a Community Smell proposed within this chapter is founded on the concept of *Developer Social Networks*, in order to retrieve information about communication and collaboration relationships within a software development community. Therefore, the identification process of every Community Smells is enabled by the availability of the communication and collaboration Developer Social Networks of a development community.

While the identification patterns of four Community Smells base their discovery logic solely on the state of a development community in a particular point in time, one identification pattern founds its detection capabilities on the additional consideration of historical information, i.e., it considers more than one temporal window.

In this master thesis, as further explained in Chapter 7, the communication DSN of a project was build mining communication information from its mailing list archives, while the collaboration DSN of a project is was build mining its Version Control System. Thus, in order to operationalise the detection and quantification of

Community Smells, based on the exploitation of identification patterns presented in this chapter, it is necessary to have access to the VCS and mailing list archives of a software development community, or to any other typology of development artefact capable of providing information related to organisational structure and technical dependencies of a software project.



(a) High-level community representation

(b) Collaboration DSN

(c) Communication DSN

Figure 4.1: Example of software development community

An explicative example of a software development community is represented in Figure 4.1a, where it is possible to notice the presence of eight community members, identified by different letters, who are connected through green links if they communicated within the mailing list and community members are connected through blue links to software components on which they worked (represented by a file icon). Given such high-level representation of a software development community, it is possible to generate the communication DSN, representing the social and organisational structure of a project, and the collaboration DSN,representing technical relationships between developers working on similar portions of a project's source code. While the communication DSN of the proposed example, represented in Figure 4.1c, can be easily generated considering exclusively communication links of the high-level representation of the development community, the generation of the collaboration DSN, represented in Figure 4.1b, requires additional reasonings because two developers are linked if and only if they worked on a similar portion of source code. Thus, since the file icon in the high-level representation of the development community identifies a similar portion of source code, all developers linked to the same source code portion have to be connected in the collaboration DSN.

Within the proposed example it is possible to consider the communication DSN constituted by two different sub-communities: developers identified by the letters B, C and D and developers identified by the letters E, F and G. It is important to highlight that a Developer Social Network is constituted only by *active* community members, with respect to the considered development aspect. Therefore, it is possible to notice that, within the previously exposed example, the collaboration DSN do not contain community members identified by the letters C and F because they are not working on any source code portion and the communication DSN do not contain the community member identified by the letter A because he or she is not communicating with anyone within the development community.

The following sections address all the Community Smells considered in the scope of this master thesis. For each Community Smell it is provided an identification pattern and a synthetic example explaining the behaviour of the identification pattern. Furthermore, each section presents a detailed explanation of the execution of the considered identification pattern within the development community presented in Figure 4.1a and a visual representation of identified occurrences of Community Smells within such example.

4.1 Organisational Silo Effect and Missing Links

Tamburri et al. [6] defined Organisational Silo Effect Community Smell as the occurrence of a too high level of decoupling between the organisational structure and development activities. Considering its definition and the literature based on Conway's law, it was possible to assimilate this Community Smell to an *imperfection in the mirroring relationship between the organisational structure and the technical structure*, constituted by collaborations of software developers.

An Organisational Silo is characterised by an independentist sub-community that is loosely dependent to other development partners and that duplicates or wastes resources over the development life cycle, due to its isolation from the rest of the software development sub-communities.

This Community Smell is associated to the main side effect of decreasing communication activities between community members belonging to different sub-communities; thus, it negatively impacts on mutual awareness and it implies a degradation of the socio-technical congruence.Another side effect that can characterise this Community Smell is the rise of a "tunnel vision" within the software development community, since members belonging to the Organisational Silo tend to limit their cooperation and communication activities with members who do not belong to their sub-community. Moreover, community members belonging to an Organisational Silo can exhibit egoistic and superior behaviours, giving rise to autonomous architectural decisions taken without the necessary authority or knowledge. A possible mitigation to this Community Smell can be the implementation of a "social wiki", that combines documentation, developer profiles and artefacts [6].

While we were pursuing our goal of identifying a possible automatic identification pattern of this Community Smell, mining commonly available software development artefacts, we focused of the most important Organisational Silo Effect side effects: decrease of communications within the community and generation of a "tunnel vision". Therefore, we proposed two different identification patterns in order to provide the ability of identifying both typologies of side effects:

1. **Organisational Silo Effect**

   The identification pattern of Organisational Silo Effect Community Smell is based on the detection of *community members who collaborate with other members but who do not communicate* within the analysed communication channel, i.e., mailing list. An explanatory example is represented in Figure 4.2, where the occurrence of this Community Smell is due to the developer identified by the number 1 because he or she does not have outgoing communication edges even if there is a collaboration with the developer identified by the number 2.

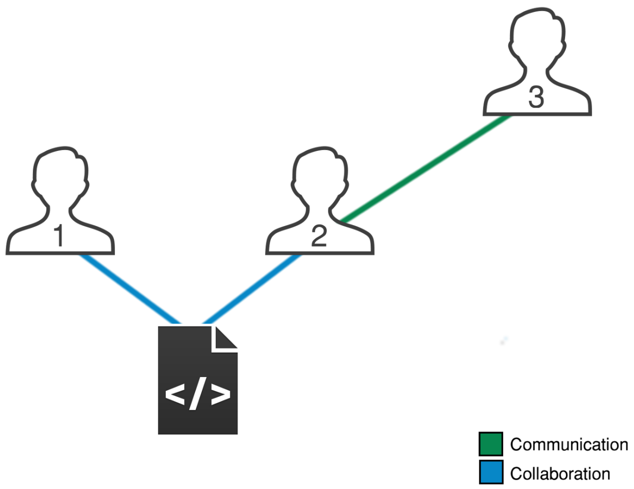


Figure 4.2: Identification pattern of Organisational Silo Effect

   This identification pattern is capable of identifying the presence of actual sub-communities present within the software development community that generate a "tunnel vision" because they do not communicate with community members external to their sub-community. Pushing to its extreme the concept of sub-community, thus considering two developers at a time, an occurrence of *Organisational Silo Effect is detected whenever one of them shows uncooperative behaviours, not participating in the mailing list of the development community.*

In order to operationalise the proposed identification pattern, the collaboration and communication Developer Social Networks, generated respectively mining the VCS and mailing list archives of a development community, should be considered at the same time and each collaboration present between two different community members contributing to adjacent parts of the source code should be considered as an atomic cooperation unit to identify possible Organisational Silo. Since we considered the absence of one developer belonging to an atomic cooperation unit as the trigger of the occurrence of Organisational Silo Effect Community Smell, it is necessary to consider every software development cooperation present within the collaboration Developer Social Network and check if the two community members are present or not in the communication Developer Social Network. We quantified the number of occurrences of Organisational Silo Effect as the number of single collaborations between different developers in which at least one of them do not participate in the communication channel. Therefore, we focused on collaborations and not on developers, in order to capture the repercussions of the absence of developers in communication activities. This choice was made because the absence of a highly active and highly connected contributor is surely more relevant than the absence of an occasional contributor.

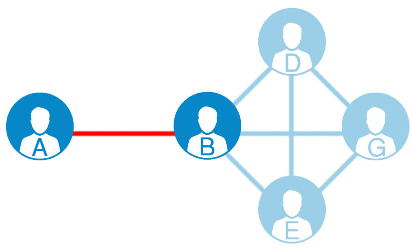


Figure 4.3: Example of occurrences of Organisational Silo Effect

Considering the example of development community proposed in Figure 4.1a, in order to execute the proposed identification pattern of this Community Smell it is possible to compare directly the collaboration DSN (Figure 4.1b) with its communication counterpart (Figure 4.1c) and verify that the developer identified by the letter A is present in the collaboration DSN but he or she is not present in the communication Developer Social Network. Since we focused on collaboration level and not on software developer level, we have to consider every collaboration link between two different community members of the collaboration DSN. The absence of the developer identified by the letter A is counted only one time since he or she is characterised by a unique outgoing link in the collaboration DSN. Therefore, within the proposed example there is just one occurrence of Organisational Silo Effect Community Smell and a graphical representation is presented in Figure 4.3, where the only occurrence of this Community Smell is highlighted in red and all the developers belonging to the collaboration DSN who are not involved in the insurgence of this Community Smell are faded out.

The operationalisation of this identification pattern should provide a correct quantification of Organisational Silo Effect in cases where both the collaborating developers do not participate within a project's communication channel. Moreover, since we focused on collaborations if in the proposed example, the collaboration Developer Social Network was characterised by an additional developer who was isolated from other developers without having any identified collaboration and he or she was not present in the communication DSN, this additional community member would not have any repercussion on the Organisational Silo Effect detection.

2. **Missing Links**

   The identification pattern of Missing Links Community Smell is based on the detection of *development collaborations between two community members that do not have communication counterparts*. An explanatory example is represented in Figure 4.4, where the occurrence of Missing Links Community Smell is due to the fact that developers identified by the number 1 and 2 are collaborating on a similar portion of source code, but they are not directly connected through a communication link.

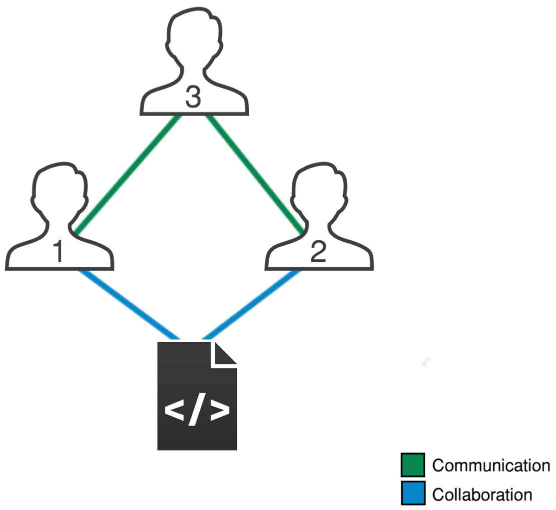


Figure 4.4: Identification pattern of Missing Links

This identification pattern has the purpose of detecting side effects as the decreasing of communication activities, which implies a negative influence on mutual awareness, and the inherent socio-technical congruence degradation caused by the presence of uncooperative Organisational Silo. Thus, while the Organisational Silo Effect identification pattern is focused on the detection of the "tunnel vision" side effect, this identification pattern is supposed to capture all the other typologies of associated side effects. Pushing to its extreme the concept of sub-community, thus considering two developers at a time, an insurgence of Missing Links is detected whenever a couple of co-committing (collaborating) developers exhibit uncooperative behaviours, not communicating in the mailing list of the development community.

Considering the given definition of this identification pattern, it is obvious that Missing Links is a more general case and actually incorporates in itself the identification pattern of Organisational Silo Effect but we considered necessary to define and consider both typologies of identification patterns because they provide different detail levels of information and they characterise and analyse different aspects associated to Community Smells.

In order to operationalise the proposed identification pattern, the collaboration and communication Developer Social Networks, generated respectively mining the VCS and mailing list archives of a development community, should be considered at the same time and each collaboration present between two different community members contributing to adjacent parts of the source code should be considered as an atomic cooperation unit. Since we considered the absence of a communication counterpart of a collaboration as the trigger of the occurrence of Missing Link Community Smell, it is necessary to consider every software development cooperation present within the collaboration Developer Social Network and check if the two community members are connected in the communication Developer Social Network.

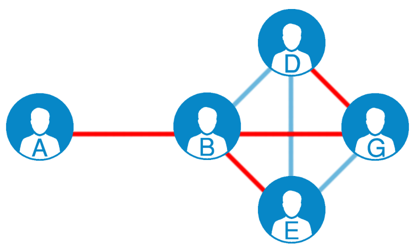


Figure 4.5: Example of occurrences of Missing Links

Considering the example of development community proposed in Figure 4.1a, in order to execute the proposed identification pattern of this Community Smell it is possible to compare directly the collaboration DSN (Figure 4.1b) with its communication counterpart (Figure 4.1c) and verify that four links within the collaboration DSN do not have corresponding links between the same community members in the collaboration DSN. Detected occurrences of Missing Links are identified in Figure 4.5 with red links and are constituted by the collaborations between developers identified by the following couple of letters: A-B, B-E, B-G and G-D. Using this example it is possible to verify that Organisational Silo Effect is a subset of Missing Links, since the missing communication link associated to the collaboration between developers identified by the letters A and B is actually an Organisational Silo Effect.

In conclusion, since the main purpose of Missing Links identification pattern is to identify and count the number of unsatisfied communication needs due technical collaborations, thus liaising it to socio-technical literature related to Conway's law, *it is possible to consider Missing Links as the opposite measurement of socio-technical congruence.* While socio-technical congruence is supposed to quantify the accordance of technical and organisational structures, Missing Links quantifies the discordance of such structures and counts the number of the technical relationship that are not mirrored within the organisational structure.

4.2 Black-cloud Effect

The identification pattern of Black-cloud Effect Community Smell is based on the detection of *isolated sub-communities that, in different and subsequent time periods, do not communicate with the exception of one communication link.* An explanatory example is represented in Figure 4.6, where the occurrence of this Community Smell is due to the fact that developers identified by the number 3 and 4 are the only two community members who communicate between the two sub-communities, over time.

Black-cloud Effect Community Smell was initially defined by Tamburri et al. [6] as a social pattern, within a software development community, characterised by the occurrence of two concurrent circumstances:

1. inability of community members to cover knowledge and experience gaps between two different software products developed within the same software community;

2. lack of periodic and official opportunities to share and exchange knowledge between all the community members.

Whenever these two different circumstances occur together, they can generate a "black cloud" within the development community, constituted by confusing and

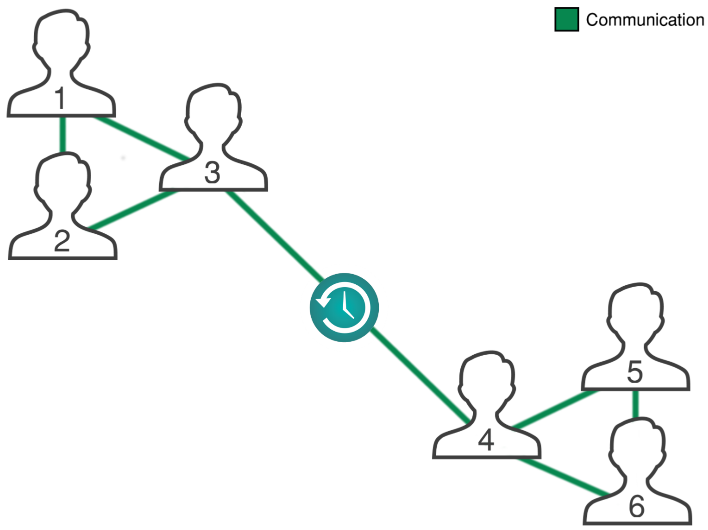


Figure 4.6: Identification pattern of Black-cloud Effect

unnecessary communications that generate a communication overload, which obfuscate the project's reality and its global and shared vision. The occurrence of this Community Smell can be generated or worsened by the following socio-technical triggers:

- absence of official protocols dedicated to knowledge sharing;
- lack of people with the objective of linking the work of their team with the one of other teams (boundary spanners);
- presence of inefficient communication filtering protocols.

We speculated that the lack of official knowledge sharing opportunities, which constitute one of the two characterising factor of this Community Smell, implies the analysis of formal communication activities, and thus the development mailing list archives of a software development community. On the other side, the inability to cover knowledge and experience gaps between two different software products developed within the same software community suggested a temporal analysis rather then exclusively consideration a snapshot of a development community in time. We used these considerations as a starting point and we defined a methodology to identify possible Black-cloud Effect present within a software development community mining communication relationships between community members, extracted from the communication Developer Social Network, and considering them along different time windows in order to achieve a temporal analysis.

We defined as a "potential" Black-cloud Effect implies that one of the two sub-community is isolated from the rest of the communication DSN external to its boundaries with the exception of a unique communication link from one of its constituting members toward another community member belonging to a different sub-community. The members who constituted the unique communication link acts as a unique knowledge broker (*boundary spanner*) toward the other sub-community. A "potential" Black-cloud Effect can be considered as an actual Black-cloud Effect Community Smell if and only if the same exact "potential" Black-cloud Effect is perpetrated in the following time window of analysis.

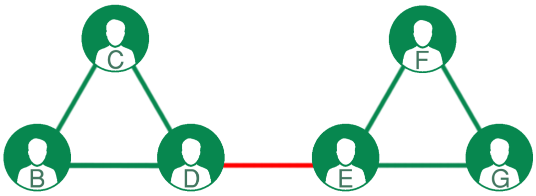


Figure 4.7: Example of occurrences of Black-cloud Effect

After the generation of the project's communication Developer Social Network it is necessary to classify all community members into clusters, with respect to their communication habits, in order to be able to identify different sub-communities present within the software development communication community. Considering the example of development community proposed in Figure 4.1a, the two sub-communities present within the development community can be easily visualised considering the density of communication links. Analysing the communication DSN (Figure 4.1c), it is evident that the communication link between community members identified by the letters D and E constitutes a unique connection between the two sub-communities and thus community members identified by the letters D and E act as unique knowledge brokers toward the other sub-community. The detected occurrence is represented in Figure 4.7 with a red link, that represents a "potential" Black-cloud Effect and it can be considered as an effective Black-cloud Effect if and only if the same "potential" Black-cloud Effect will be iterated again in the next analysed time window.

4.3 Prima-donnas Effect

The identification pattern of Prima-donnas Effect Community Smell is based on the *detection of isolated sub-communities that cooperate on similar parts of the source code but do not communicate with the exception of one communication link.* An explanatory example is represented in Figure 4.8, where the occurrence of this Community Smell is due to the fact that developers identified by the number 2, 3, 4 and 6 are collaborating on the same portion of source code but the community members identified by the numbers 3 and 4 are the only two community members

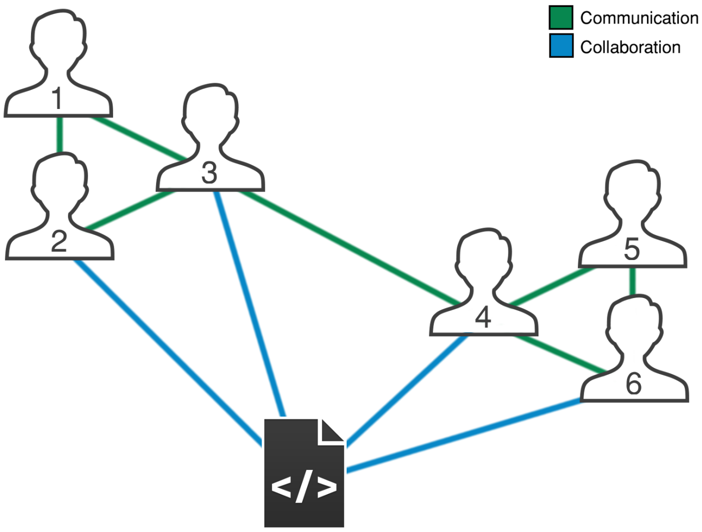


Figure 4.8: Identification pattern of Prima-donnas Effect

who communicate between the two sub-communities.

Prima-donnas Effect Community Smell was initially defined by Tamburri et al. [6] and it is characterised by the presence of a sub-community of developers which is unreceptive with respect to different influences coming from outside it boundaries, unwilling to communicate with external community members, unwilling to welcome support from other members and resistant to imposed organisational or processes changes.

In order to operationalise the proposed identification pattern, the collaboration and communication Developer Social Networks, generated respectively mining the VCS and mailing list archives of a development community, should be considered at the same time and it is also necessary to classify all community members into clusters, with respect to their communication habits, in order to be able to identify different sub-communities present within the software development communication community. The proposed of Prima-donnas Effect Community Smell can be subdivided into two identification steps:

1. **Step one:** identification of sub-communities that communicate with each-other exclusively through a unique communication link, thus it is the same concept the concept of "*potential" Black-cloud Effect Community Smell;*

2. **Step two:** Once potential problematic sub-communities with isolationist behaviours are identified ("potential" Black-clouds) in the previous step, it is necessary to consider the collaboration Developer Social Network and compute the collaboration level of developers belonging to every couple of identified sub-communities. If computed collaboration level is over a given threshold, then a Prima-donnas Effect Community Smell is detected. The threshold used

to decide if two isolationist sub-communities are collaborating, thus if they should be considered as Prima-donnas, is not universally definable because it may depend on the software development context, characteristics and research preferences.

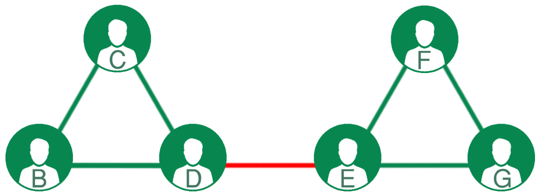


Figure 4.9: Example of occurrences of Prima-donnas Effect

Considering the example of development community proposed in Figure 4.1a, in order to execute the proposed identification pattern of this Community Smell it is necessary consider every sub-community present within the communication DSN (Figure 4.1c) and it is evident that the communication link between community members identified by the letters D and E constitutes a unique connection between the two sub-communities and thus community members identified by the letters D and E act as unique knowledge brokers toward the other sub-community. Then, it is necessary to consider the collaboration relationships between members of every couple of identified isolationist sub-communities within the collaboration Developer Social Network (Figure 4.1b) and it is evident that community members belonging to the two sub-communities are highly connected (full mesh network), so a Prima-donnas Effect Community Smell is detected. The detected occurrence is represented in Figure 4.9 with a red link, that represents the unique link connecting the two Prima-donnas.

4.4 Radio Silence

The identification pattern of Radio Silence Community Smell is based on the *detection of unique knowledge and information brokers toward different sub-communities*. An explanatory example is represented in Figure 4.10, where the occurrence of this Community Smell is due to the fact that the community member identified by the number 3 is the only member of its sub-community who communicates with the other sub-community.

Radio Silence Community Smell was initially defined by Tamburri et al. [6] and it was identified by the occurrence of the following negative characteristics within a software development community: proposed changes require an extraordinary quan-

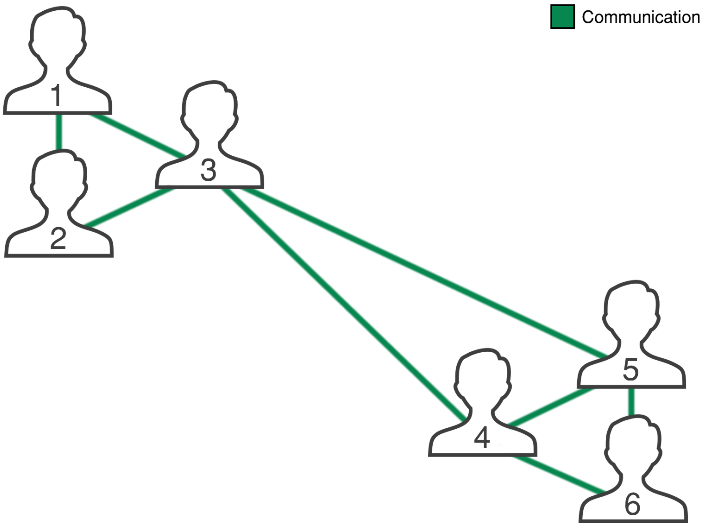


Figure 4.10: Identification pattern of Radio Silence

tity of time to be implemented, time wasting, hidden information, highly formal organisational structure, complex organisational structure, and highly regular procedures. The fundamental side effect generated by this Community Smell is a massive delay that characterises decision making processes within a software development community, due to people unavailability or to the communication overload created to cope with further generated information needs.

The presence of a unique boundary spanner toward another sub-community, i.e., a unique knowledge and information broker, implies that every information and knowledge exchange between the two sub-communities needs to be processed by that unique community member. This demanding task requires an additional amount of time from that single community member, generating delays. Moreover, the absence of a unique boundary spanner can stall every formal interaction between two sub-communities.

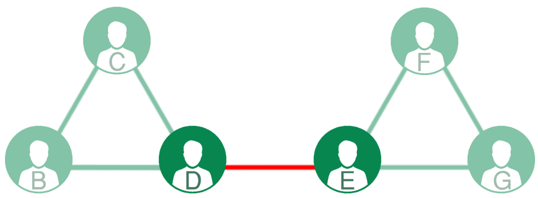


Figure 4.11: Example of occurrences of Radio Silence

Considering the example of development community proposed in Figure 4.1a, since the detection of Radio Silence Community Smell is defined through the concept of unique boundary spanner, it is necessary to identify all community members who are the only members of their sub-community that communicate with another sub-community. Therefore, we have to consider every inter-communication between different sub-communities.

The decisional process to detect Radio Silence Community Smell using the proposed identification pattern within the proposed example, considering one sub-community at a time, is the following:

- **Sub-community B-C-D**: It communicates with sub-community E-F-G exclusively through community member identified by the letter D, then this is identified as a Radio Silence occurrence and the community member identified by letter D is the knowledge and information broker toward the other sub-community;
- **Sub-community E-F-G**: It communicates with sub-community A-B-C exclusively through community member identified by the letter E, then this is identified as a Radio Silence occurrence and the community member identified by letter E is the knowledge and information broker toward the other sub-community.

The two detected occurrences are represented in Figure 4.11, where knowledge and information brokers are highlighted while all the other community members are faded out. The quantification of the Radio Silence Community Smell that we proposed is focused on the number of different developers who act as boundary spanner toward other sub-communities and not on the number of unique communication links.

Chapter 5

Socio-technical Quality Framework

After the definition of identification patterns capable of identifying Community Smells occurrences and thus enabling the ability of detecting the presence of potential Social Debt within a software development community, proposed in Chapter 4, we focused on the arduous task of identifying which socio-technical development aspects are related or responsible of the insurgence of Community Smells. Social Debt is considered an ubiquitous phenomenon within the entire software development life cycle and in all its related literature (Section 2.5) it was not possible to clearly identify specific socio-technical aspects responsible of increases or decreases of Social Debt.

Our contribution to cover this further step in the identification of socio-technical aspects which can be considered responsible of the contraction or extinction of Social Debt was the definition of a Socio-technical Quality Framework. Such Socio-technical Quality Framework is a fundamental basic building block of this master thesis, since it was used to correlate its composing quality factors with the occurrence of Community Smells within FLOSS development communities in order to understand how and to what extend Community Smells are related or caused by specific socio-technical quality factors.

The Socio-technical Quality Framework proposed in this chapter is composed by a total of 40 socio-technical quality factors, which were mainly extracted from empirical software engineering literature but the introduction of some of them were even suggested and supported by the results of the executed survey (Chapter 6). The identified quality factors composing the defined Socio-technical Quality Framework analyze many different software development aspects: from dimensional characteristics of Developer Social Networks to core community members quantification, from turnover rates to socio-technical metrics and from truck numbers to Social Network Analysis metrics.

Category	Metric ID	Socio-technical Quality Metric Description
Developer Social Network metrics	devs	Number of developers present in the global Developers Social Network
	ml.only.devs	Number of developers present only in the communication Developers Social Network
	code.only.devs	Number of developers present only in the collaboration Developers Social Network
	ml.code.devs	Number of developers present both in the collaboration and in the communication DSNs
	perc.ml.only.devs	Percentage of developers present only in the communication Developers Social Network
	perc.code.only.devs	Percentage of developers present only in the collaboration Developers Social Network
	perc.ml.code.devs	Percentage of developers present both in the collaboration and in the communication DSNs
	sponsored.devs	Number of sponsored developers (95% of their commits are done in working hours)
	ratio.sponsored	Ratio of sponsored developers with respect to developers present in the collaboration DSN
Socio-technical metrics	st.congruence	Estimation of socio-technical congruence
	communicability	Estimation of information communicability (decisions diffusion)
	num.tz	Number of timezones involved in the software development
	ratio.smelly.devs	Ratio of developers involved in at least one Community Smell
Core community members metrics	core.global.devs	Number of core developers of the global Developers Social Network
	core.mail.devs	Number of core developers of the communication Developers Social Network
	core.code.devs	Number of core developers of the collaboration Developers Social Network
	sponsored.core.devs	Number of core sponsored developers
	ratio.sponsored.core	Ratio of core sponsored developers with respect to core developers of the collaboration DSN
	global.truck	Ratio of non-core developers of the global Developers Social Network
	mail.truck	Ratio of non-core developers of the communication Developers Social Network
	code.truck	Ratio of non-core developers of the collaboration Developers Social Network
	mail.only.core.devs	Number of core developers present only in the communication DSN
	code.only.core.devs	Number of core developers present only in the collaboration DSN
	ml.code.core.devs	Number of core developers present both in the communication and in the collaboration DSNs
	ratio.mail.only.core	Ratio of core developers present only in the communication DSN
	ratio.code.only.core	Ratio of core developers present only in the collaboration DSN
	ratio.ml.code.core	Ratio of core developers present both in the communication and in the collaboration DSNs
Turnover	global.turnover	Global developers turnover with respect to the previous temporal window
	code.turnover	Collaboration developers turnover with respect to the previous temporal window
	core.global.turnover	Core global developers turnover with respect to the previous temporal window
	core.mail.turnover	Core communication developers turnover with respect to the previous temporal window
	core.code.turnover	Core collaboration developers turnover with respect to the previous temporal window
	ratio.smelly.quitters	Ratio of developers previously involved in any Community Smell that left the community
Social Network Analysis metrics	closeness.centr	SNA degree metric of the global DSN computed using closeness
	betweenness.centr	SNA degree metric of the global DSN computed using betweenness
	degree.centr	SNA degree metric of the global DSN computed using degree
	global.mod	SNA modularity metric of the global DSN
	mail.mod	SNA modularity metric of the communication Developers Social Network
	code.mod	SNA modularity metric of the collaboration Developers Social Network
	density	SNA density metric of the global Developers Social Network

Table 5.1: Summary of the Socio-technical Quality Framework

The complete list of the 40 quality factors defined within our Socio-technical Quality Framework is summarized in Table 5.1, that reports for each metric its appropriate category, identification string used in Codeface4Smells (Chapter 7) and a brief description of such quality factor. In the following sections every expressed quality factor is explained and its presence into the Socio-technical Quality Framework is motivated.

5.1 Developer Social Network metrics

Community dimensions. Since our approach is based on Developer Social Networks to identify and detect Community Smells by exploiting such generated networks to capture communication and collaboration characteristics and habits of Open Source Software community members, some of the most basic yet important metrics that characterize a development environment are the dimensional number of developers and community members who constitute the community itself. One fundamental dimension that characterizes a software development community, independently from its Open Source or Closed Source nature, is the number of people involved in it and how they interact with each other while communicating or collaborating during any software development phase. It is important to understand that, especially when analyzing and dealing with Open Source Software projects, it is necessary to consider and capture different roles and behaviors of community members, since many parallel and different contribution and participation levels are usually present within a FLOSS project development environment. The number of developers contributing to a project can be composed by formal developers, regular and constant contributors to the source code, developers who participate in the software development with a discontinued attitude and other code contributors that are more difficultly characterizable, such as: developers who contribute to the project just to fix issues that interfere with their software usage, thus potentially once in a lifetime contribution to the project, and official maintainers who are responsible of the entire project development cycle, who may or may not be directly involved in the source code development activity. A formal definition of every participation level is indeed difficult because every project and every developer can potentially imply a different contribution typology and can be associated to a different behavior. Moreover, Open Source Software communities are not only constituted by developers who contribute directly to the project's source code but they are composed even from a whole different universe of community members with different functionalities and roles not directly involved within the software development activity, performing useful services such as: provide support to software users, seek for help and thus possibly provide easy-to-find solutions for users that will have the same problematic in the future, create project documentation, translate the software and its documentation or its informational pages in different idioms, report bugs or software malfunctions and suggest possible solutions, test new software functionalities or versions and promote the software and/or its community. In order to achieve a finer grain level analysis and capture every possible community member typology involved within the development life cycle, we considered the communication Developer Social Network extracted from the development mailing list, the collaboration Developer Social Network extracted from the Version Control System history and the global Developer Social Network obtained merging the two previously described Developer Social Network. A community member is defined as a person who supports a project in at least one of the two following ways: if he or she participates within the communication channel by sending and/or receiving a message within the range in analysis or if he or she is the author of at least one software contribution (commit) to the source code within the range in analysis. These different development dimensions, extracted from the relative Developer Social Networks, are captured through the following measurements:

1. The number of *community members present only in the communication channel* (**ml.only.devs**). Thus, community members who do not directly contribute to a project's source code but that provide help, seek for help and/or participate into community's activities in different ways;

2. The number of *developers contributing to the source code but who do not communicate* (**code.only.devs**). Thus, developers who contribute to a project's source code but that are distant or not interested in participating into the community itself, rather they have specific interests only in the developed software. This dimension can capture once in a lifetime contributors or developers whom contribute to the project by fixing or implementing features that are needed by their specific usage of the software;

3. The number of *community members who contribute to the source code and communicate* (**ml.code.devs**). Thus, this dimensional metric provides information about members completely involved in different community's activities;

4. The number of global *distinct community members involved in the community* (**devs**). Thus, this dimensional metric captures every possible person involved in any possible way into a FLOSS community.

Previously listed dimensional metrics are considered even in their percentages form, in order to achieve a deeper understanding of how community members are involved within every different analyzed scenario. Therefore, the following percentages are computed by the proposed Socio-technical Quality Framework: *community members involved in the software development of a project who communicate* (**perc.ml.code.devs**), *community members involved exclusively in the communication channels* (**perc.mail.only.devs**) and *developers who contribute to a project only by committing code but that do not communicate* (**perc.code.only.devs**). These typologies of dimensional metrics, related to the size of a community's organization, are supported as significant and valid measures to be inserted into our Socio-technical Quality Framework by Conway's law [9], which is the foundation of every socio-technical empirical software engineering research, because one of its characteristics was that this his becomes more reliable as the organizational size increases.

Sponsored developers. FLOSS development is growing year after year as an important commercial activity and, as soon as an Open Source Software project becomes promising, commercial companies may be interested in actively participating into its development in order to adequate the software to their needs. Due to their status, developers sponsored by commercial companies devolve most of their time to software development and to community activities, working on most of the tasks, organizing and coordinating the community. The proposed Socio-technical Quality Framework considers the presence of software developers who contribute to a FLOSS development that are *sponsored by commercial companies* or who can be considered as *self-sponsored developers* and quantifies such phenomenon computing the total number of sponsored developers present in the collaboration Developer Social Network (**sponsored.devs**) and its relative ratio (**ratio.sponsored**). The background theory that supports the inclusion of these metrics, related to developers sponsored by commercial companies and to self-employed developers, was provided in 2014 by Riehle et al. [8], who stated that the ratio of volunteer and paid work can be considered as an indicator of the health of an Open Source Software project, as well as they may indicate the commercial attractiveness of a FLOSS project itself.

5.2 Socio-technical metrics

Software development is a complex activity which is influenced by both social and technical components of an organization, or of a community in the case of Open Source Software. As exhaustively explained in Section 2.1 and in Section 2.5, unhealthy social and technical characteristics of a software project may imply the insurgence of additional malfunctions within the produced output and affect software developers relational dynamics between themselves and with the organization itself. The interdependence of such components, affecting every development phase, was considered so prominent that the term “socio-technical” was coined, to highlight the specific effects that social and technical aspects of software development are capable to exercise in conjunction on the outcome of a software community. Since unhealthy socio-technical patterns may generate Technical Debt and its social counterpart, we included into our Socio-technical Quality Framework some specific metrics which were defined or suggested by socio-technical software engineering literature.

Socio-technical congruence. The attention on socio-technical aspects of software development phases grew in the last decade such that Cataldo et al. [5] in 2008 introduced the key idea of “socio-technical congruence”. As previously explained (Section 2.1), socio-technical congruence is meant to measure the level of agreement between communication needs of software artefacts and the actual communications that occur within a software development environment. The socio-technical congruence measurement methodology introduced by Cataldo et al. was based on the relation between product dependencies, work dependencies and their comparison to coordination activities. Instead, the *socio-technical congruence* (**st.congruence**) measurement implementation that we proposed in our Socio-technical Quality Framework is based on the direct comparison of the collaboration Developer Social Network, that represents all the development relationships and work dependencies within a software development and highlights the communication needs, to the communication Developer Social Network, that represents all the actual coordination activities and relationships within a software development community. Thus, our implementation of socio-technical congruence metric, rather than considering module dependencies as proposed by Cataldo et. al, is based on Valetto et al. [45] idea that to investigate and quantify socio-technical congruence within a software development environment, it is necessary to investigate if there is a similarity (congruence) between the coordination structure (represented by the collaboration DSN) and the social organization (represented by the communication DSN) of the project itself. Socio-technical congruence can be considered as one of the most important socio-technical quality metrics of the proposed framework because it is correlated to the Social Debt concept, since in Conway's law terms it can be considered as an indicator of the presence of possible social related issues, as it compares the social and technical processes requirements and measures their accordance. Furthermore, Cataldo et al. suggested that a higher socio-technical congruence is correlated to a higher software development performance.

Temporal and geographic dispersion. Since FLOSS development can be inherently considered a Global Software Development environment, it will be affected by all the considerations and problematics previously elicited in Section 2.2. Temporal and geographic dispersion, not to mention cultural and linguistic differences, can negatively affect software development performance, software quality and the software development community itself, in case such distances and differences are not kept under control and exploited to extract the best profit from their existence. It is known that if temporal and geographic distribution and dispersion of a project occur to be unaddressed, they can generate important side effects in the software development community or to its product outcome [21]. The temporal and geographic dispersion of contributors whom participate to software development and the related impact of such dispersions onto the software community is captured by the our Socio-technical Quality Framework through an indicator that represents the number of *different time-zones involved in the source code development* (**num.tz**).

Communicability. Another important socio-technical quality metric that influenced our proposed quality framework, was defined in a Social Debt related study conducted by Tamburri et al. focused on software architecture processes [7]; they proposed a metric called "Debt-Aimed arcHitecture-Level Incommunicability Analysis" (DAHLIA) that had the purpose of evaluating the in-communicability of architectural decisions across a software development community. Tamburri et al. defined in-communicability as the "likelihood that who should know about architecture decisions actually does not know anything about them". We introduced into our Socio-technical Quality Framework a new metric called *communicability* (**communicability**), which can be considered as the inverse of in-communicability measurement proposed by DAHLIA. Communicability was preferred for practical reasons, because given the implementation methodology briefly explained in the following, in-communicability in FLOSS development communities tends to be characterized by very small values. DAHLIA was defined considering the social-network weak-ties hypothesis and was supposed to be calculated for every project's architectural decision. In order to adapt and implement an estimation of communicability, we treated every technical collaboration between two developers represented in the project's collaboration Developer Social Network, as a possible source of design or architectural decisions. To mimic the social-network weak-tie hypothesis adopted in DAHLIA and to pander socio-technical development aspects captured by the collaboration and the communication DSNs of a software community, we concluded that developers whom should be made aware of a decision (a collaboration) are those that directly collaborate with at least one of the two developers involved in the decision making. The motivation of such decision reside in the concept that since they are related, probably they are interested and are working on the same or on a correlated part of the software code source. A developer is considered aware of a technical decision if he or she is connected with at least one of the developers implicated in the decision (collaboration) in the communication Developer Social Network. Since communicability metric is related to every existent collaboration between two different project's developers, we considered the global communicability metric estimation as the mean of the communicability metric of all the collaborations present within the collaboration Developer Social Network.

In the proposed Socio-technical Quality Framework it was introduced a metric that resides between socio-technical and Developer Social Network metrics categories, as it is directly dependent on actual identified Community Smells, explained in Chapter 4, and thus it can be considered as an indicator of the presence of Community Smells and their negative effects. Such metric is constituted by the percentage of total community members present in the global Developer Social Network who are *involved into at least one identified Community Smell* (**ratio.smelly.devs**) and its purpose is to quantify the predominance and quantity of members whom incurred in Community Smells during their activities.

5.3 Core community members metrics

Core community members. A group of dimensional metrics related to Developer Social Network metrics, which were recognized such important and specific to deserve a dedicated section, are constituted by indicators capable of measuring the number of core members involved into software development and its associated community activities. Core developers are community members who actively participate into development activities, are associated with important managerial roles, have higher decisional power and pursue the goal of supporting and soliciting software contributions and community participations. Metrics related to core community members were inserted in our Socio-technical Quality Framework because, over the years, many empirical software engineering studies used such indicators to characterize the quality of software development activities and many researchers proposed different methodologies to identify such typology of developers from Developer Social Networks. The count of core and peripheral developers of a software development community can be considered as very useful indicators of the development state and of the community general health with respect to the following hypothesis formulated by Mockus et al. [13]:

- FLOSS projects have a group of core developers (usually 10-15 people) whom control the code base and which are responsible of at least 80% of new implemented functionalities;
- Large FLOSS projects are subdivided into different groups, creating several related sub-projects;
- A successful FLOSS project has a larger group of members (one order of magnitude bigger than core developers group) committed to bug-fixing and another larger group of members (another other of magnitude) that reports problems;
- A FLOSS project that has a strong group of core developers but at the same time has a small number of developers committed to bug-fixing, will be able to implement new functionalities but it will fail as defects will not be found and corrected.

Crowston et al. [61] stated that core contributors involved in the development of a FLOSS project are very important because “many of the processes necessary for successful projects likely involve core members differently than peripheral members” and concluded that both the typologies of members (peripheral and core) should be considered separately during an empirical software engineering study to obtain valid results. In their study Crowston et al. showed that, while peripheral developers are usually involved in bug fixing or small code enhancements and contribute to the project in an irregular and short-term manner, core developers have a long-term involvement with the software project, are fundamental to perform system architecture decisions and create the general leadership structure of the project community. Terceiro et al. [62] correlated core developer behaviours to the structural complexity of a project development environment and discovered that usually core developers generate less structural complexity than peripheral developers, thus a stable and healthy team of core developers is a must for the success and sustainability of FLOSS projects. We tried to define precisely the properties that characterise core developers using different literature resources but its definition in real world Open Source communities is quite blurry because the figure of core developers and the privileges associated to this figure can change within every development community. As an example, in some communities maintainers and core developers terminologies coincide, while in others they refer to completely different concepts. In this master thesis *we considered as a core community member whoever is characterised by a high degree of centrality* within the specifically considered Developer Social Network. Therefore, the proposed Socio-technical Quality Framework contained the following metrics related to the identification of core community members:

1. Number of *core developers within the global Developer Social Network* (**core.global.devs**);

2. Number of *core developers within the communication Developer Social Network* (**core.mail.devs**);

3. Number of *core developers within the collaboration Developer Social Network* (**core.code.devs**);

4. Number of *core developers within the collaboration Developer Social Network who are sponsored* by commercial commercial companies or who can be considered as self-employed in the FLOSS project and its corresponding ratio percentage (**sponsored.core.devs**, **ratio.sponsored.core**);

5. Number of *core developers whom are present only in the communication Developer Social Network* but not in its collaboration counterpart and its corresponding ratio percentage (**mail.only.core.devs**, **ratio.mail.only.core**);

6. Number of *core developers whom are present only in the collaboration Developer Social Network* but not in its communication counterpart and its corresponding ratio percentage (**code.only.core.devs**, **ratio.code.only.core**);

7. Number of *core developers whom are present both in the communication and collaboration Developer Social Networks* and its corresponding ratio percentage (**ml.code.core.devs**, **ratio.ml.code.core**).

Truck number. As soon as peripheral and core community members of the three typologies of Developer Social Networks are identified, it is possible to compute the *truck number* (also known as bus factor) of the global DSN (**global.truck**), of the communication DSN (**mail.truck**) and of the collaboration DSN (**code.truck**). Such metric typology is important because it measures the concentration of information within every individual developer and it is considered as a good indicator to evaluate risks associated to knowledge loss due to turnover of developers [57]. The truck number is commonly defined as the quantity of members who can be unexpectedly lost ("hit by a truck") without dooming the project to failure, due to lack of knowledge or competent members. While a low truck number can highlight a possible project risk because most of the knowledge is concentrated in few people, a high truck number can represent a low risk due to developers turnover because the knowledge is spread across all the community and, if a developer will leave the project, other developers do have enough knowledge to carry on the software development activity [57]. The decision of including truck number metric into our Socio-technical Quality Framework was supported by Lavallée et al. [56] research, which considered "human resource planning" issues as one of the possible organizational factors impacting on software development quality.

5.4 Turnover

The organizational structure of a software project can be influenced from members joining and leaving the software development community and, while departing community members will lead the project to knowledge loss and relative knowledge gaps that need to be filled in order to avoid the raise of problematic situations, new members might bring innovations and new ideas into the community [14]. New developers may introduce defects and lower developed software quality due to their lack of knowledge about the system and it will require time from core community members to answer to their questions and mentoring them. Since the quality of a development community is tightly related to the knowledge and the experience of members, the longer a community member has been involved in a project the fewer defects will be present in his or her code [63]; consequently the quality of developed source code is higher when long time and more experienced developers are present within the development community.

While studying the relationship between developer-centric measures of organizational changes and the probability of customer-reported defects in the context of a large software project, Mockus [14] discovered that organizational volatility was correlated to an increased probability of customer-reported defects in the software. Therefore, experienced developers leaving an organization create gaps in tacit knowledge and, as consequence, the product quality decreases. Furthermore, he discovered that new developers bring innovation into the software community but they actually have no impact on software quality, usually the motivation of this phenomenon is that they are assigned to less critical changes [14].

As the quality of software contributions is usually related to the system knowledge of community members, the highest risk for a project is when a core developer leaves the community. This event can generate a big knowledge gap within the project that should be mitigated as soon as such phenomenon is detected, for example promoting strong developers to become core developers. Since FLOSS projects are constituted by the fundamental contributions of voluntary developers, delays in the process of meritocratic promoting a strong developer to a core developer role (e.g. granting him or her the write permission on the Distributed VCS) can cause departures of unsatisfied developers from the software community. Thus, a mechanism to early identify and monitor promising future core developers is indeed needed to ensure the successful evolution of FLOSS projects [64].

Turnover should be taken into account as it may results in a loss of productivity, personnel re-training and additional recruitment costs. In an industrial scenario replacing one worker may costs from three months to one year of salary, depending of the worker needed skills [55]. Core members turnover should be considered and handled by community maintainers as well, because a stable and healthy core team is fundamental to ensure the stability of a FLOSS project [62]. Moreover, within the software engineering literature, turnover is considered as one fundamental socio-technical quality measure which is necessary to understand a dynamics and health of a development community. The inclusion of turnover metrics in our socio-technical quality framework was further motivated by the fact that Cataldo et al. [15] empirically demonstrated that if a software development base is stable, then the socio-technical congruence of an organization will increase over time.

In our framework we considered the *turnover* of every community member present in the global DSN and in the collaboration DSN (**global.turnover**, **code.turnover**) and the turnover of core community members of the global DSN , communication DSN and collaboration DSN (**core.global.turnover**, **core.mail.turnover**, **core.code.turnover**).

A further metric present within our Socio-technical Quality Framework related to turnover of members, is the ratio of *community members who left the community and that were involved in at least one Community Smells* in the previously considered temporal range (**ratio.smelly.devs**). This metric can indicate to what extent Community Smells can imply negative effects within a development environment to the extreme of causing the abandonment of the community by previously involved members.

5.5 Social Network Analysis metrics

As exhaustively described in Section 2.2, during the last decade, empirical software engineering studies tended to consider and evaluate software development communities through the use of Social Networks, applying to them Social Network Analyses methodologies [31,32,36]. Meneely et al. [37] proved that socio-technical Developer Social Networks created observing development artifacts are representative of actual socio-technical relationships present in a software development community. Given the extensive presence of SNA studies in the empirical software engineering research field, it was considered reasonable to include within the proposed Socio-technical Quality Framework some fundamental Social Network Analysis metrics which are capable of offering an insight about a the structure of Developer Social Networks of a community and their characteristics.

Centralization. Centralization [65] is a Social Network Analysis metric that calculates the graph-level centrality score, based on node-level centrality measure of the entire considered graph. Centralization Social Network Analysis metric is founded on the concept of nodes centrality, which is associated to the *identification of nodes which are most popular and that stand in the center of attention* in a possible sociometric start concept within the network in analysis. Centralization can be considered as the extension of the node centrality concept to the whole network global level, as it refers to a network level indicator of the overall cohesion or integration of all the nodes of the network. Within our Socio-technical Quality Framework the following typologies of Social Network Analysis centrality metrics were considered for the global Developer Social Network [65]:

1. *Degree centrality* (**degree.centr**): this centrality metric was the first typology of Social Network Analysis centrality metric defined in the SNA literature and it is founded on the counting of incident edges upon a social network node, thus it represents how many connections a single network node has. Degree SNA metric represents the immediate possibility that a network node has of capturing an information freely flowing through the entire social network. Within directed graphs it is possible to consider two different typologies of degree centrality, considering the incoming or the outgoing edges from a specific network node, but as explained in Chapter 7 we considered every generated Developer Social Network as an undirected graph so we are not interested by this further distinction;

2. *Closeness centrality* (**closeness.centr**): this centrality Social Network Analysis metric is defined upon the concept of shortest paths present within the considered social network, computed using a natural distance metric between every pair of its network nodes. Closeness centrality SNA metric represents the mean number of steps that an information has to take in order to reach every other node belonging to the social network;

3. *Betweenness centrality* (**betweenness.centr**): this centrality Social Network Analysis metric is based on the quantification of the number of times in which a single network node acts like an informational bridge, thus the number of times in which it is present within the shortest path between two different network's nodes. This Social Network Analysis metric is capable of detecting the importance that each node has with respect to the relationships present in the other network entire social network between different nodes.

Modularity. Modularity [66] is a Social Network Analysis metric that measures the strength of the structure of a community: high modularity indicates that a clear definition and distinction of different sub-communities within the considered network exists, while when the network modularity tend to zero it indicates that there are no distinct sub-communities within the considered network. In the Social Network literature it was identified a threshold of 0.3 for the modularity metric [66], over which a community is considered highly modular and thus in posses of a clear distinction between sub-communities present in its development network. The inclusion of modularity in the proposed framework was further motivated by Mockus's study [13], who discovered that a higher modularity corresponds to a lower coordination need. In our Socio-technical Quality Framework modularity was computed not only for the global Developer Social Network (**global.mod**) but even for the communication DSN (**mail.mod**) and for the collaboration DSN (**core.mod**).

Density. Density [67] is a Social Network Analysis metric that indicates the ratio of the number of edges connecting different network nodes with respect to the total number of possible edges between every network's nodes. This metric is fundamental to classify a social network as a dense or sparse graph. Density (**density**) is computed only for the global Developer Social Network within the proposed Socio-technical Quality Framework.

Chapter 6

Survey

A survey was conducted among three well-known Open Source Software development communities. The main motivations and purposes that brought us to consider, design and execute a questionnaire are summarised in Table 6.1.

Although Social Debt and Community Smells were never mentioned during the survey execution, it was possible to gather vital information about them considering developer perceptions. Since the survey copes the existence and characteristics of Social Debt and Community Smells within FLOSS development communities, it constitutes an important and fundamental contribution to the Social Debt research field by extending it with further notions and developer perceptions. This is the first questionnaire ever done to assess Social Debt characteristics and perceptions from verified FLOSS community members, since we ensured that respondents of a development community contributed to the project's source code with at least one contribution.

Generally speaking, considering the results obtained within the three reference projects, FFmpeg developers seem to be the most disillusioned with respect to their project's characteristics and inner mechanisms. Respondents related to the FFmpeg community exhibited slightly intolerant behaviours toward core community members and developers sponsored by commercial companies, perceived an important lack of equality in developer opinion's importance with respect to decisions and are very

Survey motivations and goals
Validate motivations, constituent theories and assumptions of this master thesis;
Verify the existence of Social Debt and Community Smells in FLOSS communities;
Support Social Debt and Community Smells discoveries and characterisation;
Extract verified FLOSS developer perceptions about possible social related issues;
Identify further quality metrics considered important by developers;
Qualitatively detect the presence of Community Smells using developer perceptions;
Validate empirical results obtained with Codeface4Smells analysis.

Table 6.1: Goals and motivations of the survey

upset about the documentation and the clarity of rules and structures of the project.

In Section 6.1 we illustrate the structure of the questionnaire and its main sections, in which all proposed questions were subdivided. Section 6.2 summarises some characteristics and background information of interviewees who participated at the questionnaire. The existence of Social Debt within Open Source Software development communities and the validity of using mailing list in order to identify Community Smells are verified in Section 6.3. Section 6.4 constitutes a fundamental contribution of this master thesis as it verifies the existence and effectiveness of Community Smells, proposed in Chapter 4, using respondents perceptions. Finally, Section 6.5 highlights the fundamental role that the survey results had on the identification and definition of quality factors belonging to our Socio-technical Quality Framework.

The structure of the survey with the entire list of questions, the detailed characteristics of analysed projects and the complete representations of obtained responses are presented in Appendix A.

6.1 The questionnaire

The questionnaire, designed specifically for our study, was composed by a list of questions aimed to address the aspects and goals reported in Table 6.1. Social Debt and Community Smells were never mentioned during the execution of the questionnaire, in order to avoid biases in responses. The subset of proposed questions, which contains only the ones considered during our study, can be subdivided into three different sections:

1. The *first part* of the questionnaire was composed by three general questions, with the purpose of addressing the background of questionnaire respondents, and four questions dedicated to the the relationships of respondents with his or her project community. The background information that respondents had to specify during the questionnaire were: year of birth, country and occupation. Within this section, after specifying the community in which the respondent was involved, it was asked to specify the role within the project community. The community and e-mail address specified within this section were used to verify that the respondent actually committed at least one software contribution to the project's source code.

2. The *second part* of the questionnaire was composed by four questions. In this section we asked if the respondent's participation within the FLOSS community was remunerated. Then, we tried to understand, between all possible communicational channels, which channel was mainly used inside the respondent's community. Thus, we were interested in the identification of the channel

through which important project decisions were taken. Finally we tried to assess information and perceptions about possible Social Debt existence and its related issues, questioning what respondents consider as the main cause of time waste within their community during the development phases and the reason of their longest wait occurred before a proposed commit was considered by project maintainers.

3. The *third part* of the questionnaire, like the final part of its second section, was related to possible Social Debt and Community Smells aspects and perceptions, but it was composed by fourteen sentences and every respondent had to specify his or her agreement with every proposed sentence. Agreement was collected using the Likert scale, composed by the following options: strongly disagree, partially disagree, neither agree nor disagree, partially agree and strongly agree. In this part of the questionnaire we tried to capture developer perceptions about the importance of developers sponsored by commercial companies and how much a project can be influenced by the absence of a developer or the absence of a core developer. We then proceeded to understand the perceived degree of formality and if the decisional power, within FLOSS communities, was shared between every community member or it was concentrated within a limited group of members. In this part of the questionnaire we tried to assess to information and opinions related to the quality of development artefacts produced by communities, like the general and software architecture documentations, and if eventual issues in such artefacts resulted in unofficial assumptions made by single developers. We analysed the perceived importance of communications before and after software development commit activities and then we moved forward considering the clarity of community rules and structures. Finally, we tried to capture the existence of sub-communities within the three reference projects and understand some characteristics of such sub-communities, like communication behaviours, homophily and antagonisms.

The list of questions considered and used in the elaboration of this master thesis are provided in the Appendix A, together with a detailed summary of received responses relative to the third section of the questionnaire, grouped with respect to the analysed communities.

6.2 Background of respondents

The list of FLOSS development communities initially considered was composed by 10 software projects, specifically selected to cover different combinations of the following project characteristics:

1. *Project size*: a community is considered “big” if the total number of its commits is above the average of the total number of commits of all considered projects, otherwise it is considered “small”;

2. *Development activity*: a community is considered “active” if the total number of its commits during the last 12 months is above its average yearly commits number (total number of commits / years of activity), otherwise it is considered in “stall”;

3. *Project relevance*: a community has a “high” relevance if the total number of member involved in its development is above the average of the total number of developers who contributed to the source code of all the considered projects, otherwise it has a “low” relevance.

The complete list of Open Source communities initially considered and the different combinations of factors characterising every project are reported in Table 6.2, while a more detailed table explaining classification motivations can be found in Appendix A.

Project	Size	Activity	Relevance
Firefox	*BIG*	*ACTIVE*	*HIGH*
Android	*BIG*	*ACTIVE*	*HIGH*
WebKit	*BIG*	*ACTIVE*	*LOW*
LibreOffice	*BIG*	*STALL*	*HIGH*
FFmpeg	*SMALL*	*ACTIVE*	*HIGH*
OpenSSL	*SMALL*	*ACTIVE*	*LOW*
LibreSSL	*SMALL*	*ACTIVE*	*LOW*
AngularJS	*SMALL*	*STALL*	*HIGH*
Khtml	*SMALL*	*STALL*	*LOW*
libva	*SMALL*	*STALL*	*LOW*

Table 6.2: Projects initially considered for the survey

Considering the characteristics of all initially considered projects, we decided to *explicitly target our survey toward three well-known FLOSS development communities*, which covered almost every sampling combination reported in Table 6.2. The three reference projects which were the subjects of our survey are: Firefox, LibreOffice and FFmpeg.

To achieve a higher diversity of developer typologies who contributed in any possible way and in different temporal instants to every considered project, from project maintainers to once in a life time contributors, we extracted from the VCS of every considered project the entire list of authors of software contributions committed in the entire life time of a project. Such list was extracted using a git command considering every unique e-mail address present and every e-mail in an invalid format was removed with a regular expression (see Appendix A for further information).

The questionnaire was created using Google Forms and it was available from the 15th of February 2016 to the 5th of March 2016, for a total of 20 days. The link to the questionnaire and a short introduction were sent to every extracted unique e-mail. To avoid any possible bias, Social Debt and Community Smells were never mentioned during the questionnaire or in its introduction and the survey was introduced as a general FLOSS questionnaire. To mitigate the side effects of this generalisation, Open Source developers participation was stimulated offering as a prise two Amazon gift cards of the value of 25 dollars each (total jackpot 50$). The two prizes were assigned to two randomly extracted participants few weeks after that the survey time window ended.

Figure 6.1: Survey results (part one)

A total of 5169 unique valid e-mail addresses were extracted and contacted from the Version Control Systems of the three reference projects, subdivided as follows: 1057 valid e-mails of LibreOffice developers, 1162 valid e-mails of FFmpeg developers and 2950 valid e-mails of Firefox developers. At the end of the survey we collected a total of 59 valid responses. A response was considered valid if and only if the e-mail address specified by the respondent was present in list of committer e-mails of the community specified within the same response.

As showed in Figure 6.1a, the 59 valid responses obtained from our interviewees were divided between reference projects as follows: 33 replies from Firefox community which constitute the 55.93% of total valid replies, 14 replies from FFmpeg community which constitutes the 23.73% of total valid replies and 12 replies from LibreOffice community which constitute the 20.34% of total valid replies.

In the following are presented the results related to the first section of the questionnaire, in order to achieve a better understanding of the background of the interviewees and validate the representativeness of survey responses. All the replies were considered as a whole and not subdivided into a finer grain level, considering every single community.

Moving to generational characteristics of survey respondents, summarised in Figure 6.1b, we concluded that almost half of interviewees were in the range of age between 26 and 35 years old (50.85%). Developers younger then 25 years old and developers with an age between 36 and 50 years old were both almost equivalently represented by one quarter of the total respondents, while developers over the age of 50 years old represented only the 5% of the total respondents. We considered this subdivision as a reasonable representation of developers' ages in the FLOSS ecosystem.

Figure 6.1c helps to understand the working status of the survey interviewees. The majority of respondents had a full-time job (69.49%), while a small part of them had a part-time job (8.47%). One out of four developers identified himself or herself as a student (20.34%) and a small part of the interviewees did not have a job nor were students (1.69%).

Moreover, we asked respondents to specify if their contribution to the project was on a voluntary base or if their involvement within the project community was sponsored, in any form, by a commercial company. Globally we retrieved that almost three developers out of four participated in FLOSS development without any monetary interest (71.19%); considering the three reference communities, this parameter was a bit fluctuating from a minimum of 57.14% of FFmpeg community to a maximum of 78.77% of Firefox community. Moving to waged development contributions, the percentage of developers whose involvement within a FLOSS community was completely supported by a commercial company is 23.73% of the total respondents and it was stable within all the reference projects (Firefox: 21.21%; LibreOffice: 25%; FFmpeg: 28.57%). Developers who were paid only partially by a commercial company to contribute to one of the considered projects constituted the 5.08% of the total interviewees and its distribution between the three reference projects was very different: none of Firefox community respondents was partially paid by a company, while considering the other projects: 8.33% of LibreOffice developers and almost the double (14.29%) of FFmpeg developers stated that their contributions were partially paid by a commercial company.

An interesting aspect inspected by our questionnaire, in order to verify the representativeness of received replies with respect to different possible development behaviours and tasks, was the role of respondents within their FLOSS development community. As expected, given the modalities through which we retrieved the developers information, the majority of respondents classified themselves as developers (71.19%). An important fraction of interviewees specified their status of software engineer (11.86%), while just a small part of the respondents stated that they were the maintainers of the project (1.69%). Moving toward less technical tasks, we received replies from translators (3.39%) and a considerable amount of responses from interviewees that were not able to identify themselves in any of the previously listed categories (11.86%). Figure 6.1d presents a graphical representation of respondent roles distribution with their related percentages.

Country	# respondents	Percentage
United States	9	15.25%
Germany	7	11.86%
France	7	11.86%
U.K.	5	8.37%
Brazil	4	6.78%
India	4	6.78%
Austria	3	5.08%
Belgium	2	3.39%
Poland	2	3.39%
Sweden	2	3.39%
Taiwan	2	3.39%
Argentina	1	1.69%

Country	# respondents	Percentage
Canada	1	1.69%
China	1	1.69%
Estonia	1	1.69%
Spain	1	1.69%
Finland	1	1.69%
Italy	1	1.69%
Kazakhstan	1	1.69%
Romania	1	1.69%
Russia	1	1.69%
Tunisia	1	1.69%
Venezuela	1	1.69%

Table 6.3: Nationalities of survey respondents

The questionnaire had a predominant participation of developers based in the United States of America (15.25%) and in Europe, in which the most represented countries were Germany (11.86%), France (11.86%) and United Kingdom (8.37%). The geographically distribution of the survey respondents can be considered quite high since we collected replies coming from 23 different countries of the world, which are represented with their related number of replies and the associate percentage of the total responses in Table 6.3.

Considering the background information of respondents, it was possible to state that our survey involved developers of every age, geographic location, occupation and characterised with different roles within FLOSS development communities. In conclusion, these results provide the confirmation that retrieved survey responses can be considered enough heterogeneous and diversified to be representative of the entire FLOSS ecosystem.

6.3 Confirmatory role

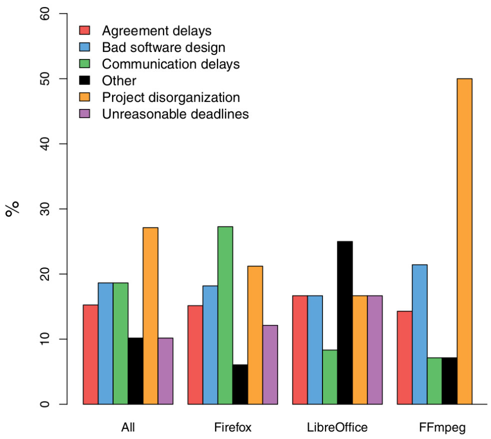
The obtained replies allowed us to verify some assumptions made during the execution of this master thesis, avert possible threats to validity of this study and support the existence of Social Debt from the view point of developer perceptions due to socio-technical related issues within FLOSS development communities. Therefore, this section has the fundamental goal of motivate and *justify this master thesis research questions*, as it verifies the existence of Social Debt within FLOSS communities through developer perceptions.

As extensively explained in Chapter 4 and Chapter 7, during the execution of this master thesis, we relied on Developer Social Networks extracted from FLOSS projects' mailing list archives, in order to capture communication habits and social aspects of a community's organisational structure. Even if in the literature there are many empirical software engineering studies that use mailing list archives to achieve such goal (listed in Section 2.4), in order to verify the effectiveness and demonstrate the validity of our decision of using mailing lists to compute communication DSNs and avoid possible threats to validity, we investigated which communication channel was the most used in FLOSS communities to settle important project decisions. Thus, we were interested in the identification of the communication channel in which the decisional component was most prominent within Open Source development communities.

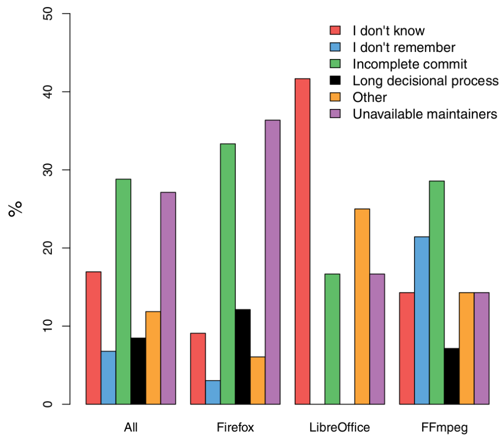
A detailed representation of retrieved responses related to this analysed aspect can be seen in Figure 6.2c. Our detections revealed that mailing lists are yet the main communicational channels used when it comes to decision making in FLOSS communities, as it was specified by 40.68% of the total questionnaire respondents, while on the basis of individual reference projects the minimum percentage was achieved by Firefox community with 27.27% of its total interviewees, followed by the 41.67% of LibreOffice community and FFmpeg community registered an outstanding agreement of 71.43%. Given such results *we verified that it is reasonable to consider mailing lists as the main communication channel through which extract FLOSS developers formal relationships and development communications.* Mailing lists are then the best candidates to be mined with the scope of generating communication Developer Social Networks and we provided a real world confirmation for our assumption, which is proved to be valid.

We further investigated if FLOSS developers perceived specific socio-technical issues that can potentially generate Social Debt within an Open Source Software development community and to what extent social related issues were perceived as impediments with respect to every software development cycle phase.

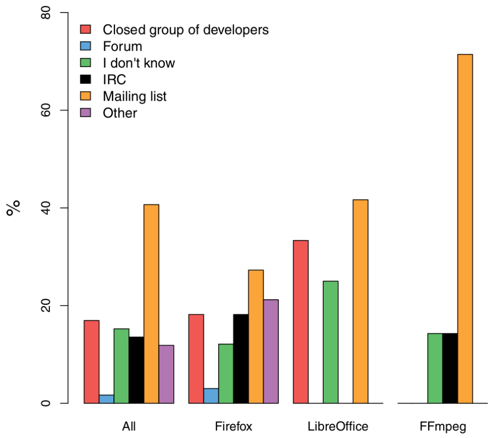
We asked to respondents to specify the reason of their longest wait before one of their software contribution to the project's source code was considered by maintain-



(a) Time waste motivation



(b) Contribution approval delay motivation



(c) Important decisions channel

Figure 6.2: Survey results (part two)

ers; responses are represented in Figure 6.2b. With respect to this specific software development phase, as expected technical related issues were predominant and identified by the 28.81% of the total responses. We also detected the importance of social related issues, represented in the questionnaire through the unavailable projects maintainers option, that globally constituted the 27.12% of the total responses. *This important result supported our research goal and provides the evidence that social related issues captured by Community Smells can cause actual delays and thus they do generate Social Debt.* Considering in detail all the reference projects: LibreOffice constituted the perfect representation of this equivalent relevance of technical and social issues since both of them were selected by the 16.67% of the project respondents, FFmpeg issues related to the maintainer unavailability were not considered predominant (14.29%), while social related issues were considered the causes of most of the delays in software contribution evaluation activity within the Firefox community (36.36%), for which they were even more relevant than the technical ones (33.33%). If we consider long decisional processes as a social related issue, then the percentage of global developers who identified social aspects of FLOSS development as the cause of delays in the software contribution phase reaches the important value of 35.59%, constituting one third of the entire survey sample. It is necessary specify that, at global level, almost one respondent out of four (23.73%) stated that he or she did not remember or did not know the reason of his or her longest wait until a proposed contribution was considered by the project's maintainers. This uncertainty can be the symptom of unhealthy FLOSS development characteristics, which can eventually generate mistrust and project abandonment, thus generating indirectly Social Debt. This consideration can be an hint for future Social Debt studies within FLOSS environments.

One of the most relevant questions of the entire second section of the questionnaire was addressed to capture the major causes of time waste within any phase of a project development. Figure 6.2a shows such results at global level and at a finer grain level, subdividing the responses for every single reference project. Above all the possible answers, project disorganization appeared to be the main disadvantage that caused time waste within all the considered projects, reaching the 27.12% of global responses with a minimum value of 16.67% inside LibreOffice community and an impressive maximum value of 50% inside FFmpeg community. The following important motivations, identified as possible delay generators within a FLOSS development environment, received the same amount of responses from all the interviewees: bad software design and communication delays. In contrast to the general concept of communication delays, agreement delay was considered the most important reason of time wasting from 15.25% of total respondents and this percentage was stable in every reference project. Unreasonable software release deadlines and other non specified motivations registered a total of 10.17% each. If we consider together the delays due to agreement or generally related to communication needs, the considerable value of 33.89% of total responses is reached: these typologies of disadvantages imply inefficient communication and organisational structures or unhealthy habits, which are the basic assumptions at the foundation of Social Debt theory. Since the project organisation is the representation of the developer social structure, we can consider even project disorganisation motivation to FLOSS development communities delays as a social related issue, thus, a total of 61.01% of the global responses suggested that social related issues are the main causes of time waste within any phase of Open Source Software development process. *This result supports our research questions, guarantees and verifies the effective existence and presence of Social Debt in Open Source Software communities and its intrinsic nature of being ubiquitous and not limited to a specific development activity.*

6.4 Validity of Community Smells

The conducted survey was conceived with a set of questions with the purpose of collecting Open Source developer perceptions related to potential socio-technical issues, in order to verify the effectiveness and validity of Community Smells definitions and proposed identification patterns, proposed in Chapter 4.

Delays were considered the main reason of time waste within any project activity by the 33.89% of total respondents: while the 18.64% blamed communication related delays as the most impacting source of delay, the remaining 15.25% believed that the most debilitating typology of delay was caused by agreement activities.

Agreement delays were specified within all the three reference projects by an almost constant percentage of respondents and it was classified as the fourth most important time waste motivation, therefore *this finding verifies the existence of Radio Silence Community Smells within FLOSS development ecosystem*, since it is one of the explicitly defined side effects of such Community Smell. Communication delays, instead, *verify the existence of Black-cloud Effect, Prima-donnas Effect and, in a more relevant manner, Radio Silence Community Smells* because all their definitions are associated to the presence of unique communication and knowledge conveyors between different sub-communities; thus, communications directed outside a sub-community will be negatively impacted by all the three listed typologies of Community Smells, generating communication delays. Conversely to agreement delays, the percentage of respondents belonging to different communities who selected communication delays as the main cause of time waste in any community activity, was very fluctuating in reference projects: from the minimum value of 7.14% achieved by FFmpeg community to the maximum of 27.27% of Firefox community.

It was recognised by three out of four global respondents (75%) that the absence of a developer can stall some community activities. Therefore, this finding surely *verifies the validity of Black-cloud Effect, Prima-donnas Effect and Radio Silence Community Smells Community Smells*, as they are characterised by the presence of unique communication and knowledge conveyors between different sub-communities within Open Source Software communities. Radio Silence is probably the Community Smell that is most verified by this result, as its definition is completely based on the concept of unique boundary spanners and problems related to their absence or due to overload of their capacities. The level of agreement with respect to this analysed aspect was slightly variable within reference projects: from the minimum value of 58% of LibreOffice community to the maximum value of 86% of FFmpeg community.

One of the agreement statements proposed in the questionnaire had the target of capturing developer perceptions about a potentially high degree formality within their software development community. The global responses suggested that, generally speaking, every developer had a personal perception about the formality level of his or her community, as the number of agreement, disagreement and neutrality responses were equally distributed.

The perception of community's rules and its constituent structure can be considered as another aspect that can represent a triggering factor of Radio Silence Community Smell due to a high level of formality and complexity, as it can be related to how developers perceive and understand their community's organisational structure. Globally, FLOSS development community rules and structures were perceived as sound and clear by almost 60% of the total respondents. While LibreOffice and Firefox respondents exhibited a similar agreement distribution, FFmpeg project recorded the most negative record between all reference projects. The global percentage of disagreement was constituted by the 15% of total responses, but it was characterised by a high fluctuation level within the three reference projects: LibreOffice community did not record any disagreement responses, Firefox community reached a value of 12% and FFmpeg community was characterised by the negative record of 38% of disagreement responses. *Therefore, the potential presence of Radio Silence Community Smell within FLOSS communities is further demonstrated.*

One out of four of total respondents declared that he or she did assumptions during development activities due to unclear requirements or documentation; this developer tendency was stable in Firefox and FFmpeg communities, characterising the behaviour of the 21% of respondents of each project, while LibreOffice was the reference community with the highest rate of developers who did assumptions within their developers activities, reaching a total of 42% of its respondents. Potentially, assumptions performed by single developers can degenerate into autonomous design and architectural decisions which are not expected, documented and performed without the necessary authority or knowledge. Therefore, *Organisational Silo Effect, Missing Links and Black-cloud Effect Community Smells existence is verified by this* *result*, as it is one of the possible triggering factors of such Community Smells.

Within the proposed questionnaire, four different agreement sentences were present in order to capture developer perceptions about Community Smells related to the presence of sub-communities within a community and their behaviours: Black-cloud Effect, Prima-donnas Effect and Radio Silence. In order to verify the existence of previously listed Community Smells and support their identification patterns through the mean of sub-communities detection, as proposed in Chapter 4, it was necessary to verify that sub-communities actually exists and within FLOSS development and that community members perceive the existence of such Community Smells; this condition was verified by the questionnaire results since the 47% of total respondents indicated that their reference development community was constituted by different subgroups. From the results of the executed survey it was evident that, within Open Source Software communities, sub-communities tend to communicate with each other, as only the 12% of total respondents believed that different subgroups rarely communicate within a software development community. Therefore, this result may indicate an *exiguous number of occurrences of Black-cloud Effect and Prima-donnas Effect Community Smells*.

Moreover, 36% of total respondents was convinced that every sub-community is composed by members with a similar mindset and that they behave as a little stand-alone community inside the actual community, thus *the existence of Black-cloud Effect and Prima-donnas Effect Community Smells within FLOSS development communities is further verified*.

An additional hint that may indicate an exiguous number of occurrences of Black-cloud Effect and Prima-donnas Effect Community Smells within Open Source Software development communities derives from the 14% of total respondents that believed that different sub-communities can be antagonists, while the 51% of them denied the existence of such phenomenon. This result suggests that sub-communities tend to collaborate and not isolate themselves, thus *it is possible to assume that Black-cloud Effect and Prima-donnas Effect are quite rare in FLOSS development communities*.

6.5 Quality factors identification

The survey had a fundamental role even in the identification of some socio-technical quality factors considered important by Open Source Software developers and, thus, that had to be included within our Socio-technical Quality Framework proposed in Chapter 5.

A couple of results obtained within the third part of the questionnaire were very important to suggest the introduction of quality factors founded on the *identification of core and peripheral community members*. Therefore, every socio-technical quality metric that distinguish between peripheral community members and core community members (e.g. turnover) is motivated by the two following findings:

1. While 75% of total respondents believed that the absence of a developer can stall some community activities, if the absent developer is a core developer such result increased and reached the 80% of agreement level. It important to highlight that such result was due to a decrease of its disagreement level, that passed from the 8% to the 3%. In is interesting to notice that FFmpeg was the only reference project that exhibited a counter-current behaviour with respect to the others reference projects, as its community members believed that the absence of general community members was more problematic that the absence of core developers, with an impressing margin of 7%;

2. Even if FLOSS communities are usually characterised by open, collaborative and welcoming development environment, the idea that they represent a democracy in which every community member has the same importance, is still an utopia and it cannot be considered a world-wide reality. Only 34% of total respondents believed that every opinion is equal in important project decisions, while almost one out of two respondents (49%) denied the existence of this equality and democratic value within Open Source Software community.

An additional contribution, obtained from the results of the questionnaire, to the definition of our Socio-technical Quality Framework suggested the importance of the i*dentification of community members supported by commercial companies* or who can be considered self-employed with respect to a software project. The inclusion of quality factors related to the identification of sponsored developers was motivated by the 63% of total respondents who specified their belief that developers sponsored by commercial companies increased the health of a project and because only the 8% of the global sample was in disagreement with such opinion.

The quality factors introduced in the out Socio-technical Quality Framework related to the identification of core developers are explained in details in Section 5.3, while the ones introduced due to the importance of developers sponsored by commercial companies within FLOSS development communities are described in Section 5.1. In this master thesis these two typologies of socio-technical quality factors were associated to the identification of a total of 19 different socio-technical quality metrics. Therefore, the conducted questionnaire, in conjunction to the study of software engineering literature, had a fundamental role in the definition of the proposed Socio-technical Quality Framework.

Chapter 7

Operationalising our Quality Framework: Codeface4Smells

This chapter presents Codeface4Smells, our software contribution to perform socio-technical analysis, compute quality factors defined in Chapter 5, identify and quantify Community Smells exploiting identification patterns defined in Chapter 4 and empirically detect relevant correlations between socio-technical quality factors and Community.

Codeface4Smells offers a lens to observe software development communities from a quality perspective and diagnose organisational issues in an automated tool-supported fashion. Therefore, it can be used for continuous Community Smells management and improvement.

Section 7.1 presents Codeface, the software project used as a development base to implement our tool. The architecture and inner workings of Codeface4Smells are explained in Section 7.2. Finally, details related to the operationalization of identification patterns of Community Smells and of socio-technical quality factors are presented respectively in Section 7.3 and Section 7.4.

7.1 Codeface

Codeface is an Open Source "framework and interactive web front-end for the social and technical analysis of software development projects" [68], which is capable to retrieve and analyse collaboration and communication relationships of a software development community using different software development artefacts (Version Control Systems and mailing lists). Codeface was created in 2010 by Wolfgang Mauerer and most of its developed is internally executed and sponsored by Siemens. It is written mainly using python and R and it is released under the GNU General Public License v2.0.

Codeface analysis results can be useful to learn more about an embedded software ecosystem and all the retrieved information about a software project may help to understand and exploit collaboration and communication patterns and characteristics, highlight development issues and assist maintainers in project control and management activities.

Codeface is composed by a variety of modules and components which cooperate and collaborate with each other, the most important are:

- An *interactive web front-end* to easily monitor and visualise analysed projects. It enables the possibility to assess retrieved information at different levels of details, from management summaries (communication, collaboration, construction, complexity) down to raw data, supporting direct comparisons of projects;
- An *analysis framework* with minimal configuration needs that can scale even to large software projects;
- An *holistic database* that allows complex insightful queries by coherently integrating all available data sources;
- A *framework* to execute quantitative analysis and objective classification of social aspects of software development.

Codeface implements an interactive web front-end which allows any user to dynamically modify its views to adapt them to every evaluation needs and personal preferences. Codeface web front-end provides three main levels of details: overviews, summaries and raw data. The web front-end of Codeface provides an overview with general information related to a considered project and a concise representation of the project's status with respect to its complexity, construction, communications and collaborations.

While the concise overview summarises the results of different categories evaluations, which are useful to formulate an idea about a project's strengths and weaknesses, the detailed summary enables to examine the status of one specific project area and see its related analysis. At a lower level, raw data allow to exam a project area in depth.

Some characterising features of Codeface are:

- Central contributors are detected using their social influence and not only static measures;
- Many available approaches to compute collaboration relationships from Version Control Systems;
- Text mining usage in communication analysis to detect most active discussion topics and discussion culture;

- Classification of active and passive contributors in the mailing lists;
- Different clustering algorithms used to detect closely interacting communities and their evolution over time;
- Range of traditional software engineering code metrics;
- Time series analysis (e.g. growth data, complexity behavior, communication volume) to estimate trends, determine regularity and self-consistency;
- Page-rank analysis to evaluate developers influence, thus not relying just to total commit count;
- Community detection;
- Time zone analysis.

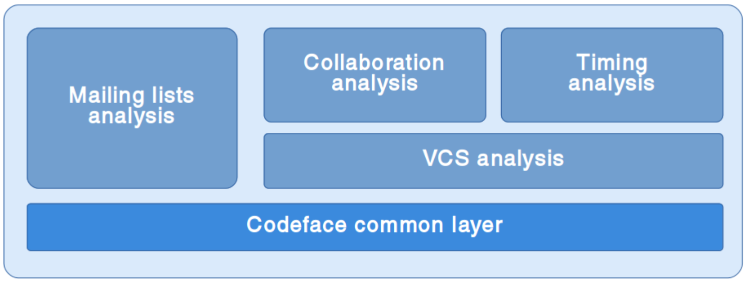


Figure 7.1: Architecture of Codeface

The software architecture of Codeface, as showed in Figure 7.1, is constituted by the following layers:

1. *Common layer*: constituted by common routines, SQL abstraction, projects configuration and logging functionalities;
2. *Mailing lists analysis*: performs communication analysis;
3. *Version Control Systems analysis*:
    - *Timing analysis*: computes evolutionary project metrics (e.g. number of files);
    - *Collaboration analysis*: creates the developer network and execute cluster analysis.

Codeface can generate a collaboration Developer Social Network using the following collaboration detection techniques:

- *Proximity analysis*: developers are linked when their commits are in close proximity (e.g. same file and nearby line numbers);

- *Committer2Author analysis*: developers are linked with directional relationships extracted from the committer and the author meta-data of every commit;
- *Tag analysis*: developers are linked with directional relationships extracted from tags placed in every commit (e.g Signed-Off, Acked-by, Reviewed-by);
- *Function analysis*: developers are linked considering proximity of their commits within the same function of a file;
- *Feature analysis*: developers are linked consider proximity of their commits within the same feature (unlike functions, features are split across different files).

In our work we used Codeface exclusively to exploit its capabilities of extraction and generation of Developer Social Networks from communication and collaboration software development artefacts of FLOSS development communities. We extended its functionalities introducing a brand new socio-technical network analysis layer, constituted by a Socio-technical Quality Framework and a Community Smells detection and quantification mechanism.

7.2 Architecture of Codeface4Smells

Codeface4Smells is build on top of Codeface's software source code, thus it can be considered as an extension of Codeface, and it has the purpose of *introducing software enhancements and new socio-technical analysis and Community Smells detection capabilities*. Considering the architecture of Codeface with the addition of Codeface4Smells extension, represented in Figure 7.2, it is possible to conclude that Codeface4Smells constitute a bran new software layer, which resides on top of all the already present Codeface architecture layers, and that its socio-technical analysis capabilities are enabled by Codeface's communication and collaboration analysis outputs.

Codeface4Smells main contributions are:

1. **Creation of a global Developer Social Network**, generated from the combination of communication and collaboration Developer Social Networks. This innovation implies the possibility of a finer-grained socio-technical analysis of a software project development ecosystem, by considering at the same time both the technical and social relationships between community members and not as two isolated and independent parts of the development process, as previously conceived in Codeface;
2. **Introduction of automatic ranges detection** to analyse at most the last three years of a project considering three months windows;

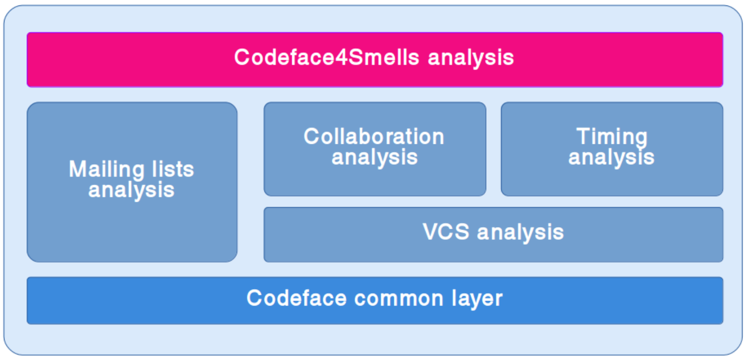


Figure 7.2: Architecture of Codeface with Codeface4Smells extension

3. **Identification of sub-communities** within the global, communication and collaboration Developer Social Networks. Codeface provided the identification of sub-communities within its collaboration analysis but it was necessary to re-implement this functionality with a different approach in order to support this feature outside the collaboration analysis, extend it to other DSN typologies and consider the undirected nature of generated networks;

4. **Identification of core developers** within the global, communication and collaboration Developer Social Networks. This feature was already present in Codeface but it was limited to collaboration analysis and thus it was necessary to modify its implementation to support our socio-technical analysis implementation and extend it to to the other DSN typologies and to the undirected nature of generated networks;

5. **Identification of developers supported by commercial companies** or self-employed within a project development community;

6. **Introduction of our Socio-technical Quality Framework** to gather more information about socio-technical characteristics of a software project development environment and its community. It is presented in details in Section 7.4;

7. **Detection and quantification of Community Smells** which may indicate the presence of Social Debt within the project development. It is presented in details in Section 7.3;

8. **Correlation analysis** (Pearson and Spearman) execution between socio-technical quality factors and the occurrence of Community Smells;

9. **Reports and graphs** generation to simplify access to socio-technical and Community Smells analysis results.

The Developer Social Networks produced by Codeface's communication and collaboration analysis are the basic building blocks on which Codeface4Smells is implemented, thus in order to be able to execute Codeface4Smells it is necessary to execute these two typologies of analysis in advance.

In Appendix B it is explained how to install, set up and execute a complete Codeface4Smells analysis from scratch.

Codeface4Smells is build on top of Codeface common layers (Section 7.1) and it is constituted by some functionalities that consider a software project in analysis at a global level and some other functionalities which are executed for every project's range specified within the configuration file. *Ranges are intervals between different source code snapshots* that are expressed in a project configuration file: manually, by specifying release versions or commit hashes, or automatically computed by Codeface4Smells, specifying "3months" analysis within the configuration file before executing Codeface4Smells analysis (consult Appendix B for further information). While range analysis is executed for each considered snapshot of a project, global analysis resides at a higher level of generality as it considers every information extracted from every executed range analysis as a whole.

While Figure 7.2 represents the general architecture layout constituted by Codeface and Codeface4Smells extension layer, which enables a deeper socio-technical analysis and resides on top of every analysis layer previously implemented in Codeface, a zoom in and detailed view of Codeface4Smells architecture is represented in Figure 7.3 and this and the following sections will provide a description of all its constituents components.

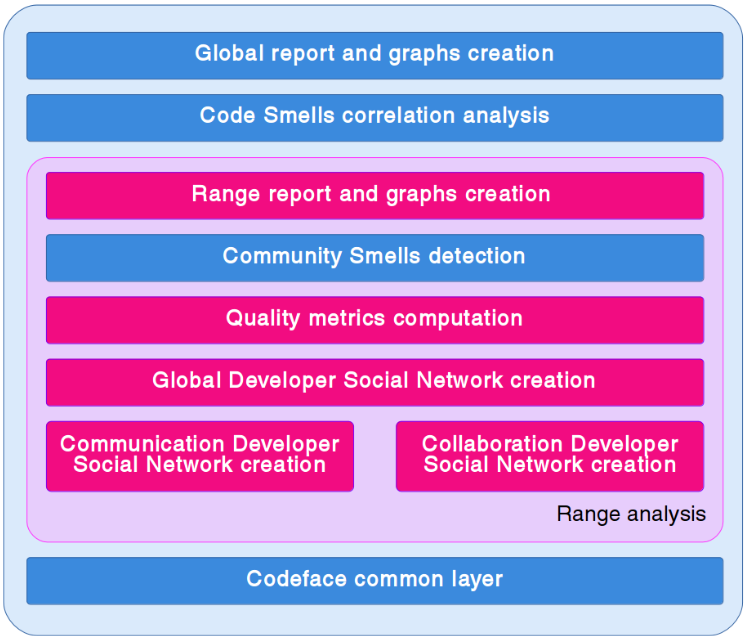


Figure 7.3: Architecture of Codeface4Smells

Codeface4Smells analysis is enabled, in addition to Codeface's collaboration and communication analysis as previously explained, by the general Codeface utility layer. This fundamental layer of Codeface is composed by low level functions capable of providing basic but useful functionalities as database connections, communication and collaboration specific database query executions and information retrieving of a project's configuration. We modified this low level layer of Codeface, that is shared within every typology of analysis, in order to allow the selection of socio-technical analysis performed by Codeface4Smells, compile produced LaTeX output to obtain PDF report files and enhance query and computational tasks capable of performing different range analysis functionalities.

Whenever Codeface4Smells analysis is started, it uses Codeface's common layer in order to extract the configuration of the project in analysis, its list of snapshots (ranges) to consider and the directory that contains Codeface's communication and collaboration analysis outputs. As soon as the list of ranges that have to be analysed is known, each range is analysed separately but in succession, because in order to compute some socio-technical quality factors (e.g. turnover) or some Community Smells (e.g. Black-cloud Effect) it is necessary to consider historical information extracted from previously analysed ranges. The sequence of steps that compose a single range analysis is summarised within the magenta coloured box of Figure 7.3 and its phases will be explained in the following.

A specific range analysis starts with the *generation of the communication Developer Social Network*. The date of the start and the end of the range to be considered are used to query Codeface's database in order to retrieve the edge list of every communication that occurred within the considered time lapse. Every element of such edge list is composed by a triplet of values that summarises the identifiers of two community members and the number of communications that were initiated by the first community member toward the second community member within the analysed range. If the communication channel was never used within the considered range we terminate Codeface4Smells analysis since it is impossible to achieve a meaningful socio-technical analysis without such fundamental data. The communication Developer Social Network is defined as an *undirected graph*, in order to achieve a higher generalisation level of a project's organisational structure and thus the number of e-mail messages exchanged between two community members is maintained as the weight of the link that connects the two community members within the communication DSN. The generated communication Developer Social Network undergoes a further processing phase that has the target of simplifying the previously generated network, by removing multiple edges and loops and substituting such identified multiple edges with one unique edge characterised by the weight resulting from the sum of all weights of deleted edges.

The data of a specific range necessary to *create the communication Developer*

Social Network are extracted from the adjacency matrix file generated by Codeface's collaboration analysis. Such text file is a representation of a square matrix in which each row/column represents a community member who contributed to the source code of the project in analysis within the considered software snapshot and the intersection element between a row and a column of the matrix represents the number of collaborations between the two developers identified by the relative row/column identifiers. Codeface4Smells exploits the retrieved adjacency matrix to generate the collaboration Developer Social Network of the analysed range, using the number of collaborations as edge weights, nullifying the diagonal of the matrix because the number of developer self-collaborations are not considered interesting within our socio-technical analysis and considering an undirected network approach, in order to achieve a higher generalisation level of a project's collaboration habits and behaviours. Similarly to the communication Developer Social Network creation process, the generated collaboration DSN undergoes to a simplifying functionality that removes multiple edges and loops and substitutes such identified multiple edges with one unique edge characterised by the weight resulting from the sum of all the weights of deleted edges.

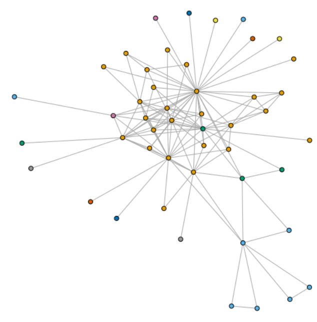
The availability of communication and collaboration Developer Social Networks enable the further *generation of the global Developer Social Network*, that is constituted by merging together the two previously computed networks. Such functionality had to be implemented from scratch in Codeface4Smells and the fusion process is performed using the communication DSN as a base, since usually it is constituted by the greater number of community members, to which are added the project's developers who were not present in the communication channel and, once every community member present in any considered development phase is present within the global DSN, an edge fusion functionality is executed in order to characterise the weight of every edge between two community members of the global DSN as the sum of communication and collaboration edges between the same two considered community members within the communication and collaboration Developer Social Networks. Once again, the generated global Developer Social Network undergoes to a simplification functionality in order to remove multiple edges and loops and it substitutes such identified multiple edges with one unique edge characterised by the weight resulting from the sum of all the weights of deleted edges. Therefore, the weights of edges belonging to the global DSN correspond to the sum of total number of collaborations and total number of communications between two community members.

Since some quality factors and Community Smells require the identification of sub-communities within the three generated Developer Social Networks, Codeface4Smells identifies the cluster (sub-community) of membership of every community member within all the three typologies of considered DSNs. Every community detection algorithm is characterised by a different approach and performance, therefore in 2014 Sousa et al. [69] conducted a study in which different community detection algorithms, implemented and available within the igraph suite (used by Codeface4Smells), were deeply analysed and ranked according to their performances in different possible scenarios. The results obtained by de Sousa et al. demonstrated that *Walk-trap algorithm outperforms every other considered community detection algorithm*, followed by multi-level algorithm and highlighted how Spin-glass approach is highly influenced by the number of nodes that constitute a network and its fragmentation degree, thus if network's modularity is low or the number of nodes increases, its performance degrades. Codeface's collaboration analysis provided a community detection functionality based on Spin-glass but in order to harmonise sub-community identification functionality to all the typologies of DSNs considered by Codeface4Smells, support undirected Developer Social Networks with edges weights and given the results obtained by de Sousa et al. [69], *Codeface4Smells comes with a new feature that exploits Walk-trap detection algorithm* implemented within the igraph package of R language to detect sub-communities within every generated Developer Social Networks.

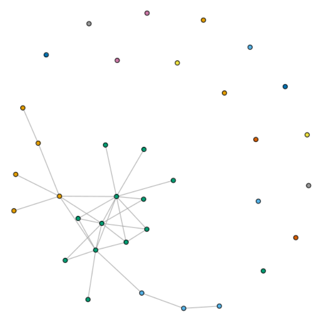
The analysis of a project's range proceeds with the computation of quality factors belonging to the Socio-technical Quality Framework defined in Chapter 5, explained in details in Section 7.4, and with the identification and quantification of Community Smells presented in Chapter 4, explained in details in Section 7.3.

The range analysis terminates with the creation of a report file that summarises, through the mean of graphs, the generated Developer Social Networks and Community Smells detected by Codeface4Smells. Figure 7.4 represents a real life example of a project's range analysis report file. Considering the report analysis of the proposed example, it is possible to see how communication and collaboration Developer Social Networks (Figures 7.4a and 7.4b) are merged together into a global DSN (Figure 7.4c) and, within the same images, it is possible to recognise community members belonging to different sub-communities, with respect to every DSN typology, considering the colour of every node. The other graphs present within the report file are related to Community Smells identified within the range in analysis. If Codeface4Smells did not discover any occurrence of a considered Community Smell then a message will be displayed, otherwise a visual representation of detected Community Smell occurrences will be displayed with the following characteristics, related to the Community Smell typology:

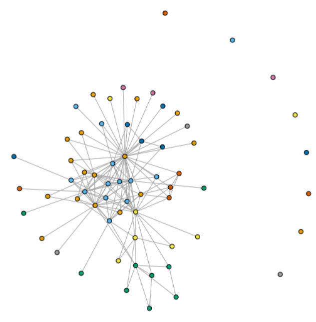
1. **Organisational Silo Effect** (Figure 7.4d): This graph represents a version of the collaboration Developer Social Network in which every couple of community members that constitute an identified Organisation Silo is highlighted and limited by colourful ellipses. Every developer involved in at least one Organisation Silo is coloured in red, while all the other developers are transparent;



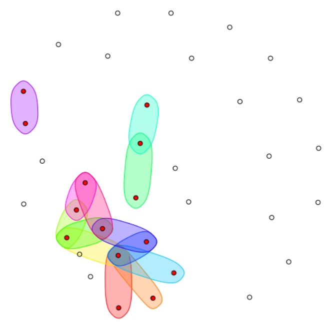
(a) Communication DSN



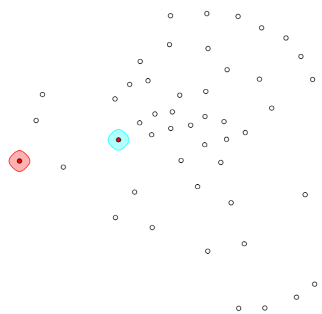
(b) Collaboration DSN



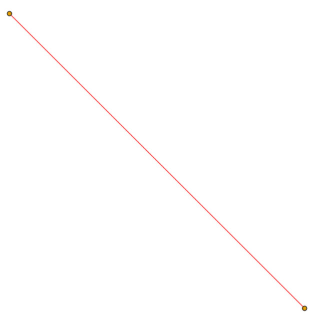
(c) Global DSN



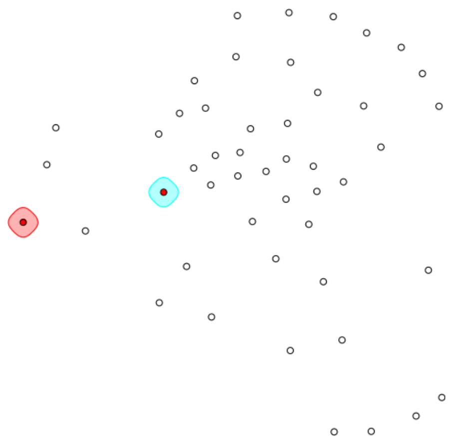
(d) Organisational Silo Effect



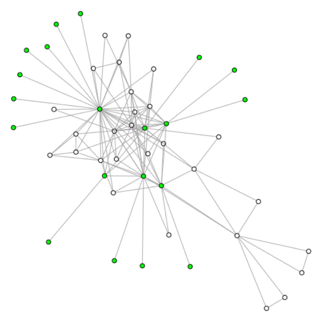
(e) Missing Links



(f) Black-cloud Effect



(g) Prima-donnas Effect



(h) Radio Silence

Figure 7.4: Example of range analysis report

2. **Missing Links** (Figure 7.4e): Same convention as Organisation Silo Effect;

3. **Black-cloud Effect** (Figure 7.4f): This graph represents all the communication links that were identified as occurrences of a Black-cloud Effect, therefore it shows the community members involved in at least one Black-cloud Effect as nodes coloured in yellow and they are connected through red edges representing the indicted Black-cloud links;

4. **Prima-donnas Effect** (Figure 7.4g): This graph represents a version of the communication Developer Social Network in which every sub-community involved in at least one Prima-donnas Effect is highlighted and limited by colourful ellipses. Every community member involved in at least one Prima-donnas Effect is coloured in red, while all the other community members are transparent;

5. **Radio Silence** (Figure 7.4h): This graph represents a version of the communication Developer Social Network in which every community member classified at least once as a unique knowledge broker toward another sub-community is coloured in green, while all the other community members are transparent.

Please notice that it is possible to enable the display of identifiers, names and e-mails of community members within every generated Developer Social Networks, but such functionality was disabled to preserve community member privacy, since in this master thesis we are not interested in evaluating single developers.

Range analysis and all its previously described constituent phases are executed for every range listed in the relative configuration file and, once all the specified ranges are analysed, Codeface4Smells execute additional global analysis phases, using the retrieved results of all analysed ranges: correlation analysis and global report.

In correlation analysis layer every identified Community Smell is considered with respect to every socio-technical quality factor and their correlations are computed, in order to *empirically identify if and how socio-technical quality frameworks are related to a specific Community Smell.* Correlation analysis takes in input two sets of data (measurements related to a specific Community Smell and to a specific socio-technical quality factor), elaborates them and returns a correlation coefficient that quantifies the extent to which the two variables tend to change together, describing both the strength, the direction and the p-value of the relationship. Within this analysis Codeface4Smells performs two two correlation analysis techniques [70]:

- **Pearson product moment correlation** evaluates linear relationships within two continuous variables, thus it explores the condition in which a change in one of the two analysed variables is associated with a proportional change in the other variable;

- **Spearman rank-order correlation** is a nonparametric (distribution-free) rank statistic proposed by Charles Spearman that evaluates monotonic relationships within two continuous or ordinal variables, thus it explores situations in which two variables tend to change in conjunction but not necessarily at a constant rate, without making any assumptions about the frequency distribution of the variables. Usually Spearman's rank correlation coefficient is considered whenever the data distribution makes Pearson's correlation coefficient undesirable or misleading.

Codeface4Smells analysis terminates with the creation of report files that recap all generated data from the execution of every Codeface4Smells socio-technical analysis phase. Real world global report examples of the three reference projects can be consulted in Appendix C. Moreover, Codeface4Smells generates comma-separated values (CSV) files of every analysed aspect of a software development community, in order to provide easy access to data and facilitate further data analysis. For each analysed project such report files contain the following information:

- Measurements of socio-technical quality factors and Community Smells of every analysed range;
- Pearson's correlation coefficient and related p-value between every socio-technical quality metric and Community Smells;
- Spearman's correlation coefficient and related p-value between every socio-technical quality metric and Community Smells;
- Graphs representing trends of the main socio-technical quality metrics.

7.3 Operationalisation of Community Smells

This section explains how identification patterns of Community Smells, proposed in Chapter 4, are operationalised in CodeFace4Smells. The general functionality of all the operationalised identification patterns of Community Smells is represented in Figure Figure 7.5 on page 97, where the operationalised identification pattern is represented by the blue box and, in order to identify and quantify the number of occurrences of its specific considered Community Smell, it needs as input the collaboration DSN, the communication DSN and the list of sub-communities present within the communication DSN. Actually, not all the operationalised identification patterns require all the information reported in the figure, as some of them do not need information related to sub-communities (e.g., Missing Links) and others do not need the collaboration DSN (e.g., Radio Silence).

The following set of Community Smells is computed for every range specified in the configuration file of a project and obtained measurements are summarised in the

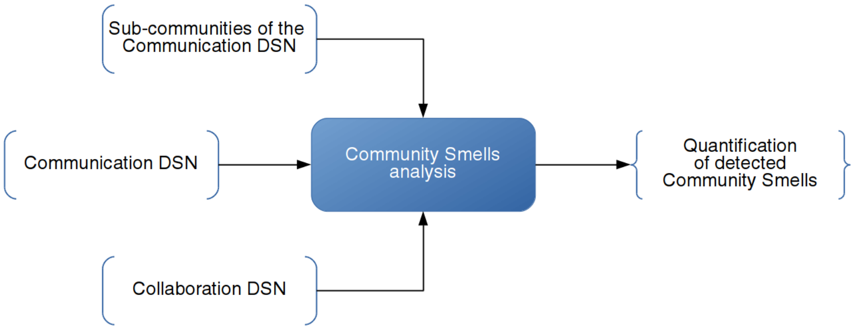


Figure 7.5: General functioning of the identification of Community Smells

socio-technical analysis report generated at the end of the global analysis performed by Codeface4Smells.

Organisational Silo Effect Community Smell measures the number of collaboration links characterised by the absence of at least one of the two developers, constituting the link, within the communication channel; therefore, the identification of such Community Smell requires as input the collaboration and communication Developer Social Networks. The list of non-communicative developers is obtained performing the complement of the collaboration DSN with respect to the communication DSN and appropriate handling functions are implemented to consider a collaboration between two non-communicative developers only once. When every collaboration belonging to non-communicative developers are counted, the number of total collaboration links in which one or both of the considered developers are absent in the communication DSN is returned as the measurement that characterises the Organisational Silo Effect Community Smell. The R implementation of this Community Smell in Codeface4Smells is proposed in Algorithm 7.1.

Algorithm 7.1 Operationalisation of Organisational Silo Effect identification pattern

```
community.smell.organisational.silo <- function (mail.graph, code.graph) {
  ## discover develpers not present in the communication DSN
  non.communicative.ids <- setdiff(V(code.graph)$id, V(mail.graph)$id)
  silos <- list()
  ## for each non communicative developer, save his collaborations
  for (vert in non.communicative.ids) {
    for (collab in neighbors(code.graph, V(code.graph)[V(code.graph)$id == vert])$
        id) {
      ## if both are non-communicative count the collaboration only once
      ## due to the undirected nature of the graph
      if ((collab %in% non.communicative.ids) & (collab < vert)) {
        next()
      }
      ## organisational silo smell detected
      silos[[length(silos) + 1]] <- c(vert, collab)
    }
  }
  return(silos)
}
```

Missing Links Community Smell measures the number of collaboration links that do not have a communication counterpart, therefore the identification of such Community Smell requires as input the collaboration and communication Developer Social Networks. The identification process considers every developer within the collaboration DSN and detects every collaboration link in which the two collaborating developers are present within the communication DSN but they are not connected through an edge in the communication Developer Social Network. The presence or absence of a communication link between two collaborating developers is obtained checking the presence of the other developer in the list of neighbors of the developer considered within the collaboration DSN. This initial identification phase does not consider collaborations in which one or both of implicated developers are not present in the communication channel due to optimisation reasons later explained and it applies appropriate handling functions within the collaboration links analysis in order to consider a Missing Links detection only once, due to the undirected nature of the collaboration Developer Social Network. The information about Missing Links related to non-communicative developers can be extracted from precomputed Organisational Silo Effect Community Smell, that can be passed to this Community Smell identification function as a parameter or otherwise it will be computed directly. This optimisation is possible due to the fact that, as explained in Section 4.1, Organisational Silo Effect is contained in Missing Links Community Smell. This Community Smell operationalisation returns the list of edges that were classified as Missing Links and not the number of them, therefore to obtain the measurement of Missing Links Community Smell it is necessary to count the number of elements returned by this identification process. The R implementation of this Community Smell in Codeface4Smells is proposed in Algorithm 7.2.

Black-cloud Effect Community Smell measures the number of communication links that identify unique communication points toward other sub-communities, therefore the identification of such Community Smell requires as input the communication Developer Social Network and the subdivision of its community members in different sub-communities. It is important to notice that Black-cloud Effect is a temporal related Community Smell, since it needs to consider historical information in order to identify actual Black-cloud Effect occurrences. Within every range analysis every sub-community of the communication DSN is considered once at the time and the number of outgoing edges from every sub-community are counted. If the number of outgoing communication edges is exactly one, then a “potential” Black-cloud Effect Community Smell is detected. This Community Smell identification function returns the list of communication edges classified as “potential” Black-cloud Effect Community Smell and, in order to be classified as actual Black-cloud Effect Community Smell within a range analysis, a “potential” Black-cloud Effect has to be present in the list of “potential” Black-cloud Effect of the previously analysed

Algorithm 7.2 Operationalisation of Missing Links identification pattern

```
community.smell.missing.links <- function (mail.graph, code.graph, precomputed.
    silo=NA) {
  missing <- list()
  for (vert in V(code.graph)$id) {
    if (!(vert %in% V(mail.graph)$id)) {
      next() # the case of one dev not present in the mailing list is handled
          later
    }
    for (coll in neighbors(code.graph, V(code.graph)[V(code.graph)$id == vert])$id
        ) {
      if (coll > vert) {
        next() # avoid to check twice a graph due to its undirected nature
      }
      if (!(coll %in% V(mail.graph)$id)) {
        next() # the case of one dev not present in the mailing list is handled
            later
      }
      ## if a missing communication link is found, it is saved
      if (!(coll %in% neighbors(mail.graph, V(mail.graph)[V(mail.graph)$id == vert
          ])$id)) {
        missing[[length(missing) + 1]] <- c(vert, coll)
      }
    }
  }

  ## if no precoumputed organisational silo, we are done
  if (length(precomputed.silo) == 0){
    return(missing)
  }

  ## If organisational silo is not pre-computed, calculate it
  if (is.na(precomputed.silo)){
    precomputed.silo <- community.smell.organisational.silo(mail.graph, code.graph
        )
  }
  ## Add the missing links due to developers absence in the mailing lists
  for (edge in precomputed.silo) {
    missing[[length(missing) + 1]] <- edge
  }

  return(missing)
}
```

Algorithm 7.3 Operationalisation of Black-cloud Effect identification pattern

```
community.smell.potential.black.cloud <- function (mail.graph, clusters) {
  black.links <- list()
  memships <- membership(clusters)
  ## For every sub-community check how many edges connect it to another
  ## sub-community. If there is just one extra-cluster edge, we have
  ## a potential black cloud
  for (clust in 1:length(clusters)) {
    extra.clust.links <- list()
    for (vert in V(mail.graph)[memships == clust]$id) {
      for (neigh in neighbors(mail.graph, V(mail.graph)[V(mail.graph)$id == vert])$id) {
        if (memships[V(mail.graph)[V(mail.graph)$id == neigh]] != clust) {
          extra.clust.links[[length(extra.clust.links) + 1]] <- c(vert, neigh)
        }
      }
    }
    if (length(extra.clust.links) == 1) {
      # Potential black cloud smell detected
      black.links[[length(black.links) + 1]] <- extra.clust.links[[1]]
    }
  }

  return(black.links)
}
```

range. Therefore, the measurement of Black-cloud Effect Community Smell is the number of communication links resulting from the intersection of the list of "potential" Black-cloud Effect of the actual range in analysis with the list of "potential" Black-cloud Effect of the previously analysed range, thus the Black-cloud Effect associated to the first analysed range will always be zero. The R implementation of "*potential*" Black-cloud Effect Community Smell in Codeface4Smells is proposed in Algorithm 7.3.

Prima-donnas Effect Community Smell measures the number of communication links that identify unique communication points toward other sub-communities in a situation in which two analysed sub-communities can be considered collaborating within the software source code development, therefore the identification of such Community Smell requires as input the collaboration and communication Developer Social Networks, the subdivision of community members belonging to the communication DSN into different sub-communities and the threshold needed to consider two distinct sub-communities as collaborating. *The default value of the collaboration threshold is setted to the 20% of total possible collaborations.* As explained in Section 4.3, the identification process of communication links that are possibly involved in a Prima-donnas Effect is the same as the "potential" Black-cloud Effect, thus in order to enable a computational optimisation it is possible to specify a precomputed list of "potential" Black-cloud Effect Community Smell, otherwise the related Community Smell identification function will be invoked. Every communication link present in the "potential" Black-cloud Effect list is considered and the number of collaborations between the two different sub-communities identified by a "potential"

Black-cloud Effect is computed. Then, the number of total possible collaborations is computed multiplying the numbers of community members constituting the two sub-communities and if the percentage of actual inter-collaborations, thus the result of the number of collaborations between the two sub-communities over the total number of possible collaborations, is greater than the given threshold then the two sub-communities are considered collaborating within the software development activity and a Prima-donnas Effect is effectively detected. Therefore, Prima-donnas Effect identification function returns the list of communication links of the communication DSN that identify the occurrence of such Community Smell and in order to obtain the related measurement it is necessary to count the number of elements that constitute the returned list. The R implementation of this Community Smell in Codeface4Smells is proposed in Algorithm 7.4.

Radio Silence Community Smell measures the number of unique knowledge brokers toward different sub-communities, therefore the identification of such Community Smell requires as input the communication Developer Social Network and the subdivision of its community members into different sub-communities. The identification process considers one by one every sub-community of the communication Developer Social Network and considers every outgoing communication link toward other sub-communities. If a sub-community is composed by only one community member, he or she is considered a unique boundary spanner without further computations, otherwise the analysis continues considering two sub-communities at the time and, if one sub-community communicates with the other one through only one community member, him or her is identified as a knowledge broker and a Radio Silence Community Smell is detected. Therefore, Radio Silence identification function returns the list of unique knowledge brokers within the sub-communities belonging to the communication Developer Social Network and in order to compute its associate measurement it is necessary to count the number of elements that constitute the returned list. The R implementation of this Community Smell in Codeface4Smells is proposed in Algorithm 7.5.

7.4 Socio-technical Quality Framework implementation

This section explains how the 40 socio-technical quality factors that constitute our Socio-technical Quality Framework proposed in Chapter 5 are implemented in CodeFace4Smells. The complete set of socio-technical quality factors is summarised in Table 5.1 and it is computed for every analysed range and obtained measurements are summarised in a socio-technical analysis report generated at the end of the global analysis performed by Codeface4Smells.

Community dimensions. Some community dimensions that consider the number of developers and members who are involved within the software project

Algorithm 7.4 Operationalisation of Prima-donnas Effect identification pattern

```
community.smell.primadonnas <- function (mail.graph, clusters, code.graph,
    collaboration=0.2, precomputed.black=NA) {
  primadonnas <- list()
  memships <- membership(clusters)
  comms <- communities(clusters)
  ## For every potential black-cloud, check collaborations of involved sub-
      communities;
  ## if it is greater than the threshold, we have two prima-donnas

  ## if no potential black-cloud, we are done
  if (length(precomputed.black) == 0){
    return(primadonnas)
  }

  if (is.na(precomputed.black)) {
    ## If potential black-cloud is not pre-computed, calculate it
    precomputed.black <- community.smell.potential.black.cloud(mail.graph,
        clusters)
  }
  for (black.link in precomputed.black) {
    sub.comm.connections <- 0
    ## retrieve cluster identifier of the two sub-communities
    id.clust1 <- memships[V(mail.graph)[V(mail.graph)$id == black.link[1]]]
    id.clust2 <- memships[V(mail.graph)[V(mail.graph)$id == black.link[2]]]
    ## count inter-collaborations of the two sub-communities
    for (dev.clust1 in V(mail.graph)[memships == id.clust1]$id) {
      if (!(dev.clust1 %in% V(code.graph)$id)) {
        next() # ignore devs present only in the communication graph
      }
      for (dev.clust2 in V(mail.graph)[memships == id.clust2]$id) {
        if (!(dev.clust2 %in% V(code.graph)$id)) {
          next() # ignore devs present only in the communication graph
        }
        if (dev.clust1 %in% neighbors(mail.graph, V(mail.graph)[V(mail.graph)$id
            == dev.clust2])$id) {
          sub.comm.connections <- sub.comm.connections + 1
        }
      }
      ## If the fraction of present collaborations over the total possible
          collaborations
      ## (Number of devs of clust1 * Number of devs of clust2) is greater than
      ## the given threshold then we have two prima-donnas
      tot.possible.collaborations <- length(comms[[id.clust1]]) * length(comms[[id
          .clust2]])
      if ((sub.comm.connections / tot.possible.collaborations) > collaboration) {
        ## prima-donnas effect detected
        primadonnas[[length(primadonnas) + 1]] <- c(id.clust1, id.clust2)
      }
    }
  }

  return(primadonnas)
}
```

Algorithm 7.5 Operationalisation of Radio Silence identification pattern

```
community.smell.radio.silence <- function (mail.graph, clusters) {
  brockers <- c()
  memships <- membership(clusters)
  ## consider every communication outside each cluster and if there is just one
  ## communication edge from a sub-community toward another one, we have a
  ## radio silence smell (unique boundary spanner)
  for (clust in 1:length(clusters)) {
    ## If a cluster has only one dev, he is an unique boundary spanner
    if (length(V(mail.graph)[memships == clust]$id) == 1) {
      brockers[length(brockers) + 1] <- V(mail.graph)[memships == clust]$id
      next()
    }
    extra.clust.links <- list()
    for (vert in V(mail.graph)[memships == clust]$id) {
      for (neigh in neighbors(mail.graph, V(mail.graph)[V(mail.graph)$id == vert])
          ) {
        ## Note: neigh is the local graph vertex id, not the developer id
        if (clust != memships[neigh]) {
          ## for each outgoing edge, save the cluster developer id and the
               destination
          ## sub-community id
          extra.clust.links[[length(extra.clust.links) + 1]] <- c(vert, memships[
               neigh])
        }
      }
    }
    ## for each outgoing edge, substitute destination vertex with its community
    if (length(extra.clust.links) > 0) {
      ## change format to enable comparisons
      extra.clust.links <- matrix(unlist(extra.clust.links), ncol=2, byrow=TRUE)
      for (outClust in unique(extra.clust.links[, 2])) {
        from.dev <- which(extra.clust.links[, 2] == outClust)
        if (length(from.dev) == 1) {
          ## radio silence community smell detected
          brockers[length(brockers) + 1] <- extra.clust.links[from.dev, 1]
        }
      }
    }
  }

  return(unique(brockers))
}
```

are retrieved considering the number of nodes that constitute the global, communication and collaboration Developer Social Networks. The total number of people involved in any possible and analysable way within the considered community in a specific range (**dev**) is obtained counting the number of nodes that constitute the Global DSN, while the number of members who are present in every aspect of a FLOSS project development (**ml.code.devs**) is obtained counting the number of nodes that constitute the intersection of the collaboration and communication Developer Social Networks. Finally, the number of developers who contribute to a project's source code development but do not participate in the communication channel (**code.only.devs**) and the number of members who participate to every activities of a community with the exception of the development phase (**ml.only.devs**) are retrieved respectively, subtracting the number of members present in every community phase (ml.code.devs) to the number of nodes that constitute the collaboration DSN and subtracting the number of members present in every community phase (ml.code.devs) to the number of nodes that constitute the communication DSN. Therefore, given the measured dimensional characteristics of a FLOSS community, it is possible to retrieve the dimensions of the communication or collaboration Developer Social Networks summing two different available metrics (ml.code.devs and ml.only.devs; ml.code.devs and code.only.devs); these aggregate dimensions were not considered within this master thesis study because we considered previously listed dimensions in their disaggregate and finer grain details level. Other insights that could help a researcher to understand how a community is structured and subdivided between communication and collaboration activities, can be the identification of how community members are spread into previously classified participation typologies. In order to capture such community's characteristics, it is calculated the percentage of people involved in code source development who communicate on the project's mailing list (**perc.ml.code.devs**), people present in the mailing list but that do not commit code contributions (**perc.ml.only.devs**) and developers that contribute to the community only by committing contributions to a project's source code (**perc.code.only.devs**).

Sponsored developers. The list of developers whom are supposed to be sponsored by commercial companies or whom can be considered self-sponsored developers with respect to the project in analysis, is retrieved applying an approach proposed by Riehle et al. [8], that considers information related to every commit pushed into a project's source code within the range in analysis. *A developer is associated with a sponsored status if at least the 95% of his or her commits are executed in working time*, from 9am to 5pm (local time) and from Mondays to Fridays. In their research Riehle et al. tried different threshold combinations in order to model different working habits present world-wide and concluded that the considered definition of working time provided an accurate approximation of the concept on a global working scale. The computed list of sponsored developers is used to compute the total number of sponsored developers within the window of analysis (**sponsored.devs**) and its related ratio with respect to the total number of developers whom contribute to a project's source code (**ratio.sponsored.devs**).

Core community members. The identification of core community members of the global, communication and collaboration Developer Social Networks is founded on a methodological approach proposed in 2016 by Joblin et al. [60]. Such identification methodology considers the degree centrality measure of every developer since, in their research, it was demonstrated that *core developers exhibit a higher global centrality in the developer network* and that they are likely to coordinate with other core developers, while peripheral developers are likely to coordinate with core developers. The method proposed by Joblin et al. uses social network analysis methodologies and it was proven to provide a better reflection of developer perception rather than count-based approaches (e.g. commit count, LOC count and mail count) [60]. Codeface was already able to classify as core or peripheral a developer belonging to the collaboration Developer Social Network, but this functionality was applied only within the collaboration analysis to developers whom contributed to a project's source code and it supported only directed graphs. Since the methodologies we proposed are based on undirected graph topologies, the preexistent solution to identify core and peripheral community members involved in a project was extended to support undirected graphs and the ability to apply such classification functionality to communication, collaboration and global Developer Social Networks. Once that core community members of the collaboration, communication and global Developer Social Networks are identified, it is possible to count the number of core members present in a community for each typology of analysed network (**core.global.devs**, **mail.global.devs**, **core.code.devs**). Since the list of developers sponsored by commercial companies or self-employed is retrieved from the collaboration DSN, it is possible to compute the number of sponsored developers whom are classified as core developers within the collaboration DSN (**sponsored.core.devs**), intersecting the two relative information and computing its related ratio (**ratio.sponsored.core**) with respect to the total number of core developers present in the collaboration Developer Social Network. A deeper understanding of how core members behave within different community activities can be achieved counting how many community members are characterised by core status both in the collaboration and in the communication Developer Social Networks (**ml.code.core.devs**), how many core members of the communication DSN are not core developers in the collaboration DSN (**mail.only.core.devs**) and how many core developers of the collaboration DSN are not core members in the communication DSN (**core.only.core.devs**). These three dimensional metrics related to core members distributions within considered Developer Social Networks are then used to compute the related ratio with respect to the total number of unique core members presents in the communication and in the collaboration Developer Social Networks and in the two different generated Developer Social Networks alone (**ratio.ml.code.core**, **ratio.mail.only.core**, **ratio.core.only.core**).

Truck number. The truck number represents the ratio of people that an activity can lose without entering into a stagnation phase. It does not exist a formal definition to calculate truck number (truck factor) [57] but within a FLOSS development community we can define as vital members associated with a core status with respect to each generated network typology. The number of core members present in the communication, collaboration and global developer Developer Social Networks previously obtained are then used to calculate the truck number relative to each network typology (**mail.truck**, **code.truck** and **global.truck**), using the following formula:

$$\textbf{Truck number} = \frac{\#peripheral\, members}{\#members} = \frac{(\#members - \#core\, members)}{\#members}$$

Turnover. Different typologies of turnover are calculated using the number of community members of the current and of the previously analysed ranges. Therefore, the turnover of the first range of an analysis will always be zero. The following formula is applied to compute turnover:

$$\textbf{Turnover} = \frac{NEEDLY}{(NEBY + NEEY)\,/\,2} * 100\%$$

Where:

- $NELDY$ is the number of members who left the project in the analysed range. It is obtained counting the number of members resulting from the intersection of members of previously analysed range and members of the actual range in analysis;
- $NEBY$ is the total number of members who constituted the community in the previously considered range;
- $NEEY$ is the total number of members who constitute the actual range in analysis.

Turnover metrics are characterised by a temporal nature because in order to be computed they need to have access to historical development analysis information. The following typologies of turnover are calculated using previously explained formula:

1. turnover of global members (**global.turnover**);
2. turnover of collaboration members (**code.turnover**);

3. turnover of global core members (**core.global.turnover**);
4. turnover of communication core members (**core.mail.turnover**);
5. turnover of collaboration core members (**core.code.turnover**).

Temporal and geographic dispersion. The temporal and geographic dispersion of a software project is calculated as the number of different and unique time-zones involved in every source code contribution to the source code within the range in analysis (**num.tz**). Codeface's collaboration analysis populates a database table with all retrievable details of commits and their relative author, hour, date and time-zone. Codeface4Smells comes with a functionality capable to query such database table in order to retrieve all the commits information related to the range in analysis, extract their associated time-zones and return the number of unique different time-zones that were involved within the project development in the considered range.

Socio-technical congruence. Socio-technical congruence (**st.congruence**) is measured as the number of development collaborations that do have a communication counterpart over the total number of collaboration links present in the collaboration Developer Social Network. Development collaborations that do have a communication counterpart are identified analysing one by one the collaboration links that connect different developers present in the collaboration Developer Social Network and check within the communication Developer Social Network if such developers are present and connected through a communication link. Therefore, socio-technical congruence can be computed using Missing Links Community Smell metric as follows:

$$\textbf{Socio-technical congruence} = \frac{\#collaborations - \#missingLinks}{\#collaborations}$$

Communicability. Each collaboration between two developers (A and B) in the software development network is considered as a possible source of architectural and design decision, therefore a developer is considered aware of a decision if he or she is strongly connected to at least one of the two developers whom generated the decision. In-communicability is related to every collaboration within the collaboration DSN and it is based on Tamburri et al.'s formulation [DEBT-2]:

$$\text{MAI} = DEM - DAM$$

$$\text{DEM} = \frac{\#collaborators\,of\,the\,two\,developers}{\#developers}$$

$$\text{DAM} = \frac{\#collaborators\,of\,the\,two\,developers\,whom\,communicate\,with\,them}{\#developers}$$

Therefore, global in-communicability can be defined as the mean MAI over the entire collaboration network. Communicability is a global indicator which consists in the mean of all local communicability measures, calculated for every collaboration between two developers within the collaboration Developer Social Network in the range in analysis. Communicability was preferred to in-communicability in order to simplify measurement comprehension, because in-communicability tend to be characterised by measurements that tend to zero. Communicability is computed as:

$$\textbf{Communicability} = 1 - \textit{incommunicability} = 1 - \frac{1}{n}\sum MAI$$

Social Network Analysis metrics. We used some Social Network Analysis methodologies available in R language to calculate the following factors:

- centrality of the global Developer Social Network computed considering closeness (**closeness.centr**), betweenness (**betweenness.centr**) and degree (**degree.centr**);
- density of the global Developer Social Network (**density**);
- modularity of the global Developer Social Network (**global.mod**), communication Developer Social Network (**mail.mod**) and collaboration Developer Social Network (**code.mod**).

Smelly developers and smelly quitters. Two socio-technical quality metrics related to the outcome of Community Smells identification analysis, specifically computed using the list of unique community members involved within at least one Community Smell, are the ratio of smelly developers and the ratio of smelly quitters. The ratio of smelly developers (**ratio.smelly.devs**) is the ratio of community members who are involved in at least one Community Smell with respect to the total number of unique members who constitute the global Developer Social Network. The ratio of smelly quitters (**ratio.smelly.quitters**) represents the ratio of developers who were involved in at least one Community Smell in the previously analysed range that left the software development community within the range in analysis. The ratio of smelly quitters is characterised by a temporal characteristic because in order to be computed it needs to have access to historical development analysis information, therefore a list of every community member and of smelly developers of the previously analysed range is kept and passed to the next range analysis.

Chapter 8

Evaluation

This chapter presents the evaluation of obtained results inherent to the presence, identification and characteristics of Social Debt and Community Smells within Open Source Software development communities, achieved through the discussed work.

Considering Pearson and Spearman correlation analysis results of the entire set of 60 analysed projects, it was possible to empirically identify quality factors that can be considered correlated to the occurrence of Community Smells. Within our analysis we considered a correlation as relevant if and only if its associated p-value was less than 0.05 and we imposed a minimum threshold of 20% to the number of relevant correlations found within all the projects or within the different dimensional categories, in order to determine which quality factors had to be considered associated to a variation of Community Smells.

We evaluated the results of Codeface4Smells analysis of all the 60 considered projects and empirically identified relevant correlations between some specific quality factors and the occurrence of Community Smells. Section 8.1 presents the evaluation related to the existence, occurrence and characteristics of considered Community Smells within Open Source Software projects. Moreover, we highlighted thresholds of such quality factors correlated to the insurgence of Community Smells, in order to provide precious hints to keep Community Smells under control. This evaluation is presented in Section 8.2 and it addresses the first Research Question (*RQ1*), its sub-questions and the second Research Question (*RQ2*). A further evaluation was performed considering together the results related to developer perceptions, obtained with the executed survey, and the results of the three reference projects obtained executing Codeface4Smells analysis, in order to verify if particular perceptions of developers can be used as indicators of a higher or lower number of Community Smells. This part of the evaluation process is presented in Section 8.3 and it addresses the third Research Question (*RQ3*). Section 8.4 summarises the answers to the research questions formulated in Chapter 3. Finally, Section 8.5 specifies potential threats that could affect the validity of results, evaluations and conclusions of in this study.

8.1 Occurrences of Community Smells

Considering the number of analysed projects affected by all the typologies of considered Community Smells, summarised in Table 8.1, it was possible to conclude that Radio Silence Community Smell was implicit in every FLOSS development community and even Organisational Silo Effect and Missing Links Community Smells can be considered omnipresent in Open Source projects, as they were detected in the 98% of analysed projects. Black-cloud Effect Community Smell occurred only in one out of two FLOSS projects and Prima-donnas Effect Community Smell was the rarest form of Community Smell within FLOSS communities as it was detected only in the 42% of analysed projects.

Dimensional category	Organisational Silo	Missing Links	Radio Silence	Black-cloud	Prima-donnas
<50	19	19	20	8	3
50-150	20	20	20	11	8
>150	20	20	20	12	14
ALL	59	59	60	31	25

Table 8.1: Number of analysed projects with Community Smells

Even if it was not possible to empirically correlate additional occurrences of Black-cloud Effect and Prima-donnas Effect Community Smells to the number of trimestral community members, as demonstrated in Section 8.2 for the other three typologies of Community Smells, it was possible to conclude that such Community Smells are more frequent in high dimensional communities:

- Black-cloud Effect Community Smell was detected in the 40% of analysed projects with less than 50 trimestral community members and in the 55-60% of analysed projects with more than 50 trimestral community members.
- Prima-donnas Effect Community Smell was detected in the 15% of analysed projects with less than 50 trimestral community members, in the 40% of analysed projects with more than 50 but less than 150 trimestral community members and in the 70% of analysed projects with less than 150 trimestral community members. Therefore, a project with more than 150 trimestral community members had more than the 467% of chances of incurring into Prima-donnas Effect Community Smell compared to a project with less than 50 trimestral community members.

Except for the correlation between Missing Links and Organisation Silo Effect Community Smells, that was expected since one includes the definition of the other, globally it was not found any correlation between different Community Smells. This finding is very important as it highlights that *Community Smells considered and implemented in this master thesis are independent and analyse different aspects and risks*, within a software development community.

Metric	Mean	Min	Max	St.deviation
Organisational Silo	164.46	0	10224	916.55
Missing Links	190.59	0	10327	940.65
Radio Silence	31.38	0	309	41.23
Black-cloud	0.18	0	7	0.65
Prima-donnas	0.24	0	8	0.85

(a) Trimestral variability of Community Smells (with outliers)

Metric	Mean	Min	Max	St.deviation
Organisational Silo	13.65	0	91	19
Missing Links	25.62	0	151	36.17
Radio Silence	21.52	0	96	23.59
Black-cloud	0	0	0	0
Prima-donnas	0	0	0	0

(b) Trimestral variability of Community Smells (without outliers)

Table 8.2: Trimestral variability of Community Smells

A general overview of the occurrences of Community Smells within FLOSS development communities, summarised by the results of analysed projects, can be obtained consulting the Table 8.2a, that for each Community Smell presents its mean, minimum, maximum and standard deviation trimestral values. Since the standard deviation of almost all Community Smells was very relevant, *it was necessary to apply a data cleaning process capable of removing outliers*, in order to achieve a higher data quality and a more realistic overview of occurrences of Community Smells within FLOSS development communities. Moreover, the elimination of outliers enabled a higher quality and more realistic prediction of growth trends within scatter plots, as used in Section 8.2. The data cleaning process was based on a R language standard function (boxplot.stats) that classifies as an outlier any data point that is located outside 1.5 times the interquartile range above the upper quartile and bellow the lower quartile. Table 8.2b represents retrieved information about analysed Community Smells without considering results classified as outliers.

Considering the survey results highlighted in Chapter 6, the fact that Black-cloud Effect and Prima-donnas Effect Community Smells were not correlated at global level to any metric belonging to the proposed Socio-technical Quality Framework, the fact that their trimestral average values were less then 1 considering every obtained result and that data cleaning process removed every occurrence of both typologies of Community Smells, it is possible to conclude that *Black-cloud Effect and Prima-donnas Effect Community Smells are not frequent nor particularly predominant in Open Source Software development communities.*

8.2 Quality factors correlated to Community Smells

This section addresses the first Research Question (***RQ1***), all its sub-questions and the second Research Question (***RQ2***). In the following we highlight important findings extracted from relevant correlations obtained considering all the 60 projects together and subdividing them into equal groups of similar dimension. For some findings it was possible to identify *quality thresholds* from *scatter plots,* built on top of trimestral results of all analysed projects. Such scatter plots were generated by eliminating outliers of occurrences of Community Smells, as previously explained in Section 8.1.

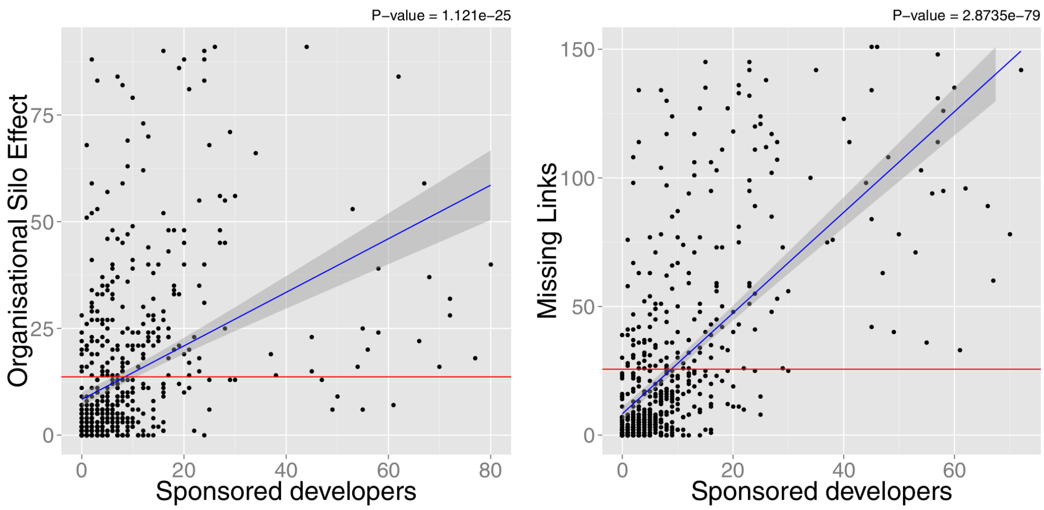


Figure 8.1: Scatter plots of sponsored developers

Community Smells increase with number of developers sponsored by commercial companies. We formulated the *research sub-question* ***RQ1a*** because the literature suggested that a higher number of paid developers (sponsored by commercial companies or self-employed) is correlated to a higher attractiveness and health of an Open Source project [8]. therefore, it was expected that a higher number of sponsored developers was associated to a lower number of Community Smells. Our results denied completely such hypothesis and, moreover, proved that a higher number of sponsored developers was actually associated to an increase of the number of occurrences of Organisational Silo Effect and Missing Links Community Smells in the 25% of analysed projects. Therefore, *research sub-question RQ1a was negatively answered and its anti-thesis was proven to be valid.* Considering the scatter plots represented in Figure 8.1 and their linear regressions, it was possible to identify the *quality threshold of 10 trimestral sponsored developers.* Over this threshold the amount of detected Community Smells were over the average values. Moreover, the role of sponsored developers in the generation of additional Community Smells assumed different importance with respect to the dimension of software projects:

- In projects with less than 50 trimestral community members, socio-technical quality factors related to the identification of sponsored developers were not correlated at all to the generation of additional Community Smells;
- In projects with more than 50 but less than 150 trimestral community members, the number of occurrences of Organisational Silo Effect and Missing Links Community Smells was not only positively correlated to the number of sponsored developers of a community, but even to the number of sponsored developers who were associated to the role of core developer. Furthermore, a higher ratio of sponsored core developers was associated to an increase of the number of occurrences of Organisational Silo Effect; this finding highlighted that in software communities with 50-150 trimestral members, *core sponsored developers tend to isolate themselves and not participate in a project's communication channel.*

Community Smells are not influenced by temporal and geographic dispersion. We formulated the *research sub-question **RQ1b*** because the literature suggested that temporal and geographic dispersion can generate socio-technical issues within a software development community or to its product outcome [21]. Therefore, it was expected that the number of time-zones involved in a software development community, used as an indicator of geographic and temporal dispersion, was positively correlated to an increment of detected Community Smells. This expected correlation was detected only in the 5% of analysed projects and thus *research sub-question RQ1b was negatively answered.* A possible explanation to this finding is that, nowadays, distributed development is the standard for software development environments and it implies only delays in communication activities between developers. Moreover, FLOSS is founded on the concept of Global Software Development and so there might exist implicit coordination or development mechanisms capable of avoiding the raise of additional Community Smells due to temporal and geographic dispersion.

The higher the socio-technical congruence, the lower the number of Community Smells. We formulated the *research sub-question **RQ1c*** because Cataldo et al. [5] stated that a higher socio-technical congruence was correlated to a higher software development performance and thus, considering the concept of socio-technical congruence as an indicator of the accordance between a organisational structure and its technical requirements, it was reasonable to suppose that a higher level of socio-technical congruence was associated to a lower number of occurrences of Community Smells within a software development community. Our results verified this hypothesis since in the 50% of analysed projects, an increment of socio-technical congruence was correlated to a decrease of the number of occurrences

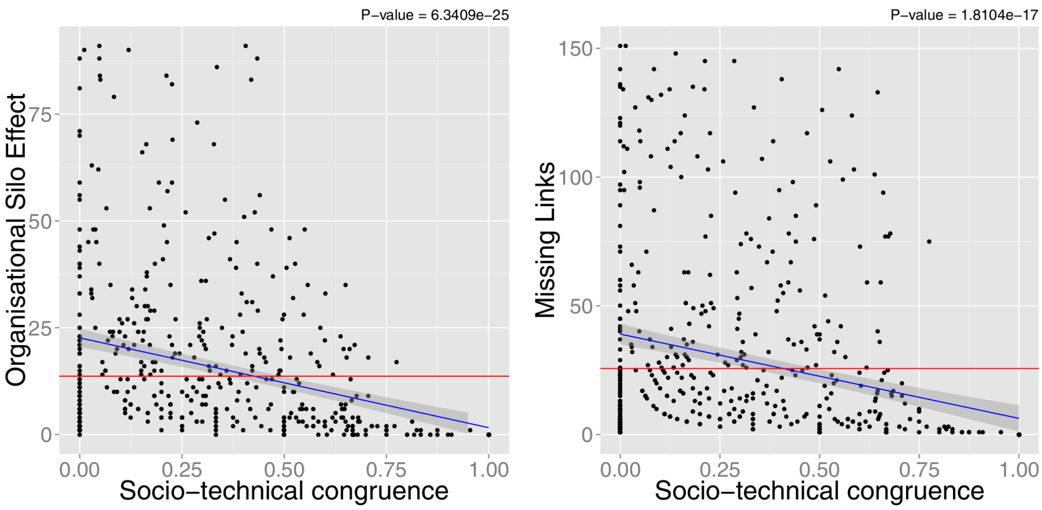


Figure 8.2: Scatter plots of socio-technical congruence

of Organisational Silo Effect and Missing Links Community Smells. Therefore, *research sub-question RQ1c was positively answered*. Considering the scatter plots represented in Figure 8.2 and their linear regressions, it was possible to identify the *quality threshold of 0.5 for socio-technical congruence*. Over this threshold the amount of detected Community Smells were over the average values.

The higher the communicability, the lower the number of Community Smells. We formulated the *research sub-question **RQ1d*** because the very same conjecture was formulated within a research related to Social Debt in software architectures [7]. Our results demonstrated that in the 62% of analysed projects an increase of communicability was associated to a decrease of the number of occurrences of Organisational Silo Effect Community Smell and in the 70% of analysed projects it was associated to a decrease of the number of occurrences of Missing Links Community Smell. Therefore, *research sub-question RQ1d was positively answered*. Even if it was empirically demonstrated that communicability influences the presence of Community Smells, it was not possible to unequivocally identify a quality threshold.

A high community modularity of communication DSN yields a lower number of Community Smells. We formulated the *research sub-question **RQ1e*** because the literature suggested that a higher modularity, within developer networks, implies higher interdependency of sub-communities and thus it was possible to suppose that modularity was associated to a higher value of socio-technical congruence and, consequently, to a lower number of occurrences of Community Smells. This hypothesis was verified as in the 40% of analysed projects an increase of communication DSN's modularity was correlated to a decrease of the number of occurrences of Radio Silence Community Smell. This finding can be explained by the fact that a higher modularity corresponds to a lower coordination need and, thus, the role of unique knowledge and information broker tends to lose its importance. Therefore, *research sub-question RQ1e was positively answered.* Even if it was empirically demonstrated that modularity in the communication DSN influences the presence of Community Smells, it was not possible to unequivocally identify a quality threshold.

The higher the turnover of core developers, the lower the number of Community Smells. We formulated the *research sub-question **RQ1f*** because Cataldo et al. [15] empirically demonstrated that if a software development base is stable, then the socio-technical congruence of an organization will increase over time and thus, considering research sub-question RQ1c, it was reasonable to suppose that a lower turnover was associated to a lower number of occurrences of Community Smells within a software development community. Our results revealed that a higher turnover of *core* developers belonging to the collaboration DSN was correlated to a decrease of the number of occurrences of Missing Links Community Smell. In projects with less then 50 trimestral community members this correlation was not considered as relevant and more precisely none of the turnover metrics were found to have a relevant correlation with any of Community Smell. While, within projects with more than 150 trimestral community members the turnover of core developers was also negatively correlated to the insurgence of additional Organisational Silo Effect Community Smell and the turnover of global core community members were found to be negatively correlated with the additional insurgence of both Organisational Silo Effect and Missing Links Community Smells. Therefore, *research sub-question RQ1f was negatively answered.* Even if it was empirically demonstrated that turnover of core developers influences the presence of Community Smells, it was not possible to unequivocally identify a quality threshold.

While answering the first Research Question, it was possible to identify further quality factors, not considered by any research sub-questions, that can influence the number of Community Smells within a FLOSS development community. Such findings are reported in the following part of this section.

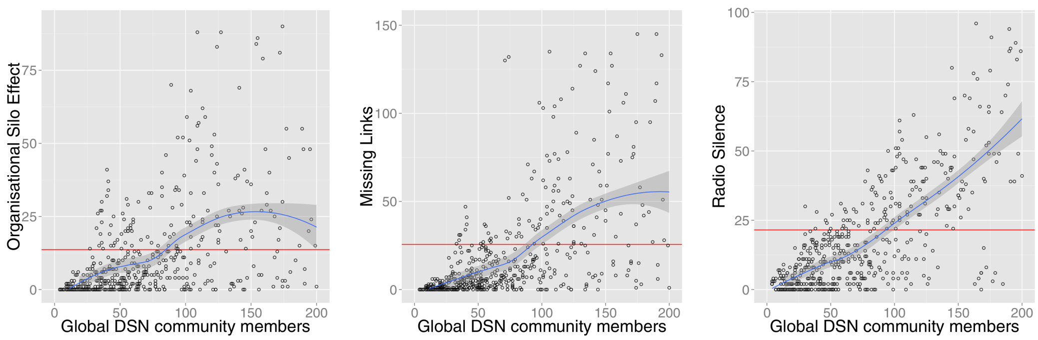


Figure 8.3: Scatter plots of global DSN community members

Community Smells increase with number of community members. An increment of the number of community members participating to a project's activities was associated to additional occurrences of Organisational Silo Effect (25% of analysed projects), Missing Link (28% of analysed projects) and Radio Silence (45% of analysed projects) Community Smells. Considering the scatter plots represented in Figure 8.3 and their local regressions (loess), it is possible to understand that *Community Smells increase quadratically with the linear growth of the number of community members until the threshold of 150 community members*, after which it tends to stabilise itself in the cases of Organisational Silo Effect and Missing Links Community Smell. Moreover, we discovered that in the 32% of analysed projects the number of occurrences of Radio Silence Community Smell was positively correlated to the number of community members who were present both in the communication and in the collaboration Developer Social Networks. This finding allows us to suppose that, within FLOSS environments, *developers who are present in the communication channel of a project tend to assume the privileged role of knowledge and information broker* within the communication channel of a development community.

Community Smells increase with number of Core community members. An increment of the number of core community members of the global DSN was correlated to additional occurrences of Organisational Silo Effect (27% of analysed projects), Missing Link (28% of analysed projects) and Radio Silence (30% of analysed projects) Community Smells. More specifically, obtained results demonstrated that an increment of the number of core developers belonging to the collaboration DSN was associated to a higher number of occurrences of Organisational Silo Effect Community Smell in the 80% of analysed projects and of Missing Links Community Smell in the 93% of analysed projects. It is important to highlight that such positive correlation was found both with respect to the number of developers who were considered core members only in collaboration DSN, that with respect to the number of developers who were considered at the same time core members within the collaboration and the communication DSNs. Therefore, an increment of the number of every typology of core developer within the collaboration DSN was correlated to an increment of the number of occurrences of Organisational Silo Effect and Missing Links Community Smells. The incidence of core developers in the generation of additional Community Smells was remarked even by the finding that a higher truck number related to the collaboration DSN, thus a higher ratio of peripheral developers, was correlated to a decrease of the number of occurrences of Organisation Silo Effect and Missing Links Community Smells, respectively in the 42% and in the 52% of analysed projects. Additionally, obtained results highlighted that, in the 27% of analysed projects, an increment of the number of core community members belonging to the communication DSN was correlated to a higher number of occurrences of Radio Silence Community Smell. In projects with more than 150

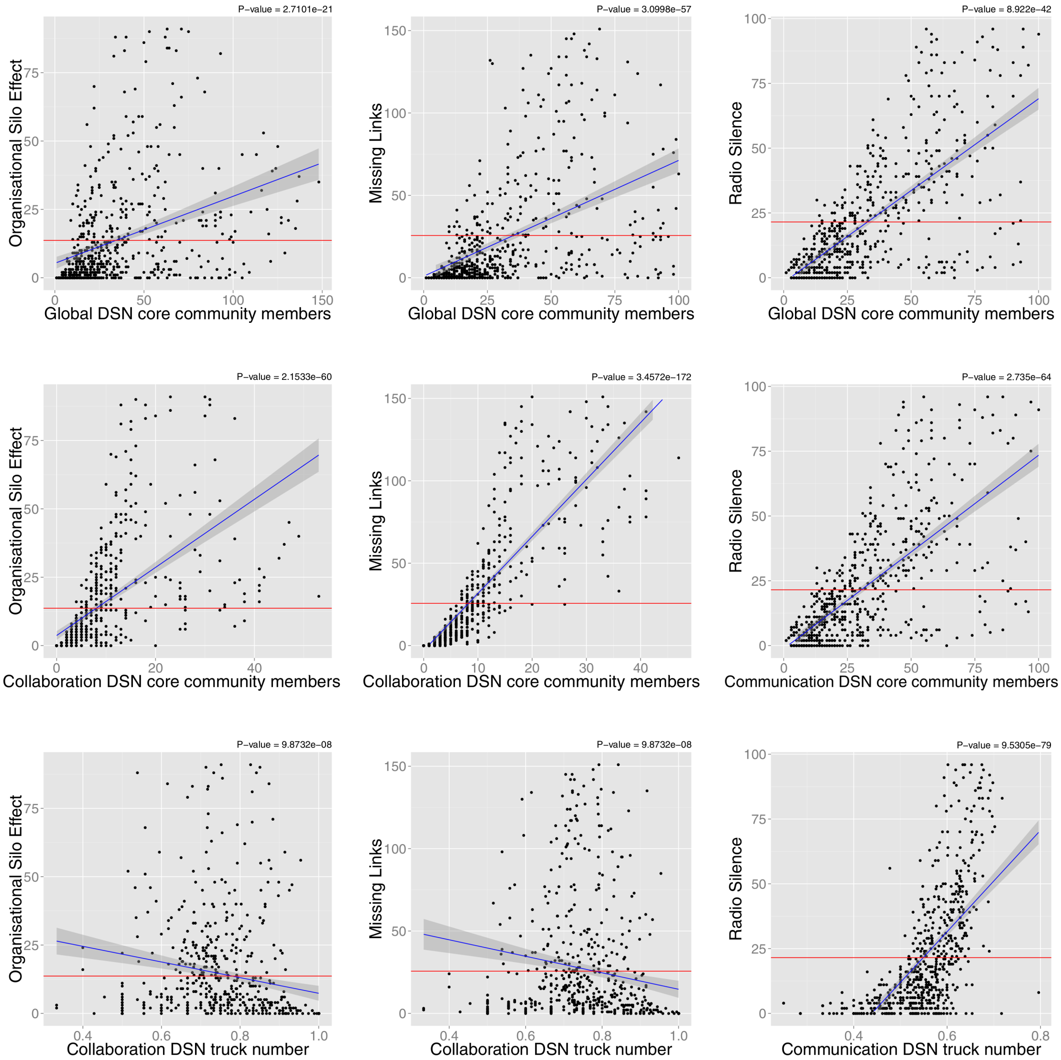


Figure 8.4: Scatter plots related to core community members

trimestral community members, the correlation between the number core community members belonging to the global or communication DSNs and the number of occurrences of Radio Silence Community Smell was not found to be relevant. A possible explanation can be that within big communities, the relevance of Community Smells generated by core community members tends to decrease with respect to the one generated by normal community members. Thus, core community members tend to lose their role of unique knowledge and information broker in highly dimensional development communities. This explanation is further motivated by retrieved results which revealed that in the 43% of total analysed projects, a higher truck number related to the communication DSN, thus a higher ratio of peripheral community members, was correlated to an increment of the number of occurrences of Radio Silence Community Smell. Therefore, Radio Silence Community Smell is generated by both core and peripheral community members belonging to the communication DSN but additional occurrences of Radio Silence Community Smell generated by core members tend to be irrelevant in big communities. Considering the scatter plots represented in Figure 8.4 and their linear regressions, it was possible to identify the following *quality thresholds,* for which the amount of detected Community Smells were over the average values:

- *25 trimestral core community members in the global DSN.* The quality threshold with respect to Missing Links and Radio Silence Community Smells was a bit higher (30 in both cases), but since the threshold with respect to Organisational Silo Effect Community Smell was 25, it was selected as the global threshold for the number of core community members within the global DSN;
- *9 trimestral core developers in the collaboration DSN;*
- *30 trimestral core community members in the communication DSN;*
- *0.8 for the truck number of the collaboration DSN;*
- *0.55 for the truck number of the communication DSN.*

Within small communities, the abandonment of members previously involved in Community Smells generates additional Community Smells. In projects constituted by less than 50 trimestral community members it was found a relevant positive correlation between the ratio of smelly quitters, thus the ratio of members who left the community and were implied in at least one Community Smell in previously analysed range, and the number of occurrences of Organisational Silo Effect and Missing Links Community Smells.

Even if the *centrality* of global Developer Social Network was not associated to an increase or decrease of the number of occurrences of Community Smells at a global lever, some relevant correlations were found considering different dimensional categories of FLOSS development communities:

- in projects with less then 50 trimestral community members, the increase of closeness centrality was associated to additional occurrences of Organisational Silo Effect Community Smell;
- in projects with more than 150 trimestral community members, the increase of betweenness centrality was associated to additional occurrences of Black-cloud Effect Community Smell.

All the relevant correlations found between quality factors and Community Smells, that allowed us to respond to the first Research Question, are summarised in Table 8.3, where positive correlations are identified by the plus sign while the negative ones with a minus sign.

Quality factor (ID)	Organisational Silo Effect	Radio Silence	Missing Links
devs	+	+	+
ml.only.devs		+	
code.only.devs	+		+
ml.code.devs		+	
perc.ml.only.devs	-		-
perc.code.only.devs	+		+
sponsored.devs	+		+
st.congruence	-		-
communicability	-		-
ratio.smelly.devs	+	+	+
core.global.devs	+	+	+
core.mail.devs		+	
core.code.devs	+		+
mail.truck		+	
code.truck	-		-
mail.only.core.devs		+	
code.only.core.devs	+		+
ml.code.core.devs	+		+
ratio.mail.only.core	-		-
ratio.code.only.core	+		+
core.code.turnover			-
mail.mod		-	

Table 8.3: Summary of quality factors correlated to Community Smells

8.3 Qualitative indicators of Community Smells

This section addresses the third Research Question (***RQ3***). We wanted to verify if developer perceptions about specific characteristics of their project could act as good indicators of the presence of Community Smells within FLOSS development communities. Fundamental prerequisites to the execution of this analysis were the results obtained from the proposed survey and the results obtained performing Codeface4Smells analysis of the development communities of the three reference projects (i.e., Firefox, LibreOffice and FFmpeg).

Results related to the number of occurrences of Community Smells of every reference project were normalized: for each range of analysis it was computed the number of occurrences of every typology of Community Smells per community member, thus dividing the total number of occurrences of each Community Smell typology for the total number of members constituting the community, and then it was calculated the average number of such metric, considering all the analysed ranges of the reference project.

In order to empirically verify if developer perceptions were actually associated to a lower or higher number of Community Smells, estimated considering the mean number of trimestral occurrences of Community Smells per community member, the following list of steps was executed for every factor considered by every question of the executed survey:

1. If the aspect in analysis was addressed within the third part of the questionnaire, decide if it is preferred to consider agreement or disagreement results;
2. Define if, for the analysed aspect, it is expected a growth or reduction trend of the mean number of occurrences of Community Smells per community member;
3. Arrange reference projects in ascendant order with respect to their results related to the analysed aspect, considering the decision taken in Step 1;
4. If the trend speculated in Step 2 is verified in the order of the three reference projects identified in Step 3, then developer perceptions is empirically proven to be a significant indicator of the presence of additional Community Smells within a FLOSS development community, with respect to the analysed aspect.

Table 8.4 represents the summary of the *mean number of trimestral occurrences of Community Smells per community member* with respect to every reference project. Detailed reports of the three reference projects, related to Codeface4Smells analysis results and responses of the survey, can be found respectively in Appendix C and Appendix A.

Project	Organisational Silo	Missing Links	Radio Silence	Black-cloud	Prima-donnas
Firefox	10.44444	10.60712	0.02469979	0	0.000527224
FFmpeg	0.3967442	0.5849786	0.3842722	0.004148463	0.001766914
LibreOffice	2.261178	3.265845	0.2768581	0.001033613	0.006896172

Table 8.4: Average number of Community Smells per community member

By performing the previously defined list of steps, it was possible to empirically verify the validity of the following developer perceptions as significant indicators of the presence of Community Smells :

1. **Better consideration of developers sponsored by commercial companies is correlated to a lower number of Community Smells.** A higher agreement with the statement that asserted that sponsored developers increased the health of the project, was correlated to a lower mean number of trimestral occurrences of Radio Silence and Black-cloud Effect Community Smells per community member. The same result was obtained considering the disagreement levels and correlating them to an increase of Community Smells.
2. **Higher perceived decision importance inequality is correlated to a higher number of Community Smells.** A higher disagreement with the statement that asserted that every opinion was equal in project's important decisions, was correlated to a higher mean number of trimestral occurrences

of Radio Silence and Black-cloud Effect Community Smells per community member.

3. **Higher perceived importance of communications is partially correlated to a lower number of Community Smells.** A higher agreement with the statement that asserted that frequent communication before and after commit activities was essential for the project, was correlated to a lower mean number of trimestral occurrences of Radio Silence and Black-cloud Effect Community Smells per community member. It is interesting to highlight that the perceived importance of communications within the reference projects was directly proportional to the mean number of trimestral occurrences of Organisational Silo Effect and Missing Links Community Smells per community member. Therefore, communities with well developed communication culture are intended to generate less Community Smells.

4. **Higher perceived quality of documentation is correlated to a lower number of Community Smells.** A higher agreement with the statement that asserted that project's software architecture was well documented, easily accessible and understandable and with the one that asserted that project's documentation was understandable and helpful, were correlated to a lower mean number of trimestral occurrences of Radio Silence and Black-cloud Effect Community Smells per community member. This finding highlights the important role of documentation within a development community in order to perform informed decisions, understand architectural decisions and achieve a shared and coherent project knowledge, because Black-cloud Effect and Radio Silence Community Smells are characterised by low mutual awareness, lack of knowledge exchanges, hidden information and tunnel vision associated to autonomous architecture decision making activities.

5. **More personal assumptions are correlated to a higher number of Community Smells.** A higher disagreement with the statement that asserted that the respondent did assumptions during development due to unclear requirements or documentation, was correlated to a lower mean number of trimestral occurrences of Prima-donnas Community Smell per community member. It was necessary to consider the disagreement and the decreasing trend of occurrences of Community Smells because Firefox and FFmpeg communities achieved the same exact agreement level. This result further verifies the consideration of previous finding.

6. **Higher perceived antagonism between sub-communities is correlated to a higher number of Community Smells.** A higher agreement with the statement that asserted that sometimes different subgroups were antagonists,

was correlated to a higher mean number of trimestral occurrences of Organisational Silo Effect and Missing Links Community Smells per community member.

7. **Higher perceived communication delays are correlated to a higher number of Community Smells.** A higher number of responses indicating communication delays as the main reason of time waste within any phase of the project's development, was correlated to a higher mean number of trimestral occurrences of Organisational Silo Effect and Missing Links Community Smells per community member.

Moreover, it was possible to empirically demonstrate the importance of the analysed communication channel to be able to achieve a complete and consistent overview of occurrences of Community Smells within a software development community. A higher number of responses indicating the project's mailing list as the communication channel through which important project's decisions were made, was correlated to a higher mean number of trimestral occurrences of Radio Silence and Black-cloud Effect Community Smells per community member. Therefore, it was proven that considering the most used and information-rich communication channel within Community Smells analysis, is fundamental in order to capture more occurrences of Community Smells.

8.4 Summary of Research Questions

It was possible to *positively answer to the first and second Research Questions*, since it was possible to empirically identify quality factors that were correlated to Community Smells present within software development communities and, for same of them, it was possible to elicit quality thresholds that can be used by maintainers to improve the health of communities. Considering the second Research Question, Table 8.5 summarises all the identified quality thresholds that can be used to keep Community Smells under control.

It was possible to *positively answer to the third Research Question*, since it was possible to identify specific developer perceptions that were actually correlated to the amount of Community Smells present in the reference projects and thus, such identified developer perceptions, can be used to obtain *qualitative* indications about the presence of Community Smells and achieve a deeper understanding of Community Smells within FLOSS environments.

In conclusion, Table 8.6 briefly summarises the answers to the research questions formulated in Chapter 3.

Quality factor	Healthy value
# trimestral sponsored developers	< 10
Socio-technical congruence	> 0.5
# trimestral core members in global DSN	< 25
# trimestral core members in collaboration DSN	< 9
# trimestral core members in communication DSN	< 30
Truck number of collaboration DSN	> 0.8
Truck number of communication DSN	< 0.55

Table 8.5: Summary of socio-technical quality thresholds

ID	Research Question	Answer
RQ1	Are there quality factors that can influence the emergence of Community Smells?	**YES**
RQ1a	Do sponsored developers decrease Community Smells?	**NO**
RQ1b	Do temporal and geographic dispersion increase Community Smells?	**NO**
RQ1c	Do high socio-technical congruence decrease Community Smells?	**YES**
RQ1d	Do high communicability decrease Community Smells?	**YES**
RQ1e	Do high modularity decrease Community Smells?	**YES**
RQ1f	Do low turnover decrease Community Smells?	**NO**
RQ2	If such quality factors exist, is it possible to identify quality thresholds?	**YES**
RQ3	Do developer perceptions indicate the presence of Community Smells?	**YES**

Table 8.6: Summary of Research Questions

8.5 Threats to validity

This section highlights potential threats that could affect the validity of the results, evaluations and conclusions proposed in this master thesis.

Construct Validity. Threats to construct validity are related to the relationships between theory and observations and, generally, this typology of threat is mainly constituted by imprecisions in performed measurements. In the part of the study related to the identification and quantification of Community Smells and quality factors, this typology of threat to validity is mainly concerned to how quality factors and identification patterns of Community Smells, were implemented in Codeface4Smells. Also, part of this research is based on results obtained using a survey, thus construct validity may be compromised by biased developer perceptions and because it was used a questionnaires as a measure of comprehension. Therefore the study can be affected by construct validity.

Internal Validity. Threats to internal validity are related to factors that could have influenced our results. In the part of the study related to the identification an quantification of Community Smells, a factor that can potentially impact on our ability to correctly detect Community Smells is that the considered development mailing list is indeed the main communicational channel used by community members. We were not able to check whether this was so in all the projects composing our sample, however, confirmatory survey results strongly highlight that mailing lists are indeed the key communication channel used within FLOSS development communities. Concerning the survey, it was impossible for us to ensure that respondents had a good knowledge of all analysed aspects of the software development community through which they were contacted. Moreover, a factor that could have influenced our results was the questionnaire response rate, that was quite low (1.14% with respect to the three reference projects) with respect to expected return rates of this typology of studies (20% - 11% [51, 71]). One motivation that may partially explain the low response rate is that we extracted information related to developers, considering the entire life time of analysed projects, therefore considering even very old information (more than 12 years old).

External Validity. Threats to external validity are related to the generalization of obtained results. Codeface4Smells currently identifies and quantifies five different Community Smells and considers them as indicators of potential risk related to Social Debt existence, but there exist other Community Smells not operationalised yet [6], that may act as significant indicators of the risk of Social Debt. Moreover, we conducted our analysis on a total of 60 FLOSS development projects, ensuring a high generalisation level of our findings, but these results might be influenced by the temporal window that we selected for the analysis (i.e., 3 months).

Chapter 9

Conclusions and future work

The study presented in this master thesis demonstrated that Open Source Software development communities are not immune to Community Smells, i.e., nasty socio-technical and organisational circumstances that may lead to project delays or the insurgence of Social Debt [6].

By conducting a survey, we confirmed that developers perceive the presence of Social Debt and socio-technical issues within FLOSS development communities and we empirically identified specific developer perceptions that can be used as qualitative indicators of the presence of additional occurrences of Community Smells.

This master thesis elaborates, operationalises, validates and discusses a Socio-technical Quality Framework for software development communities, constituted by the measurement of socio-technical quality factors and by the identification and quantification of Community Smells. During our empirical software engineering research, it was possible to identify several quality factors correlated to the presence of Community Smells within software development communities. Above all, our results suggested that the role of core community member is fundamental in the generation of additional occurrences of Community Smells and, therefore, core community members should make a greater effort to limit their anti-social and non-optimal behaviors in order to increase the chances of success of their development community. Moreover, while evaluating of our Socio-technical Quality Framework it was possible to unequivocally identify quality thresholds of some specific socio-technical quality factors. Therefore, we concluded that our framework, operationalised in a tool called Codeface4Smells, does in fact offer a lens to observe software development communities from a quality perspective and diagnose organisational issues in an automated tool-supported fashion.

The evaluation of our work, presented in Chapter 8, and the results of the executed survey, allowed us to conclude that Community Smells and Social Debt can actually interfere with the well-being of software development communities. Therefore, while executing project performance analysis, it is important to consider even the social and organisational aspects besides the technical ones, in order to lower the barriers that can influence the success of the development community.

With the goal of further increasing the identification capabilities of quality factors capable of influencing the presence of Community Smells within a software development community, a possible extension of this work can be the elimination from our framework of quality factors that were not correlated to the insurgence of Community Smells and the introduction of new quality factors. Furthermore, a possible future enhancement can be the definition and operationalisation of identification patterns of additional Community Smells. Moreover, a possible evolution of this software engineering research consists in the evaluation of the proposed Socio-technical Quality Framework from a technical perspective, with the purpose of identifying correlations between Community Smells and Code Smells.

Finally, we plan to merge the developed tool-support in the main Codeface distribution, potentially transforming Codeface into a full-fledged continuous software development community improvement platform.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] Nachiappan Nagappan, Brendan Murphy, and Victor Basili. The influence of organizational structure on software quality: an empirical case study. In *Proceedings of the 30th international conference on Software engineering*, pages 521–530. ACM, 2008.

[2] Damian A Tamburri, Patricia Lago, and Hans van Vliet. Organizational social structures for software engineering. *ACM Computing Surveys (CSUR)*, 46(1):3, 2013.

[3] Damian A Tamburri, Philippe Kruchten, Patricia Lago, and Hans van Vliet. What is social debt in software engineering? In *Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop on*, pages 93–96. IEEE, 2013.

[4] Philippe Kruchten, Robert L Nord, and Ipek Ozkaya. Technical debt: From metaphor to theory and practice. *Ieee software*, 29(6), 2012.

[5] Marcelo Cataldo, James D Herbsleb, and Kathleen M Carley. Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 2–11. ACM, 2008.

[6] Damian Andrew Tamburri, Philippe Kruchten, Patricia Lago, and Hans van Vliet. Social debt in software engineering: insights from industry. *J. Internet Services and Applications*, 6(1):10:1–10:17, 2015.

[7] Damian A Tamburri and Elisabetta Di Nitto. When software architecture leads to social debt. In *Software Architecture (WICSA), 2015 12th Working IEEE/IFIP Conference on*, pages 61–64. IEEE, 2015.

[8] Dirk Riehle, Philipp Riemer, Carsten Kolassa, and Michael Schmidt. Paid vs. volunteer work in open source. In *2014 47th Hawaii International Conference on System Sciences*, pages 3286–3295. IEEE, 2014.

[9] Melvin E Conway. How do committees invent. *Datamation*, 14(4):28–31, 1968.

[10] James D Herbsleb and Rebecca E Grinter. Architectures, coordination, and distance: Conway's law and beyond. *IEEE software*, 16(5):63, 1999.

[11] Frederick P Brooks Jr. The mythical man-month (anniversary ed.). 1995.

[12] David Lorge Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

[13] 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 (TOSEM)*, 11(3):309–346, 2002.

[14] Audris Mockus. Organizational volatility and its effects on software defects. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, pages 117–126. ACM, 2010.

[15] Marcelo Cataldo, Patrick A Wagstrom, James D Herbsleb, and Kathleen M Carley. Identification of coordination requirements: implications for the design of collaboration and awareness tools. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 353–362. ACM, 2006.

[16] Anita Sarma, Jim Herbsleb, and André Van Der Hoek. Challenges in measuring, understanding, and achieving social-technical congruence. In *Proceedings of Socio-Technical Congruence Workshop, In Conjuction With the International Conference on Software Engineering*, 2008.

[17] Lyra J Colfer and Carliss Y Baldwin. The mirroring hypothesis: Theory, evidence and exceptions. *Harvard Business School Finance Working Paper*, (16-124), 2016.

[18] Seiji Sato, Hironori Washizaki, Yoshiaki Fukazawa, Sakae Inoue, Hiroyuki Ono, Yoshiiku Hanai, and Mikihiko Yamamoto. Effects of organizational changes on product metrics and defects. In *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*, volume 1, pages 132–139. IEEE, 2013.

[19] `https://en.wikipedia.org/wiki/Globalization`.

[20] Kenneth R Gray and Thomas L Friedman. The world is flat: A brief history of the twenty-first century, 2005.

[21] Juyun Cho. Globalization and global software development. *Issues in information systems*, 8(2):287–290, 2007.

[22] Bogdan Vasilescu, Alexander Serebrenik, and Vladimir Filkov. A data set for social diversity studies of github teams. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 514–517. IEEE, 2015.

[23] Eric Molleman and Jannes Slomp. The impact of team and work characteristics on team functioning. *Human Factors and Ergonomics in Manufacturing & Service Industries*, 16(1):1–15, 2006.

[24] Christopher P Earley and Elaine Mosakowski. Creating hybrid team cultures: An empirical test of transnational team functioning. *Academy of Management Journal*, 43(1):26–49, 2000.

[25] `http://www.gnu.org/philosophy/free-sw.html`.

[26] `https://opensource.org/osd.html`.

[27] `http://www.gnu.org/philosophy/open-source-misses-the-point.html`.

[28] `http://www.gnu.org/philosophy/floss-and-foss.html`.

[29] Bogdan Vasilescu, Daryl Posnett, Baishakhi Ray, Mark GJ van den Brand, Alexander Serebrenik, Premkumar Devanbu, and Vladimir Filkov. Gender and tenure diversity in github teams. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 3789–3798. ACM, 2015.

[30] Christopher Vendome, Mario Linares-Vásquez, Gabriele Bavota, Massimiliano Di Penta, Daniel M German, and Denys Poshyvanyk. When and why developers adopt and change software licenses. In *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*, pages 31–40. IEEE, 2015.

[31] Qiaona Hong, Sunghun Kim, Shing Chi Cheung, and Christian Bird. Understanding a developer social network and its evolution. In *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, pages 323–332. IEEE, 2011.

[32] Fu-ren Lin and Chun-hung Chen. Developing and evaluating the social network analysis system for virtual teams in cyber communities. In *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*, pages 8–pp. IEEE, 2004.

[33] James Howison, Keisuke Inoue, and Kevin Crowston. Social dynamics of free and open source team communications. In *IFIP International Conference on Open Source Systems*, pages 319–330. Springer, 2006.

[34] Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz, and Anand Swaminathan. Mining email social networks. In *Proceedings of the 2006 international workshop on Mining software repositories*, pages 137–143. ACM, 2006.

[35] Gregory Madey, Vincent Freeh, and Renee Tynan. The open source software development phenomenon: An analysis based on social network theory. *AMCIS 2002 Proceedings*, page 247, 2002.

[36] Luis Lopez-Fernandez, Gregorio Robles, Jesus M Gonzalez-Barahona, et al. Applying social network analysis to the information in cvs repositories. In *International workshop on mining software repositories*, pages 101–105, 2004.

[37] Andrew Meneely and Laurie Williams. Socio-technical developer networks: should we trust our measurements? In *Proceedings of the 33rd International Conference on Software Engineering*, pages 281–290. ACM, 2011.

[38] Roozbeh Nia, Christian Bird, Premkumar Devanbu, and Vladimir Filkov. Validity of network analyses in open source projects. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, pages 201–209. IEEE, 2010.

[39] Caius Brindescu, Mihai Codoban, Sergii Shmarkatiuk, and Danny Dig. How do centralized and distributed version control systems impact software changes? In *Proceedings of the 36th International Conference on Software Engineering*, pages 322–333. ACM, 2014.

[40] Andrejs Jermakovics, Alberto Sillitti, and Giancarlo Succi. Mining and visualizing developer networks from version control systems. In *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 24–31. ACM, 2011.

[41] Mitchell Joblin, Wolfgang Mauerer, Sven Apel, Janet Siegmund, and Dirk Riehle. From developer networks to verified communities: a fine-grained approach. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, pages 563–573. IEEE Press, 2015.

[42] Christian Bird, Nachiappan Nagappan, Harald Gall, Brendan Murphy, and Premkumar Devanbu. Putting it all together: Using socio-technical networks to predict failures. In *2009 20th International Symposium on Software Reliability Engineering*, pages 109–119. IEEE, 2009.

[43] Yonghee Shin, Andrew Meneely, Laurie Williams, and Jason A Osborne. Evaluating complexity, code churn, and developer activity metrics as indica-

tors of software vulnerabilities. *IEEE Transactions on Software Engineering*, 37(6):772–787, 2011.

[44] Ning Nan and Sanjeev Kumar. Joint effect of team structure and software architecture in open source software development. *IEEE Transactions on Engineering Management*, 60(3):592–603, 2013.

[45] Giuseppe Valetto, Mary Helander, Kate Ehrlich, Sunita Chulani, Mark Wegman, and Clay Williams. Using software repositories to investigate socio-technical congruence in development projects. In *Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 25. IEEE Computer Society, 2007.

[46] Jorge Colazo. Structural changes associated with the temporal dispersion of teams: Evidence from open source software projects. In *2014 47th Hawaii International Conference on System Sciences*, pages 300–309. IEEE, 2014.

[47] Ward Cunningham. The wycash portfolio management system. *SIGPLAN OOPS Mess.*, 4(2):29–30, December 1992.

[48] Nicolli SR Alves, Leilane F Ribeiro, Vivyane Caires, Thiago S Mendes, and Rodrigo O Spínola. Towards an ontology of terms on technical debt. In *Managing Technical Debt (MTD), 2014 Sixth International Workshop on*, pages 1–7. IEEE, 2014.

[49] William H Brown, Raphael C Malveau, Hays W McCormick, and Thomas J Mowbray. *AntiPatterns: refactoring software, architectures, and projects in crisis*. John Wiley & Sons, Inc., 1998.

[50] Martin Fowler. Refactoring: Improving the design of existing code. In *11th European Conference. Jyväskylä, Finland*, 1997.

[51] Fabio Palomba, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Denys Poshyvanyk, and Andrea De Lucia. Mining version histories for detecting code smells. *IEEE Transactions on Software Engineering*, 41(5):462–489, 2015.

[52] Ivaldir H de Farias Junior, Ryan R de Azevedo, Hermano P de Moura, and Dennis S Martins da Silva. Elicitation of communication inherent risks in distributed software development. In *2012 IEEE Seventh International Conference on Global Software Engineering Workshops*, pages 37–42. IEEE, 2012.

[53] Neil A Ernst, Stephany Bellomo, Ipek Ozkaya, Robert L Nord, and Ian Gorton. Measure it? manage it? ignore it? software practitioners and technical debt. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 50–60. ACM, 2015.

[54] Aniket Potdar and Emad Shihab. An exploratory study on self-admitted technical debt. In *ICSME*, pages 91–100, 2014.

[55] Nianjun Zhou, Wesley M Gifford, Krishna Ratakonda, Gregory H Westerwick, and Carl Engel. On the quantification of global team performance and profitability. In *Services Computing (SCC), 2014 IEEE International Conference on*, pages 378–385. IEEE, 2014.

[56] Mathieu Lavallée and Pierre N Robillard. Why good developers write bad code: An observational case study of the impacts of organizational factors on software quality. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, pages 677–687. IEEE Press, 2015.

[57] Valerio Cosentino, Javier Luis Cánovas Izquierdo, and Jordi Cabot. Assessing the bus factor of git repositories. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 499–503. IEEE, 2015.

[58] Ran Mo, Yuanfang Cai, Rick Kazman, Lu Xiao, and Qiong Feng. Decoupling level: a new metric for architectural maintenance complexity. In *Proceedings of the 38th International Conference on Software Engineering*, pages 499–510. ACM, 2016.

[59] Damian A Tamburri, Elisabetta Di Nitto, and Patricia Lago. "let me measure my self!" said open-source. (unpublished).

[60] Mitchell Joblin, Sven Apel, Claus Hunsen, and Wolfgang Mauerer. Classifying developers into core and peripheral: An empirical study on count and network metrics. *arXiv preprint arXiv:1604.00830*, 2016.

[61] Kevin Crowston, Kangning Wei, Qing Li, and James Howison. Core and periphery in free/libre and open source software team communications. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*, volume 6, pages 118a–118a. IEEE, 2006.

[62] Antonio Terceiro, Luiz Romario Rios, and Christina Chavez. An empirical study on the structural complexity introduced by core and peripheral developers in free software projects. In *Software Engineering (SBES), 2010 Brazilian Symposium on*, pages 21–29. IEEE, 2010.

[63] Audris Mockus and David M Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2):169–180, 2000.

[64] Akinori Ihara, Yasutaka Kamei, Masao Ohira, Ahmed E Hassan, Naoyasu Ubayashi, and Ken-ichi Matsumoto. Early identification of future committers in open source software projects. In *2014 14th International Conference on Quality Software*, pages 47–56. IEEE, 2014.

[65] Linton C Freeman. Centrality in social networks conceptual clarification. *Social networks*, 1(3):215–239, 1978.

[66] Mark EJ Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

[67] Stanley Wasserman and Katherine Faust. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994.

[68] `http://siemens.github.io/codeface`.

[69] Fabiano Berardo de Sousa and Liang Zhao. Evaluating and comparing the igraph community detection algorithms. In *Intelligent Systems (BRACIS), 2014 Brazilian Conference on*, pages 408–413. IEEE, 2014.

[70] Jan Hauke and Tomasz Kossowski. Comparison of values of pearson's and spearman's correlation coefficients on the same sets of data. *Quaestiones geographicae*, 30(2):87–93, 2011.

[71] Yehuda Baruch. Response rate in academic studies-a comparative analysis. *Human relations*, 52(4):421–438, 1999.

[72] `https://github.com/smnmgn/codeface`.

[73] `https://www.vagrantup.com`.

[74] `https://www.virtualbox.org`.

[75] `https://github.com/smnmgn/master_thesis`.

Appendix A

Survey

This Appendix lists the subset of questions of the survey that were considered in this master thesis in order to retrieve information about Social Debt and Community Smells, validate research assumptions and capture developer perceptions. Moreover, it presents the characteristics of all initially considered projects and explains how such characteristics were computed. Finally, this appendix provides a graphical representation of results related to the third part of the questionnaire.

A.1 The questionnaire

This section contains the list of questions that composed the survey, which was specifically designed to conduct our study.

Part one:

1. Country
2. Year of birth
3. Occupation
   (a) Student
   (b) Part time employee
   (c) Full time employee
   (d) Unemployed
   (e) Retired
4. Community

5. Role
   (a) Developer
   (b) Maintainer
   (c) Software engineer
   (d) Translator
   (e) Graphic
   (f) Other

Part two:

1. Do you contribute to this project as an individual or because your company is involved in it?
   (a) Paid by a company
   (b) Partially paid by company
   (c) Voluntary developer
2. Through which channel are important project decisions made?
   (a) Mailing list
   (b) Forum
   (c) IRC
   (d) Closed group of developers
   (e) I don't know
   (f) Other
3. What is the major cause of time waste within any phase of the project's development?
   (a) Delays in developer communications
   (b) Long time to reach an agreement within the community
   (c) Development and bug fixing can't keep up with deadlines
   (d) Bad or poor software design decisions end up in reengineering and fixup code
   (e) Project disorganization
   (f) I don't know

(g) Other

4. What was the reason of your longest wait for a commit approval?

    (a) Unavailable maintainers stalled the process
    (b) Long discussion between developers to decide if accept or deny
    (c) Proposed code didn't follow community standards
    (d) Code had to be rewritten to be accepted
    (e) I don't know
    (f) I don't remember
    (g) Other

Part three:

1. Developers sponsored by commercial companies increase project health
2. Absence of a developer can stall some community activities
3. Absence of a core developer can stall some community activities
4. The project development has a high degree of formality
5. Every opinion is equal in project's important decisions
6. Frequent communication before and after commit activities are essential
7. Community rules and structures are sound and clear
8. Software architecture is well documented, easily accessible and understandable
9. I did assumptions during development due to unclear requirements or documentation
10. Documentation is understandable and helpful
11. There are different subgroups within the community
12. Different subgroups rarely communicate
13. Subgroups have similar mindset and act as little communities inside the community
14. Different subgroups are sometimes antagonists

A.2 Characteristics of initially considered projects

Table A.1 explicits the numerical values used to classify FLOSS projects factors, extracted for every project the 8th of February 2016.

Project	#Years	#Commits	#last year commits	#Committers	Size	Activity	Relevance
Firefox	18	460959	50320	4560	*BIG*	*ACTIVE*	*HIGH*
Android	7	213252	31331	1734	*BIG*	*ACTIVE*	*HIGH*
WebKit	14	172077	12931	622	*BIG*	*ACTIVE*	*LOW*
LibreOffice	6	387763	19290	1441	*BIG*	*STALL*	*HIGH*
FFmpeg	15	78350	8893	1460	*SMALL*	*ACTIVE*	*HIGH*
OpenSSL	17	15162	2312	262	*SMALL*	*ACTIVE*	*LOW*
LibreSSL	2	494	285	27	*SMALL*	*ACTIVE*	*LOW*
AngularJS	6	7497	1051	1547	*SMALL*	*STALL*	*HIGH*
Khtml	2	264	56	32	*SMALL*	*STALL*	*LOW*
libva	9	1136	73	101	*SMALL*	*STALL*	*LOW*

Table A.1: Characteristics of projects initially considered for the survey

The commands used to quantify the value of analysed project characteristics were the following:

- *Years of activity*

```
git log --pretty=format:%ar |
tail -1
```

- *Total number of commits*

```
git rev-list --count HEAD
```

- *Numbers of commits during the last 12 months*

```
git log --since='last 12 months' --pretty=format:'%h' |
wc -l
```

- *Total number of project committers*

```
git log --all --pretty=format:"%aE"  |
sort -u -f  |
wc -l
```

A.3 Likert scale results

This section contains the likert graphs of agreement questions gathered from the third part of our survey. The first likert graph (Figure A.1) represents all the retrieved responses as a whole, while the following three graphs present the likert graphs decomposing responses with respect to the three reference projects. The code that identifies a question result within a likert graph is related to the question associated with the same number in the third part of the questionnaire, elicited in the first section of this appendix.

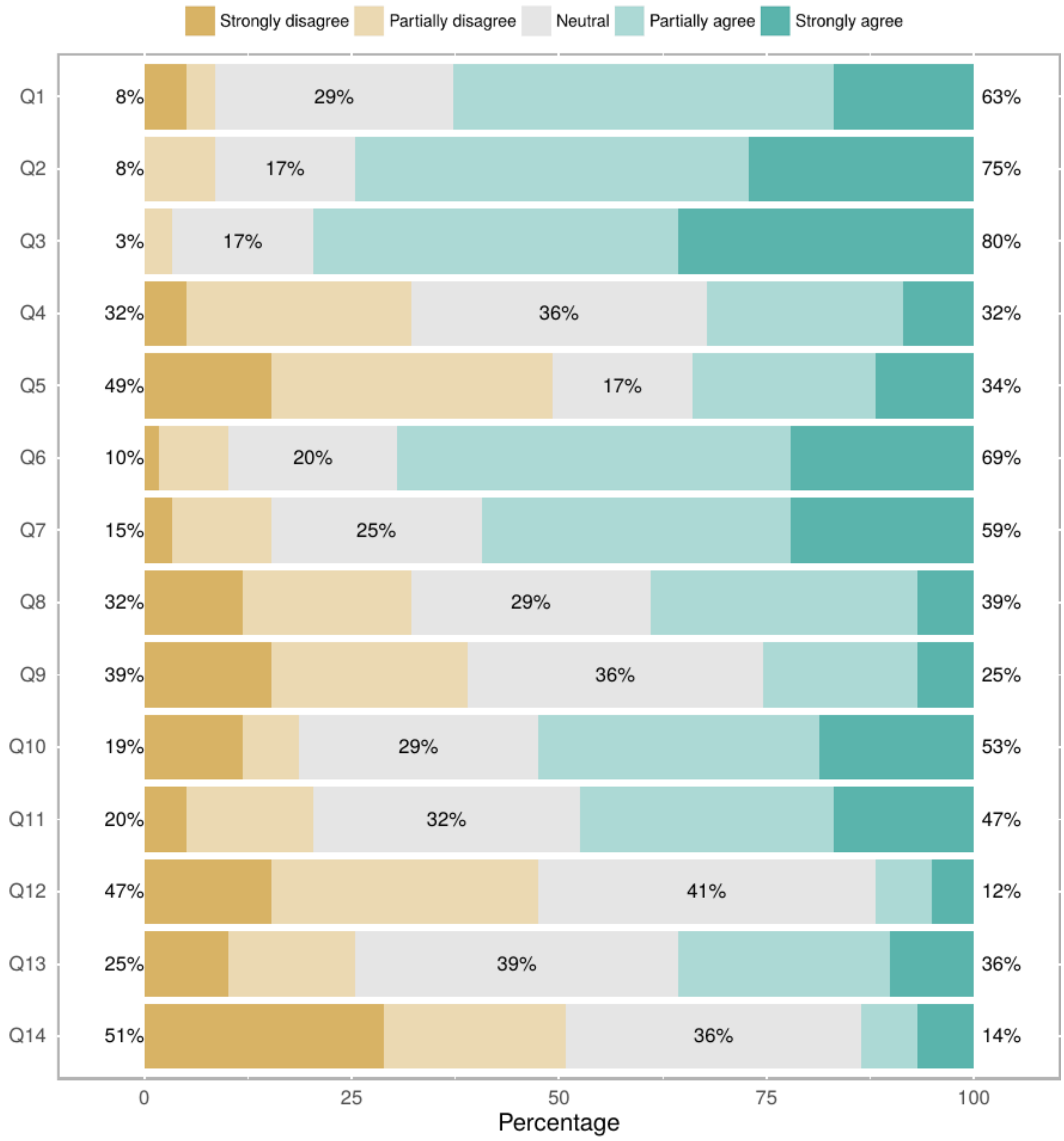


Figure A.1: Survey results (part three) - All respondents

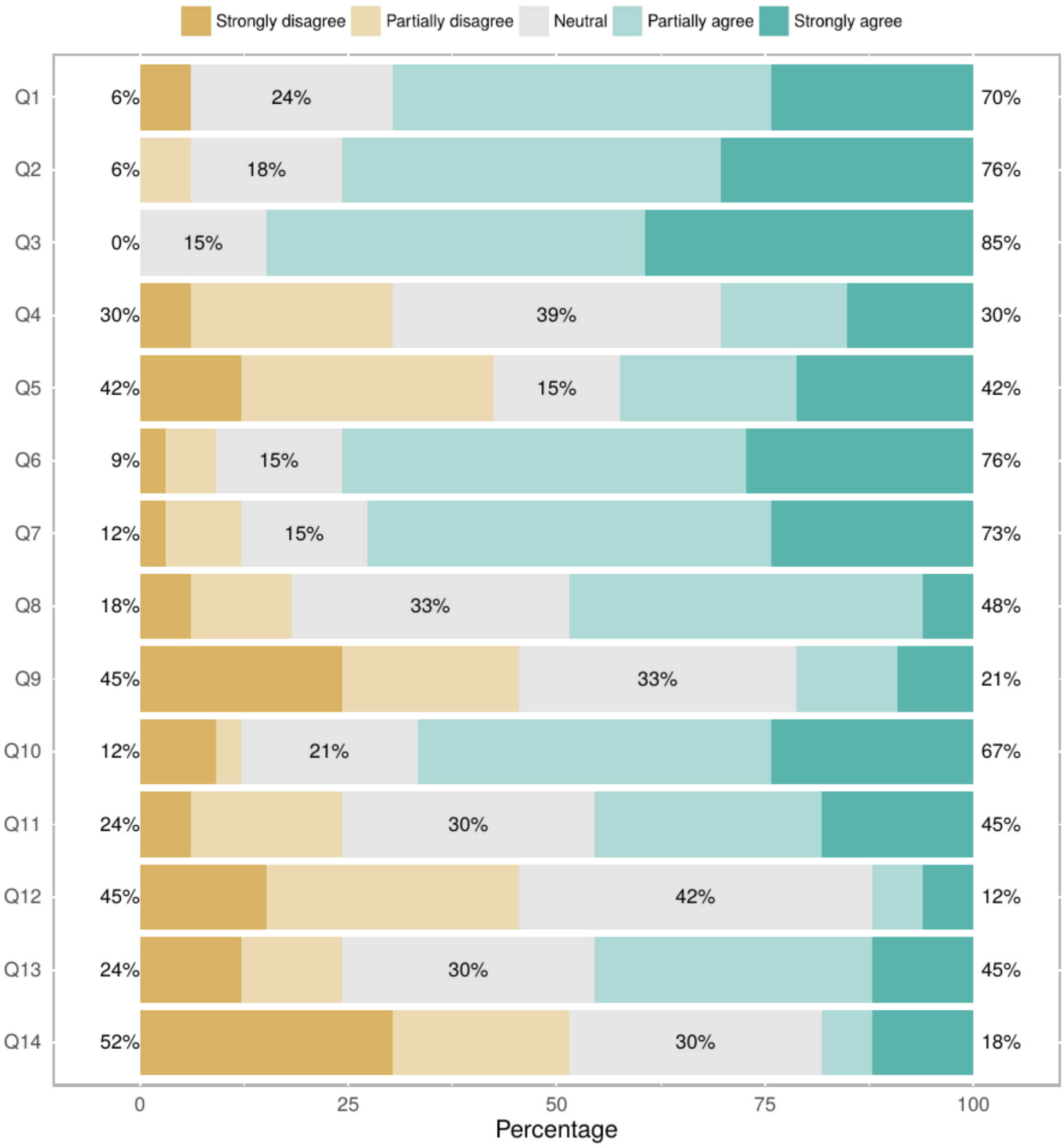


Figure A.2: Survey results (part three) - Firefox

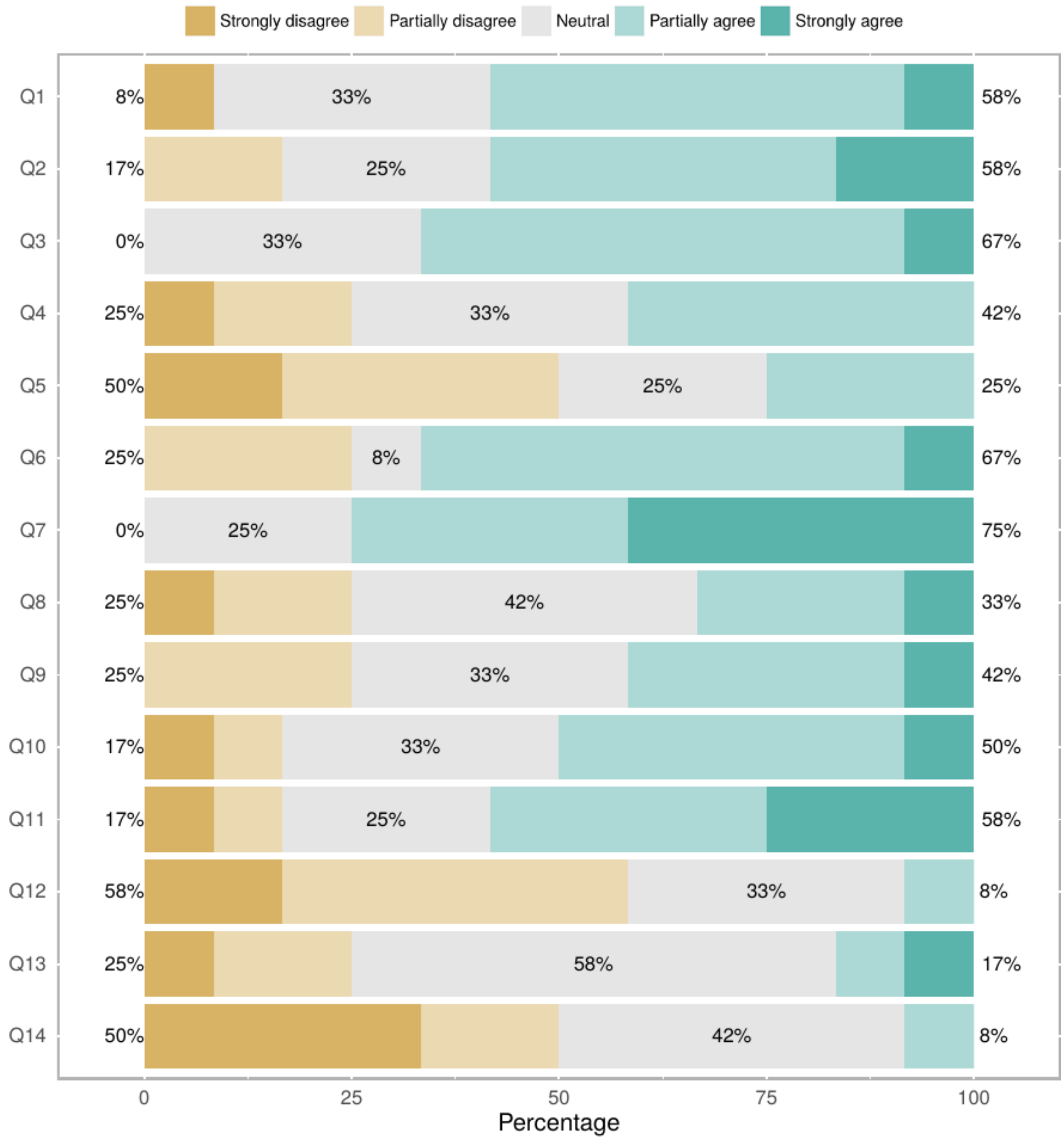


Figure A.3: Survey results (part three) - LibreOffice

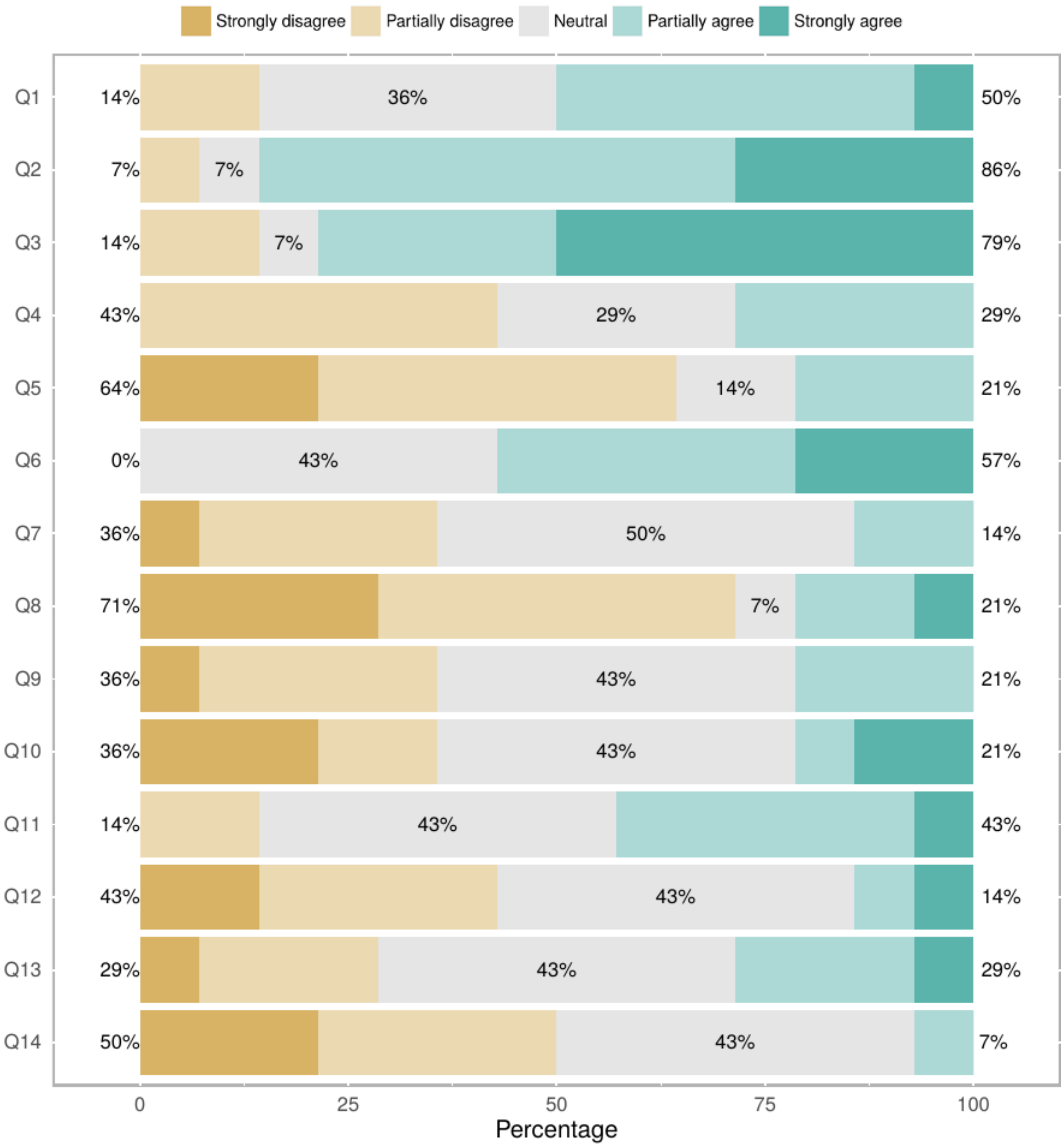


Figure A.4: Survey results (part three) - FFmpeg

Appendix B

Codeface4Smells

This Appendix describes how to install, set-up and execute Codeface4Smells, the tool developed during this master thesis, through which we operationalised our Socio-technical Quality Framework and the identification patterns of Community Smells, explained respectively in Chapter5 and Chapter 4. Moreover, we report a brief set of lessons learned, in order to provide useful notions, about how to write a proper configuration file and successfully complete the collaboration analysis of a high-volume software project using Codeface. Finally, we present some utility scripts specifically developed to cope with particular needs emerged within this study.

B.1 Set-up and analysis execution

Codeface4Smells is hosted on GitHub and then it is possible to download it manually or cloning its associated git repository [72]. As explained in Chapter 7, Codeface4Smells extends Codeface and thus, it shares its inner workings. Therefore, since Codeface is supposed to be executed in a virtual machine, for practical reasons, the presence of Vagrant [73] and VirtualBox [74] on the machine is a necessary precondition in order to continue with the set-up of our tool.

Once the source code of Codeface4Smells is present on the machine, it is possible to initialise the virtual machine and then access it, using the following set of commands:

```
# vagrant up
# vagrant ssh
```

The first command initialises the virtual machine. If it is the first time that the considered instance of Codeface4Smells is executed, then Vagrant will download the virtual machine image and some initialisation scripts will download and install all the necessary software to correctly execute the tool. The second command establishes an ssh connection with the previously initialised virtual machine, in order to have access to its console.

Whenever the virtual machine is not needed anymore, it is possible to shut it down using the following command:

```
# vagrant halt
```

Before being able to analyse a project with Codeface4Smells, it is necessary to have access to the following project resources: source code, mailing list archive files (mbox format) and a configuration file. Moreover, before performing the actual Codeface4Smells analysis it is necessary to execute the collaboration and communication analysis, provided by Codeface, in advance. Therefore, the list of commands necessary to correctly execute Codeface4Smells analysis is:

```
# cd /vagrant/id_service/;
        nodejs id_service.js ../codeface.conf &
        cd ..
# codeface run -p conf_file dir_out dir_git
# codeface ml -p conf_file dir_out dir_git
# codeface st -p conf_file dir_out
```

Where the following parameters should be substituted:

- *conf_file*: path to the project's configuration file;
- *dir_git*: path to the directory that contains the source code of the project;
- *dir_ml*: path to the directory that contains the mailing list archive files;
- *dir_out*: path to the directory that will contain the analysis output.

The first command starts the ID manager service, that is needed to handle the identification and management of developer identities. The second command executes the collaboration analysis and the third one executes the communication analysis. The last command starts Codeface4Smells analysis, which uses both the collaboration and communication analysis output generated by previous commands.

B.2 Project configuration

To perform an analysis with Codeface and Codeface4Smells, it is necessary to specify a configuration file that contains all the necessary information related to the project in analysis. The configuration file has to be specified in every command that perform an analysis, it must have a ".conf" extension and it should contain at least the following parameters:

- *project*: name of the project to analyse;
- *repo*: name of the directory containing the source code;
- *mailinglists*: lists of mailing lists names and typologies;

- *description*: description of the project to analyse;
- *revisions*: lists of versions to analyse. If this parameter is not present all the project history will be analysed using three months windows. It is possible to set this parameter to “3months” in order to analyse at most the last three years of activity using three months windows;
- *tagging*: collaboration detection method.

An example of a valid configuration file, which allows to analyse Cassandra project, is the following:

```
project: Cassandra
repo: cassandra
mailinglists:
        - {name: gmane.comp.db.cassandra.devel,
           type: dev, source: gmane}
description: Cassandra project
revisions: 3months
tagging: proximity
```

B.3 Analyse high-volume communities

In order to positively conclude collaboration analysis, provided by *Codeface*, of high-volume software development communities (e.g., Firefox and LibreOffice) it was necessary to make the following changes:

- Increase the value of the variable «*max_packet_size*» in “*codeface/dbmanager.py*”
- Add the following parameters to MySQL configuration file:

```
[mysqld]
innodb_buffer_pool_size = 3G
innodb_buffer_pool_instances = 3
wait_timeout = 3600
innodb_log_buffer_size = 800M
net_read_timeout = 600
net_write_timeout = 600
innodb_lock_wait_timeout = 500

[mysqldump]
max_allowed_packet = 1024M
```

B.4 Utility tools

This section illustrates all utility scripts specifically developed to address some problematics or necessities occurred during this master thesis execution. The first script can be used before the collaboration analysis in order to purge unsupported characters from a mailing list archive, to avoid undesired behaviors. The second script generates an Excel file that sums up all the relevant correlations found between Community Smells and quality factors of a set of Codeface4Smells analysis results. Finally, the third and last script generates scatter plots of some specific quality factors, considering a set of Codeface4Semlls analysis results. All the developed utility scripts are available on-line [75].

1. **Purge mailing list archive**

   Codeface comes with an R file which allows to download a specific project's mailing list archive from `www.gmane.org`, generating a valid «.mbox» file that can be used to execute Codeface collaboration analysis. In some unlucky cases it is possible that downloaded mailing list archives contain non-readable or invalid characters, which cause Codeface to crash without successfully terminating the collaboration analysis of a project. This situation is likely to happen when many Asiatic or non-common characters are used in mailing list messages or within user e-mail addresses.

   A python script was created to purge a mailing list archive from all these invalid characters and enhance the success probability of communication analysis. The sequence of steps that should be followed to obtain a bullet proof mailing list archive is the following:

   (a) Download a mailing list archive using the script provided by Codeface: «*codeface/R/ml/download.r*»;

   (b) Rename the mailing list archive as «mail.mbox» and move it in the directory with our script;

   (c) Execute the script «*clean_mail.py*»;

   (d) Rename «*mail-clear.mbox*» to the appropriate mailing list name specified in the configuration file.

2. **Correlations summary**

   Codeface4Smells enables to measure at what extent Community Smells are influenced by socio-technical quality factors. These relationships are explored by calculating Pearson and Spearman correlation analysis and the results of such analysis are included in the project report produced by Codeface4Smells. Pearson and Spearman methodologies are both considered because the first focuses on linear relationships while the last one focuses on not linear ones.

The «correlation_results.r» R script was developed to automatically generate an Excel file that summarises all the relevant correlation found analysing FLOSS projects. It considers all Codeface4Smells analysis results present at its same hierarchical level and extracts all the correlation information from them, identifies all the important correlations (a correlation is interesting if its p-value is less than 0.05) and groups relevant correlations with respect to the software development community size. If both Pearson and Spearman methodologies detect a relevant correlation between a Community Smell and a socio-technical quality factor, the generated summary will show the one with the minor p-value.

3. **Scatter plots**

   In order to understand to what extent Community Smells are influenced by different socio-technical quality factors, we generated scatter plots of every relevant correlation found between Community Smells and quality factors belonging to our Socio-technical Quality Framework. Scatter plots were fundamental to identify thresholds capable of suggesting healthy values of some socio-technical quality factors. To provide useful models the script removes outliers and provides two regression model approaches:

   (a) loess model: «*scatter_plots_loess.r*»

   (b) linear model: «*scatter_plots_lm.r*»

Appendix C

Reports of reference projects

This appendix includes detailed results obtained from the execution of Codeface4Smells analysis on the following three reference software projects: Firefox, LibreOffice and FFmpeg. For each reference project the following information are presented:

1. *Report of Codeface4Smells analysis*: constituted by the quantification of quality factors, belonging to our Socio-technical Quality Framework, and Community Smells for all the twelve ranges, of three months each, considered in the analysis;
2. *Pearson's correlation analysis*: constituted by the correlation coefficient and its associated p-value of every different couple of quality factors, belonging to the Socio-technical Quality Framework, and/or Community Smells;
3. *Spearman's correlation analysis*: as Pearson's correlation analysis, it is constituted by the correlation coefficient and its associated p-value of every different couple of quality factors, belonging to the Socio-technical Quality Framework, and/or Community Smells.

The complete datasets of results and graph reports, that summarise trends of important quality factors with respect to every range and to the entire analysis time period, of the three reference projects are accessible on-line [75], where it is also possible to retrieve Codeface4Smells analysis results of all the 60 software development communities analysed during the evaluation part of this master thesis.

C.1 Firefox

	range.date	devs	ml.only.devs	code.only.devs	ml.code.devs	perc.ml.only.devs	perc.code.only.devs	perc.ml.code.devs	sponsored.devs	ratio.sponsored	sponsored.core.devs	ratio.sponsored.core	num.tz	core.global.devs	core.mail.devs	core.code.devs	org.silo	prima.donnas	radio.silence	black.cloud	missing.links	st.congruence	communicability	global.turnover	code.turnover
1	2013-06 – 2013-09	568	45	483	40	0.0792	0.8504	0.0704	66	0.1162	2	0.0038	28	188	40	168	5400	0	24	0	5472	0.0074	0.8225	0.0000	0.0000
2	2013-09 – 2013-12	593	39	510	44	0.0658	0.8600	0.0742	90	0.1518	4	0.0072	32	164	38	155	6698	0	19	0	6784	0.0029	0.8040	0.2705	0.2470
3	2013-12 – 2014-03	652	32	587	33	0.0491	0.9003	0.0506	104	0.1595	4	0.0065	30	187	30	182	6707	0	7	0	6784	0.0019	0.8313	0.2779	0.2572
4	2014-03 – 2014-06	642	32	570	40	0.0498	0.8879	0.0623	80	0.1246	10	0.0164	31	188	32	179	7732	0	22	0	7801	0.0028	0.8090	0.3648	0.3496
5	2014-06 – 2014-09	723	44	643	36	0.0609	0.8893	0.0498	88	0.1217	6	0.0088	31	220	40	213	10224	0	14	0	10327	0.0039	0.8062	0.2930	0.2839
6	2014-09 – 2014-12	552	29	488	35	0.0525	0.8841	0.0634	55	0.0996	3	0.0057	32	160	30	153	3797	0	4	0	3841	0.0044	0.8749	0.4831	0.4576
7	2014-12 – 2015-03	695	33	621	41	0.0475	0.8935	0.0590	71	0.1022	5	0.0076	30	180	31	175	7169	0	5	0	7317	0.0044	0.8303	0.2566	0.2363
8	2015-03 – 2015-05	590	31	525	34	0.0525	0.8898	0.0576	63	0.1068	6	0.0107	36	153	28	145	6414	2	16	0	6522	0.0044	0.8109	0.4405	0.4292
9	2015-05 – 2015-08	681	68	578	35	0.0999	0.8488	0.0514	71	0.1043	4	0.0065	31	188	50	168	9841	2	10	0	9994	0.0027	0.7734	0.2785	0.2935
10	2015-08 – 2015-11	542	37	469	36	0.0683	0.8653	0.0664	47	0.0867	2	0.0040	34	144	33	132	4578	0	2	0	4660	0.0051	0.8131	0.4481	0.4132
11	2015-11 – 2016-02	744	51	638	55	0.0685	0.8575	0.0739	78	0.1048	10	0.0144	31	191	47	179	9892	0	29	0	10165	0.0093	0.7990	0.2146	0.1987
12	2016-02 – 2016-05	459	65	349	45	0.1416	0.7603	0.0980	34	0.0741	6	0.0152	34	140	48	112	1745	0	27	0	1788	0.0116	0.8807	0.6101	0.6495

	core.global.turnover	core.mail.turnover	core.code.turnover	ratio.smelly.quitters	ratio.smelly.devs	global.truck	mail.truck	code.truck	closeness.centr	betweenness.centr	degree.centr	global.mod	mail.mod	code.mod	density	mail.only.core.devs	code.only.core.devs	ml.code.core.devs	ratio.mail.only.core	ratio.code.only.core	ratio.ml.code.core
1	0.0000	0.0000	0.0000	0.0000	0.7870	0.6690	0.5294	0.6788	0.0030	0.0686	0.2882	0.4853	0.1881	0.4809	0.0381	33	161	7	0.1642	0.8010	0.0348
2	0.4318	0.7436	0.3963	0.6051	0.7437	0.7234	0.5422	0.7202	0.0028	0.0795	0.3467	0.4800	0.4531	0.4809	0.0401	28	145	10	0.1530	0.7923	0.0546
3	0.3134	0.6471	0.3145	0.5376	0.7408	0.7132	0.5385	0.7065	0.0024	0.0997	0.3590	0.5443	0.4648	0.5421	0.0328	20	172	10	0.0990	0.8515	0.0495
4	0.3893	0.4839	0.4044	0.5254	0.7882	0.7072	0.5556	0.7066	0.0027	0.1172	0.4067	0.4826	0.2101	0.4831	0.0395	21	168	11	0.1050	0.8400	0.0550
5	0.2892	0.4167	0.2857	0.5800	0.7870	0.6957	0.5000	0.6863	0.0024	0.0370	0.2833	0.5648	0.3846	0.5680	0.0408	26	199	14	0.1088	0.8326	0.0586
6	0.5000	0.6857	0.5137	0.6169	0.7355	0.7101	0.5312	0.7075	0.0029	0.0603	0.1879	0.5946	0.4845	0.5948	0.0262	17	140	13	0.1000	0.8235	0.0765
7	0.3000	0.7213	0.3049	0.5125	0.6892	0.7410	0.5811	0.7356	0.0021	0.0775	0.3418	0.5285	0.5928	0.5381	0.0314	18	162	13	0.0933	0.8394	0.0674
8	0.4745	0.7458	0.4938	0.4346	0.7305	0.7407	0.5692	0.7406	0.0026	0.0431	0.2562	0.5546	0.3806	0.5549	0.0392	18	135	10	0.1104	0.8282	0.0613
9	0.2170	0.3846	0.2620	0.5311	0.7313	0.7239	0.5146	0.7259	0.0025	0.0767	0.3494	0.5125	0.3648	0.5078	0.0462	37	155	13	0.1805	0.7561	0.0634
10	0.5000	0.9157	0.4867	0.5365	0.7472	0.7343	0.5479	0.7386	0.0031	0.1423	0.4062	0.4726	0.4125	0.4717	0.0337	22	121	11	0.1429	0.7857	0.0714
11	0.1552	0.3750	0.1543	0.5145	0.7661	0.7433	0.5566	0.7417	0.0022	0.0411	0.2891	0.4695	0.3499	0.4743	0.0393	25	157	22	0.1225	0.7696	0.1078
12	0.5801	0.5684	0.6598	0.6458	0.6667	0.6950	0.5636	0.7157	0.0034	0.1546	0.3044	0.5355	0.3805	0.5524	0.0231	31	95	17	0.2168	0.6643	0.1189

Figure C.1: Codeface4Smells analysis report - Firefox

	devs	ml.only.devs	code.only.devs	ml.code.devs	perc.ml.only.devs	perc.code.only.devs	perc.ml.code.devs	sponsored.devs	ratio.sponsored	sponsored.core.devs	ratio.sponsored.core	num.tz	core.global.devs	core.mail.devs	core.code.devs	org.silo	prima.donnas	radio.silence	black.cloud	missing.links	st.congruence	communicability	global.turnover
org.silo	0.94	0.10	0.91	0.10	-0.36	0.49	-0.63	0.69	0.39	0.43	0.09	-0.36	0.82	0.22	0.82	-	0.26	0.06	-	1.00	-0.41	-0.83	-0.86
prima.donnas	0.09	0.26	0.07	-0.37	0.11	0.05	-0.35	-0.09	-0.14	-0.03	-0.03	0.40	-0.10	0.11	-0.12	0.26	-	-0.10	-	0.26	-0.24	-0.45	0.01
radio.silence	-0.03	0.43	-0.15	0.71	0.43	-0.53	0.60	-0.03	-0.02	0.58	0.67	-0.06	0.11	0.54	-0.04	0.06	-0.10	-	-	0.06	0.61	-0.05	0.03
black.cloud	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
missing.links	0.94	0.10	0.91	0.12	-0.36	0.48	-0.62	0.69	0.38	0.43	0.10	-0.36	0.81	0.23	0.82	1.00	0.26	0.06	-	-	-0.40	-0.83	-0.86

	code.turnover	core.global.turnover	core.mail.turnover	core.code.turnover	ratio.smelly.quitters	ratio.smelly.devs	global.truck	mail.truck	code.truck	closeness.centr	betweenness.centr	degree.centr	global.mod	mail.mod	code.mod	density	mail.only.core.devs	code.only.core.devs	ml.code.core.devs	ratio.mail.only.core	ratio.code.only.core	ratio.ml.code.core
org.silo	-0.82	-0.90	-0.62	-0.91	-0.45	0.52	0.28	-0.36	0.05	-0.81	-0.61	0.17	-0.19	-0.10	-0.23	0.82	0.14	0.78	0.21	-0.36	0.38	-0.26
prima.donnas	0.05	-0.12	-0.12	-0.04	-0.55	-0.15	0.33	-0.05	0.36	-0.15	-0.28	-0.11	0.17	-0.07	0.12	0.48	0.20	-0.10	-0.13	0.15	-0.06	-0.12
radio.silence	0.11	-0.08	-0.54	-0.02	0.10	0.18	-0.31	0.14	-0.16	0.17	-0.05	-0.09	-0.40	-0.64	-0.35	0.16	0.44	-0.09	0.35	0.42	-0.46	0.31
black.cloud	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
missing.links	-0.82	-0.91	-0.62	-0.92	-0.46	0.51	0.29	-0.35	0.06	-0.81	-0.61	0.17	-0.19	-0.10	-0.24	0.82	0.14	0.78	0.22	-0.35	0.38	-0.25

(a) Pearson's correlation (coefficient)

	devs	ml.only.devs	code.only.devs	ml.code.devs	perc.ml.only.devs	perc.code.only.devs	perc.ml.code.devs	sponsored.devs	ratio.sponsored	sponsored.core.devs	ratio.sponsored.core	num.tz	core.global.devs	core.mail.devs	core.code.devs	org.silo	prima.donnas	radio.silence	black.cloud	missing.links	st.congruence	communicability	global.turnover
org.silo	0.00	0.76	0.00	0.75	0.24	0.11	0.03	0.01	0.21	0.16	0.77	0.25	0.00	0.49	0.00	-	0.41	0.85	-	0.00	0.18	0.00	0.00
prima.donnas	0.79	0.41	0.82	0.23	0.73	0.89	0.26	0.79	0.67	0.93	0.92	0.20	0.77	0.74	0.70	0.41	-	0.77	-	0.41	0.45	0.15	0.99
radio.silence	0.92	0.16	0.64	0.01	0.16	0.08	0.04	0.93	0.94	0.05	0.02	0.85	0.74	0.07	0.90	0.85	0.77	-	-	0.84	0.03	0.88	0.93
black.cloud	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
missing.links	0.00	0.75	0.00	0.72	0.25	0.11	0.03	0.01	0.22	0.16	0.77	0.25	0.00	0.48	0.00	0.00	0.41	0.84	-	-	0.20	0.00	0.00

	code.turnover	core.global.turnover	core.mail.turnover	core.code.turnover	ratio.smelly.quitters	ratio.smelly.devs	global.truck	mail.truck	code.truck	closeness.centr	betweenness.centr	degree.centr	global.mod	mail.mod	code.mod	density	mail.only.core.devs	code.only.core.devs	ml.code.core.devs	ratio.mail.only.core	ratio.code.only.core	ratio.ml.code.core
org.silo	0.00	0.00	0.04	0.00	0.16	0.09	0.38	0.26	0.89	0.00	0.03	0.60	0.56	0.75	0.47	0.00	0.67	0.00	0.51	0.26	0.22	0.42
prima.donnas	0.87	0.73	0.73	0.91	0.08	0.65	0.30	0.89	0.24	0.64	0.38	0.72	0.60	0.83	0.72	0.11	0.53	0.75	0.69	0.64	0.86	0.72
radio.silence	0.75	0.81	0.09	0.95	0.77	0.57	0.33	0.67	0.62	0.60	0.88	0.79	0.19	0.02	0.27	0.62	0.15	0.77	0.27	0.17	0.13	0.32
black.cloud	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
missing.links	0.00	0.00	0.04	0.00	0.16	0.09	0.36	0.27	0.86	0.00	0.03	0.61	0.55	0.76	0.46	0.00	0.67	0.00	0.49	0.26	0.23	0.44

(b) Pearson's correlation (p-value)

Figure C.2: Pearson's correlation analysis - Firefox

	devs	ml.only.devs	code.only.devs	ml.code.devs	perc.ml.only.devs	perc.code.only.devs	perc.ml.code.devs	sponsored.devs	ratio.sponsored	sponsored.core.devs	ratio.sponsored.core	num.tz	core.global.devs	core.mail.devs	core.code.devs	org.silo	prima.donnas	radio.silence	black.cloud	missing.links	st.congruence	communicability	global.turnover
org.silo	0.95	0.24	0.92	0.05	-0.19	0.24	-0.52	0.71	0.48	0.52	0.33	-0.46	0.86	0.24	0.84	-	0.13	0.15	-	1.00	-0.45	-0.73	-0.69
prima.donnas	0.06	0.06	0.06	-0.49	0.16	-0.06	-0.39	-0.16	-0.06	0.07	0.03	0.30	-0.07	0.00	-0.23	0.13	-	-0.06	-	0.13	-0.26	-0.39	0.07
radio.silence	0.10	0.49	-0.01	0.62	0.47	-0.50	0.50	0.11	0.21	0.56	0.56	-0.06	0.29	0.50	0.08	0.15	-0.06	-	-	0.16	0.37	-0.24	-0.18
black.cloud	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
missing.links	0.95	0.25	0.90	0.08	-0.18	0.21	-0.49	0.71	0.48	0.52	0.34	-0.44	0.85	0.26	0.82	1.00	0.13	0.16	-	-	-0.44	-0.76	-0.70

	code.turnover	core.global.turnover	core.mail.turnover	core.code.turnover	ratio.smelly.quitters	ratio.smelly.devs	global.truck	mail.truck	code.truck	closeness.centr	betweenness.centr	degree.centr	global.mod	mail.mod	code.mod	density	mail.only.core.devs	code.only.core.devs	ml.code.core.devs	ratio.mail.only.core	ratio.code.only.core	ratio.ml.code.core
org.silo	-0.70	-0.94	-0.67	-0.91	-0.43	0.41	0.29	-0.22	0.03	-0.82	-0.48	0.16	-0.16	-0.19	-0.11	0.73	0.14	0.73	0.28	-0.25	0.27	-0.22
prima.donnas	0.22	-0.15	0.00	-0.07	-0.45	-0.39	0.32	0.00	0.39	-0.13	-0.26	-0.13	0.19	-0.19	0.19	0.39	0.10	-0.26	-0.13	0.26	-0.19	0.00
radio.silence	-0.14	-0.19	-0.51	-0.12	-0.05	0.23	-0.24	0.17	-0.05	0.07	-0.10	-0.13	-0.34	-0.74	-0.22	0.21	0.50	0.01	0.14	0.48	-0.31	-0.03
black.cloud	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
missing.links	-0.71	-0.94	-0.66	-0.91	-0.42	0.41	0.30	-0.21	0.05	-0.80	-0.48	0.15	-0.18	-0.19	-0.12	0.76	0.15	0.71	0.28	-0.23	0.24	-0.21

(a) Spearman's correlation (coefficient)

	devs	ml.only.devs	code.only.devs	ml.code.devs	perc.ml.only.devs	perc.code.only.devs	perc.ml.code.devs	sponsored.devs	ratio.sponsored	sponsored.core.devs	ratio.sponsored.core	num.tz	core.global.devs	core.mail.devs	core.code.devs	org.silo	prima.donnas	radio.silence	black.cloud	missing.links	st.congruence	communicability	global.turnover
org.silo	0.00	0.46	0.00	0.88	0.55	0.46	0.08	0.01	0.12	0.09	0.29	0.13	0.00	0.45	0.00	-	0.69	0.65	-	0.00	0.14	0.01	0.02
prima.donnas	0.84	0.84	0.84	0.11	0.61	0.84	0.21	0.61	0.84	0.84	0.92	0.35	0.84	1.00	0.48	0.69	-	0.84	-	0.69	0.41	0.21	0.83
radio.silence	0.75	0.10	0.99	0.03	0.13	0.10	0.10	0.75	0.51	0.06	0.06	0.85	0.36	0.10	0.79	0.65	0.84	-	-	0.62	0.23	0.44	0.60
black.cloud	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
missing.links	0.00	0.43	0.00	0.80	0.59	0.50	0.10	0.01	0.11	0.09	0.28	0.15	0.00	0.42	0.00	0.00	0.69	0.62	-	-	0.15	0.00	0.02

	code.turnover	core.global.turnover	core.mail.turnover	core.code.turnover	ratio.smelly.quitters	ratio.smelly.devs	global.truck	mail.truck	code.truck	closeness.centr	betweenness.centr	degree.centr	global.mod	mail.mod	code.mod	density	mail.only.core.devs	code.only.core.devs	ml.code.core.devs	ratio.mail.only.core	ratio.code.only.core	ratio.ml.code.core
org.silo	0.02	0.00	0.03	0.00	0.19	0.19	0.35	0.50	0.92	0.00	0.12	0.62	0.62	0.56	0.75	0.01	0.67	0.01	0.37	0.43	0.40	0.50
prima.donnas	0.51	0.66	1.00	0.83	0.17	0.21	0.30	1.00	0.21	0.69	0.42	0.69	0.55	0.55	0.54	0.21	0.76	0.42	0.68	0.42	0.55	1.00
radio.silence	0.69	0.58	0.11	0.73	0.90	0.47	0.44	0.59	0.89	0.83	0.77	0.70	0.29	0.01	0.48	0.51	0.10	0.97	0.66	0.12	0.32	0.92
black.cloud	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
missing.links	0.02	0.00	0.03	0.00	0.20	0.18	0.35	0.50	0.88	0.00	0.11	0.63	0.57	0.55	0.71	0.00	0.63	0.01	0.37	0.48	0.45	0.50

(b) Spearman's correlation (p-value)

Figure C.3: Spearman's correlation analysis - Firefox

C.2 LibreOffice

	range.date	devs	ml.only.devs	code.only.devs	ml.code.devs	perc.ml.only.devs	perc.code.only.devs	perc.ml.code.devs	sponsored.devs	ratio.sponsored	sponsored.core.devs	ratio.sponsored.core	num.tz	core.global.devs	core.mail.devs	core.code.devs	org.silo	prima.donnas	radio.silence	black.cloud	missing.links	st.congruence	communicability	global.turnover	code.turnover
1	2013-06 – 2013-09	275	120	78	77	0.4364	0.2836	0.2800	37	0.1345	2	0.0129	24	78	69	48	392	2	93	0	807	0.1505	0.7374	0.0000	0.0000
2	2013-09 – 2013-12	240	90	90	60	0.3750	0.3750	0.2500	35	0.1458	8	0.0533	28	82	52	56	682	4	66	0	901	0.1175	0.7296	0.5437	0.4525
3	2013-12 – 2014-03	240	90	88	62	0.3750	0.3667	0.2583	23	0.0958	4	0.0267	26	76	53	54	943	0	60	0	1306	0.0705	0.6298	0.4333	0.3400
4	2014-03 – 2014-06	224	85	84	55	0.3795	0.3750	0.2455	34	0.1518	0	0.0000	27	61	54	43	754	0	43	0	987	0.0827	0.6336	0.4914	0.4567
5	2014-06 – 2014-09	212	76	80	56	0.3585	0.3774	0.2642	27	0.1274	0	0.0000	28	63	50	43	569	2	61	0	778	0.0922	0.6908	0.4587	0.3709
6	2014-09 – 2014-12	197	71	71	55	0.3604	0.3604	0.2792	25	0.1269	0	0.0000	36	61	47	38	355	5	49	0	593	0.1671	0.7137	0.4890	0.4351
7	2014-12 – 2015-03	224	83	88	53	0.3705	0.3929	0.2366	36	0.1607	2	0.0142	30	68	44	41	397	0	53	0	537	0.1435	0.7112	0.3895	0.3371
8	2015-03 – 2015-05	223	82	91	50	0.3677	0.4081	0.2242	45	0.2018	3	0.0213	32	61	40	44	507	0	70	0	661	0.1337	0.6786	0.4743	0.4113
9	2015-05 – 2015-08	182	70	57	55	0.3846	0.3132	0.3022	18	0.0989	2	0.0179	33	60	46	38	337	0	49	0	489	0.1877	0.6846	0.6074	0.5217
10	2015-08 – 2015-11	214	74	79	61	0.3458	0.3692	0.2850	28	0.1308	1	0.0071	36	63	47	45	485	0	33	0	717	0.1115	0.6514	0.3535	0.2857
11	2015-11 – 2016-02	250	97	72	81	0.3880	0.2880	0.3240	40	0.1600	2	0.0131	32	68	51	43	283	6	102	1	504	0.1385	0.7384	0.4138	0.3276
12	2016-02 – 2016-05	238	93	82	63	0.3908	0.3445	0.2647	17	0.0714	2	0.0138	35	67	44	46	426	0	83	2	609	0.0978	0.7041	0.4713	0.3624

	core.global.turnover	core.mail.turnover	core.code.turnover	ratio.smelly.quitters	ratio.smelly.devs	global.truck	mail.truck	code.truck	closeness.centr	betweenness.centr	degree.centr	global.mod	mail.mod	code.mod	density	mail.only.core.devs	code.only.core.devs	ml.code.core.devs	ratio.mail.only.core	ratio.code.only.core	ratio.ml.code.core
1	0.0000	0.0000	0.0000	0.0000	0.6327	0.7164	0.6497	0.6903	0.0076	0.1013	0.3084	0.4273	0.4072	0.4657	0.0419	44	23	25	0.4783	0.2500	0.2717
2	0.4125	0.5289	0.4038	0.5857	0.6542	0.6583	0.6533	0.6267	0.0089	0.1046	0.3631	0.3886	0.2987	0.3884	0.0553	34	38	18	0.3778	0.4222	0.2000
3	0.3165	0.3619	0.2909	0.5385	0.6292	0.6833	0.6513	0.6400	0.0089	0.1239	0.4652	0.5132	0.3688	0.5140	0.0662	36	37	17	0.4000	0.4111	0.1889
4	0.4672	0.4486	0.4124	0.4737	0.5938	0.7277	0.6143	0.6906	0.0093	0.1415	0.3974	0.2907	0.4294	0.3052	0.0555	36	25	18	0.4557	0.3165	0.2278
5	0.2097	0.3077	0.1395	0.4500	0.6792	0.7028	0.6212	0.6838	0.0097	0.1314	0.3779	0.3460	0.3634	0.3469	0.0534	34	27	16	0.4416	0.3506	0.2078
6	0.4839	0.4124	0.5679	0.6300	0.6142	0.6904	0.6270	0.6984	0.0089	0.0969	0.3890	0.2165	0.1864	0.2180	0.0651	25	16	22	0.3968	0.2540	0.3492
7	0.3566	0.4176	0.3038	0.4268	0.6250	0.6964	0.6765	0.7092	0.0086	0.1435	0.4550	0.1745	0.3654	0.1699	0.0472	29	26	15	0.4143	0.3714	0.2143
8	0.5116	0.5000	0.4000	0.5566	0.6726	0.7265	0.6970	0.6879	0.0082	0.1381	0.4258	0.0231	0.2773	0.0270	0.0426	23	27	17	0.3433	0.4030	0.2537
9	0.4628	0.3953	0.5610	0.6341	0.6099	0.6703	0.6320	0.6607	0.0122	0.0972	0.3958	0.2644	0.3102	0.2767	0.0628	27	19	19	0.4154	0.2923	0.2923
10	0.2927	0.3656	0.2410	0.5286	0.5841	0.7056	0.6519	0.6786	0.0089	0.2193	0.5321	0.1021	0.3326	0.1086	0.0501	26	24	21	0.3662	0.3380	0.2958
11	0.2748	0.3265	0.4091	0.4583	0.6920	0.7280	0.7135	0.7190	0.0080	0.2574	0.3473	0.1185	0.2298	0.1931	0.0342	31	23	20	0.4189	0.3108	0.2703
12	0.3111	0.4842	0.3146	0.5739	0.6891	0.7185	0.7179	0.6828	0.0089	0.2226	0.4246	0.2464	0.1779	0.2844	0.0396	26	28	18	0.3611	0.3889	0.2500

Figure C.4: Codeface4Smells analysis report - LibreOffice

	devs	ml.only.devs	code.only.devs	ml.code.devs	perc.ml.only.devs	perc.code.only.devs	perc.ml.code.devs	sponsored.devs	ratio.sponsored	sponsored.core.devs	ratio.sponsored.core	num.tz	core.global.devs	core.mail.devs	core.code.devs	org.silo	prima.donnas	radio.silence	black.cloud	missing.links	st.congruence	communicability	global.turnover
org.silo	0.14	0.01	0.60	-0.26	-0.20	0.48	-0.52	-0.06	-0.10	0.34	0.31	-0.54	0.34	0.16	0.67	-	-0.33	-0.30	-0.33	0.95	-0.79	-0.67	-0.00
prima.donnas	0.19	0.17	-0.24	0.49	0.08	-0.39	0.49	0.26	0.19	0.13	0.09	0.07	0.24	0.23	0.00	-0.33	-	0.44	0.09	-0.22	0.30	0.67	0.06
radio.silence	0.73	0.77	0.03	0.75	0.69	-0.59	0.28	0.32	0.06	0.24	0.21	-0.25	0.45	0.35	0.25	-0.30	0.44	-	0.68	-0.15	0.08	0.65	-0.01
black.cloud	0.45	0.58	-0.07	0.55	0.60	-0.47	0.28	-0.27	-0.42	-0.04	-0.05	0.36	0.06	-0.18	0.03	-0.33	0.09	0.68	-	-0.30	-0.14	0.35	-0.09
missing.links	0.33	0.23	0.52	-0.01	0.02	0.25	-0.37	-0.05	-0.16	0.31	0.27	-0.65	0.48	0.44	0.73	0.95	-0.22	-0.15	-0.30	-	-0.73	-0.57	-0.06

	code.turnover	core.global.turnover	core.mail.turnover	core.code.turnover	ratio.smelly.quitters	ratio.smelly.devs	global.truck	mail.truck	code.truck	closeness.centr	betweenness.centr	degree.centr	global.mod	mail.mod	code.mod	density	mail.only.core.devs	code.only.core.devs	ml.code.core.devs	ratio.mail.only.core	ratio.code.only.core	ratio.ml.code.core
org.silo	-0.00	-0.03	0.12	-0.34	-0.10	-0.18	-0.21	-0.34	-0.64	-0.01	-0.26	0.26	0.58	0.49	0.51	0.48	0.38	0.76	-0.39	-0.01	0.56	-0.69
prima.donnas	0.03	-0.08	-0.17	0.31	0.05	0.34	-0.08	0.06	0.20	-0.30	0.07	-0.60	-0.01	-0.44	0.06	-0.12	0.10	-0.16	0.38	0.14	-0.33	0.30
radio.silence	-0.19	-0.29	0.02	-0.02	-0.15	0.78	0.36	0.64	0.24	-0.51	0.30	-0.65	0.08	-0.29	0.21	-0.69	0.31	0.11	0.24	0.14	-0.02	-0.09
black.cloud	-0.27	-0.34	0.12	-0.08	0.02	0.59	0.41	0.72	0.25	-0.22	0.69	-0.15	-0.13	-0.63	-0.01	-0.63	-0.20	-0.00	0.09	-0.25	0.10	0.06
missing.links	-0.07	-0.09	0.02	-0.32	-0.08	-0.20	-0.16	-0.37	-0.60	-0.15	-0.29	0.12	0.71	0.54	0.67	0.43	0.58	0.69	-0.12	0.16	0.36	-0.59

(a) Pearson's correlation (coefficient)

	devs	ml.only.devs	code.only.devs	ml.code.devs	perc.ml.only.devs	perc.code.only.devs	perc.ml.code.devs	sponsored.devs	ratio.sponsored	sponsored.core.devs	ratio.sponsored.core	num.tz	core.global.devs	core.mail.devs	core.code.devs	org.silo	prima.donnas	radio.silence	black.cloud	missing.links	st.congruence	communicability	global.turnover
org.silo	0.66	0.98	0.04	0.42	0.54	0.12	0.09	0.84	0.76	0.28	0.32	0.07	0.29	0.61	0.02	-	0.30	0.34	0.32	0.00	0.00	0.02	1.00
prima.donnas	0.56	0.61	0.46	0.10	0.81	0.21	0.11	0.42	0.56	0.70	0.78	0.83	0.46	0.47	0.99	0.30	-	0.15	0.79	0.48	0.35	0.02	0.86
radio.silence	0.01	0.00	0.92	0.01	0.01	0.04	0.38	0.31	0.85	0.45	0.51	0.44	0.14	0.26	0.44	0.34	0.15	-	0.02	0.63	0.81	0.02	0.97
black.cloud	0.16	0.06	0.84	0.08	0.05	0.14	0.40	0.43	0.20	0.92	0.89	0.28	0.85	0.60	0.93	0.32	0.79	0.02	-	0.37	0.69	0.29	0.79
missing.links	0.29	0.46	0.08	0.97	0.94	0.44	0.24	0.88	0.62	0.33	0.40	0.02	0.11	0.16	0.01	0.00	0.48	0.63	0.37	-	0.01	0.05	0.87

	code.turnover	core.global.turnover	core.mail.turnover	core.code.turnover	ratio.smelly.quitters	ratio.smelly.devs	global.truck	mail.truck	code.truck	closeness.centr	betweenness.centr	degree.centr	global.mod	mail.mod	code.mod	density	mail.only.core.devs	code.only.core.devs	ml.code.core.devs	ratio.mail.only.core	ratio.code.only.core	ratio.ml.code.core
org.silo	0.99	0.93	0.72	0.30	0.77	0.58	0.51	0.27	0.02	0.97	0.41	0.41	0.05	0.11	0.09	0.12	0.22	0.00	0.20	0.98	0.06	0.01
prima.donnas	0.94	0.80	0.63	0.35	0.87	0.28	0.80	0.85	0.54	0.35	0.82	0.04	0.98	0.15	0.86	0.72	0.75	0.61	0.22	0.66	0.29	0.35
radio.silence	0.57	0.39	0.95	0.96	0.67	0.00	0.24	0.02	0.46	0.09	0.34	0.02	0.81	0.37	0.52	0.01	0.32	0.73	0.45	0.66	0.94	0.78
black.cloud	0.43	0.30	0.73	0.82	0.96	0.05	0.21	0.01	0.46	0.51	0.02	0.66	0.70	0.04	0.97	0.04	0.55	1.00	0.80	0.45	0.76	0.85
missing.links	0.84	0.80	0.95	0.33	0.82	0.52	0.61	0.24	0.04	0.64	0.36	0.70	0.01	0.07	0.02	0.16	0.05	0.01	0.71	0.62	0.25	0.04

(b) Pearson's correlation (p-value)

Figure C.5: Pearson's correlation analysis - LibreOffice

	devs	ml.only.devs	code.only.devs	ml.code.devs	perc.ml.only.devs	perc.code.only.devs	perc.ml.code.devs	sponsored.devs	ratio.sponsored	sponsored.core.devs	ratio.sponsored.core	num.tz	core.global.devs	core.mail.devs	core.code.devs	org.silo	prima.donnas	radio.silence	black.cloud	missing.links	st.congruence	communicability	global.turnover
org.silo	0.11	0.08	0.75	-0.15	-0.30	0.59	-0.67	-0.09	-0.03	0.18	0.18	-0.49	0.18	0.29	0.55	-	-0.37	-0.23	-0.42	0.87	-0.85	-0.64	0.09
prima.donnas	0.25	0.23	-0.36	0.37	0.06	-0.34	0.38	0.29	0.12	-0.10	-0.23	-0.02	0.32	0.37	-0.04	-0.37	-	0.43	0.20	-0.11	0.32	0.79	0.07
radio.silence	0.67	0.75	0.12	0.51	0.56	-0.30	0.08	0.39	0.16	0.47	0.30	-0.28	0.53	0.07	0.40	-0.23	0.43	-	0.66	-0.07	0.08	0.62	-0.05
black.cloud	0.50	0.66	-0.20	0.67	0.68	-0.58	0.42	-0.13	-0.20	0.08	-0.07	0.28	0.25	-0.15	0.18	-0.42	0.20	0.66	-	-0.35	-0.03	0.42	-0.20
missing.links	0.38	0.33	0.52	0.17	-0.07	0.23	-0.43	0.01	-0.07	0.14	0.03	-0.68	0.40	0.66	0.70	0.87	-0.11	-0.07	-0.35	-	-0.72	-0.40	0.02

	code.turnover	core.global.turnover	core.mail.turnover	core.code.turnover	ratio.smelly.quitters	ratio.smelly.devs	global.truck	mail.truck	code.truck	closeness.centr	betweenness.centr	degree.centr	global.mod	mail.mod	code.mod	density	mail.only.core.devs	code.only.core.devs	ml.code.core.devs	ratio.mail.only.core	ratio.code.only.core	ratio.ml.code.core
org.silo	0.15	0.06	0.19	-0.42	-0.14	-0.10	-0.13	-0.22	-0.50	0.30	0.02	0.38	0.42	0.45	0.42	0.39	0.36	0.77	-0.54	-0.14	0.69	-0.73
prima.donnas	-0.01	-0.20	-0.24	0.31	0.02	0.42	-0.02	-0.05	0.31	-0.28	-0.21	-0.81	0.11	-0.32	0.17	-0.13	0.14	-0.27	0.42	0.27	-0.32	0.15
radio.silence	-0.19	-0.26	0.11	-0.09	-0.06	0.91	0.34	0.58	0.16	-0.63	0.20	-0.55	0.10	-0.31	0.19	-0.69	0.16	0.24	0.01	0.04	0.12	-0.20
black.cloud	-0.35	-0.43	-0.03	0.05	-0.04	0.66	0.50	0.67	0.26	-0.35	0.66	-0.26	-0.20	-0.61	-0.05	-0.66	-0.14	0.04	0.20	-0.12	-0.04	0.13
missing.links	0.06	-0.03	0.10	-0.41	-0.15	-0.11	-0.06	-0.34	-0.41	0.09	-0.14	0.08	0.64	0.58	0.66	0.32	0.62	0.56	-0.15	0.13	0.38	-0.56

(a) Spearman's correlation (coefficient)

	devs	ml.only.devs	code.only.devs	ml.code.devs	perc.ml.only.devs	perc.code.only.devs	perc.ml.code.devs	sponsored.devs	ratio.sponsored	sponsored.core.devs	ratio.sponsored.core	num.tz	core.global.devs	core.mail.devs	core.code.devs	org.silo	prima.donnas	radio.silence	black.cloud	missing.links	st.congruence	communicability	global.turnover
org.silo	0.73	0.81	0.01	0.65	0.34	0.04	0.02	0.78	0.94	0.57	0.58	0.10	0.57	0.36	0.06	-	0.23	0.48	0.20	0.00	0.00	0.03	0.80
prima.donnas	0.44	0.46	0.25	0.23	0.85	0.28	0.22	0.37	0.72	0.75	0.47	0.95	0.31	0.24	0.90	0.23	-	0.17	0.55	0.73	0.32	0.00	0.83
radio.silence	0.02	0.01	0.71	0.09	0.06	0.34	0.80	0.21	0.62	0.12	0.34	0.38	0.07	0.84	0.20	0.48	0.17	-	0.03	0.82	0.80	0.03	0.88
black.cloud	0.12	0.03	0.55	0.03	0.02	0.06	0.20	0.69	0.55	0.82	0.84	0.41	0.45	0.66	0.60	0.20	0.55	0.03	-	0.29	0.94	0.20	0.55
missing.links	0.22	0.30	0.08	0.60	0.83	0.46	0.17	0.97	0.83	0.67	0.93	0.02	0.20	0.02	0.01	0.00	0.73	0.82	0.29	-	0.01	0.20	0.97

	code.turnover	core.global.turnover	core.mail.turnover	core.code.turnover	ratio.smelly.quitters	ratio.smelly.devs	global.truck	mail.truck	code.truck	closeness.centr	betweenness.centr	degree.centr	global.mod	mail.mod	code.mod	density	mail.only.core.devs	code.only.core.devs	ml.code.core.devs	ratio.mail.only.core	ratio.code.only.core	ratio.ml.code.core
org.silo	0.65	0.86	0.58	0.20	0.69	0.75	0.70	0.48	0.10	0.34	0.96	0.22	0.18	0.15	0.18	0.21	0.26	0.00	0.07	0.67	0.02	0.01
prima.donnas	0.98	0.56	0.47	0.36	0.95	0.18	0.96	0.88	0.32	0.39	0.50	0.00	0.74	0.30	0.59	0.68	0.66	0.40	0.18	0.39	0.31	0.65
radio.silence	0.58	0.45	0.74	0.80	0.86	0.00	0.29	0.05	0.62	0.03	0.53	0.06	0.76	0.33	0.55	0.01	0.62	0.45	0.98	0.91	0.72	0.54
black.cloud	0.29	0.19	0.94	0.87	0.91	0.03	0.12	0.02	0.45	0.29	0.03	0.45	0.55	0.05	0.87	0.03	0.69	0.91	0.55	0.72	0.91	0.69
missing.links	0.86	0.95	0.78	0.21	0.67	0.73	0.85	0.28	0.18	0.79	0.67	0.82	0.03	0.05	0.02	0.31	0.03	0.06	0.65	0.70	0.22	0.06

(b) Spearman's correlation (p-value)

Figure C.6: Spearman's correlation analysis - LibreOffice

C.3 FFmpeg

	range.date	devs	ml.only.devs	code.only.devs	ml.code.devs	perc.ml.only.devs	perc.code.only.devs	perc.ml.code.devs	sponsored.devs	ratio.sponsored	sponsored.core.devs	ratio.sponsored.core	num.tz	core.global.devs	core.mail.devs	core.code.devs	org.silo	prima.donnas	radio.silence	black.cloud	missing.links	st.congruence	communicability	global.turnover	code.turnover
1	2013-04 – 2013-07	155	61	28	66	0.3935	0.1806	0.4258	19	0.1226	1	0.0106	24	50	44	23	86	0	70	0	127	0.3351	0.8662	0.0000	0.0000
2	2013-07 – 2013-10	162	64	39	59	0.3951	0.2407	0.3642	18	0.1111	0	0.0000	29	48	38	26	147	0	78	1	164	0.2579	0.8185	0.5047	0.4479
3	2013-10 – 2014-01	174	67	41	66	0.3851	0.2356	0.3793	16	0.0920	1	0.0093	32	53	43	16	90	0	66	1	95	0.0104	0.8542	0.4821	0.4585
4	2014-01 – 2014-04	552	56	400	96	0.1014	0.7246	0.1739	24	0.0435	3	0.0060	31	60	47	34	145	0	92	2	189	0.2759	0.9676	0.1019	0.0232
5	2014-04 – 2014-07	190	68	44	78	0.3579	0.2316	0.4105	23	0.1211	3	0.0246	28	58	47	34	120	0	94	0	145	0.2857	0.8657	1.2102	1.3948
6	2014-07 – 2014-10	228	173	30	25	0.7588	0.1316	0.1096	9	0.0395	1	0.0182	30	67	59	15	63	0	72	0	66	0.0294	0.7500	0.4785	1.0508
7	2014-10 – 2015-01	662	52	505	105	0.0785	0.7628	0.1586	27	0.0408	2	0.0033	31	59	52	28	56	2	63	0	85	0.4408	0.9832	0.1551	0.0301
8	2015-01 – 2015-04	208	87	33	88	0.4183	0.1587	0.4231	26	0.1250	3	0.0248	31	63	58	30	91	0	83	0	138	0.4052	0.9077	1.2828	1.4802
9	2015-04 – 2015-07	189	68	40	81	0.3598	0.2116	0.4286	28	0.1481	3	0.0248	31	57	52	25	55	0	74	2	107	0.3554	0.8998	0.6297	0.5537
10	2015-07 – 2015-10	215	87	32	96	0.4047	0.1488	0.4465	24	0.1116	4	0.0312	30	68	59	36	83	0	100	2	177	0.4197	0.8895	0.4554	0.4498
11	2015-10 – 2016-01	190	88	21	81	0.4632	0.1105	0.4263	27	0.1421	3	0.0294	31	58	52	27	48	0	86	1	117	0.4682	0.9050	0.5827	0.6087
12	2016-01 – 2016-04	220	100	22	98	0.4545	0.1000	0.4455	23	0.1045	2	0.0167	30	62	58	28	35	4	114	2	99	0.5580	0.9398	0.4488	0.4324

	core.global.turnover	core.mail.turnover	core.code.turnover	ratio.smelly.quitters	ratio.smelly.devs	global.truck	mail.truck	code.truck	closeness.centr	betweenness.centr	degree.centr	global.mod	mail.mod	code.mod	density	mail.only.core.devs	code.only.core.devs	ml.code.core.devs	ratio.mail.only.core	ratio.code.only.core	ratio.ml.code.core
1	0.0000	0.0000	0.0000	0.0000	0.6516	0.6774	0.6535	0.7553	0.0182	0.4164	0.6626	0.1738	0.1614	0.2293	0.0517	28	7	16	0.5490	0.1373	0.3137
2	0.4082	0.5122	0.3673	0.6375	0.7284	0.7037	0.6911	0.7347	0.0198	0.4473	0.6277	0.4346	0.1263	0.4467	0.0493	25	13	13	0.4902	0.2549	0.2549
3	0.2970	0.2963	0.8095	0.7407	0.5402	0.6954	0.6767	0.8505	0.0142	0.4054	0.5823	0.4179	0.1678	0.3759	0.0478	40	13	3	0.7143	0.2321	0.0536
4	0.3894	0.3778	0.2800	0.6757	0.2500	0.8913	0.6908	0.9315	0.0012	0.0624	0.2506	0.2844	0.1679	0.4297	0.0053	30	17	17	0.4688	0.2656	0.2656
5	0.4915	0.4468	0.4706	0.1670	0.7526	0.6947	0.6781	0.7213	0.0139	0.5297	0.6935	0.2914	0.0841	0.4604	0.0366	29	16	18	0.4603	0.2540	0.2857
6	0.4480	0.3962	1.0612	0.7300	0.4430	0.7061	0.7020	0.7273	0.0175	0.5442	0.7183	0.2304	0.1610	0.4857	0.0439	55	11	4	0.7857	0.1571	0.0571
7	0.5238	0.5405	0.2791	0.4203	0.1586	0.9109	0.6688	0.9541	0.0008	0.0452	0.2103	0.3092	0.0909	0.5509	0.0030	32	8	20	0.5333	0.1333	0.3333
8	0.3934	0.3818	0.5517	0.0950	0.6058	0.6971	0.6686	0.7521	0.0127	0.4993	0.7003	0.3070	0.1752	0.5314	0.0388	38	10	20	0.5588	0.1471	0.2941
9	0.4833	0.5455	0.6909	0.5840	0.6032	0.6984	0.6510	0.7934	0.0112	0.4971	0.6894	0.2595	0.2306	0.3422	0.0341	38	11	14	0.6032	0.1746	0.2222
10	0.3200	0.3784	0.3279	0.5435	0.6977	0.6837	0.6776	0.7188	0.0161	0.3761	0.6334	0.3073	0.1625	0.4603	0.0442	37	14	22	0.5068	0.1918	0.3014
11	0.4762	0.4324	0.5714	0.7034	0.6158	0.6947	0.6923	0.7353	0.0141	0.3248	0.6040	0.1053	0.0740	0.3710	0.0521	36	11	16	0.5714	0.1746	0.2540
12	0.3167	0.2727	0.4727	0.6413	0.6455	0.7182	0.7071	0.7667	0.0151	0.4335	0.6524	0.1759	0.0874	0.4115	0.0417	34	4	24	0.5484	0.0645	0.3871

Figure C.7: Codeface4Smells analysis report - FFmpeg

	devs	ml.only.devs	code.only.devs	ml.code.devs	perc.ml.only.devs	perc.code.only.devs	perc.ml.code.devs	sponsored.devs	ratio.sponsored	sponsored.core.devs	ratio.sponsored.core	num.tz	core.global.devs	core.mail.devs	core.code.devs	org.silo	prima.donnas	radio.silence	black.cloud	missing.links	st.congruence	communicability	global.turnover
org.silo	0.06	-0.39	0.15	-0.11	-0.32	0.31	-0.12	-0.17	-0.13	-0.16	-0.42	-0.18	-0.37	-0.65	0.30	-	-0.51	-0.01	-0.03	0.74	-0.37	-0.14	0.09
prima.donnas	0.28	0.04	0.21	0.44	-0.13	0.10	0.00	0.18	-0.19	-0.06	-0.18	0.10	0.17	0.33	0.08	-0.51	-	0.41	0.17	-0.36	0.53	0.46	-0.28
radio.silence	-0.15	0.11	-0.23	0.43	0.07	-0.27	0.39	0.28	0.22	0.50	0.42	0.03	0.43	0.38	0.63	-0.01	0.41	-	0.49	0.43	0.53	0.27	0.14
black.cloud	-0.07	-0.23	-0.08	0.37	-0.18	-0.05	0.34	0.27	0.23	0.28	0.08	0.20	0.00	0.00	0.26	-0.03	0.17	0.49	-	0.39	0.29	0.32	-0.45
missing.links	0.02	-0.47	0.07	0.35	-0.40	0.16	0.25	0.31	0.19	0.37	0.02	-0.16	-0.08	-0.28	0.75	0.74	-0.36	0.43	0.39	-	0.20	0.22	0.07

	code.turnover	core.global.turnover	core.mail.turnover	core.code.turnover	ratio.smelly.quitters	ratio.smelly.devs	global.truck	mail.truck	code.truck	closeness.centr	betweenness.centr	degree.centr	global.mod	mail.mod	code.mod	density	mail.only.core.devs	code.only.core.devs	ml.code.core.devs	ratio.mail.only.core	ratio.code.only.core	ratio.ml.code.core
org.silo	0.03	-0.10	0.10	-0.35	-0.16	0.07	0.14	0.01	0.08	-0.04	-0.10	-0.18	0.61	0.13	0.09	-0.14	-0.49	0.73	-0.12	-0.45	0.83	-0.06
prima.donnas	-0.32	-0.16	-0.28	-0.24	0.06	-0.18	0.28	0.36	0.23	-0.19	-0.21	-0.21	-0.25	-0.48	0.17	-0.23	-0.10	-0.70	0.48	-0.10	-0.69	0.51
radio.silence	0.07	-0.41	-0.50	-0.38	-0.09	0.38	-0.14	0.51	-0.33	0.11	0.10	0.18	-0.30	-0.32	0.09	0.08	-0.18	0.02	0.63	-0.47	-0.13	0.52
black.cloud	-0.58	-0.50	-0.24	-0.27	0.52	0.14	-0.02	0.06	0.07	-0.02	-0.16	-0.06	-0.13	0.31	-0.66	0.02	-0.21	0.03	0.25	-0.23	0.01	0.22
missing.links	-0.09	-0.24	0.02	-0.67	-0.20	0.24	0.06	-0.04	-0.07	-0.06	-0.19	-0.14	0.26	0.10	-0.01	-0.10	-0.60	0.60	0.41	-0.76	0.58	0.38

(a) Pearson's correlation (coefficient)

	devs	ml.only.devs	code.only.devs	ml.code.devs	perc.ml.only.devs	perc.code.only.devs	perc.ml.code.devs	sponsored.devs	ratio.sponsored	sponsored.core.devs	ratio.sponsored.core	num.tz	core.global.devs	core.mail.devs	core.code.devs	org.silo	prima.donnas	radio.silence	black.cloud	missing.links	st.congruence	communicability	global.turnover
org.silo	0.85	0.21	0.64	0.74	0.32	0.33	0.70	0.60	0.69	0.62	0.18	0.57	0.23	0.02	0.35	-	0.09	0.97	0.92	0.01	0.24	0.66	0.80
prima.donnas	0.38	0.90	0.51	0.15	0.68	0.76	1.00	0.57	0.55	0.85	0.58	0.75	0.59	0.30	0.81	0.09	-	0.18	0.61	0.25	0.08	0.13	0.40
radio.silence	0.65	0.73	0.48	0.17	0.84	0.40	0.21	0.38	0.50	0.10	0.18	0.94	0.16	0.22	0.03	0.97	0.18	-	0.12	0.16	0.08	0.40	0.69
black.cloud	0.83	0.50	0.82	0.27	0.59	0.89	0.30	0.42	0.50	0.40	0.82	0.56	1.00	1.00	0.43	0.92	0.61	0.12	-	0.24	0.39	0.34	0.16
missing.links	0.95	0.13	0.84	0.26	0.20	0.61	0.43	0.33	0.55	0.24	0.95	0.63	0.80	0.38	0.01	0.01	0.25	0.16	0.24	-	0.53	0.49	0.84

	code.turnover	core.global.turnover	core.mail.turnover	core.code.turnover	ratio.smelly.quitters	ratio.smelly.devs	global.truck	mail.truck	code.truck	closeness.centr	betweenness.centr	degree.centr	global.mod	mail.mod	code.mod	density	mail.only.core.devs	code.only.core.devs	ml.code.core.devs	ratio.mail.only.core	ratio.code.only.core	ratio.ml.code.core
org.silo	0.94	0.76	0.77	0.28	0.64	0.83	0.66	0.98	0.81	0.91	0.75	0.58	0.04	0.69	0.78	0.66	0.11	0.01	0.72	0.14	0.00	0.84
prima.donnas	0.34	0.63	0.41	0.47	0.87	0.58	0.38	0.25	0.48	0.56	0.52	0.52	0.43	0.12	0.60	0.48	0.75	0.01	0.11	0.75	0.01	0.09
radio.silence	0.83	0.21	0.12	0.25	0.79	0.23	0.67	0.09	0.29	0.73	0.75	0.58	0.35	0.31	0.79	0.81	0.57	0.95	0.03	0.12	0.70	0.08
black.cloud	0.06	0.12	0.49	0.42	0.10	0.69	0.95	0.86	0.83	0.94	0.64	0.86	0.70	0.36	0.03	0.95	0.53	0.93	0.46	0.49	0.98	0.51
missing.links	0.79	0.48	0.96	0.02	0.56	0.45	0.85	0.91	0.83	0.86	0.55	0.67	0.42	0.76	0.99	0.76	0.04	0.04	0.18	0.00	0.05	0.22

(b) Pearson's correlation (p-value)

Figure C.8: Pearson's correlation analysis - FFmpeg

	devs	ml.only.devs	code.only.devs	ml.code.devs	perc.ml.only.devs	perc.code.only.devs	perc.ml.code.devs	sponsored.devs	ratio.sponsored	sponsored.core.devs	ratio.sponsored.core	num.tz	core.global.devs	core.mail.devs	core.code.devs	org.silo	prima.donnas	radio.silence	black.cloud	missing.links	st.congruence	communicability	global.turnover
org.silo	-0.25	-0.48	0.50	-0.36	-0.36	0.60	-0.44	-0.38	-0.13	-0.15	-0.41	-0.20	-0.26	-0.56	0.21	-	-0.53	-0.02	-0.20	0.63	-0.67	-0.32	0.13
prima.donnas	0.50	-0.01	0.01	0.64	-0.08	-0.06	0.05	0.17	-0.37	-0.14	-0.30	0.04	0.20	0.24	0.13	-0.53	-	0.06	0.05	-0.44	0.59	0.57	-0.51
radio.silence	0.20	0.46	-0.29	0.36	0.27	-0.46	0.55	0.14	0.22	0.59	0.46	-0.27	0.44	0.34	0.70	-0.02	0.06	-	0.47	0.57	0.40	0.20	-0.03
black.cloud	-0.08	0.00	-0.17	0.32	-0.03	-0.17	0.57	0.19	0.20	0.30	0.14	0.11	-0.02	0.00	0.17	-0.20	0.05	0.47	-	0.40	0.20	0.20	-0.37
missing.links	-0.16	-0.26	0.13	0.11	-0.20	0.20	0.24	0.13	0.31	0.45	0.10	-0.28	-0.02	-0.23	0.68	0.63	-0.44	0.57	0.40	-	-0.03	0.08	0.06

	code.turnover	core.global.turnover	core.mail.turnover	core.code.turnover	ratio.smelly.quitters	ratio.smelly.devs	global.truck	mail.truck	code.truck	closeness.centr	betweenness.centr	degree.centr	global.mod	mail.mod	code.mod	density	mail.only.core.devs	code.only.core.devs	ml.code.core.devs	ratio.mail.only.core	ratio.code.only.core	ratio.ml.code.core
org.silo	0.01	-0.10	0.01	-0.32	-0.19	0.22	-0.08	-0.13	-0.15	0.10	0.15	-0.03	0.57	0.27	0.24	-0.03	-0.40	0.62	-0.23	-0.50	0.70	-0.17
prima.donnas	-0.51	0.01	-0.13	-0.34	-0.12	-0.22	0.57	0.24	0.43	-0.22	-0.29	-0.29	-0.04	-0.39	0.22	-0.36	-0.12	-0.60	0.56	-0.12	-0.65	0.65
radio.silence	-0.04	-0.33	-0.45	-0.28	-0.21	0.52	-0.12	0.48	-0.45	0.05	0.09	0.17	-0.23	-0.22	0.01	-0.02	-0.17	0.28	0.60	-0.50	0.20	0.31
black.cloud	-0.54	-0.51	-0.34	-0.13	0.27	0.10	-0.02	0.10	0.13	0.07	-0.37	-0.27	-0.20	0.27	-0.71	0.07	-0.07	0.20	0.17	-0.10	0.22	0.07
missing.links	-0.15	-0.18	-0.09	-0.54	-0.32	0.50	-0.30	-0.05	-0.31	0.04	-0.13	-0.09	0.17	0.17	-0.05	0.03	-0.53	0.60	0.28	-0.73	0.63	0.17

(a) Spearman's correlation (coefficient)

	devs	ml.only.devs	code.only.devs	ml.code.devs	perc.ml.only.devs	perc.code.only.devs	perc.ml.code.devs	sponsored.devs	ratio.sponsored	sponsored.core.devs	ratio.sponsored.core	num.tz	core.global.devs	core.mail.devs	core.code.devs	org.silo	prima.donnas	radio.silence	black.cloud	missing.links	st.congruence	communicability	global.turnover
org.silo	0.44	0.11	0.10	0.26	0.25	0.04	0.15	0.22	0.68	0.64	0.19	0.54	0.42	0.06	0.51	-	0.07	0.96	0.55	0.03	0.02	0.31	0.71
prima.donnas	0.10	0.97	0.97	0.02	0.80	0.86	0.88	0.60	0.24	0.68	0.34	0.89	0.52	0.45	0.69	0.07	-	0.86	0.88	0.15	0.04	0.05	0.11
radio.silence	0.53	0.13	0.35	0.25	0.39	0.13	0.07	0.67	0.48	0.04	0.14	0.40	0.15	0.27	0.01	0.96	0.86	-	0.14	0.06	0.20	0.53	0.95
black.cloud	0.80	1.00	0.62	0.33	0.92	0.62	0.07	0.58	0.55	0.37	0.69	0.75	0.96	1.00	0.62	0.55	0.88	0.14	-	0.22	0.55	0.55	0.26
missing.links	0.62	0.42	0.70	0.73	0.53	0.54	0.44	0.70	0.33	0.15	0.75	0.37	0.96	0.47	0.02	0.03	0.15	0.06	0.22	-	0.92	0.82	0.86

	code.turnover	core.global.turnover	core.mail.turnover	core.code.turnover	ratio.smelly.quitters	ratio.smelly.devs	global.truck	mail.truck	code.truck	closeness.centr	betweenness.centr	degree.centr	global.mod	mail.mod	code.mod	density	mail.only.core.devs	code.only.core.devs	ml.code.core.devs	ratio.mail.only.core	ratio.code.only.core	ratio.ml.code.core
org.silo	0.99	0.78	0.99	0.34	0.58	0.48	0.81	0.70	0.65	0.75	0.65	0.94	0.06	0.40	0.46	0.92	0.20	0.03	0.47	0.10	0.01	0.60
prima.donnas	0.11	0.97	0.69	0.31	0.72	0.49	0.05	0.46	0.16	0.49	0.36	0.36	0.91	0.21	0.49	0.25	0.70	0.04	0.06	0.70	0.02	0.02
radio.silence	0.92	0.33	0.17	0.40	0.54	0.09	0.71	0.12	0.14	0.89	0.78	0.60	0.47	0.50	0.99	0.96	0.60	0.37	0.04	0.10	0.53	0.33
black.cloud	0.09	0.11	0.31	0.69	0.42	0.77	0.96	0.77	0.69	0.84	0.26	0.42	0.55	0.42	0.01	0.84	0.84	0.55	0.62	0.77	0.52	0.84
missing.links	0.67	0.60	0.80	0.09	0.34	0.10	0.35	0.89	0.32	0.90	0.70	0.78	0.59	0.59	0.89	0.92	0.08	0.04	0.38	0.01	0.03	0.59

(b) Spearman's correlation (p-value)

Figure C.9: Spearman's correlation analysis - FFmpeg