Electronic Communications of the EASST
Volume 48 (2012)

Proceedings of the Fifth International Workshop on on Foundations and Techniques for Open Source Software Certification (OpenCert 2011)

Learning and Activity Patterns in OSS Communities and their Impact on Software Quality

Antonio Cerone

21 pages

Guest Editors: Luís Soares Barbosa, Dimitrios Settas
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Learning and Activity Patterns in OSS Communities and their Impact on Software Quality

Antonio Cerone[1]

[1] antonio@iist.unu.edu,
UNU-IIST — International Institute for Software Technology
United Nations University, Macau SAR China

Abstract: This paper presents a framework to identify and analyse learning and activity patterns that characterise participation and collaboration of individuals in Open Source Software (OSS) communities. It first describes how participants' activities enable and drive a learning process that occurs in individual participants as well as in the OSS project community as a whole. It then explores how to identify and analyse learning patterns at both individual level and community level. The objective of such analysis is to determine the impact of these patterns on the quality of the OSS product and define a descriptive approach to quality that is concerned less with standards than with the facts of OSS peer-review and peer-production.

Keywords: Open Source Software, activity patterns, learning patterns, software quality.

1 Introduction

Open Source Software (OSS) communities have been representing during the last decades an interesting phenemenon of collaborative work leading to the successful delivery of high-quality software products. Popular OSS products range from operating systems, such as Linux, Ubuntu and BSD distributions, to network services, such as Apache, high-end applications, such as MySQL, and Learning Management Systems (LMS), such as Moodle. Popularity and widespread use of such products are *de facto* indicators of their high quality.

Although the high-quality of such OSS products has been accepted as a fact and is confirmed by their widespread use not only in the academic world but also in the industrial world and in public administration, it is still largely unclear how such high-quality emerges from the "bazaar-style" activities of an OSS community. Recently several studies have analysed activities, practices and collaboration processes that occur within OSS communities and projects, as well as the quality of OSS products, in the attempt to identify relevant factors that foster the emergence of high quality in product releases.

Halloran and Scherlis [HS02] review a number of notable quality practices on some popular OSS projects, of which good project communication and management is highlighted. Coverity has been analysing the quality of Open Source Software since 2006, using Coverity Scan, a tool for automated static analysis of source code [Cov]. Coverity reports emphasise the high-quality of OSS products, comparable with if not better than closed-source proprietary software, and express the expectation that as open source continues to mature, more and more projects will begin

to adopt stronger quality practices [Cov11]. According to Newman, OSS "has the potential of being better [than closed-source proprietary software] if its development process addresses many factors that are not normally experienced in mass-market proprietary software" [Neu05]. Zhao and Elbaum [ZE00] undertake a small survey to examine the factors underlying quality assurance methods of open source developers. Their work characterises the general attitude and practices of the open source community towards quality, realising that quality assurance practices are somewhat different to those prevalent in traditional software development. McConnell [McC99] emphasises the need for a comprehensive methodology for open source development. This is a fundamental need for OSS products to be used as safety-critical or security components of high quality complex software systems. In fact, Schneider [Sch00] finds that Open Source Software still falls short of requirements for security systems. Moreover, the lack of central management in OSS projects [MHP05, Mic05] makes it difficult to define a standard that could suggest indicators of the technical rigour used by a distributed community of volunteers and identify the human processes involved in the project.

OSS communities are heterogeneous groups of volunteers that are loosely organised in a "democratic" fashion based on the principles of freedom and equality of participants. However, this does not mean that OSS communities are disorganised entities. In fact, they develop a natural form of self-organisation in which participants play distinct roles, have various forms of engagement, develop different levels of knowledge, produce a range of contributions and are driven by a large variety of intrinsic and extrinsic motivations. Roles, engagement, knowledge, contribution and personal motivations build up reputation and support the emergence of driving personalities and forms of leadership. In addition to this implicit bottom-up organisational characteristics, which naturally emerge within the community, some forms of top-down organisation may be explicitly superimposed by the project initiator, who can be an individual, a team, a consortium, an organisation, or even a private company.

Collaboration among peers is the productive engine of the OSS community and determine the typical OSS development model, which has potential benefits that include "the ability to more easily carry out open peer reviews, add new functionality either locally or to the mainline products, identify flaws, and fix them rapidly — for example, through collaborative efforts involving people irrespective of their geographical locations and corporate allegiances" [Neu05]. Additional testimonials about the impact that peer-review has on the product of OSS development are given by Raymond, who claims that "The high level of quality of free software is partly due to the high degree of peer review and user involvement" [Ray99], and McConnell, who acknowledges the efficiency of extensive field testing and peer review in open source development [McC99].

OSS projects can also be considered as learning and development environments in which heterogeneous communities get together to exchange knowledge through discussion and put it into practice through actual contributions to software development, revision and testing [CSb]. OSS communities are open participatory ecosystems in which actors create not only source code but a large variety of resources that include the implicit and explicit definitions of learning processes and the establishment and maintenance of communication and support systems.

The rest of this paper is organised as follows. Section 2 introduces typical roles of participants to OSS projects, describes their high-level activities and breaks them down into four basic activities. Section 3 describes the learning process of individual actors of an OSS community as the result of communication and development activities. Section 3.1 identifies the learning stages

that characterise OSS ecosystems and relates them to participants' activities. Section 3.2 reviews the literature concerning the identification and analysis of learning patterns. Section 4 presents a descriptive approach to quality based on the identification of learning and activity patterns in data collected from OSS project repositories and the analysis of their impact on the quality of the OSS product.

2 Contributors' Roles and Activities

Paricipants in OSS projects may play a large varieties of roles, have various levels of engagements, both within the project and the community, and contribute to the project development in different respects. Typical participants' roles are

observer who plays a passive role in which there is neither interaction with the community nor production of artifacts;

supporting user who often plays a very active role in providing feedback, helping new users, recommending the project to others, requesting new features, but does not produce artifacts;

developer who actively writes and updates software, documentation and/or creates artwork;

tester who actively performs testing and reports and possibly fixes bugs;

translator who translates software and/or documentation into another language.

The **observer** performs a large range of activities including: accessing and reading reports, documentation and tool manuals, possibly using the tools, looking at data in the repositories, reading posted message without posting or replying, looking at the code and possibly running it. However, this role is *passive* in terms of interaction with the community, although it may characterise an *active* but **non-supporting user** (the usage actually occurs outside the community, with no impact on the community or the project). It is therefore a role with neither *supporting* nor *productive* finality, but involving two kinds of *basic activities*: ***observe*** and ***use***. Here the finality of the participant is actually *learning*, which is usually the main goal during the first stage of the participation of a contributor in a project. We will deeply investigate this aspect in Section 3.1.

The other roles are all *active*. A **supporting user** not only uses tools and code but also provides feedback to the community as well as support to new users, and/or requests new features. These kinds of services to the communities are provided through another *basic activity*, ***post***, which consists in making available to the community messages containing questions, requests, advices and/or critics. This typically occurs in discussion fora.

The last three roles are not only *active* but also *productive*. Produced artifacts are software, documentation, artwork (by a **developer**), bug reports, fixed code (by a **tester**), translated software, translated documentation (by a **translator**). Productive activities are enabled by a fourth *basic activity*, ***commit***, which is the process of adding an artifact to a project repository and make it part of the project product, to be potentially deployed with the next release. Commit may be direct, if the contributor has commit right, or may occur through an approval process mediated by a leader or leading team.

To summarise, the four *basic activities* that enable all contributors' activities are:

observe reports, documentation, tool manuals, data, posts, code;

use code, tools;

post questions, requests, advices, critics;

commit software, documentation, artwork, bug reports, fixed code, translations.

However, the way these four basic activities are combined and result in the activity pattern of a contributor depends on multiple factors, including intrinsic and extrinsic motivations, maturity levels, technical and social skills. This yields a large heterogeneity of activity patterns at both individual level and community level.

The development of a ***post*** into an exchange of messages is the engine of discussion fora and enables the *interaction process* that occurs within the OSS community. Such interaction process has two components:

learning sub-process in which the exchange of knowledge between individual and community results in the growth of knowledge at both the individual level and the team or community level;

contribution sub-process in which a contribution in terms of commit of code, bug report, etc. is the result of an exchange of communications.

The learning component of the interaction process is essentially a *collaborative learning* process, in which knowledge is built through social constructivism and is part of a more complex learning process that will be detailed in Section 3. The contribution component of the interaction process can be seen as a *peer-production* process, in which the creative energy of large numbers of individuals is remotely coordinated, usually through the Internet, into large, meaningful projects mostly without traditional hierarchical organisation [Ben02]. Therefore, contribution, which is based on ***commit***, is the result of communication, which is based on ***post***, individual learning, which is based on ***observe*** and ***use***, and collaborative learning, which is based on ***post***. Furthermore, the interaction process is cyclic on its two components, in the sense that both the knowledge that results from the learning sub-process and the artifacts that result from the contribution sub-process feed a new iteration of the interaction process. This cyclic nature of the interaction process is the basis of the individual-team interplay that occur in OSS communities [CFS12]. Finally, we can say that also the learning process, and as a result the entire interaction process, can be seen as an instantiation of the peer-production model [Fut06].

Recent research on contributors' activity patterns has been carried out by utilising data from a single repository to analyse code contribution of developers [RG06, GKS08], trends and inequality in posting and replying activities in Apache and Mozilla [MFH02], KDE [Kuk06], Debian [SSA08], and FreeBSD [DB05]. Data on communications (***post***) from mailing lists [SSA08] and development activities (***commit***) are extracted from revision control systems such as CVS (Concurrent Versions System) and SVN (Apache SubVersioN), which are part of the Source Configuration Management (SCM) used to coordinate the coding activities of software developers and manage software builds and releases. A number of tools [JG06, RGCH09, SSSA08]

can be used to retrieve data from SCM sytems using tools that store committers' attributes into various tables and extract [SSSA08] one or more mailing list archives of a particular project.

Data are then analysed to identify posters and committers and, after appropriate data cleaning [SCb] and alias unmasking [BGD+06, SSA08], descriptive statistics is used to show developers posting and committing activities and patterns [SAS06, SCb] as well as activity patterns recurring in various OSS projects. Many of these studies highlight the essence of communication as a means to foster long term success of software projects [Bro75]. Although a strong linkage exists between the information in OSS repositories (e.g. bug reports and source code repositories [DB07, ZPZ07]) and SVN and mailing lists [SCb]), few researchers strive to understand how contributions vary across repositories. In the next sections we attempt a deeper investigation of such linkage to better understand

1. how communication and development enable a natural *learning process* (Section 3);
2. how the linkage between learning process and basic activities drives evolution of activity patterns and maturation at individual level as well as at community level (Section 3.1);
3. how activity patterns can be analysed to identify the presence of *learning patterns* (Section 3.2);
4. how evolution and maturation of the activity patterns foster the *productive process* and what their impact on software quality is (Section 4).

3 Learning Process

Freedom and equality of participants constitute a "democratic" basis for analysing OSS communities as *communities of practice*. Novices are always welcome by OSS communities, in which they undergo through a gradual process of social integration and skill development that allows them to earn a reputation as reliable developers and then move towards the leading positions in the community [Tuo05]. OSS communities are in this sense open participatory ecosystems [MGS08, MGS09], in which actors produce not only source code but a large variety of resources that include the implicit and explicit definitions of learning processes and the establishment and maintenance of communication and support systems. Furthermore these resources are made visible and available to other actors. Therefore development (source code), support (tools) and learning (knowledge) emerge as the product of a continuous socialisation process in a virtual environment.

Development of source code is enabled by building up knowledge about already produced code, through direct observation, review, modification as well as discussion with other actors, and about support tools, through direct interaction as well as access to documentation and discussion with other actors. As suggested by Sowe and Stamelos [SS08a] the learning process of individual actors can be divided in four phases through which knowledge evolves. Cerone and Sowe [CSb] give their slightly different characterisation of these phases. We complete such a characterisation by associating relevant *basic activities* with each phase as follows:

socialise by *implicitly sharing knowledge* (enabled by ***post***);

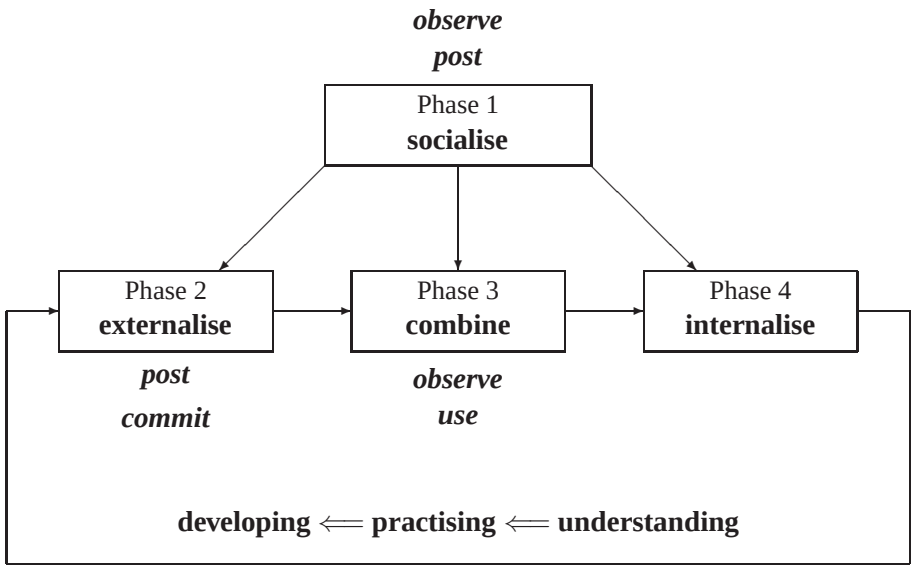

Figure 1: Learning process of individual actors in OSS communities

externalise *tacit knowledge* by making it explicit to the community (enabled by ***post***, ***commit***);

combine community *explicit knowledge* and organise it as abstract knowledge (enabled by ***observe***, ***use***);

internalise *abstract knowledge* by absorbing it and combining it with own knowledge and experiences to produce new tacit knowledge.

The four phases are not fully sequential but overlap in a certain measure, as shown in Figure 2. In particular, socialisation, after playing the role of initiating the learning process, is still active during the other phases for which it is actually an enabling factor.

Socialisation in a virtual environment, specifically through the Internet, already permeates our daily life, especially through social networks, as well as informal and formal education settings, especially through e-Learning tools and environments [Imm] such as Moodle [Moo] and Second Life [Sec]. In an OSS context the socialisation phase is enabled by specific mechanisms and tools used by OSS communities, such as discussion fora, and may be initiated

- either by the ***observe*** activity, that triggers a communication with an "observed" community member;
- or by the ***post*** activity, that triggers a reply to the posted message.

After being initiated, socialisation can continue through posting, also outside the OSS community tool infrastructure, for instance through social networks.

Externalisation naturally occurs in an implicit way through socialisation tools such as discussion fora, with ***post*** as the enabling basic activity, or in an explicit way through ***commit***. Intrinsic motivations, such as

- feel passionate about particular areas of expertise
- enjoy self-satisfaction from sharing knowledge and skills
- have a sense of belonging to a community

are all examples of strong drivers for externalisation. There are also a number of extrinsic motivations that contribute to externalisation, which include

- solve particular technical problems/needs by exploiting Linus' Law: "given enough eyeballs, all the bugs are shallow" (from Linus Torvalds);
- public visibility to increase reputation and peer recognition.

Combination of knowledge is incremental and consists of two main activities:

- multiple interactions with knowledge-management tools as well as with other members of the community to identify and extract relevant bits of explicit knowledge;
- combination and organisation of such bits of explicit knowledge to produce meaningful abstract knowledge.

The interaction with knowledge-management tools is obviously enabled by the ***use*** and ***observe*** basic activities. Organisation of explicit knowledge and production of meaningful abstract knowledge are cognitive activities within the ambit of *knowledge representation*. Without entering the realm of cognitive theories that aim to explain knowledge representation within the human mind, we can say that the way individuals combine explicit knowledge is affected by the accessibility, structure and presentation of the contents of such knowledge and by own personal learning attitudes.

Internalisation of knowledge is a cognitive activity that proceeds at an unconscious level (*unconscious assimilation*) and results in the acceptance of the newly produced tacit knowledge. Being an unconscious process, it is not affected by any basic activities. Internalisation is driven by both intrinsic and extrinsic motivations. Examples of *Intrinsic motivations* for internalisation are:

- revel in creating something new or better;
- have a personal sense of accomplishment and contribution.

Examples of *Extrinsic motivations* for internalisation are:

- improve technical knowledge base;
- pass examinations;
- develop the solution to a technical problem.

3.1 Learning Stages

We can identify three stage that characterise the evolution of activity patterns and maturation of an OSS community participant.

understanding The most critical part of the learning process develops through an initial but often significantly long stage in which communication is heavily used to capture, describe and understand contents, while no production activities are performed. That is, during this stage the project participants mainly access project repositories and exchange emails and post messages with the purpose of understanding contents without producing any code or reports and without performing any commits. This stage might be very brief and marginal for an OSS expert who just joined a new project, but is significantly long and important for a novice.

practising In this second stage the role of communication gradually moves to the proposal of new contents, the defense of the proposed contents and the criticism to existing contents or contents proposed by others. At the same time, during this stage, production activity starts as a trial and error process with a consequent low quality in the resultant product and little or no immediately visible impact on project productivity. This stage is usually significantly long and important for a novice.

developing Only during this third stage the quality level of developed code and reports becomes important and communication is mainly used to support own productive contributions and contrast them to others' contributions.

In Figure 2 contributors' *learning stages* are linked to the contributors' *basic activities* identified in Section 2. The segmented curve represents the *maturation* of the contributor through the three *learning stages*. For each *learning stage* the shaded area highlights which *basic activities* are carried out during that stage, with no quantitative meaning associated with the area size. We claim that although during the **understanding** stage ***observe*** activities are carried out, there is no actual evolution through the learning stages (i.e. no *maturation*) until ***use*** activities are also carried out. This is represented in Figure 2 by a 0 gradient curve segment for ***observe*** activities during the **understanding** stage. The low gradient of the curve segment for ***use*** activities aims to explain the fact that, although there is some form of learning at the individual level there is instead no learning at all at community level. The learning process described in Section 3 only starts when *socialisation* and *externalisation* occur, enabled by ***post*** activities. Initially ***post*** activities just contribute to *socialisation* and only when they start to be used for *externalisation* the curve gradient considerably increases to denote a faster progress through the *learning stages*. Contributors enter the **practising** stage only when they start to carry out ***commit*** activities as a trial and error process. With practice, the quality of the resultant product gradually increases, which results in the fastest learning progress, and finally the **developing** stage is entered.

Note that we have used a segmented curve to highlight progress from one *learning stage* to the next as well as milestones within stages. In reality the *maturation* process gradually evolves in an almost continuous way, though milestones still occur both at individual and community level.

Figure 3 shows how *actual activities* carried out by contributors, as instantiations of their basic activities, are linked to their *maturation* through *learning stages*.

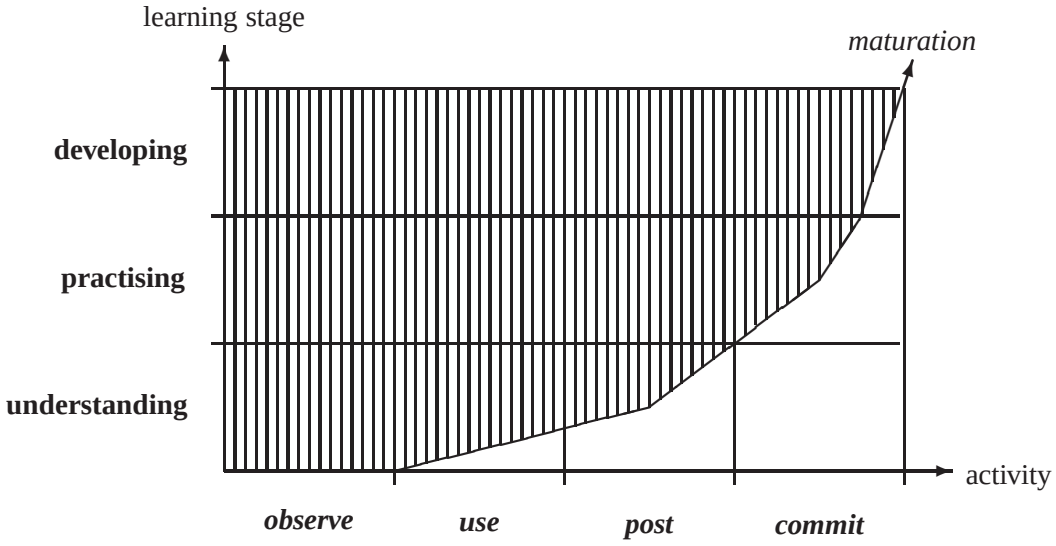

Figure 2: Learning process of individual actors in OSS communities

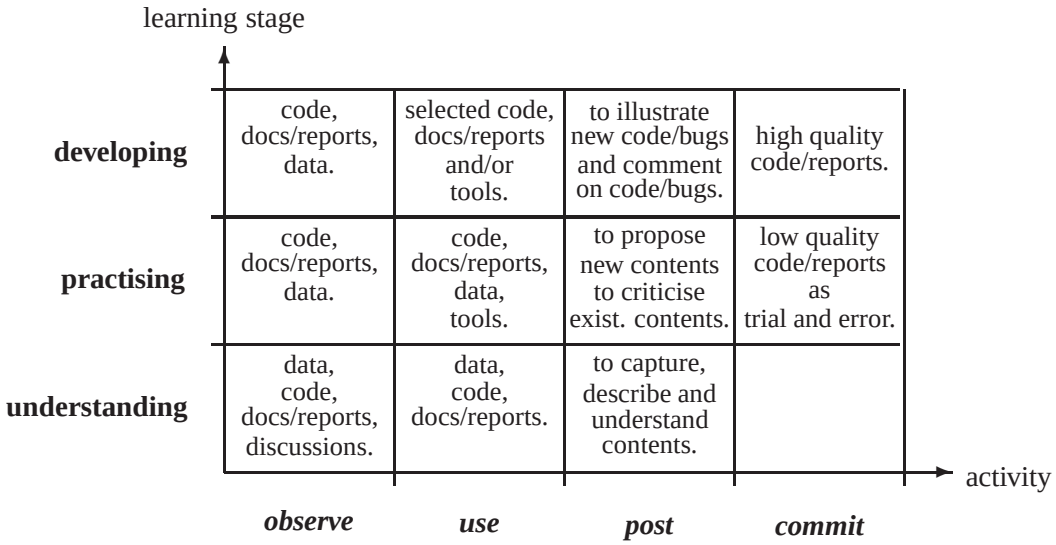

Figure 3: Activity contents

It is evident that the first two *learning stages* (**understanding** and **practicing**) are the most relevant for the understanding of the learning processes occurring within the OSS community. Since the third *learning stage* (**developing**) mainly involves *production activities*, at first thought we might think that on one hand such stage does not contribute to the learning process at all, on the other hand this is the only one among the three stages that contributes to productivity. However, we have to consider that any form of productivity counts on knowledge and skills built through learning processes and training activities. Therefore, in the case of OSS communities

- productivity is affected by the the learning process: it is actually the final act of a very long and complex evolution and growth in the individual and collective knowledge and practice;
- productivity re-enters the learning process through the *externalisation* phase and thus contributes to iteratively enable the learning process.

In Section 3.2 we highlight some analysis steps that are essential in understanding the learning processes occurring within individuals and the knowledge growth at group and community level and in identifying *learning patterns*. Moreover, in Section 4 we illustrate how it is possible to extract indicators of the quality of the OSS product from learning patterns.

3.2 Learning Patterns

Collaboration in OSS projects is highly mediated by the usage of tools, such as versioning systems, mailing lists, reporting systems, etc. These tools serve as repositories which can be data mined to understand the identities of the individuals involved in a communication, the topics of their communication, the amount of information exchanged in each direction, as well as the amount of contribution in terms of code commits, bug fixing, reports and documentation produced and email postings. This large amount of data can be selectively collected and then analysed not only by using inferential statistics to identify activity patterns but also by using ontology engineering formalisms that support the extraction of semantic information. In the area of Empirical Software Engineering, cyber-archeology [SSS07] has been applied to these repositories to learn and better understand the patterns of contribution of OSS developers in the projects concerned [SFF+06, SSS07, GKS08, SCb].

In order to investigate learning processes, the data mining analysis has to be carried out on communications and activities related to learning. In previous work data collection has involved communications mainly in terms of participants, quantity and sometimes topics but neglected the objective collection of actual communication contents. At most, content data has been collected through questionnaire and surveys [SS08b] or through written student reports describing the encountered risks [AP09], thus providing subjective rather than objective data. This is not a problem when the research goal is the analysis of level, type and quality of productive participation in the project, since communication data is usually integrated by objective data on contribution in terms of commits, bug reports, bug fixing and on the approval and inclusion of the resultant artifacts in a software release [SCb].

When the research goal is the analysis of learning patterns, objective data have to be extracted from communications and activities releated to learning The identification of the learning stage of

the OSS community member whose data is collected can be a challenge during this preliminary data collection from repositories of OSS projects. Text mining of communications can be used to identify keywords and phrases that may indicate the learning maturity level in the learning environment.

The technology of antipatterns [BMM+98] can provide an important support to both data mining and text mining. A number of OSS communication antipatterns have been identified and made available [Leu08, Ber, Nea11, ant11, CSa]. OSS participant communication analysis using data mining can reveal commonly occurring problematic OSS practices or problematic processes that occur during the learning process and have negative consequences for the development process and/or for the learning process itself. Moreover, text mining can reveal the communication and collaboration antipatterns that exist in OSS projects. Further reasoning on determining which antipatterns exist in specific OSS projects can be achieved using software tools that support Semantic Web technology [Tap08, SMSB11]. The knowledge representation formalism of ontology [CRP06] can define a common dictionary of OSS community antipattern terms and can be used as an extensible knowledge base of an intelligent system that can detect antipatterns in specific OSS projects. Ontology editors support collaborative Web-based ontology enrichment and editing [TVN08]. An example of this approach is the e-Learning environments designed by Settas and Cerone [SC11]. Similar environments can be designed to provide OSS antipattern contributors with the motivation to contribute antipatterns by communicating with other contributors and by supporting discussions and voting mechanisms for ontology changes. In this way the analysis of antipatterns in general, and of the learning-related antipatterns in particular, may be carried out by the OSS community itself, which would thus actively contribute to the analysis and refactored solution of its own 'negative' activity patterns in general, and of its own 'negative' learning patterns in particular.

4 Descriptive Approach to Quality

The traditional approach to quality assurance is based on the conformance to *prescriptive* standards and strict guidelines, which are sometimes incorporated in legal prescriptions for software certification. Moreover, standards and guideline refer to clearly defined software development methodologies and are the basis for the definition of quality metrics to be used in the certification process.

The lack of accurate information on how quality emerges from the large amount of loosely organised activities of an OSS community makes it difficult to apply traditional quality metrics and certification processes to OSS products. For instance, if we consider McCall's production revision quality factors [MRW77], can we claim that an OSS product lacks *maintainability* because there are no defined coding standards and guidelines to which programming has adhered? The philosophy of freedom and absence of hierarchical organisation typical of OSS communities results in collaborative production environments in which there is no space for *prescriptive* standards and strict guidelines. Communication and collaboration as well as individual preferences, skills and intrinsic and extrinsic motivations are the drivers of such production environments and naturally determine the evolution of programming practices within teams of contributors and across the OSS community, even beyond a specific OSS project. In such a context, a *descriptive*

approach that analyses the OSS community of practise and its activities is likely to define better indicators of the quality of the software product than a *prescriptive* approach that tries to check whether these activities follow prescribed standards and guidelines.

This distinction between *prescriptive* and *descriptive* approach can in general refer to the correctness and quality evaluation of artifacts that are the result of evolving community processes. For example, language emergence and evolution is a natural community process whose artifacts are speeches and written texts. The early approach to linguistic analysis was based on the *prescriptive* tradition: the correctness and quality of a speech or written text are evaluated in terms of its conformance to strict rules defined by grammarians. In 1761, Joseph Priestley, in his *Rudiments of English Grammar*, proposed an alternative view by claiming that "the custom of speaking is the original and only just standard of any language". This started the *descriptive* approach to linguistic analysis, which, in David Crystal's words "is concerned less with 'standards' than with the *facts* of linguistic usage" [Cry97].

If we transfer Priestley's claim to the OSS world we can say that

> the custom of OSS individual and community activities, learning and patterns is the original and only just standard of any OSS project.

And a paraphrase of Crystal's words for the OSS world could be

> the *descriptive* approach is concerned less with standards than with the *facts* of OSS peer-review and peer-production.

In Section 3.2 we have seen that cyber-archeology [SSS07] has been applied to OSS repositories to learn and better understand the patterns of communication and contribution of OSS developers [SFF+06, SSS07, GKS08, SCb]. The findings in these studies have promoted the conjecture that patterns of communication and contribution have implications for the quality of code of the OSS product. When a large number of developers externalise and discuss their coding activities with other community members in mailing lists, the high magnitude of the community ***engagement*** may enable developers to improve the quality of their code base, do more refactoring and learn about how the quality of the produced code may be improved [SCb]. Moreover, the engagement of an individual in the community can be related to three notions of quality identified by Shaikh and Cerone [SCa]: quality by access, quality by development and quality by design. Every activity of an individual can be classified under an appropriate category of quality and marked to contribute to the final software product accordingly [CFS12]. Bug reporting, testing and reviews enhance quality by development, the media and format used to externalise such contributions affect quality by access, whereas evidence of planning and design, and validation of software code contribute to quality by design.

The number of commits describes how much the individual delivers in terms of product and is therefore an indicator of the individual's ***productivity***. Although there is no guarantee on the quality of the product delivered, number of commits can be considered by itself an important parameter in evaluating the quality of the individual as a contributor. Moreover, in the analysis of level, type and quality of productive participation in the project, the mere number of commits can be integrated by other data related to the object of a considered commit activity. Such related data can be extracted from communications as well as other commit activities and processed

using text mining and ontology engineering techniques. In this wider context, productivity is intended in terms of community activities rather than individuals' contribution. More precisely, data mining may identify clusters of interrelated commit and post activities that refer to a specific artifact, whose development is likely to involve several contributors; in addition, text mining of the post activities can provide information on the quality of the considered artifact.

Evidence suggests [RHS06] that ***reputation*** serves as a major source of motivation for developers to participate in a community. Moreover, reputation builds on a history of participation in OSS projects that have successfully delivered high quality software products. It is therefore perceived as a warranty for quality. Communications among members of an OSS community can be analysed to extract information about the reputation that an individual has achieved within the community. Text mining of communications can be used to identify keywords and phrases that may indicate whether an individual is asking or providing support and whether an answer or suggestion is taken on board or refuted by others.

Cerone, Fong and Shaikh [CFS12] consider ***engagement***, ***productivity*** and ***reputation*** as key factors to evaluate the quality of an individual's contribution to an OSS project.

We have also seen in Section 3.1 that the learning stages of OSS community members affect the kind of activities carried out and have an impact on the quality of the artifacts they produce. For instance the code committed by a contributor in the **practising** stage has on average a lower quality than the code committed by a contributor in the **developing** stage. Therefore, the learning stage can be seen as a weight to associate with productivity in the evaluation of the quality of individuals' contribution and community activities. Moreover, learning stages of individuals belonging to a team or to a project community affect collaboration patterns and have an impact on the collaboration effectiveness, that is, the quality of collaboration within the community. For example, individuals in the **understanding** stage are not expected to contribute to team decisions and commit activities, whereas individuals in the **developing** stage are expected to contribute at least to commit activities and possibly to team decisions [CFS12]. We can, therefore, consider ***learning*** as a fourth key factor to evaluate the quality of an individual's contribution to an OSS project.

Our descriptive approach to quality is summarised in Figure 4. The flow on top of the Figure describes the extraction of data from OSS Project Repositories, their processing using a range of Analysis Technologies to characterise patterns that describe OSS Community Activities, then quantified through metrics for Quality. Typical data to be analysed includes: communications, commits, code, reports as well as organisational structure and processes. Data extracted from repositories are processed using data mining and text mining and applying categories of quality, collaboration models, trust models and ontology engineering to identify patterns for engagement, productivity, learning and reputation, as well as positive (effective) collaboration patterns and negative patterns (antipatterns) at the community level.

Quality of individuals' contribution can be characterised by defining separate metrics for engagement, productivity, learning and reputation of individuals [CFS12]. However, how to combine such metrics into a global metric that could quantify the quality of an individuals contribution to a specific OSS project is still unclear. Another challenge is represented by possible interrelations between the four metrics; for instance, we can note that the level of engagement of an individual within a project is visible to the entire project community and, therefore, implicitly affects that individual's reputation. We have also pointed out that learning affects productivity:

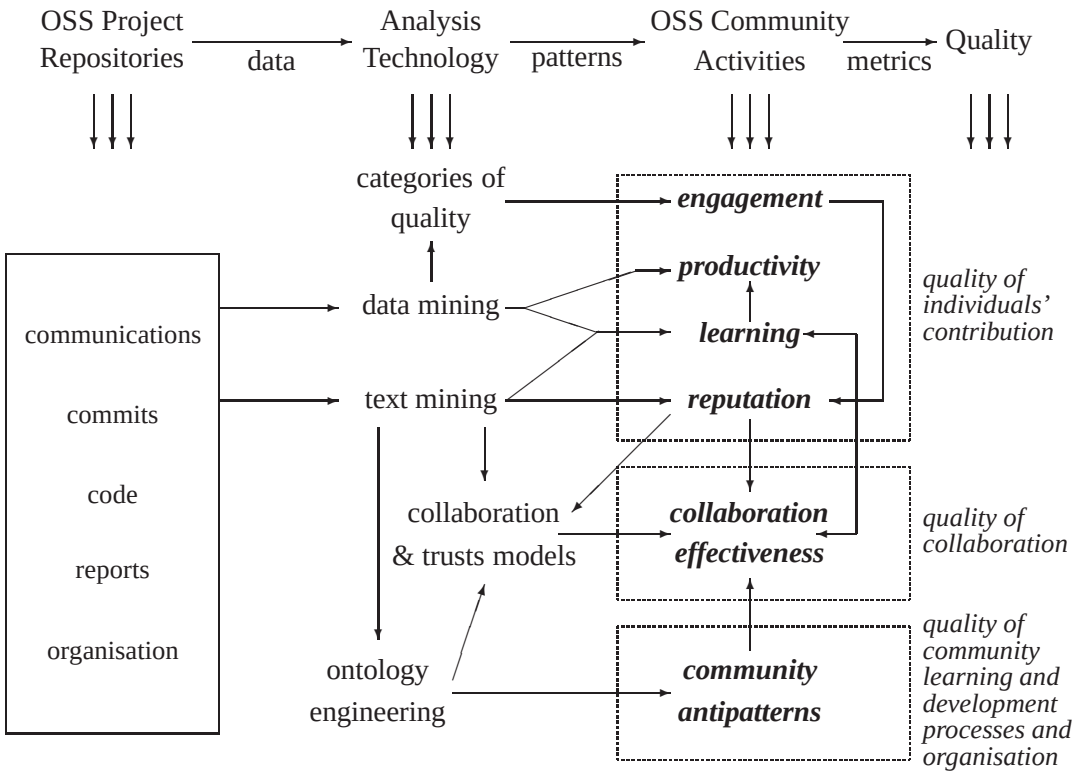

Figure 4: Descriptive approach for OSS quality analysis

a learning stage can be seen as a weight to associate with productivity in defining a metric for the quality of individuals' contribution [CFS12]. These two interrelations are described by arrows in Figure 4. Finally, although this is not captured by the diagram in Figure 4, there are other social parameters that affect reputation [HZC12], demonstrating how reputation obeys the laws of cumulative advantage (through higher likelihood of attracting good reputation given past reputation) and homophily (providing advantage through shared affiliations in terms of common OSS projects) [CFS12].

Quality of collaboration can be defined by applying metrics to patterns identified using collaboration and trust models to describe ***collaboration effectiveness***. These models have to be instantiated using content information extracted from communication through text mining carried out with the support of appropriate ontologies aiming to identify patterns, progress, evolution and achievements in the collaboration process occurring within groups of participants. Moreover, online trust may be inferred via a hierarchical metrics model which consists of explicit trust by relation, and implicit trust by reputation [CFda]. Therefore, reputation, in the form of implicit trust, is incorporated within trust models. Finally, the two-way relation between learning and collaboration effectiveness shown in Figure 4 highlights what we have already observed:

- that learning stages of individuals belonging to a team or to a project have impact on the

collaboration effectiveness (as mentioned earlier in this section);

- that learning in OSS communities is essentially a collaborative process (it is actually a sub-process of the interaction process illustrated in Section 2).

Quality of community processes (such as *learning* and *development*) and *organisation* can be characterised by identifying communication, learning and collaboration antipatterns occurring in OSS communities. In this sense Cerone and Settas [CSa] propose a framework to provide a measure that quantifies the negative effect of such antipatterns on quality. Note that the analysis we propose here considers ***community antipatterns*** rather than individual-based antipatterns; thus learning antipatterns contribute to measure collaborative learning — hence collaboration effectiveness, rather than individual learning. Therefore, there is an arrow in Figure 4 from community antipatterns to collaboration effectiveness, yet there is no connection between learning and community antipatterns.

5 Conclusion and Future Directions

The descriptive approach to quality presented in Section 4 shows that learning is a key factor to evaluate not only the quality of an individual's contribution but also the quality of community activities. The two-way arrow between learning and collaboration effectiveness in Figure 4 shows that individual learning affects collaboration effectiveness through the learning stages of the collaborating participants and that, in the other directions, individuals actually learn through collaborative efforts. Thus, we can claim that, on one hand, *learning affects quality*, since collaboration effectiveness is an indicator of quality (collaboration quality), and, on the other hand, *quality affects learning*.

Our descriptive approach to quality aims to "deduce" the quality of the OSS product from the quality of the production process and its enabling components: individual engagement, productivity, learning and reputation, as well as collaboration effectiveness and solution of community antipatterns. Therefore, our approach leads to a measure of the quality of the process rather than the quality of the product: only the former is measured whereas the latter is deduced. Although this notion of quality is *potential* rather than *assured*, it is warranted by the popularity and widespread use of many OSS products and is supported by the complex and multifaceted "open-source paradigm", which actually "has significant potential that is much more difficult to attain in closed-source proprietary systems" [Neu05]. This "potential of being better" is exactly what our approach aims to measure.

After having deeply investigated the impact of learning on quality, we would like to spend a few words to illustrate how the quality of the production process could be exploited to improve learning. We have already seen in Section 2 that the artifacts that result from the contribution sub-process feed a new iteration of the interaction process, thus also driving the next step of the learning sub-process. We also agree that OSS projects provide a meaningful alternative learning context to expose students to real-world software development activities [SSD06]. The proposal to use OSS projects as e-Learning tools by injecting students into OSS communities [CSb] can be see in this perspective. Moreover, such proposal has the potential to offer great advantages to OSS communities themselves:

- expose the community to innovative technologies brought in by postgraduate students who work at the edge between academic research and industrial development with a constant look at the future of software engineering;
- integrate tools to support students and educational objectives, which is likely to lead to improvements in tool usability and acceptance by the community of effective social protocols, with a positive impact on collaboration and on the externalisation and combination phases of the learning process;
- adopt new e-Learning technology that can benefit the entire community;
- produce more documentation as part of student tasks or even student assignments;
- expand the OSS community with new categories of actors (hence with new roles) who, for instance, reverse engineering code into formal models, refine and extend formal models, verify formal models, certify code and community activities [CS08].

Finally, we would like to conclude with the hope (and confidence) that introduction of new e-Learning technologies, injection of students as contributors, gradual acceptance and practise of innovative technologies will all have a strong impact on OSS communities by increasing their learning capability and their connection with the research and development world.

Acknowledgements: Several colleagues and friends have contributed through discussions and research collaborations to the development of the ideas presented in this paper: Luís S. Barbosa, Gabriella Dodero, Sara Fernandes, Simon Fong, Donatella Persico, Francesca Pozzi, Barbara Russo, Dimitrios Settas, Siraj A. Shaikh and Sulayman K. Sowe.

This work has been supported by Macao Science and Technology Development Fund, File No. 019/2011/A1, in the context of the PPAeL project.

Bibliography

[ant11] Community Management Wiki. 2011.
http://communitymgt.wikia.com/wiki/Category:Anti-patterns

[AP09] T. Ahtee, T. Poranen. Risks in Students' Software Projects. In *In Proceedings of the 22nd Conference on Software Engineering Education and Training (CSEET)*. Pp. 154–157. IEEE Computer Society, 2009.

[Ben02] Y. Benkler. Coase's Penguin, or, Linux and The Nature of the Firm. *The Yale Law Journal* 212:369–446, 2002.
http://www.yalelawjournal.org/images/pdfs/354.pdf

[Ber] J. Berkus. How to destroy your community.
http://lwn.net/Articles/370157/

[BGD+06] C. Bird, A. Gourley, P. Devanbu, M. Gertz, A. Swaminathan. Mining email social networks. In *MSR '06: Proceedings of the 2006 international workshop on Mining software repositories*. Pp. 137–143. ACM Press, New York, NY, USA, 2006.

[BMM+98] W. J. Brown, R. C. Malveau, H. W. S. McCormick, T. J. Mowbray, T. Hudson (eds.). *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. John Wiley, 1998.

[Bro75] F. Brooks. *The Mythical Man-Month. Essays on Software Engineering.* Addison-Welsey Publishing, 1975.

[CFda] W. Chen, S. Fong. Social Network Collaborative Filtering Framework and Online Trust Factors: a Case Study on Facebook. In *The 5th International Conference on Digital Information Management (ICDIM 2010)*. Pp. 266–273. July 2010, Thunder Bay, Canada.

[CFS12] A. Cerone, S. Fong, S. A. Shaikh. Analysis of Collaboration Effectiveness and Individuals' Contribution in FLOSS Communities. In *Proceedings of OpenCert 2011*. Volume 48 of Electronic Communications of the EASST. 2012.

[Cov] Coverity Scan.
http://scan.coverity.com/about.html

[Cov11] Coverity. Open Source Reports. 2008–2011.
http://www.coverity.com/resource-library

[CRP06] C. Calero, F. Ruiz, M. Piattini. *Ontologies for Software Engineering and Software Technology*. Springer, 2006.

[Cry97] D. Crystal. *The Cambridge Encyclopedia of Language*. Cambridge University Press, 1997.

[CSa] A. Cerone, D. Settas. Using Antipatterns to Improve the Quality of FLOSS Development. In *Proceedings of OpenCert 2011*. Volume 48 of Electronic Communications of the EASST, 2012.

[CSb] A. Cerone, S. K. Sowe. Using Free/Libre Open Source Software Projects as E-Learning Tools. In *Proceedings of OpenCert 2010*. Volume 33 of Electronic Communications of the EASST, 2010.

[CS08] A. Cerone, S. A. Shaikh. Incorporating Formal Methods in the Open Source Software Development Process. In *Proceedings of the OpenCert and FLOSS-FM 2008 joint Workshop*. UNU-IIST Research Report 398, pp. 26–34. 2008.

[DB05] T. T. Dinh-Trong, J. M. Bieman. The FreeBSD Project: A Replication Case Study of Open Source Development. *IEEE Transactions on Software Engineering* 31(6):481–494, 2005.

[DB07] J. M. Dalle, M. den Besten. Different Bug Fixing Regimes? A Preliminary Case for Superbugs. In Feller et al. (eds.), *Open Source Development, Adoption and Innovation*. IFIP International Federation for Information Processing 234, pp. 247–252. Springer, September 7–10 2007.

[Fut06] Futurelab. The potential of open source approaches for education. Opening Education Report. Published online, 2006.
http://www.futurelab.org.uk/resources/

[GKS08] G. Gousios, E. Kalliamvakou, D. Spinellis. Measuring developer contribution from software repository data. In *MSR '08: Proceedings of the 2008 International Workshop on Mining Software Repositories*. Pp. 129–132. ACM, 2008.

[HS02] T. J. Halloran, W. L. Scherlis. High Quality and Open Source Software Practices. In *2nd Workshop on Open Source Software Engineering*. May 2002.

[HZC12] D. Hu, J. L. Zhao, J. Cheng. Reputation management in an open source developer social network. *Decision Support Systems* 53(3):526–1058, Jun 2012.

[Imm] Immersive Education.
http://immersiveeducation.org/

[JG06] G. R. Juan Jose Amor, J. M. Gonzalez-Barahona. Discriminating Development Activities in Versioning Systems: A Case Study. In *Proceedings PROMISE 2006: 2nd. International Workshop on Predictor Models in Software Engineering co-located at the 22th International Conference on Software Maintenance (Philadelphia, Pennsilvanya, USA)*. 2006.

[Kuk06] G. Kuk. Strategic Interaction and Knowledge Sharing in the KDE Developer Mailing List. *Management Science* 52, 2006.

[Leu08] T. Leung. Open Source Community Antipatterns. In *O'Reilly OSCON Open Source Convention*. 2008.

[McC99] S. McConnell. Open Source Methodology: Ready for Prime Time? *IEEE Software* 16(4):6–8, Jul/Aug 1999. IEEE Computer Society.

[MFH02] A. Mockus, R. Fielding, J. Herbsleb. Two case studies of open source software development: Apache and Mozilla. *Transactions on Software Engineering and Methodology* 11(3):1–38, 2002.

[MGS08] A. Meiszner, R. Glott, S. K. Sowe. Free/Libre Open Source Software (FLOSS) Communities as an Example of successful Open Participatory Learning Ecosystems. *The European Journal for the Informatics Professional, UPGRADE* IX(3):62–68, 2008.

[MGS09] A. Meiszner, R. Glott, S. K. Sowe. Preparing the Ne(x)t Generation: Lessons learnt from Free/Libre Open Source Software — Why free and open are pre-conditions

and not options for higher education! In *Proceedings of the 4th International Barcelona Conference on Higher Education*. Volume 2. Knowledge technologies for social transformation. Barcelona, Spain, 15–19 July 2009.

[MHP05] M. Michlmayr, F. Hunt, D. Probert. Quality Practices and Problems in Free Software Projects. In Scotto and Succi (eds.), *Proceedings of the First International Conference on Open Source Systems*. Pp. 24–28. Genova, Italy, 2005.

[Mic05] M. Michlmayr. Quality Improvement in Volunteer Free Software Projects: Exploring the Impact of Release Management. In Scotto and Succi (eds.), *Proceedings of the First International Conference on Open Source Systems*. Pp. 309–310. Genova, Italy, 2005.

[Moo] Moodle.
http://moodle.org/

[MRW77] J. A. McCall, P. K. Richards, G. F. Walters. Factors in Software Quality. Volume I. Concepts and Definitions of Software Quality. 1977.
http://www.dtic.mil/dtic/tr/fulltext/u2/a049014.pdf

[Nea11] D. Neary. Community Antipatterns. In *Free and Open Source Software Developers European Meeting*. 2011.

[Neu05] P. Neumann. Attaining Robust Open Source Software. In Feller et al. (eds.), *Perspectives on Free and Open Source Software*. Chapter 7, pp. 123–126. MIT Press, 2005.

[Ray99] E. S. Raymond. *The cathedral and the bazaar*. O'Reilly, 1999.

[RG06] G. Robles, J. Gonzalez-Barahona. Contributor Turnover in Libre Software Projects. In Damiani et al. (eds.), *IFIP International Federation for Information Processing, Open Source Systems*. Volume 203, pp. 273–286. Springer, 2006.

[RGCH09] G. Robles, J. Gonzalez-Barahona, D. Cortazar, I. Herraiz. Tools for the study of the usual data sources found in libre software projects. *International Journal of Open Source Software and Processes* 1(1):24–45, Jan-Mar 2009.

[RHS06] J. Roberts, I. Hann, S. Slaughter. Understanding the motivations, participation, and performance of open source software developers: a longitudinal study of the Apache projects. *Management Science* 52(7):984–999, 2006.

[SAS06] S. K. Sowe, L. Angelis, I. Stamelos. Identifying Knowledge Brokers that Yield Software Engineering Knowledge in OSS Projects. *Information and Software Technology* 48:1025–1033, 2006.

[SCa] S. A. Shaikh, A. Cerone. Towards a metric for Open Source Software Quality. In *Proceedings of OpenCert 2009*. Volume 20 of Electronic Communications of the EASST, 2009.

[SCb] S. K. Sowe, A. Cerone. Integrating Data from Multiple Repositories to Analyze Patterns of Contribution in FOSS Projects. In *Proceedings of OpenCert 2010*. Volume 33 of Electronic Communications of the EASST, 2010.

[SC11] D. Settas, A. Cerone. An Ontology Based E-Learning System Using Antipatterns. In *Proceedings of ICWL 2011*. Lecture Notes in Computer Science 7048, pp. 243–252. Springer, 2011.

[Sch00] F. B. Schneider. Open Source in Security: Visting the Bizarre. In *2000 IEEE Symposium on Security and Privacy*. May 14–17, 2000, Berkley, California, USA, pp. 126–127. IEEE Computer Society, 2000.

[Sec] Seconf Life.
http://secondlife.com/

[SFF+06] W. Scacchi, J. Feller, B. Fitzgerald, S. A. Hissam, K. Lakhani. Understanding Free/Open Source Software Development Processes. *Software Process: Improvement and Practice* 11(2):95–105, 2006.

[SMSB11] D. L. Settas, G. Meditskos, I. G. Stamelos, N. Bassiliades. SPARSE: A Symptom-based Antipattern Retrieval Knowledge-based System Using Semantic Web Technologies. *Expert Systems with Applications* 38(6), June 2011.

[SS08a] S. K. Sowe, I. Stamelos. Reflection on Knowledge Sharing in F/OSS Projects. In *Open Source Development, Communities and Quality*. IFIP International Federation for Information Processing 275, pp. 351–358. 2008.

[SS08b] S. K. Sowe, I. G. Stamelos. Involving Software Engineering Students in Open Source Software Projects: Experiences from a Pilot Study. *Journal of Information Systems Education (JISE)* 18(4):425–435, 2008.

[SSA08] S. K. Sowe, I. Stamelos, L. Angelis. Understanding Knowledge Sharing Activities in Free/Open Source Software Projects: An Empirical Study. *Journal of Systems and Software* 81(3):431–446., 2008.

[SSD06] S. K. Sowe, I. Stamelos, I. Deligiannis. A Framework for Teaching Software Testing using F/OSS Methodology. In *Proceedings of the 2nd International Conference on Open Source Systems (OSS2006)*. Como, Italy, 8–10 June 2006.

[SSS07] S. K. Sowe, I. G. Stamelos, I. M. Samoladas (eds.). *Emerging Free and Open Source Software Practices*. IGI Global, 2007.

[SSSA08] S. K. Sowe, I. Samoladas, I. Stamelos, L. Angelis. Are FLOSS developers committing to CVS/SVN as much as they are talking in mailing lists? Challenges for Integrating data from Multiple Repositories. In *3rd International Workshop on Public Data about Software Development (WoPDaSD). September 7th - 10th 2008, Milan, Italy.* 2008.

[Tap08] J. Tappolet. Semantics-aware Software Project Repositories. In *In Proceedings of the European Semantic Web Conference, Ph.D. Symposium*. Pp. 78–82. June 2008.

[Tuo05] I. Tuomi. The future of open source: Trends and prospects. In Wynants and Cornelis (eds.), *How open is the future? Economic, social and cultural scenarios inspired by free and open source software*. Pp. 429–459. Vrjie Universiteit Press, 2005.

[TVN08] T. Tudorache, J. Vendetti, F. N. Noy. Web-Protege: A Lightweight OWL Ontology Editor for the Web. In *In Proceedings of the fourth Workshop in the The OWL: Experiences and Direction (OWLED)*. 2008.

[ZE00] L. Zhao, S. Elbaum. A survey on quality related activities in open source. *ACM SIGSOFT Software Engineering Notes* 25(3):53–57, May 2000. ACM Press New York, NY, USA.

[ZPZ07] T. Zimmermann, R. Premraj, A. Zeller. Predicting Defects for Eclipse. In *PROMISE '07: Proceedings of the Third International Workshop on Predictor Models in Software Engineering*. P. 9. IEEE Computer Society, Washington, DC, USA, 2007.