TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Terhi Kilamo

Essential Properties of Open Development Communities

Supporting Growth, Collaboration, and Learning

Julkaisu 1194 • Publication 1194

Tampere 2014

Tampereen teknillinen yliopisto. Julkaisu 1194
Tampere University of Technology. Publication 1194

Terhi Kilamo

Essential Properties of Open Development Communities

Supporting Growth, Collaboration, and Learning

Thesis for the degree of Doctor of Science in Technology to be presented with due permission for public examination and criticism in Tietotalo Building, Auditorium TB109, at Tampere University of Technology, on the 7th of March 2014, at 12 noon.

Tampereen teknillinen yliopisto - Tampere University of Technology
Tampere 2014

ISBN 978-952-15-3237-5 (printed)
ISBN 978-952-15-3265-8 (PDF)
ISSN 1459-2045

Abstract

Open development has emerged as a method for creating versatile and complex products through free collaboration of individuals. This free collaboration forms globally distributed teams. Similarly, it is common today to view business and other human organizations as ecosystems, where several participating companies and organizations co-operate and compete together. For example, open source software development is one area where community driven development provides a plausible platform for both development of products and establishing a software ecosystem where a set of businesses contribute their own innovations. Equally, open learning environments and open innovation platforms are also gaining ground. While such initiatives are not limited to any specific area, they typically offer a technological, legal, social, and economic framework for development. Moreover, they always rely on the associated community, the people.

Open development would not exist without the active participation of keen developers. However, people are fickle. Firstly, as one of the main driving forces for participation is own interest, "scratching your own itch", the question of how to grow and support open development rises to the forefront. Further it leads to ask what contributes to making open development successful. This is especially crucial when the product has business value. Secondly, as open development has its own governance methods and development guidelines, one is led to ask, how learning these could be facilitated, and how community participation could be supported.

This doctoral dissertation gives insight on tools and techniques that help in dealing with the multi-faceted challenge of working with and growing an open development community. It discusses these through a framework covering the five key aspects of open development: *the people* in and *the purpose* of the community, *the product* developed by the community and *the policies* and *the platform* the community needs to function.

The thesis presents work on establishing and monitoring an open development community in two different settings: a Free/Libre/Open Source Software(FLOSS) business environment and open education. The research covers going ahead with open development within the FLOSS ecosystem both from the point of view of the product and the business environment. Additionally, this thesis offers research on how developers can learn open development methods. It introduces academic open development communities through which the developers can adopt collaborative development skills. The research presented paves the way for gaining further knowledge in growing thriving open development communities.

Keywords: Open development, community driven development, open source software, Free/Libre/Open Source Software, collaborative software development, software engineering education, participatory learning, open innovation

Preface

This doctoral dissertation would not exist without the support and guidance of many people. First and foremost, I want to thank my supervisor Docent Imed Hammouda for his guidance, ideas, and encouragement throughout the process. A very special thank you to Professor Tommi Mikkonen, my second supervisor, for his comments, insight, and support, and for believing in me when I didn't. I also thank my co-authors, who have been a tremendous help and a source of support and ideas. Additionally I want to thank my pre-examiners Professor Matti Rossi and Associate Professor Björn Lundell for their review and comments, and my opponent Associate Professor Gregorio Robles for his insight and remarks.

This work has been supported by Nokia Foundation, Ulla Tuomisen säätiö, and Tuula ja Yrjö Neuvon rahasto. I thank them for the encouragement.

I am grateful for having had the opportunity to work on this thesis in company of such great colleagues. Thank you Janne Lautamäki, Tuomas Turto, Timo Aaltonen, and Matti Rintala among others. I also want to thank my friends, especially Suvi for the therapeutic jogs, cycling, and gym sessions, and my brilliant association of lovely ladies. You are all dear and this work has partly been possible because you exist.

Lastly, I want to thank my family, Jussi and Nikolas, I love you, and my parents, who have always been there for me. Extra thanks to my mother for helping with the house and taking care of a lot of things throughout this work.

Tampere, Finland February 2, 2014

Terhi Kilamo
terhi.kilamo@tut.fi

> *"And by that destiny to perform an act*
> *Whereof what's past is prologue, what to come*
> *In yours and my discharge."*-Antonio, The Tempest. Act 2, Sc 1

Contents

List of Figures

List of Included Publications

Case I: Free/Libre/Open source software development:

[I] T. Kilamo, I. Hammouda, T. Mikkonen, and T. Aaltonen. From Proprietary to Open Source – Growing an Open Source Ecosystem. In *The Journal of Systems and Software (JSS)*. Volume 85, Issue 7, pages 1467-1478. July, 2012, Elsevier Science Inc.

[II] T. Kilamo, T. Aaltonen, and T.J. Heinimäki. BULB: Onion-Based Measuring of OSS Communities. In *Proceedings of the 6th International IFIP WG 2.13 Conference on Open Source Systems (OSS'10)*, pages 342–347. Notre Dame, IN, USA, May 30 – June 2, 2010, Springer.

[III] T. Kilamo, T. Aaltonen, I. Hammouda, T.J. Heinimäki., and T. Mikkonen. Evaluating the Readiness of Proprietary Software for Open Source Development, In *Proceedings of the 6th International IFIP WG 2.13 Conference on Open Source Systems (OSS'10)*, pages 143–155. Notre Dame, IN, USA, May 30 – June 2, 2010, Springer.

Case II: Open learning environments:

[IV] T. Kilamo, I. Hammouda, and M.A. Chatti. Teaching Collaborative Software Development: A Case Study. In *Proceedings of the 34th International Conference on Software Engineering (ICSE'12)*, pages 1165–1174. Zurich, Switzerland, June 2 – 9, 2012, Institute of Electrical and Electronics Engineers (IEEE).

[V] T. Kilamo, I. Hammouda, V. Kairamo, P. Räsänen, and J.P. Saarinen. Applying Open Source Practices and Principles in Open Innovation: the Case of Demola Platform. In *Proceedings of the 7th International IFIP WG 2.13 Conference on Open Source Systems (OSS'11)*, pages 307–311, Salvador, Brazil, October 6 – 7, 2011, Springer.

[VI] T. Kilamo. The Community Game: Learning Open Source Development Through Participatory Exercise. In *Proceedings of the 14th International Academic MindTrek Conference: Envisioning Future Media Environments (MindTrek'10)*, pages 55–60. Tampere, Finland, 2010, Association for Computing Machinery (ACM).

Author's Contribution

The candidate has been the main contributing author or one of the main authors of each of the publication included in this thesis. The candidate's role is discussed paper by paper in the following.

I **From Proprietary to Open Source – Growing an Open Source Ecosystem.**

The candidate was the main responsible author for writing the text for the publication. Each author participated in the conception of the paper, while the candidate had the main responsibility of closing the gaps and bringing everything together with previously published work in this publication. Although the paper draws together research conducted by the research group over several years and is hence a product of an interactive and iterative process, the candidate was the main author responsible for the syntheses of the group's previous work. The candidate together with the second author further kept in contact with the representative of the example case which was used to report the framework as a whole for the first time. This work is further extended with the candidate's contribution in [53] with a business perspective.

II **BULB: Onion-Based Measuring of OSS Communities**

The candidate was the main responsible author writing this publication. The data analysis for the publication was conducted together with the second author as well as writing the text. All authors jointly worked on the design and the conception of the work. The third author was mainly responsible for the constructive setting and the data collection.

III **Evaluating the Readiness of Proprietary Software for Open Source Development**

The candidate was the author responsible for the writing process. The candidate further contributed to the paper by collecting the information on the company's experiences with applying the approach in the case

in Section 4.2 of the paper (the Gurux case) and by reporting the findings in the paper. The publication was written in a collaborative and interactive process. The research project this dissertation work was conducted as a part of was developing the R3 framework depicted in the paper at the time the candidate joined the project. Most of that work is prior to the candidate's.

IV **Teaching Collaborative Software Development: A Case Study**

The candidate conducted the research together with the second author and did the data analysis by herself. The third author of the publication contributed theoretical background to the work while the rest of the paper was solely the candidate's responsibility.

V **Applying Open Source Practices and Principles in Open Innovation: the Case of Demola Platform**

The candidate was responsible for the entire publication: the conception, design, analysis, and writing. The concept of Demola – the case study context in the publication – was conceptualized and created by the third, fourth, and fifth authors but the work conducted for the publication is the candidate's.

VI **The Community Game: Learning Open Source Development Through Participatory Exercise**

The conception of the exercise depicted in the publication was an interactive effort where the candidate had a major contributing role. The design, conception, and analysis is the candidate's and the candidate is the sole author of the paper.

Chapter 1

Introduction

Openness is becoming prevalent. The open data movement [75], Free/Libre/-Open Source Software (FLOSS) [77], open government [76] initiatives[1], and open standards [95] are all examples where open access, transparency, and participation is valued and promoted. From the business perspective, open innovation [12] and open development [4] are allowing a wider audience of interested individuals to participate in endeavors that have company interest and business value as well. While traditionally such advantages have been kept within the company, openness allows a wider group of partners to strengthen and expand the scope. A closed approach would be tied to the resources the company has at hand and restricted by the company policies and know-how. Similarly, from a learning perspective open education offers a wider access and more versatile learning opportunities to learners around the globe through services like the Khan Academy[2] or open education providers like the Open University[3].

A common way today is to view businesses and other human organizations as *ecosystems* instead of separate actors [73]. The term ecosystem has further emerged as a commonly used notion in software economy [69]. In a business ecosystem companies function as a single unit. They co-operate and compete to produce new innovations and products, and to satisfy their customers' needs. In turn, the goal in a learning ecosystem is for the individuals to gain competitive skills and expertise that are beneficial from an organizational viewpoint as well. Ecosystems such as the software ecosystem, for example, typically rely on a shared platform on top of which different parties contribute their own innovations specific to their area of expertise [5]. This way each

[1]Finnish initiative: `http://www.vm.fi/vm/en/05_projects/0238_ogp/index.jsp`, US initiative: `http://www.whitehouse.gov/open`

[2]`https://www.khanacademy.org/`

[3]`http://www.open.edu/openlearn/`

participant is free to extend the ecosystem and still all the participants can gain benefits and utilize each contribution.

Such sharing of ideas and contributions is a form of openness, at least within the scope of the ecosystem. In open development a group of peers distributed worldwide collaboratively produce content, artifacts, knowledge, or similar shared resource. Generally common aspects of open development include transparency of development and the freedom to extend the available artifacts into new and more complex products and tools, and thus open development can be seen as a plausible platform for an ecosystem to build upon.

Open development is done by people, in collaboration with other people and, most commonly, focusing on a product that is meant for people to use and gain benefits from. The participating people may work in varying environments, may have their particular backgrounds and act under different conditions [9], yet sharing in many cases the same activities and interests. Furthermore, their motivation for participation may be driven by a variety of factors from employment and business drivers to altruistic incentives and self-improvement [58, 90]. For example recent software development settings such as pair programming [79], global software engineering [47], and open source software [85], exhibit the trend where individuals – from developers to users – join together to perform common software engineering activities. Similarly, there are initiatives such as Teaching Open Source [26] and Free Knowledge Institute [25] that focus on research and promotion of open and collaborative learning. In such online settings, participants need to work in collaboration with limited face-to-face contact. This leads to a variety of challenges including the technical, the organizational, and the social. Hence open development needs to address a number of challenges such as motivation, integration, and exploitation of innovation [111]. There is a need for a governance framework [30] that enables organizational alignment of the different partners, proper handling of intellectual property rights issues, and the emergence of new kinds of business opportunities.

1.1 Motivation

The rise of the Internet has enabled engaging participants worldwide in ventures from all walks of life ranging from crowd funding the Death Star[4] (and

[4] `http://www.kickstarter.com/projects/461687407/kickstarter-open\`
`\-source-death-star`

naturally also the rebel alliance's X-wing squadron[5]) to developing healthcare software[6]. People participating out of their own personal interests raises the question of how to support open development and what contributes to making open development successful. This is especially crucial from an ecosystem perspective where the product has business value to a company or an organization.

Open development relies on the developer community. Such communities typically work online, and thrive on participants' communication and interaction to generate member-driven content [61]. A healthy, working community is typically seen as a layered onion-like structure [16, 74], with different participant roles from core developers to users of the developed product. It is important for a healthy community to have participants on each level of participation [17]. Success of open communities relies on a large number of identifiable metrics [15, 82] over a life-cycle from birth to maturity [50].

Discussion on communities often regard them through expressions such as building, constructing, or forging a community [36, 54, 99, 112]. However, open development – and communities in general – flourish through the active participation of individuals, who in the case of development communities share the common goal of developing a product. Still, their personal interests may vary and focus on a wide range of different areas of development [43, 44]. People are not built or forged, and thus the focus should be put, as the title of this thesis suggests, on growing communities, growing as one would grow crops instead of a simply focusing on growth as an outcome of building.

Growing is an activity where growth happens when certain prerequisites and conditions are met. The motivation of this thesis is the viewpoint of these conditions: what are the essential properties of open development and how to handle them in order to support collaboration and grow communities in the open development domain. The question is, how to approach open development to allow the community to work together and grow in a healthy, sustainable way. Open development is thus also not without its own governance methods and development guidelines [43, 81, 85]. Understanding team work, tools, and open development methods is of more and more value as more and more development is done in global teams and through open participation initiatives [9]. Hence, how participants can learn these and how community participation can be supported are of further importance.

[5] `http://www.kickstarter.com/projects/simonkwan/crowdfunding-rebel\`
`\-alliance-x-wing-squadron`

[6] `http://en.wikipedia.org/wiki/List_of_open-source_healthcare_software`

1.2 Scope of Work

This doctoral dissertation discusses the challenges of open development focusing on the properties that are pivotal in growing an open development community and in supporting the growth of the community. This work investigates how open development can act as a platform for learning skills that are essential in the Web 2.0 era [78] where the Internet is enabling all kinds of forms of collaboration. The thesis presents research work on two open development community cases: Free/Libre/Open Source Software ecosystem and open education, and draws their findings together under a common framework. The research questions answered are: 1) What aspects of open development are essential in supporting the growth of open development communities and at the same time supporting the collaboration of the participants? 2) What kind of activities are needed to ensure and support community growth? and 3) How can collaboration be learned through participation in an open development community? In order to formulate the research questions and when developing the framework, a literature review approach has been used [63, 110].

In this thesis, the aim is to recognize what kind of aspects need to be addressed in order to establish an open development community and further, in order to give the community a chance to grow and thrive. The objective is not so much to give a cookbook-like set of rules and guidelines for building a perfect community, but to focus on the essential aspects needed to enable growth of the community and to support collaboration of participants.

1.3 Contribution

The thesis consists of six publications and an introductory part that summarizes the publications through a novel framework as a thesis contribution. The research drawn together in this thesis covers what is needed to grow a successful open development community and how people can adopt open development skills. The thesis further addresses supporting learning how to work with open development methods, as people are a pivotal part of open development. Specifically, the thesis contributes: *a framework of open development communities* that addresses their essential aspects, a set of *guidelines* for establishing and growing a FLOSS business community and an approach to *enable learners to attain understanding of open development and the skills required* with an open educational development environment. The thesis statement can be formulated based on the research as: *An open development community is characterized not only by the community itself but also by the*

product the community aims to develop and improve. As such, an open development community needs to be viewed through all of its essential elements which include: people, purpose and product, and the community policies and the development platform that support them.

Work on two cases of open development is included in this work. First as a contribution, a common framework derived based on the cases, called five Ps, is presented. It is then further discussed through the two case contexts as intances of the framework. Case I provides the OSCOMM approach to growing open software development communities in a software business setting. In Case II, an open education approach is taken with an open development environment for learning, KommGame. The open development community as a community of practice [59], where participation incites learning, ties the case viewpoints together. In Case II the open innovation platform Demola acts as the case study context and it incorporates elements of both approaches. The work done there contributes insight on how both aspects —FLOSS ecosystem and open education— can appear in a common setting and support each other.

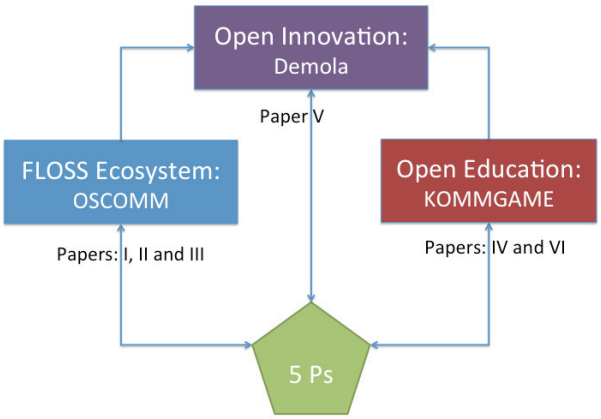


Figure 1.1: Contribution of the Thesis

Figure 1.1 shows the work in relation to all of its elements. The contribution of the five Ps framework is derived based on the work done with the two cases of open development communities and based on the existing literature.

In addition to the included publications, the candidate has been the main contributing author in Kilamo *et al.* [53], which is a supporting publication to Case I. In it growing open source communities is discussed from the business perspective as well as from the viewpoint of the software product and the software ecosystem surrounding it. The publication complements Publication I included in the thesis. A supporting publication to Case II, Goduguluri

et al. [41], takes a software platform perspective with the candidate as a supporting author. The candidate had an active part in the conception of the environment described in the publication. The environment depicted acts as the community platform in Case II.

1.4 Outline of the Thesis

The organization of the introductory part of this thesis is the following. Chapter 2 sets the background of the work. It discusses the topic of openness and open development and presents related research relevant to this thesis. Chapter 3 describes the theoretical background of the research approach and methodology used in this thesis work and gives the target questions. In addition, the chapter describes how the research was conducted and explains the used research method.

Chapter 4 introduces the open development community framework adopted for this work. Then Chapter 5 goes on to describe and discuss the two cases of open development: open source software development and open education. The chapter, through the framework introduced in Chapter 4, shows the developed OSCOMM framework, the people aspect of open development, and how participation and learning participatory skills can be supported with an open development platform called KommGame.

To conclude the introductory part, Chapter 6 gives a summary of the included publications. The chapter further summarizes and concludes this thesis with an evaluation of the thesis questions and provides possible directions for future research.

Chapter 2

Background

Communities are about people. They flourish on the notion of collaboration of people. Communities, however, are not built nor do they emerge out of thin air [81]. As mentioned, works like [36, 99, 112] use the wording ”build”, but what they actually discuss is more in the lines of nurturing instead of building. Hence the claim that supporting communities to emerge and grow is not construction work. The idea of ”scraching your own itch” [85] – joining the community out of your own motivation – further supports this. The overarching approach in the thesis is that establishing communities should be treated as an endeavor of nurturing or growing, not building of an inanimate object.

Taking a philosophical approach, open indicates the notion of allowing any keen participant to join and contribute. Openness incorporates the impilicit notion that anybody can ”hack anything” [85]. However, it is worth noting straightaway that communities are a group of individuals discussing, sharing ideas, mingling, and getting to know each other over a shared interest, but they are not necessarily open for anyone to join [43]. Some communities are closed [22] or gated [43, 93] where terms, licences, and guidelines delineate who and how people can participate. Even when the community is open, moving ahead may be more difficult in some than in others.

Open development communities work on the basis of commons based peer production [4]. It relies on a group of like-minded participants, the developer community, to build and improve a product: software, content, or relevance to name a few. The following discusses the traits common to open communities, including open development communities specifically.

2.1 Openness and Communities

Open communities, such as online forums or Free/Libre/Open Source Software development communities, rely on a group of individuals to carry out discussions, to build a product or otherwise interact with each other. In [54] Amy Jo Kim defines a community as:

> "a group of people with a shared interest, purpose, or goal, who get to know each other better over time."

In [81, p. 10] Jenny Preece extends this definition in the scope of online communities to include policies that guide people's interactions and computer systems that support and mediate social interaction in the community. Through these definitions, we can conclude that an open community consists of the *people*, the community's *purpose*, its *policies* and the software *platform* the community runs upon.

To describe a community with a few more words, interacting with each other through conversations about and around their shared interest molds a group of *people* with a shared preoccupation into a community [81]. Some further perform special tasks for the good of the community. These people, participating in the community, share a common *purpose* which ties the people together as such and which draws cooperative individuals into the community. Additionally, a community needs a set of *policies* that guide the communication and co-operation in it. Policies also set the decision making processes. Software systems, on the other hand, are needed to form a common *platform* that enables interaction, participation, and creates an impression of togetherness for the participants as in these cases the community typically works through the Internet. The following sections discuss each of these four in more detail. Open development brings in a fifth element, the *product*, and as one of the contributions of the thesis, this topic is discussed in Chapter 4. Lately, there has been a new rise in so called hackerspaces or makerspaces which combine the virtual working environment with a physical one by having real world meeting places [71]. In this thesis the focus is kept on communities that work online.

2.1.1 People

Open development needs people participating in the development community. There are varying reasons for what motivates people to participate in open communities. Access to information and expertise useful to the participant, getting help to a specific problem, or personal gain can motivate participation. The motivation can also be self-actualization and participation is done

for finding joy and challenges to personal, existing knowledge [108]. Gaining reputation, often a professional one, is a significant driver for contribution as well [109]. Such motivational aspects can further be divided and refined into extrinsic and intrinsic ones [90].

Intrinsic motivators are those that are enjoyable or interesting to the individual inherently while extrinsic refer to a separable reward or outcome that motivates the individual's actions. Within the context of open development communities the intrinsic factors are further divided into enjoyment based and community-based motivations [58]. Consequently, all three types of motivations can be found in people participating in open development.

The characteristics of the open community influence the motivations. For example, open innovation online communities report intellectual challenges, fun, interest towards the topic, possibility to influence and creativity as the main motivational factors [2]. In turn in FLOSS communities creative project experience and being paid rank high [58].

When viewing the community through the people, the social structure of the community has three aspects to be considered: individuals, their actions, and interactions [16]. The people themselves give an idea about the community and its size. How the people interact and work in the community gives further insight on how the community functions. The social structures of communities vary. There typically are central figures at the heart of the community [48]. A large majority of participants get involved only for a small amount of time.

2.1.2 Purpose

A community grows around a shared purpose, interest, or goal. It is the glue that ties a seemingly random group of people together [54]. The purpose can and will evolve over time. However, the purpose of the community has influence on people's interactions within the community [106]. The purpose also has relevance to who gets involved —a wider purpose draws a wider group of people into the community [81].

One key aspect for stating the purpose is for the community to have a mission statement [54]. Through it, the type of the community, why it exists, and who are the intended people can be communicated. The mission statement expressing the purpose clearly and compellingly determines what kind of people get involved [43]. Telling a vibrant story about the purpose and the goals of the project is a way to get people enthusiastic about participation.

2.1.3 Policies

Communities, open communities in particular, need a set of policies to direct and support them [81]. Some are clearly stated while some are less formal or even unwritten codes of conduct formed by the community. Nonetheless, the policies affect who join the community.

A governance model decides the participants' roles in the community, what can and cannot be done, and how decisions in the project are made. Governance models can take a form of a meritocracy, a consensus-based democracy [33], or follow more on the lines of a benevolent dictatorship [38], where the final say in matters falls on a single individual. For example, the Linux community follows the latter approach whereas the Apache HTTP Server Project[1] [32] is an example of the former. Typically open communities do in any case expect some level of acknowledgement of merit, regarless of the main governance model. The governance model affects the volunteer participation [93].

In the case of open development communities, the governance policies are affected by policies regarding the intellectual property rights of the contributors. These include possible contributor licence agreements (CLA) [87] and the licencing details of the product [62]. As in open development a community works on a shared product or artefact, and thus each own copyright to their respective contributions, licencing allows the community to grant others rights to the product. Contributor licence agreements are a way of defining terms under which participants' intellectual property is contributed to the development of the product and it may require the contributor to assign the copyright to the governing body of the community. Major open development organizations such as Apache Software Foundation[2], Eclipse[3] and Free Software Foundation[4] utilize CLAs. It helps in the management of product licencing, possible relicencing, and in cases of copyright dispute.

One key issue among participants is trust. Showing mutual respect enables cooperation. In electronic environments the issue of trust that relates to open communities falls into the trustworthiness of information, information systems and online relationships [13]. Trust is developed in an online community much like in the real world based on positive experiences and when people meet their promises and expectations [81].

[1] `http://httpd.apache.org/`
[2] `http://www.apache.org/licenses/`
[3] `http://www.eclipse.org/legal/CLA.php`
[4] `http://www.gnu.org/licenses/why-assign.html`

2.1.4 Platform

Open communities typically work on an online software platform even if they might have real world activities as well. Typical platforms are online discussion forums, different types of chats, and mailing lists [81].

The platform must suit the community needs, as its role is to support the community. This leads to the need to understand participants' wishes, as a platform that is suitable and usable for the participants supports learning, creativity, and productivity. The software platform affects the participants' activities and thus has an effect on the communication and interactions within the community.

2.2 Collaborative Development

Open collaborative development has its roots in peer-production where participants, a group of peers, interact with each other and collaborate [4]. While Free/Libre/Open Source may be the most famous example of such collaborative effort, it is not the first nor the only domain where peer-production is in use. Communities have influenced the development of things such as cars [56] or sport-related consumer products [34]. In addition peer-production is visible throughout the web where people provide content[5], relevance or value to goods[6], or distribute content[7].

Globally distributed development of products is not necessarily always collaborative in nature but such multi-site projects can also work on a joint effort and in collaboration with each other. Human aspects such as social ties and knowledge sharing as well as tools and procedures are emphasized when development teams are located globally and work on a shared project [57]. Hence, the community aspects are visible also there.

Open development communities encompass the aspects of open communities and peer-production as well as product related details such as licensing and project processes. As a working definition for the thesis, an open development community is a community that keen participants can join with their own contributions. It is further a development method where a transparent process is applied for a plethora of different types of goods through peer-production.

[5] `http://www.wikipedia.org`
[6] `http://www.amazon.com`
[7] `http://www.gutenberg.org/`

2.3 Related Research

The main goals of this work are identifying the essential properties for open development communities with an ecosystem mindset and studying these in the context of two different settings: industrial Free/Libre/Open Source Software[8] and collaborative learning. Albeit not directly bound to a systematic literature review, the study context is grounded in the current body of knowledge. Here, the focus is on the related body of knowledge of both cases. Case I is approached from the point of view of starting FLOSS communities, evaluation of existing communities' health, and community lifecycle research. Case II is rooted in the existing knowledge from the point of view of usage of FLOSS in software development education and collaboration. Both are approached with a literature review using the relevant scientific databases (IEEExplore, ACM Digital Library, Springerlink, Google Scholar, ScienceDirect) with the recommended approach from the senior editors of MISQ [110].

Through ecosystems the work is related to ecosystem health research – a healthy ecosystem has the means to grow. In their paper [49] Iansiti and Levien conceptualize ecosystem health through three measures: productivity, robustness and niche creation. Health measures for business ecosystems health are further developed by den Hartigh et al. [20]. However, these works are strictly business oriented and on a general level. The thesis focuses on open development community specifics within an ecosystem. The viewpoint is community-centric and the ecosystem is viewed as a working environment. In [103] the authors present a model that describe the key characteristics of a software ecosystem. The work is related to this thesis as it investigates software ecosystems through defining their key characteristics that include issues such as platform planning and community building discussed also here. However, the focus of this work is on open development settings beyond a strict business need. Furthermore, the thesis is not limited to software development only but takes a more general view.

Case I: Free/Libre/Open Source Software Development

Case I investigates establishing and fostering open development communities in the Free/Libre/Open Source Software ecosystem. Therefore, work on these relates to that presented here.

Fogel [33] presents a cookbook-like guide for starting and running a free

[8]It is acknowledged here that there are lobbies for both free software and open source software vernacular, and that there are several flavours and stances to the use of either term. However, here, the term FLOSS is used due to its neutrality as the demarcation in the terminology is of no essence from the viewpoint of the thesis.

software project in general. The book takes a very detailed and step-by-step approach to Free/Libre/Open Source Software with the different perspectives ranging from instructions for the technical infrastructure to the underlying political issues. In comparison, this work approaches open development through grouping of the essential elements instead of trying to cover the minutiae.

Establishing a Free/Libre/Open Source Software development community in an industrial setting is a key issue of Case I. Thus the attributes of FLOSS are an important angle to it. Open source maturity models such as OSMM^{TM}(Open Source Maturity Model) [42], QSOS (Qualification and Selection of Open Source Software) [83], and BRR^{TM}(Business Readiness Rating) [8] aim to aid companies in adopting FLOSS. While these models evaluate existing Free/Libre/Open Source Software projects this thesis focuses on giving birth to and fostering FLOSS development.

In their paper [91] the authors define a lifecycle model for Free/Libre/Open Source Software based on existing FLOSS projects. The paper argues that Free/Libre/Open Source Software projects go through three phases: a cathedral phase, a transition phase, and a bazaar phase. However, in contrast to the metaphor of the "cathedral and the bazaar" [85], the phases are complementary and represent common evolution phases of most open source projects. The lifecycle model is further empirically assessed through historical data using a number of case studies by Capiluppi et al. [10]. The three-phase classification fits the work on FLOSS communities presented in this thesis as the framework discussed in Subsection 5.1.2 follows the lifecycle model. The life-cycle of online communities in general together with information systems lifecycle viewpoint is taken in [50]. The authors use a literature review to propose success conditions to improve the chances of the community becoming a successful one. In this thesis the communities follow a similar lifecycle approach: inception, creation, growth, maturity. In contrast, the framework proposed here discusses the properties of communitites over the lifecycle and doesn't simply tie them to a specific stage.

The problem of open source community building has also been researched in the licentiate thesis by Stürmer [99], who describes qualitative research on eight successful FLOSS projects. The study is based on interviews with a representative of each project. The interviews go through the lifecycle of the projects from inception to present. The main contribution of the study is in describing how to initialize a Free/Libre/Open Source Software project and how the project is promoted. In comparison to the work in this thesis, the main difference is in the research approach. The work here is not an outcome of hindsight, but comes out of active participation in the process. Furthermore, the researcher has an active role instead of one of an observer.

Stephanie Freeman's doctoral dissertation [36] investigates OpenOffice.org and the Finnish public sector as an adopter of FLOSS from the points of view of developer motivation, the users, and leadership. Again, in comparison to the work here, Freeman has an observer's view. Additionally, the focus is on the people and community-adopter interaction on a single case while this thesis takes a more of a holistic view.

Case II: Open Learning Environments

Case II looks into incorporating collaborative aspects to education and how open development can be taught in an environment where learners work together in a social and learner-centric setting that resembles a real world context. The work relates to open learning ecosystems [6] and utilizing games in education [52] that are both gaining more ground in education. Initiatives like Teaching Open Source [26] and Free Knowledge Institute [25] further illustrate this trend.

Pedagogical principles and their relation to development of collaborative virtual environments is one key aspect of open learning environments and their role as a community of practice [88]. While not directly tied to the work here, the pedagogical principles have bearing with the case and principles such as "facilitating knowledge building" and "providing support for community building" were regarded in the work. The thesis is further related to massive open online courses (MOOC) as they convey the idea of a course provided online with an open access to learners. The environment used as the case context for KommGame in Case II does follow a lot of the same principles as MOOC platforms [65].

Borrowing practices from open development, mainly FLOSS, has been tried out in education. An open source ecosystem can be used in teaching software engineering in general as suggested in [97]. In [68] a hybrid approach blending open source community principles with education was used. The approach shares similarities with the ideas behind the FLOSS game environment in Case II.

In papers [39] and [64] the authors report cases of teaching open development through FLOSS. The courses require participation in, and contribution to a Free/Libre/Open Source Software project. Similarly, large companies have organized coding events such as the Google Summer of Code[9] since 2005 to promote open source development to students and to provide them with an opportunity to work on things related to their studies while engaging them in real-life software development. A similar approach is the Apache Software

[9] `https://developers.google.com/open-source/soc/`

Foundation's (ASF) Mentor Programme[10] where ASF projects provide mentors for project newcomers with the aim to allow mentees to learn how to work and make valuable contributions to Apache projects. The difference to the work done for this thesis is that here a learning community is established to support learning of the key issues instead of utilizing an existing, working product project.

From the collaboration viewpoint, the authors in [102] present how requirements analysis is taught by using collaborative students teams and students themselves playing also the part of the client. Collaborative development, and getting to know distributed, collaborative working environments, is in focus in [29] where student teams work on software development each assuming a role in the project. Also in [9] a multidisciplinary distributed collaborative environment is presented. The authors discuss using a process which makes students face real world issues and combines different disciplines that together are needed in a completed software product. Teaching aspects of collaborative development from a software development point of view are also in focus in [45]. The paper discusses characteristics of effective collaborative learning and shares similarities with this thesis in using a collaborative tool. The tool here, however, addresses all aspects of open development.

One essential element of Case II is use of reputation systems [86] as part of the community. In [98, 101] Temperini and Sterbini present a reputation tool for e-learning called SOCIALX, aiming to increase motivation, collaboration, sharing, and critical thinking and to this way meet the learning objectives better much like in Case II in this thesis. The authors do not aim at forming a functioning learning community but the paper shows use of reputation in an online environment paving ground for its use as a learning community tool as well. Cruz et al. [18] discuss the use of reputation system in Communities of Practice [59]. The goal is the same as in Case II: to promote trust among participants, and thus encourage participation.

[10] `http://community.apache.org/mentoringprogramme.html`

Chapter 3

Research Approach

Developing artifacts in a group, team, or community of people as a discipline, software engineering (SE) being one, encompasses not only technological but social aspects as well [23]. Furthermore, development communities such as Free/Libre/Open Source Software development form their own type of social and communication structures [16, 74]. Hence research of such disciplines is often conducted within actual, largely industrial settings. This further makes research a collaboration of the researchers and the practitioners. Methodologies used must hence be suitable for studying current phenomena within their natural context. Such approaches include case studies [89] and action research [19], which further emphasizes the need to address actual organizational needs while acquiring scientific knowledge as well. In the following sections, this chapter gives the theoretical foundations of the research approach adopted in the work this thesis presents. It gives the research questions to which the thesis answers. In addition, it is discussed how the research was conducted. The following also describes the research method used in connection with the research.

3.1 Research Questions

The research presented in the thesis focuses on investigating open development in two types of settings: Free/Libre/Open Source Software and open education. The work examines establishing and growing an open development community in both settings as well as connecting the two. While establishing an open development setting is straightforward, getting people excited and to participate is not. This research aims to answer what open development essentially requires, in order to be able to draw people to participate and to grow the community.

The research questions the work aims to answer are:

Q1: What aspects of open development are essential in supporting both the growth of open development communities and their participants' collaboration?

Q2: What kind of activities are needed to ensure and support community growth?

Q3: How can collaboration be learned through participation in a open development community?

The research method used to approach the questions followed the collaborative practice research approach. The approach was chosen as it allowed the combination of addressing practitioners' needs with the additional goal of contributing new knowledge by deriving answers to the research questions. The methodological approach was especially suitable as it allowed the possibility to create three types of complementary knowledge.

3.2 Collaborative Practice Research

Collaborative practice research (CPR) [66] is an approach that combines action research, conventional practice studies, and empirical experiments. Establishing well functioning relations between research and practice is its main focus. The action research method gives direct access to practice. Experiments similarly provide access to practice in a way at least partly controlled by the researcher. On the other hand, practice studies, such as case and field studies, come without direct involvement of the researcher in the studied practices but with a focus on understanding them. Collaborative practice research combines these so that the research is organized as an action research effort but is complemented with experiments and practice studies for providing support for more rigorous research results.

Action Research

Action research [3] is most commonly referred as a method that:

> "aims to contribute both to the practical concerns of people in an immediate problematic situation and to the goals of social science by joint collaboration within mutually acceptable ethical framework." [84]

as defined by Rapoport.

Action research is viewed as a cyclical process, a five stage cycle defined by Susman and Evered [100]. Figure 3.1 shows the five phases which are:

1) diagnosing, 2) action planning, 3) action taking, 4) evaluating, and 5) specifying learning. The diagnosing phase where the problem is defined is the starting point for the iterations and findings of each iteration act as input for the problem diagnosis in the next cycle.

Figure 3.1: Action Research Cyclical Process

Case Study Research

A case study is a method that studies phenomena in their real life context rather than in isolation [114]. In case studies, boundaries between the studied phenomenon and its context are not clearly separable, and case study research utilizes multiple sources of evidence. Software engineering is one field of research where the questions this thesis aims to answer are such that case study research is a plausible research method option [89]. Software development is a multidiciplinary field and the studied artefact and its environment cannot be clearly separated when studying software engineering activities.

Case study research contains five major steps: 1) defining objectives, 2) defining data collection procedures, 3) data collection, 4) analysing the data, and 5) reporting.

Empirical Experiments

In a controlled experiment some testable hypothesis is studied by manipulating one or more independent variables in order to measure their effect on one or more dependent variables [23]. For example, in software engineering experiments include practitioners performing some task or duty. In software engineering research a controlled experiment is normally either a randomized experiment, where a random process is used in deciding between alternative paths, or a quasi-experiment, where randomization of conditions is not used [96]. In controlled experiments software engineering practitioners or teams perform one or more engineering tasks with the objective of comparing different processes, methods, or techniques.

3.3 Research Method

The research method used in this dissertation work falls under collaborative practice research as research was conducted in close collaboration with practitioners – the open source businesses, and software engineering students and teachers. While action research is in direct connection with practice the research process can be difficult to control when making improvements to practice is the primary goal. Therefore, this work utilizes experiments [55] and case studies [89] to conduct the research within the action research iterations.

The method was chosen for two reasons. First, it provides an ideal setting to conduct research activities in project-bound work contexts. It enables working collaboration with the practitioners and fulfilling research interests while contributing to the practical concerns of the project partners. Secondly, through the specified learnings and problem diagnosis in the iterations, findings toward the thesis aim of identifying the essential properties for growth could be made.

The dual imperative of action research [67] involves both problem solving and research interests into the aims of research. The knowledge, depicted in Figure 3.2, these aim to create is three-fold: 1) to develop *understanding* of the systems engaged in, 2) to build new knowledge to *support* practice, and 3) to *improve* practice.

These three goals are distinct and do not need to be targeted simultaneously. However, as Figure 3.2 suggests, the different research activities complement and support each other within the iterative cycle of action research.

In the publications included in the Case I of this thesis, Publication [III]

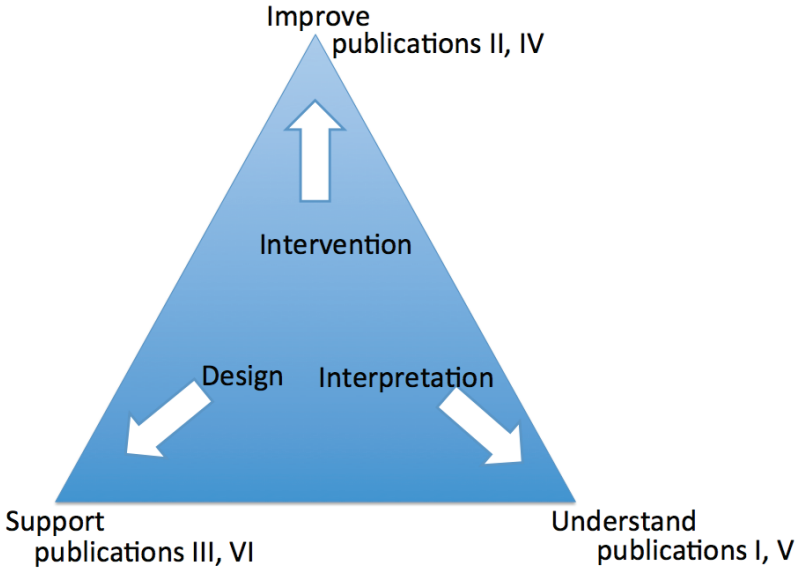


Figure 3.2: Different Knowledge and Types of Research Activities

aims to support practice, Publication [II] focuses on a method to improve practice and Publication [I] draws these together with understanding of the FLOSS ecosystem, the supporting publication [53] extending the work further with a business perspective. For Case II, Publication [V] makes interpretations and collects understanding on open innovation practice, Publication [VI] falls into the category 2: building new knowledge on adoption of community practices, and, finally, Publication [IV] leans toward improvement of practices. However, the second case is clearly more constructive in nature.

Figure 3.2 further represents the three different types of research activities knowledge can be gathered with. As pointed out, action research is an iterative process where each cycle acts as the initiator and input for the next one; here the cyclical process defined by Susman and Evered [100] (see Figure 3.1) is used as the frame of the research work. The two cases of open development communities studied in the scope of this work both went through three iterations. Tables 3.1 and 3.2 summarize the iterations. The iterations for both cases are discussed in more detail in the following.

3.3.1 Case I: FLOSS Software Development

Case I focused on studying FLOSS development (see Section 5.1) within an industrial setting. In initiating the case, the problem definition was how to establish a development community for a previously in-house product. An

action to identify the possible issues and hindrances in practice was planned. The project developed a way to approach this problem concretely – a framework to run through as a checklist called the R3 framework [III]. The flow of the research activities from iteration to iteration are presented in Table 3.1.

	Initiating	**1st Iteration**	**2nd Iteration**	**3rd Iteration**	**Closing**
Problem diagnosis	How to establish a FLOSS community	Evaluate the readiness to release	Monitor the progress of a community	How to grow an ecosystem	
Planned action	Identify needs for improvement in practice	Implement a checklist for essential elements	Measure activity and participation in community	Draw best practices together	
Action	R3 framework	Applying R3 in practice	Establishing BULB in industrial setting	Applying OSCOMM framework in practice	
Exit					Action part closed
Evaluate experiences		Bottlenecks identified What next?	Complete framework is needed		
Specified learning		R3 helps making preparations for software release	BULB gives valuable, up-to-date information	Required elements clarified and documented	Results to Q1 and Q2

Table 3.1: Case I: Action Research Performed

1st Iteration

The first iteration worked on evaluation of the readiness to start growing a community. As the planned action, a checklist to go through the essential elements of software development through open development was drafted. The checklist was applied in practice to two industrial cases. Using a case study was a natural method for this iteration as case studies are conducted in real-life settings and do not require phenomena to be studied in isolation. Here the research aimed at providing relevant information to the businesses with a real-life goal to open their source code.

The cases showed that the designed evaluation process sheds light on possible matters of improvement and can act as a set of recommendations on working with the software. The work done within that scope is outside this thesis. However, the specified learning was that the developed framework identifies the points that need to be solved before open development, and further, it raised a question that as development progresses, how can the community be monitored and supportive actions taken based on current, prevalent information.

2nd Iteration

The problem situation was that business decisions were made on a scarce set of data in companies basing their business on open development. Furthermore, not a lot of information on how the community was doing got collected. In the second iteration, an action to measure the participation and the activity of the community continuously was taken. A method to follow several data sources in a continuous yet simple way was developed and established in an industrial case. Again, a case study was a suitable research approach as it allows the community to be studied in its own context.

The second iteration showed that measuring the community gave valuable information through which the progress of the community could be monitored. It became apparent during the iteration that different stages of growing a community should not be considered as separate issues but a complete framework is needed instead. In addition, an open development community is not an island, and a wider ecosystem scope should be chosen.

3rd Iteration

As the research entered its third iteration the problem was formulated as growing a software ecosystem for a software product. The action planned was to draw together the best practices from the earlier iterations. A framework for growing an open software development community with an ecosystem scope in mind was established and applied in practice in an industrial case.

The third iteration closed the action part. It clarified and documented all the required elements and provided further support in the form of a complete industrial case. From the point of view of this thesis and its research questions answers to questions one (Q1) and two (Q2) were gained.

3.3.2 Case II: Open Learning Environments

The initial problem of the open learning environment case was specified as how to incorporate collaboration into learning and through that enable students to learn aspects of community driven development. A participatory exercise – a ”community game” – where a group of students formed a open development community within a classroom setting was devised [VI]. The flow of the research activities from iteration to iteration for Case II is presented in Table 3.2.

1st Iteration

The first iteration of Case II aimed at assessing how working in collaboration as a community is usable as a tool for learning. The action taken was to run a controlled experiment of a self-organizing developer community of software engineering students within a computer classroom. The students of an elective seminar course acted as the developers.

The experiment gave initial results to indicate that students can learn community driven development by doing and working together over an exercise such as the experiment. There were also indications that there was little need for strict boundaries and outside direction. However, it became apparent that the short timeframe and limitation of the classroom was not sufficient. While the scope of the experiment was specifically open development and communities, the question rose if skills similar to these were applied elsewhere.

2nd Iteration

In the second iteration a local open innovation platform which works through self-organizing developer teams of students was investigated as a case. In the study similarities between the working processes of the innovative teams and open development were found in motivation, collaboration, and legal concerns.

The two first iterations specified communities as a learning tool and the skills applicable outside the scope of the learning community. As the fol-

	Initiating	**1st Iteration**	**2nd Iteration**	**3rd Iteration**	**Closing**
Problem diagnosis	Collaboration and community in learning	Assess community as learning tool	Skills applied in a different setting	How to grow a learning community	
Planned action	Participatory exercise for learners	Self-organization in an in-class community	Identify common practices to open innovation development	Teaching collaborative development	
Action	The community game	Monitoring students work together	Monitoring open innovation student teams	Introducing KommGame	
Exit					Action part closed
Evaluate experiences		Community approach needs wider frame	Collaborative methods in use	Platform and policies support	
Specified learning		Self-organization, learning through community and collaboration	Community skills utilized in various settings	Learning through participation & communities.	Results to Q1, Q2 and Q3.

Table 3.2: Case II: Action Research Performed

lowup into the third iteration, growing a wider learning community with motivational aspects as well as collaboration support was planned.

3rd Iteration

The third iteration set out to find ways to support the growth of a learning community. This was done as an experimental case study where an online environment was used as a learning environment as well as a development community for educational content. The learning environment included motivational aspects as well as tools for supporting collaboration.

The third iteration closed the action part of the second case. It drew together parts of the research and supported the idea of a learning community as a learning platform. The results address all three research questions of this thesis.

3.3.3 Limitations

The research method itself renders the research with a number of limitations. When a researcher works in collaboration with the practitioners there is the risk of researcher bias, lending way to a threat to the internal validity of the results. In order to mitigate this risk and to ensure validity of the results, the findings of the research have been reviewed and discussed frequently with the practitioners. They represent, however, only a small sample of the group – a single representative or small team of practitioners was involved in the research. This can, in part, narrow the scope. However, there were several companies involved in the study of Case I and there were three different types of student groups and learning situations in Case II, which both widen the scope and bring further rigor. In Case II qualitative research methods would have been beneficial to gather more in-depth results on the participant's behavior especially in the first iteration of the case.

The most significant threat to external validity is that open development communities are established in rather case specific environments and thus some things may be unique to a particular working environment or purpose. This risk is mitigated by investigating the different aspects through separate cases and combining the findings under a common framework. Replication of some of the results is, furthermore, impossible with the exact same parameters and conditions as a software can only be released once. The results of Case II are more generalizable in this respect.

For the open learning environment case, Case II, there is the limitation of the iterations all being a single case or experiment. Having several cases or more iterations of the experiment could in part have brought to light things

not commonplace to the practitioners involved here and thus improved the external validity. The third iteration did, however, include aspects investigated in the first iteration.

The work covers two different types of open development communities. The work here relates to the company core scope of business based communities and, on the other hand, the developer focus is taken in the second case with the educational communities. The second iteration of Case II can be seen as a bridge between the industrial and the educational setting. However, more work on bridging the two is still needed.

Chapter 4

Five Ps Framework

The aim of the thesis is to recognize the aspects needed to be addressed when establishing and ensuring the future of an open development community. Characteristics of open development communities range over several dimensions, and have community specific flavors to them. Some of the aspects are, furthermore, overlapping and interleaving. It may not be possible to separate legalities from business goals or to cover the technological infrastructure apart from the governance issues. While existing literature [33,43,81,99] tries to address these, they end up with overly detailed and case specific results. A framework of the essential properties from a broader perspective seems to be missing. The five Ps framework discussed in this chapter addresses the need to have an overarching umbrella under which the community specific traits can be considered. The five Ps is derived based on the two research cases and the existing background knowledge on open communities and peer-production.

The different aspects of an open development communities from the viewpoint of this thesis are depicted in Figure 4.1. An open development community is characterized here by five aspects: 1) *people*, 2) *purpose*, and 3) *product*, the 4) *policies* that help the community to operate and the software 5) *platform* that provides the means for communication, development processes, and community group awareness. All aspects are linked to each other, and thus, are an inseparable part of community. Similarly, each aspect builds on top of the underlying issue, like peas in a pod. Figure 4.1 shows them in this order – from the foundation forming platform towards the more organic and fluctuating issues. The framework is named as the ”five Ps” as an abbreviation of its components. It extends the more general definition from Chapter 2 with the product.

4.1 Product

Open development communities differ from general online communities in that they do not exist simply for conversing and interacting with likeminded individuals. While the other aspects discussed in Section 2.1 are common, open development further aims to create and improve a product or a commodity. The definition of ”product” depends on the community. Software [77], through Free/Libre/Open Source Software, is the most common example of open development. However, open development is not by far limited to software. The product can be cinema content [113], an eCorolla electronic car [24], or anything that has value to and is a shared interest of the people in the community [4].

Even though the product itself can be viewed as the shared preoccupation of the community members, it is, in fact, more of an umbrella for a variety of such concerns. People's interests often focus on a specific issue or aspect of the product and its development [43]. Instead of being a single unit, an open development community has many internal communities, sub-communities if you will, that each have their own view of the product. Each of these focus on an integral part of the development of the project. It can be documentation, user studies, marketing – it depends on the project – but it is that shared interest that draws participants together with the purpose of developing the product as a whole as a goal too. Thus the definition of an open development community should include the product as a separate element.

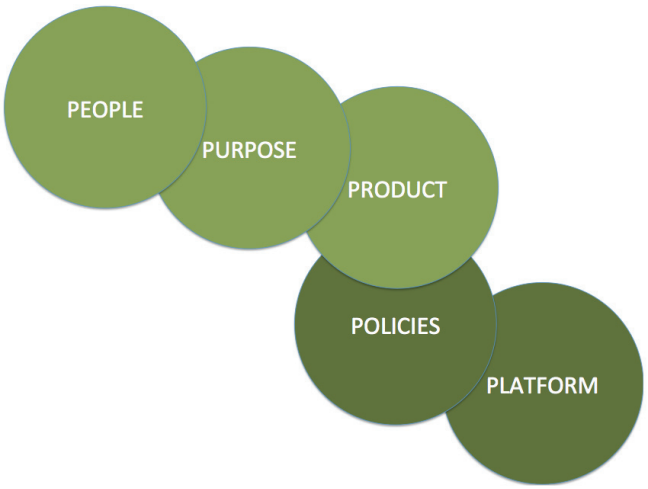


Figure 4.1: Open Development Community

4.2 Interdependent Elements

There is an interdepedency between each of the elements: one is affected by the others and none stands alone. Ergo, as the five Ps do not simply build on top of each other, they need to be viewed through their individual relations. For example, people and policies are entwined – the policies are molded by the people and in turn affect people and their participation. Hence, the framework is shaped through the relationships of its five aspects. This interdependency of the elements is summarized in Table 4.1 and discussed in more detail in the following.

Effect on	**Platform**	**Policies**	**Product**	**People**	**Purpose**
Platform	enabling factor	set requirements	defines	communication medium	suitability requirements
Policies	distributed work	enabling factor	license contribution	mold	set requirements
Product	mediator for artifacts	clear access	community body	different interesting aspects	sub-communities, further development
People	creativity, productivity, communication	enabling participation	draws together	community body	draws to join
Purpose	inline with	roadmap	shared	mold	community body

Table 4.1: Interdependency of Elements

The platform and the policies function as the enabling factors of the community [72]. The platform lays the ground for the entire community. It provides not only a communication medium but is also a mediator of development artifacts. As it acts as an easy access to the product's work items and the platform is a critical supporting and enabling element in distributed work in open development. The policies direct and support the community. Distributed work needs the social and legal conventions the policies provide, not only to give clear access to the work items but also to setup rewarding policies for sharing that in turn enables participation. The people, the shared purpose, and the product form the body of the community. Figure 4.2 illustrates the five Ps through their respective connections.

Purpose is tied to the product. The community has a shared purpose: to develop the product, but while developing the product is the goal of the community, the purpose on its own has a focus on the product and the roadmap ahead to develop it further. Growing communities have a tendency to organize themselves into an aggregate of smaller sub-projects and the community ends up with multiple centers [43, 48] forming an ecosystem. As large communities divide into sub-communities, each has a purpose of their own within the community. Furthermore, the purpose draws people into the

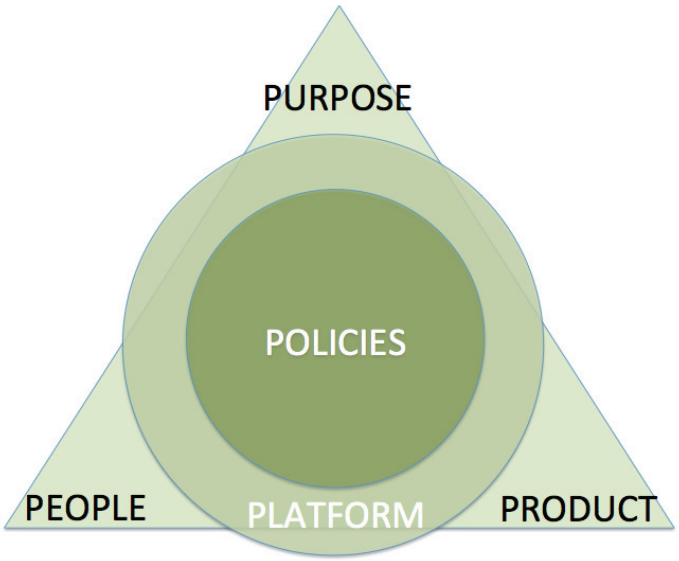


Figure 4.2: Five Ps Model

community. Even if they are enthusiastic about the product, if people don't agree with the purpose of the community, it is unlikely they wish to join and participate. The purpose also affects people's interactions [106] – we are more likely to interact with likeminded individuals.

As the product and purpose are connected, the product is further linked to the people in the community as the product or an aspect of it is an interest shared by the people in the community. The product and improving it are also one of the reasons behind people's participation. Different people focus on different things in the product. As Linus Torvalds, paraphrased by E.S. Raymond [85], states, the person who first finds the problem is not usually the same person who later understands or fixes the problem. The product impacts the policies as well through product specific needs such as licensing issues or methods of contributing.

The biggest single determinant of the platform is the product as the software systems need to be suitable for the development of the product. However, the platform needs to be a good fit for the people as well as it supports creativity and improves productivity [81]. People's communication needs and tastes also affect what kind of a platform is needed and chosen. The platform needs to be suitable from the viewpoint of the purpose as well: a software business targeted community may require a different platform than a community with a pedagogical purpose or one developing art content.

The policies are community specific but in general include issues such

as joining the community or accepted conduct between members. The policies have an influence on the people and who will join the community along the product and the purpose [81]. People's activities in turn affect the policies. The governance of a community may further evolve over time as the community grows and sub-communities emerge.

Chapter 5

Open Development Communities

Open sharing of software code is an idea that originates from the 1960s. The idea was driven by the scarcity of available computers in research labs such as the Artificial Intelligence Laboratory at Massachusetts Institute of Technology [85]. With the Internet, sharing code with others world wide became possible. Today online forges host tens of thousands of open source projects[1], and open source has witnessed an exponential growth in the number of projects [21]. The concept of commons based peer production [4] has further extended the idea of open community driven development from software to other fields as well. Open resources like the online dictionary Wikipedia[2] or the video sharing platform YouTube[3] are examples of the trend of open participation and content peer production.

The following Sections present two research cases on open development communities. The five Ps framework is instantiated in both to establish open development in two different contexts. While Free/Libre/Open Source Software is the best known example, and Case I here, the thesis takes a more general view of open development communities as communities of practice. The five Ps framework is used to show they share properties regarless of the specific domain of the community, while the individual details are community and domain specific. The social aspect of learning further ties the cases together.

[1] `http://www.ohloh.net`
[2] `https://en.wikipedia.org/wiki/Main_Page`
[3] `http://www.youtube.com/`

5.1 Case I: Free/Libre/Open Source Software Development

Free/Libre/Open Source Software (FLOSS) is a piece of software for which the source code, documentation, bug lists, and other artifacts related to the software product are available, and which gives users the right to use, modify, and redistribute the software. In addition Free/Libre/Open Source Software is a development method with a collection of processes and tools that allow a wide variety of users and developers to contribute to the development of the software product.

There are several terms used when talking about FLOSS, open source, and FLOSS development, each taking a slightly different stance, and putting emphasis on different aspects. It is thus worth to note that "open source" as a term can have several meanings depending on the context [37]. Free/Libre Software relates to the end user's freedom to use, study, share, and modify the software, and is more of a social movement or a political stance [35]. The term Open Source Software in turn covers the development methodology and business model approach on top of the availability of source code [77]. The thesis follows this Open Source Initiative's (OSI) definition as it incorporates the software artefact, the development process, and business aspects, in addition to the licensing details. Still, the term Free/Libre/Open Source Software is used here as it covers both aspects, and in addition comes without bias towards either approach or political viewpoint [31, p. 89]. Nonetheless, within the scope of this thesis, the term FLOSS is interchangeable with open source software (OSS).

5.1.1 Free/Libre/Open Source Software

Free/Libre/Open Source Software builds on the general idea of open and available source code and, furthermore, reaches beyond it by being both a software development method and a software business model. It is further defined by the license used to grant users and developers additional rights to the code. FLOSS can be freely used, studied, and modified. Free software is not necessarily cost free as such and thus *free* does not relate to monetary cost but to freedom or liberty (libre instead of gratis). Copying and redistribution is allowed as well but can be restricted by the license as it may require a need to grant the same rights to the modified versions as well.

The first definition of free software was formulated by Richard Stallman in 1986. In it free software is defined as any piece of software that grants anyone with a copy the freedom to run, study, redistribute or improve it.

That is, these are the **four freedoms** [40] of FLOSS, which are indexed from zero as a geek homage to zero-based numbering often used in computer systems.

0. The freedom to *run* the program for any purpose.

1. The freedom to *study* how the program works, and change it to make it do what you wish.

2. The freedom to *redistribute* copies so you can help your neighbor.

3. The freedom to *improve* the program, and release your improvements (and modified versions in general) to the public, so that the whole community benefits.

Open Source Software Communities

At the heart of FLOSS is the developer community – a social ecosystem on its own. The structure of the community is often depicted with a layered onion model [74] where the users of the software form the outmost layer and the most prominent developers and the leader of the project are at the core. The model is a generic representation and the amount of different layers may vary from project to project [II].

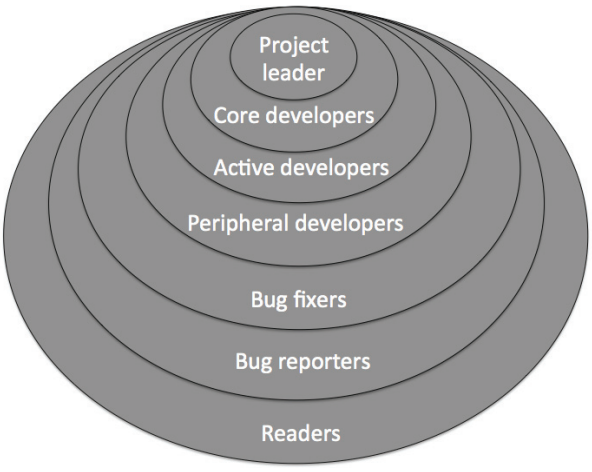


Figure 5.1: Onion Model of Open Source Communities

Taking an ecosystem view on the topic, the key characteristics of a software ecosystem vary depending on the viewpoint. From the engineering perspective, a software ecosystem provides the technology for implementation,

environment for the overall software project infrastructure, and a development methodology that is aligned with the goals of the ecosystem. Additionally for the ecosystem to foster, social, legal, and business aspects must also be considered along the technological. Consequently, the ecosystem can be viewed as a business and governance model with marketing as one of the strategic advantages.

FLOSS presents solutions for each of the aspects needed in a growing, sustainable ecosystem. While commonly addressed with a single term 'open source', the set of principles, practices, culture, and licenses of FLOSS communities differ from each other in various ways and bring diversity into the ecosystem [85]. For example, some communities are geared towards the business environment and companies, and make long-term plans, whereas there are communities that focus on individual contributors' innovative ideas to make the community succeed. Furthermore, the license plays a major role as some open source licenses are permissive[4] [87] and as such introduce only mild obligations and constraints to the user/modifier or the redistribution of the product. In contrast the so called strong copyleft licenses require any derived work to contain the same rights. Nonetheless, the generally common aspects of FLOSS include the transparency of development and the freedom to build more complex systems out of readily available building blocks. These in turn provide the means to grow an ecosystem around such pieces of software and also have an ecosystem that benefits from the members' open contributions.

Stakeholders

The onion community model for FLOSS focuses on the developer community alone. However, FLOSS – especially industrial FLOSS – forms an ecosystem involving many stakeholders that have their own objectives and whose interests may be in conflict with each other. Business and industrial partners and other similar interest groups are key participants in the ecosystem and have influence on the community while they are not a part of the core model. Similarly taking different viewpoints offers divergent looks on the community as one community can contain underlying sub-projects.

The different stakeholders, including the end users of the software product, are the *people* element in FLOSS development communities. The three main groups are the *publisher*, *the industrial partners*, and existing *open source communities* and *other individuals* [I, 53] as shown in Figure 5.2.

Several factors like familiarity with the project, objectives, and availability of skilled resources affect the position of each of three groups of stakehold-

[4] `http://en.wikipedia.org/wiki/Permissive_free_software_licence`

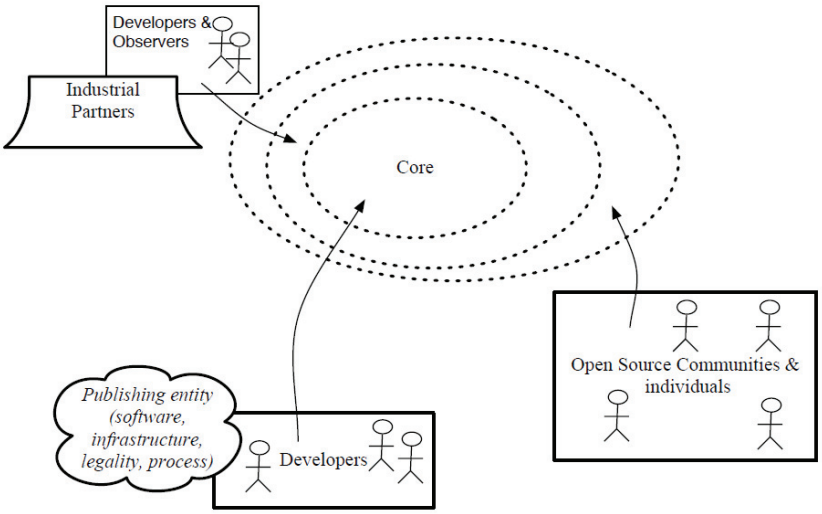


Figure 5.2: FLOSS Stakeholders [III]

ers in the community whether within the onion or outside its scope. Compared to the other two groups, the publishing entity is most familiar with the software product and commonly most willing to invest in the process of fostering the community. Thus, an active role is essential for this stakeholder throughout the development lifecycle. It is the task of the publishing entity to make decisions on the proper project platform and infrastructure and set initial policies such as the open source software license. Additionally, decisions where both the software and the social ecosystem have influence, need to be made on the role and type of the developer community. As a community policy, platform, and purpose element the publishing entity needs to evaluate which of the alternatives, company-based, volunteer, or mixed type [70] of community, would best suit the project and the ecosystem it will live in. In addition, attention needs to be paid to the governance model and the dynamics of the onion model structure (closed or gated). As a policy the selection of such a fundamental governance structure affects the entire community as it has impact on motivation and participation [93]. The publisher also has the means to allocate the community of *skilled developers* to lead the development and to interact with the rest of the community.

When talking about industrial FLOSS, an ecosystem scope reaches beyond the core community. It brings both enthusiastic and conservative partners into the community scope, and has influence on the place of the community in the ecosystem. Enthusiastic partners invest in the success of the

community, and thus usually participate with developers. They contribute to the development of the software and stay close to its evolution, so the community gets developers that are closer to the core team and might have key roles. The conservative partners in turn are more likely to be just interested in the evolution of the software and the outcome of the community. Nonetheless, they too may have their dedicated developers in the community.

The *existing open source communities* and *other individuals* are a vital element in the community growing process. The project coexists with other development projects as part of the software ecosystem they create. The individuals may not all be simply keen volunteers but could as well represent the interests of companies that are not partnering with the publishing entity. This group also represents the pool of potential contributors who could join the project if they get interested and motivated.

The individuals participating in the projects in the ecosystem form a social ecosystem – the developer community. A well-defined developer promotion policy is needed as more people join the project, especially if an open core structure is chosen. It is possible that individuals joining the community do not have any earlier experience with open source projects. In any case, the newcomers typically join the community as passive users and then may take key roles as they show commitment and value, thus penetrating the onion structure inward from outer layers [16]. The community policies support this migration.

Legalities

Copyright and intellectual property rights issues can be a tricky business. With development of industrial software through open communities the most commonly considered element is the legalities. While this is a broad topic and extends towards legal sciences, legal issues are one key element included in the policies of FLOSS communities. The license, furthermore, is an inseparable element of the developed product.

The Free/Libre/Open Source Software license is a permanent policy that affects the community throughout its life and can have influence on the willingness of developers to participate in the community. Yet, it is a policy decision often made before the community is even born. The choice of license affects other developers' access to the code. At least the type of the product, compatibility with licenses of related systems, the intended users and partners, possible need for other open source components, and the desire to provide the possibility of derived projects should be considered. Then, finally, matters of personal taste influence the selection of license.

5.1.2 Helping Open Source Communities Grow

The three-phased process of growing viable FLOSS ecosystems, the OSCOMM framework, was initially published in piecemeal fashion in Publications [II, III] and in [92]. Publication [I] is the first publication where the framework is published as a complete concept.

The phases in the framework are: *evaluating the readiness of the project* for being opened, *open source engineering* the product based on the findings of the readiness evaluation, and *measuring* the ecosystem once the project is open. The approach addresses all of the elements in the five Ps framework. This is discussed further in Subsection 5.1.3.

Evaluating the readiness for open development is done to identify possible bottlenecks both from the product perspective as well as from the business angle. Making decisions on open development, its goals, and improving the product together with defining the possible business model and goals are all key actions to make sure the community can flourish. A growing community needs to be monitored and it is possible that even drastic decisions may also be needed from the publisher and the core of the community. The OSCOMM framework does not make decisions, it only provides information to support decision making.

Product Viewpoint

Figure 5.3 shows the three phases from the product angle. The arrows depict the flow of the work from phase to phase.

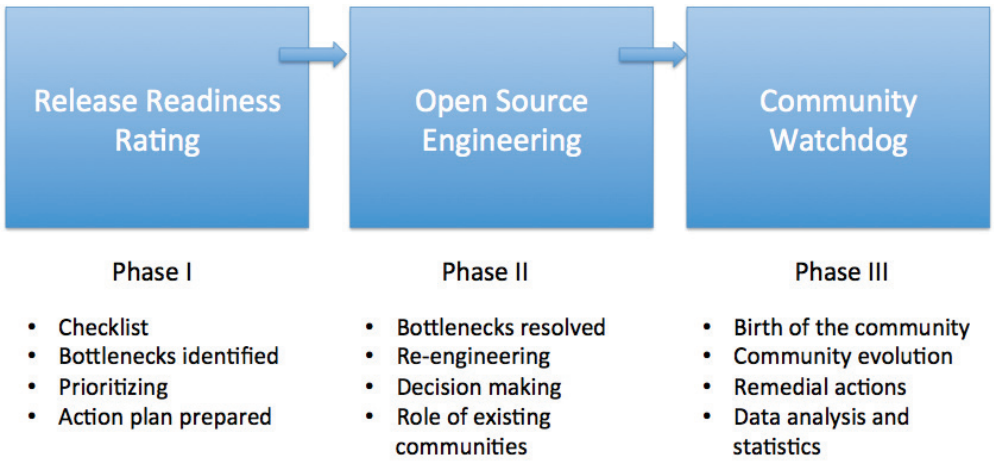


Figure 5.3: OSCOMM Framework

The OSCOMM supports the process of establishing an open development

community for industrial software regarding the five different elements in what is called on the lines of Eric S. Raymond [85] the "pre-bazaar" phase and in the bazaar – in the open development environment.

Evaluating the release readiness: Regardless of whether the product has been on the market as proprietary or not, it needs to be prepared for the open source ecosystem as it will need to be approachable to people not necessarily formerly familiar with it. The idea is to make an evaluation of the intended community from the viewpoint of the five elements. In this R3 framework [III] the product – the software itself – is evaluated based on its quality attributes, the architecture and source code, as any FLOSS project fundamentally deals with source code. Policies such as coding conventions are needed. The released system should have some practical importance and relevance to outside developers – participation is fueled by versatile factors. Thus, the purpose of the community needs to be given careful thought beyond the business goal, more as a mission statement and a roadmap. This phase should also figure out who are the people. The evaluation focuses on the potential user community and the possible partners that can form the ecosystem. Further assessment is done for the legalities and the core of the new community – the releasing authority and the code developers. The evaluation yields information that can be used to prepare for open development in the next phase.

Open source engineering: Phase I renders a set of recommendations based on which the software under evaluation and its development environment then goes through a so-called open source engineering process [92] – the Phase II. The process focuses on eliminating the problems and shortcomings that were identified in the evaluation making sure the achievability of a viable community for open development is rooted from the start. The open source engineering process itself is driven by the same elements as open development in general. The conceptual framework for this phase is not the work of the candidate and is thus left outside the thesis.

The community watchdog: The core of the community needs to support and follow the daily ongoing activities of the open development community. Additionally, having up-to-date information often plays a key role in making business decisions. Instead of relying on simple, single measures such as the number of downloads or the amount of messages on the community mailing list that can lead astray, the community watchdog [II] considers a wider set of data sources. Phase III focuses on gathering and analyzing data from the both facets of the ecosystem – the software and the social model. There are three different types of things to assess: the *community*, the *software* and *how well the objectives of the releasing authority are met*. How the first two plot out can be seen based on the measures set up for them. The third

requires assessment on the measures: Are they moving to the right direction or are remedial actions required? The community wathdog gives valuable insight on the evolution of the ecosystem by continuously measuring the key data points. The data and possible trends in its fluctuation can be used as indicators for the state of the community.

Business Model

While not all FLOSS projects have business goals, within the scope of this thesis, industrial FLOSS projects are investigated. This means FLOSS as a business model needs to be taken into account as well. In [53], Publication [I] is augmented with the business point of view. Moving the product from proprietary to the open is a large business decision changing the company's entire business and revenue model. Moreover, business reasons are usually the main motivation for the transition to industrial FLOSS. It is further worth to note that most pieces of software start as in-house products. While going open source may be a choice made from the get-go, the same phases are still applicable and the same issues must be addressed.

Taking a business viewpoint illuminates the purpose elements that act as the drivers to go for open development. These can be versatile ranging from increased visibility to a better way to engage in discussion with and get feedback from the company's customers. There can be goals of improving software quality or gaining a wider customer base [43].

Figure 5.4 augments the OSCOMM framework to address planting and growing a FLOSS developer community from the business perspective. As with the product, the business aspect is a three-phased process: evaluating the *business readiness*, building a *business strategy* and *growing business* in an open source ecosystem [53].

Business readiness: Moving to an open source business model radically changes how a company makes business. Such actions are rarely taken if there are no business bottlenecks. Changing the business model is a major discontinuity, which naturally introduces risks with it.The motivation for open development may come from external factors, such as a competitor that provides similar facilities already as open source, or from internal issues, for example multi-licensing, where a commercial license offers additional options for customers. Enabling the participation of developers in the maintenance of the system without commercial agreements can make the system more attractive for business and be the driver for open development. At the same time, some businesses are extremely conservative, which may mean that more traditional business models are more suitable. Some conservative partners may not be that familiar with open source and base their decisions on per-

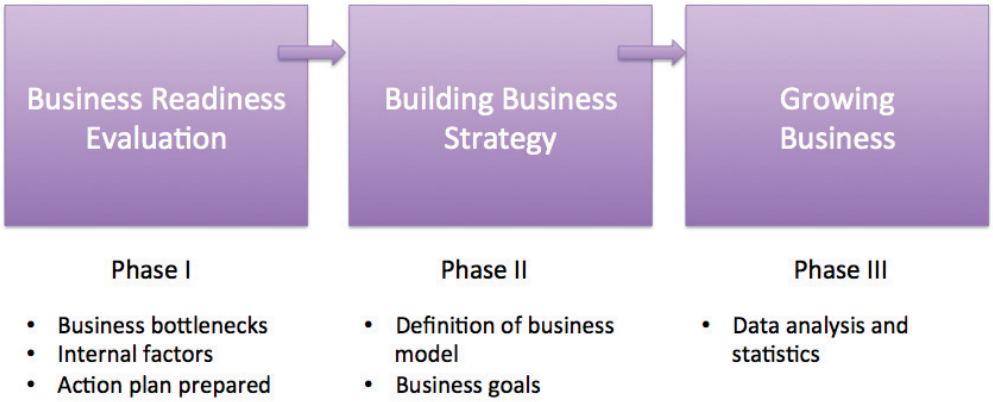


Figure 5.4: OSCOMM from Business Perspective

ceptions, even misconceptions, or they can find it intimidating. Additionally, it may take longer to make business with open source than with proprietary software, as business comes out of dual-licensing, maintenance services, or product customization to name a few. Therefore, there is an need for a action plan to make the transition towards open development.

Building the business strategy: There are numerous subtle details that provide different ways of making business from a growing open source ecosystem. Therefore, an exact definition of the new business model needs to be construed. This is something that needs to be done before considering the open source engineering phase for the product as it may be affected by the selected open source and business strategies here. In addition, the business goals of the company will also determine many of the parameters of the community watchdog – if the business plan is to benefit from the community in various ways, measuring the sufficienct of community actions is a necessity. In general in an open source ecosystem, the users are gained first and only later will they become business customers. Hence, a large mass of users is commonly needed for a sufficient amount to turn into clients.

Growing the business: With the data from Phase III for the product, the community watchdog, it is possible to keep track of the community together with the business trends. Assuming that the parameters that are used to monitor what is going on in the community have been carefully thought out and that the business is truly building on community related aspects, the resulting data – together with the data from business actions – can be used to characterize the growth of the ecosystem.

5.1.3 Discussion

The work supports the five Ps framework as shown in Figure 5.5. The five Ps framework is further summarized by breaking the elements into more detailed subsets in Table 5.1. Establishing and growing FLOSS software is investigated here within an industrial setting. However, many of the issues apply even if there are no business goals imputed onto open develoment.

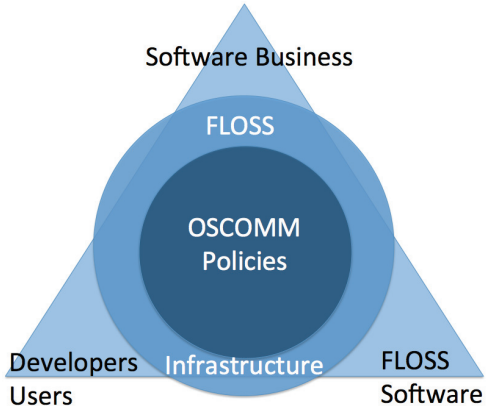


Figure 5.5: Five Ps in Industrial FLOSS

Element	FLOSS
Platform	software ecosystem environment technical platform project infrastructure
Product	Free/Libre/Open Source Software software licence
Purpose	business goals use and improvement mission statement
People	publisher and core developers industrial partners other developers users
Policies	software licence governance structure promotion policies release readiness coding conventions community watchdog

Table 5.1: Summary of Five Ps

Mapping the work onto the FLOSS lifecycle model [10, 91] the different steps fall into the lifecycle neatly. Release and business readiness evaluation occurs at the end stages of the cathedral phase, when the decision to select an open development approach is made. Open source engineering and building the business strategy fall into to the transition phase of the lifecycle. During that time the software and its environment are improved before the actual release and a FLOSS based business model is defined. The final phase, phase III, coincides with the bazaar phase where project evolution is driven by the open development community and growing business.

The OSCOMM approach provides insight on FLOSS development communities crossing all the essential properties formulated in the five Ps. It expresses and documents the elements required for establishing and sustaining a FLOSS based ecosystem. The work presented is a product of combining experiences from several industrial settings and shows what kind of activities and issues ensure and support growth. The goal is to turn the task of establishing FLOSS development based ecosystems into an engineering and business discipline with clear stepping stones and focus.

5.2 Case II: Open Learning Evironments

As open development has gained popularity in different fields, it has also acquired aspects that require sskills and knowledge. Furthermore, the rise of business ecosystems, especially within the software industry, has emphasized the need for open innovation settings as ecosystems typically rely on a shared platform on top of which different parties contribute their own, company-specific innovations [5]. Initiatives such as MIT's OpenCourseWare[5], the Open Source Courseware project [1] and openSE[6] are examples where open approaches to education can be the root of an ecosystem. This merging of open development, innovation and education leads not only to ask how participation in such endeavors could be supported, but also to the question if collaboration could be taught to interested participants, this way charting the path to large scale open development communities.

5.2.1 Collaborative Development in Education

Learning is a disciple where the learners themselves have the most active role. Modern pedagogical learning theories[7] [104] emphasize the learners central role in learning – the learner's activities and motivation, influenced by earlier knowledge, creates learning. In today's world, information is abundant and easy to access. Hase and Kenyon [46] argue that self-determined learning approaches can best meet the learner's needs in the modern world. They state that individuals and organizations are in need of flexibility, creativity, ability to work together in teams, and to apply skills to different situations and that flexible, learner-centric environments provide a way to learn and acquire such skills. Personal learning environments (PLE) are such environments that put the learner in control. In personal learning environments self-organized learning is put into actual practice giving the central role to the learner and thus control over their own learning experience [11].

Learning is furthermore a social activity. Communities of practice (CoP) [59,60] convey tacit knowledge and learning is partly an outcome of participation in a community. Similarly, most of the literature on computer-supported collaborative learning (CSCL) builds on the socio-cultural theory of learning, such as social constructivism [80, 105] and activity theory [27, 105]. Situated learning theory [14] further gives the learning context more weight emphasizing the context-dependency of learning. In [94] the author discusses the shortening lifecycle and the ever growing abundance of knowledge. The

[5] `http://ocw.mit.edu/index.htm`
[6] `http://opense.net/`
[7] `http://en.wikipedia.org/wiki/Constructivism_(learning_theory)`

author proposes a model of learning, connectivism, which emphasizes that learning and knowledge lie in an abundance options and that learning is a process of creating networks and connecting information sources. Knowledge is bound to the activities, context and culture that develop it and where it is used and hence learning should include all three aspects [7].

In [51] the authors talk about learning environments:

> "Learning, we believe, can be best facilitated through the design and implementation of constructivist tools and learning environments that foster personal meaning-making and discourse among communities of learners (socially negotiating meaning) rather than by instructional interventions that control the sequence and content of instruction and that seek to map a particular model of thinking onto the learners."

Following that opinion, this thesis includes an approach where students collaborate together in different settings to achieve a common goal, thus addressing the learner-centric and social aspects of learning. Emphasis is put on working in a community and networking to create, accumulate, and share knowledge collaboratively. Publications [VI] and [IV] investigate the use of the open development community approach in education with the aim to teach both substance as well as the working practices of communities. Furthermore, Publication [IV] considers the importance of the platform and the policies for communities. Publication [V] investigates the application of skills that are useful in open development in an alternate setting and finds them similarly to be an aid in open innovation teams.

Reputation Systems and KommGame Community

Reputation systems are used web wide to measure and quantify individuals' contribution to online communities of all sorts. One important role of reputation systems, on top of encouraging participation, is to build trust among community members online where there is limited face-to-face contact [86]. A recent example of social media reputation use is the Finland-based online sports and activity environment called HeiaHeia[8] where sport enthusiasts log their exercises on a daily basis. They can cheer each other's activities, comment on them and through this gain achievements and reputation status. Similar systems are used throughout the Internet in eCommerce, social media, and blogs.

The reputation of an entity can be determined with versatile methods: feedback, points, favorites, voting, and reviews [28]. These can be combined

[8] `http://www.heiaheia.com/`

into more complex methods. One such is *karma* where participation and quality of contribution are combined and weighted to give a participating entity points or marks. The characteristics and arithmetic aspects of forming the karma are presented in [107] and vary depending on community and goals.

KommGame [41] is an environment mimicking a FLOSS community for a learner community to participate in. It has evolved from a student-centric classroom exercise [VI] into a computer-mediated learning environment [IV]. It is a concrete instance of an open development learning community that addresses the five Ps framework (see Figure 5.6 and Subection 5.2.2).

As one of its community policies, KommGame utilizes the karma model to support participant collaboration and to act as a motivation for further participation. The karma is accumulated both from online activities in the community (participatory contribution) and other participants' expressed acceptance of the contributions (quality) based on equation 5.1.

$$\begin{aligned} Karma = \sum_{k=1}^{n} (& f_k(Contribution) \\ & + f(Favorites) \\ & + g(WeeklyQualityTokens)) \end{aligned} \tag{5.1}$$

The weight function for each is chosen based on the importance and priority of the contribution to the community.

The KommGame environment embodies the required elements of an open development community in addition to the motivation and trust building policy of the reputation system. The platform follows the traditional open development community structure with FLOSS development emphasis [41]. As the environment is an e-learning platform, the developed product is learning content such as essays and info pages in wiki format. In addition there are community activities teaching the day-to-day practices and activities of open development. KommGame is an approach to answer to the real life ecosystem need of skilled professionals.

5.2.2 Discussion

In [6] the authors start with the claim that well-educated workforce is pivotal to an ecosystem. They go further to state that ecosystems must also support continuous learning and creation of new ideas and skills. Then, the authors launch the idea of demand-pull approach to learning where students gain an access to learning communities built around practice. The open learning environment case follows this idea.

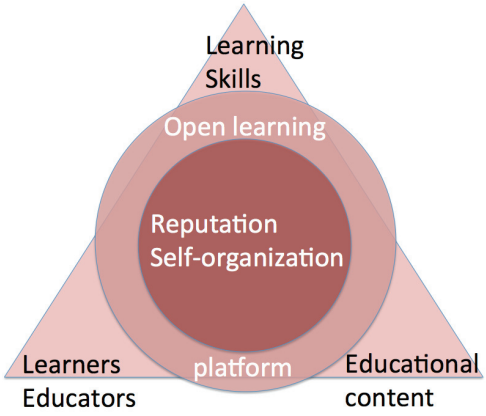


Figure 5.6: Five Ps in Open Learning

Element	Education
Platform	KommGame open platform
Product	educational content skills knowledge
Purpose	learning skills
People	learners educators
Policies	reputation systems self-organization

Table 5.2: Summary of Open Learning Communities

By mimicking real-world situations and problems, the KommGame acts as a situated learning platform. Similarly, as the open innovation environment gets the projects from the industry, they have real-world meaning. As an online distance learning setting the KommGame further fulfills the requirements presented in [51] for a constructive learning environment: context, construction of knowledge, collaboration, and conversation. There are real world features included. Knowledge is created through the participants' experiences within the context. The learners – the participants – collaborate and discuss the problems they have encountered, and further engage in conversation with each other.

Table 5.2 has the five elements of the five Ps framework further summarized through their individual aspects. From the learning perspective the product depends on the intended learning outcome. It can be text content, participation, or contribution of artifacts, software code for example. The knowledge the learners gain is one of the products of such an environment and developed by each learner through their own individual process. As a specific policy, the role of the karma model is to support collaboration and learning by motivating and rewarding. In open approaches where the learners act out of their own interest, self-organization and allowing it is an important factor.

5.3 Drawing Cases Together

Case I and Case II approach open development from two different viewpoints and in two separate settings. At first glance, they have distinct views on open development and its elements. However, FLOSS itself can be viewed as a community of practice. Furthermore, open education communities use development processes similar to FLOSS in order to produce learning content. As open environments flourish they offer a platform for open innovation ecosystems where business and educational objectives can meet and work together for the benefit of both.

Open Innovation Environments

The innovation of the future businesses relies heavily on the local schools and universities teaching the future practitioners. This emphasizes the need for early and wide collaboration between industry, teaching and research staff in educational institutes, and students. Open innovation, however, comes with a number of challenges such as motivation, integration, and exploitation of innovation [111] putting open innovation in a need of a governance framework [30] that enables organizational alignment of the different partners, proper handling of intellectual property rights issues, and the emergence of new kinds of business opportunities. These challenges have to be taken into account when building any open innovation platform with the goal of driving future development and solutions.

Publication [V] discusses the required practices and principles of open innovation in a local open innovation ecosystem New Factory[9] and its innovation platform Demola[10]. One of the aims of the platform is a multidisciplinary and agile innovation environment where innovation can flow freely and which is not restricted to any artificial process or framework that must be obeyed in order to benefit from it. Demola incorporates elements from the approaches applied in the two cases: it is both a learning environment and a facilitator to development of proof of concept-like products that have industrial relevance, and can act as a stepping stone for new business opportunities for the participants. Table 5.3 shows the elements specific to the Demola case. Publication [V] focuses on policies, but the other open development elements of Demola are included for a complete general view.

As one of its policies Demola emphasizes co-creation and relies on self-motivated participants in the development of innovative products and demos. The project ideas come from the local industry and public organizations and

[9] `http://newfactory.fi/`
[10] `http://tampere.demola.fi/`

Element	Education
Platform	Demola platform
Product	innovation project results: demos and proofs of concept
Purpose	open innovation emerging business
People	academia industrial partners
Policies	co-creation IPR

Table 5.3: Summary of Demola specific aspects

thus have practical, factual business importance. Both the industrial and the academic partners participate in the ecosystem and provide guidance to the participants. The interaction of the different participants is shown in Figure 5.7. In addition to producing innovation demos, Demola supports the emergence of new business ideas. Immaterial rights are a part of any such environment and Demola offers an approach that respects the authors without hindering commercializing the results at the same time. As a working platform Demola offers workspaces for the participant teams.

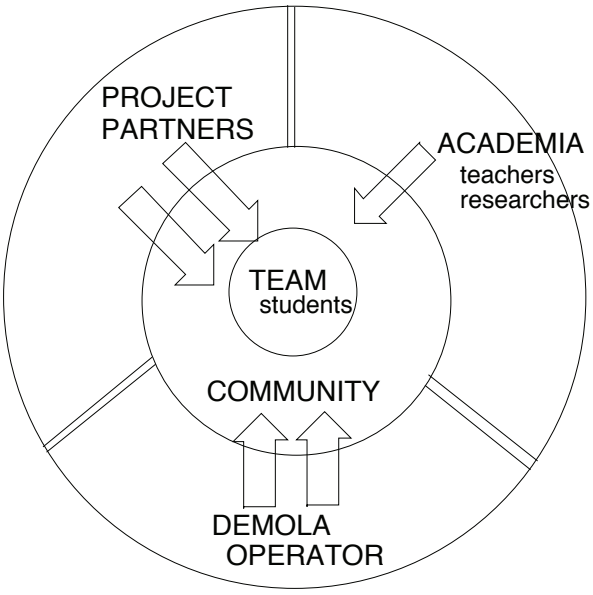


Figure 5.7: Demola Partners

The Demola approach addresses the five key elements for open development further bridging between open education and open software development towards a joint open ecosystem. From a learning perspective the two environments complement each other. The KommGame is a computer-supported collaborative learning environment while in Demola the students work with real life projects in a real life setting. As the project topics come from the local industry the business environment aspects become important. The role of Demola in this work is to show that parallels are visible in such environments with heavy business focus and open development, which supports the notion that open development skills are applicable in different settings.

Chapter 6

Conclusions

The open development model has risen to the forefront in many contemporary fields. At the same time ecosystems have become a way for companies and individuals to cooperate through a shared platform. This leads one to ask what is needed to make open development successful and how to best foster an open development community.

This doctoral thesis answers the questions through looking at the challenges of open development through the properties of open development communities that need to be addressed to make them succeed. How to support growth and how people can be given the required skills to work in open development communities through situated learning are also discussed here. The thesis offers as a concrete contribution a framework of open development communities that addresses their essential aspects drawn from the two research cases. In addition, it presents a set of guidelines for establishing and growing an open development community in an industrial FLOSS context, and an approach to teach learners open development with an open educational development environment.

6.1 Summary

Establishing an open development setting is straightforward, but bringing participants to join and get enthusiastic about the product is not. This thesis presents work on establishing and monitoring an open development community through Free/Libre/Open Source Software business and open education. Parallels between the two are also presented. The research covers going ahead with open development within the FLOSS ecosystem and open development as an learning environment offering insight on how developers can learn open development methods in an environment with real-world context. This thesis

Purpose	**Product**
I. Software business II. Learning and skills	I. FLOSS Software II. Educational content
People	**Policies**
I. Developers and users II. Learners and educators	I. OSCOMM Framework II. Reputation, self-organization
Platform	
I. FLOSS Ecosystem II. KommGame, Demola	

Table 6.1: Contribution of the Thesis

answers the question: what open development essentially requires in order to grow and to be able to draw people to participate.

This thesis comprises six publications. The candidate's contribution to each of the publications is defined separately in the beginning of the thesis on page xiii. Next, a summary of the individual publications is given with the division of the contributions of the research in them.

The publications included dicuss the different aspects of an open development community along with growing and supporting them. Their contributions can be mapped onto the five Ps model given in Section 2.1 as shown in Figures 5.5 and 5.6 and summarized in Table 6.1. The thesis covers all the specific aspects from the point of view of the two cases.

Firstly, Publication [I], reports on the problem of establishing and growing open source development communities for formerly closed industrial software. The publication draws together several earlier results and presents the OSCOMM framework for moving software from proprietary to open source development. The publication gives insight on the entire frame of establishing open development. Its main focus is on the software product and the policies needed to migrate from proprietary into the open.

A method of collecting relevant information about the development community gets introduced in Publication [II]. Information about the community, the amount of different types of community members and how actively they participate is valuable information when evaluating the growth of the com-

munity. Such data is needed in order to make decisions relating to the community purpose, along with defining policies. Both the size of the community and the activity over time can be monitored with the proposed method.

Publication [III] proposes an evaluation framework for the release readiness of a software product as open source. In addition to the product the evaluation emphasizes the community, the policies, and the provided infrastructure as well. The publication discusses two cases where the framework has been utilized and shortly adresses the steps that need to be taken based on the results of the evaluation. A part of the OSCOMM framework, introduced as a whole in [I], includes the work presented here.

Publication [IV] studies a student centric approach to teaching collaborative software development by utilizing an online platform, a learning environment called KommGame that acts as the basis of the learner community. KommGame includes a set of policies, including a reputation model to support the social aspects and the learners' collaboration. The publication shows practical results of applying the environment on an elective course.

Publication [V] identifies the similarities between an open innovation platform intended for students and community driven develoment on the whole. The main parallels are drawn to open source software development. This paper also bridges between the two open development community cases with business aspects. It addresses development policies, the product and the key stakeholders.

In conclusion, the final Publication [VI] discusses how software engineering students can be taught open software development and the special characteristics of the open source software developments community through a participatory exercise that mimics the day-to-day activities of open development of a product in a classroom setting. The paper includes results on using the exercise called the community game on a master's and postgraduate level course. It dicusses open development from the viewpoint of the participants and the platform as well as the policies.

6.2 Research Questions Revisited

This thesis answers three research questions through two cases. The questions are:

Q1: What aspects of open development are essential in supporting both the growth of open development communities and their participants' collaboration?

A1: The five Ps framework addresses the essential elements. It is drawn from and in practice supported by two open development cases. The el-

ements: platform, product, purpose, people, and policies are identified in both. These are summarized in Subsections 5.1.3 and 5.2.2. The OSCOMM approach documents the elements required for establishing and sustaining a FLOSS based ecosystem. The KommGame acts as a situated learning platform. Both studied learning environments include all the essential elements and through that complement each other. The two cases demonstrate the aspects of open development: peer-production, product related details, and the project processes, and show how the community specific aspects can differ while the essential elements identified in five Ps are common.

Q2: What kind of activities are needed to ensure and support community growth?

A2: Community is not built as an afterthought. Preparation of the product and the platform is needed long before moving into the open development model. Additionally policies that support collaboration and enforce trust are needed. The community needs to be kept an eye on through some way suitable for the community. Core participation and data support are needed. Addressing and encouraging motivational aspects are also essential.

Q3: How can collaboration be learned through participation in a open development community?

A3: As learning is situation dependent as well as an individual and social endeavor, learning open development skills can be strengthened through an environment that itself acts as an open development community. An educational tool such as the KommGame can incorporate all the elements of open development while still meeting the pedagogical demands. There is evidence to support that learning within an actual context is carried onto other settings as well.

6.3 Future Work

The work presented here is far from completed. There are many directions yet to investigate and to deepen. One such direction is to conduct a retrospective study on one of the FLOSS communities established during Case I. After several years in business in a FLOSS ecosystem there is a lot of data that can provide interesting details especially considering the community watchdog. From the people perspective, working on who the participants are and what are their motivators would be an interesting future direction.

Qualitative research on the KommGame as an open learning ecosystem would further strengthen the work presented here. It would also widen the scope of the research. Demola offers a versatile environment with both business and policy – especially legality – directions to study further. It would

also be beneficial to make a cross-team study on the day-to-day activities of the student teams.

The novel thesis contribution, the five Ps framework, should also be studied further. As a contribution to the dissertation it has been drawn on the basis of the two research cases. It could further be strengthened with a retrospective case study of an open development community. Additionally, it could be investigated from a more of an industrial viewpoint. The rise of the so-called app economy has created ecosystems that are, on one hand, open but, on the other, heavily gated or even closed. Nonetheless, as communities they lend room for studying them from the point of view of the framework.

"This is my timey-wimey detector. It goes ding when there's stuff." -The 10th Doctor

Bibliography

[1] K. Ala-Mutka and T. Mikkonen. Experiences with Distributed Open Source Courses. *Informatica-Ljubljana*, 27(3):243–254, 2003. Special Issue: Information and Communication Technology at European Universities.

[2] M. Antikainen and H. Väätäjä. "Innovating is fun" – Motivations to Participate in Online Open Innovation Communities. In K. Huizingh, M. Torkkeli, S. Conn, and I. Bitram, editors, *Proceedings of the First ISPIM Innovation Symposium Singapore: Managing Innocation in a Connected World*. International Society for Professional Innovation Management (ISPIM), 2008.

[3] D. Avison, F. Lau, M. Myers, and P. A. Nielsen. Action Research. *Communications of the ACM*, 42(1):94–97, 1999.

[4] Y. Benkler. Coase's Penguin, or, Linux and The Nature of the Firm. *Yale Law Journal*, pages 369–446, 2002.

[5] J. Bosch. From Software Product Lines to Software Ecosystems. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 111–119. Carnegie Mellon University, 2009.

[6] J. S. Brown and R. P. Adler. Open Education, the Long Tail, and Learning 2.0. *Educause review*, 43(1):16–20, 2008.

[7] J. S. Brown, A. Collins, and P. Duguid. Situated Cognition and the Culture of Learning. *Educational Researcher*, 18(1):32–42, 1898. Available at `http://www.exploratorium.edu/ifi/resources/museumeducation/situated.html`.

[8] Business Readiness Rating. `http://www.openbrr.org/`, 2005. Last visited December 2013.

[9] L. J. Burnell, J. W. Priest, and J. R. Durrett. Teaching Distributed Multidisciplinary Software Development. *IEEE Software*, 19(5):86–93, 2002.

[10] A. Capiluppi and M. Michlmayr. From the Cathedral to the Bazaar: An Empirical Study of the Lifecycle of Volunteer Community Projects. In *Proceedings of the 3rd International Conference on Open Source Systems*, OSS '07, pages 31–44. Springer, June 2007.

[11] M. A. Chatti, M. R. Agustiawan, M. Jarke, and M. Specht. Toward a Personal Learning Environment Framework. *International Journal of Virtual and Personal Learning Environments*, 1(4):71–82, 2010.

[12] H. Chesbrough. *Open Innovation: Researching a New Paradigm*, chapter Open Innovation: A New Paradigm for Understanding Industrial Innovation. Oxford University Press, Oxford, UK, 2006.

[13] K. Chopra and W. A. Wallace. Trust in Electronic Environments. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, HICSS '03. IEEE Computer Society, 2003.

[14] W. J. Clancey. Representations of Knowing: In defense of cognitive apprenticeships. *Journal of Artificial Intelligence in Education*, 3(2):139–168, 1992.

[15] J. Cothrel and R. L. Williams. On-line communities: helping them form and grow. *Journal of knowledge management*, 3(1):54–60, 1999.

[16] K. Crowston and J. Howison. The Social Structure of Free and Open Source Software Development. *First Monday*, 10(2), 2005.

[17] K. Crowston and J. Howison. Assessing the health of open source communities. *Computer*, 39(5):89–91, 2006.

[18] C. C. P. Cruz, M. T. A. Gouvêa, C. L. R. Motta, and F. M. Santoro. Towards Reputation Systems Applied to Communities of Practice. In *Proceedings of 11th International Conference on Computer Supported Cooperative Work in Design*, CSCWD '07, pages 74–79, 2007.

[19] R. M. Davison, M. G. Martinsons, and N. Kock. Principles of Canonical Action Research. *Information Systems Journal*, 14:65–86, 2004.

[20] E. den Hartigh, M. Tol, and W. Visscher. The Health Measurement of a Business Ecosystem. In *Proceedings of the European Network on Chaos and Complexity Research and Management Practice Meeting*, 2006.

[21] A. Deshpande and D. Riehle. The Total Growth of Open Source. In *Proceedings of the Fourth Conference on Open Source Systems*, OSS '08, pages 197–209. Springer Verlag, 2008.

[22] J. Dinkelacker, P. K. Garg, R. Miller, and D. Nelson. Progressive Open Source. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, pages 177–184. ACM, 2002.

[23] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian. *Guide to Advanced Empirical Software Engineering*, chapter Selecting Empirical Methods for Software Engineering Research, pages 285–311. Springer, London, UK, 2008.

[24] The eCars - Now! community. `http://www.sahkoautot.fi/eng`. Last visited December 2013.

[25] Free knowledge institute. `http://freeknowledge.eu/`. Last visited December 2013.

[26] Online community for open source software educaton. `http://www.teachingopensource.org`. Last visited December 2013.

[27] Y. Engeström. *Learning by Expanding: An Activity – Theoretical Approach to Developmental Research*. Helsinki: Orienta-Konsultit, 1987. Retrieved from `http://lchc.ucsd.edu/MCA/Paper/Engestrom/expanding/toc.htm`.

[28] F. R. Farmer and B. Glass. *Building Web Based Reputation Systems*. O'Reilly Media / Yahoo Press, Sebastopol, CA, USA, 2010.

[29] J. Favela and F. Pena-Mora. An Experience in Collaborative Software Engineering Education. *IEEE Software*, 18(2):47–53, 2001.

[30] J. Feller, P. Finnegan, J. Hayes, and P. O'Reilly. Institutionalising Information Asymmetry: Governance Structures for Open Innovation. *Information Technology & People*, 22(4):297–316, 2009.

[31] J. Feller, B. Fitzgerald, S. A. Hissam, and K. R. Lakhani, editors. *Perspectives On Free And Open Source Software*. MIT Press, Cambridge, MA, USA, 2005.

[32] R. T. Fielding. Shared Leadership in the Apache Project. *Communications of the ACM*, 42(4):42–43, 1999.

[33] K. Fogel. *Producing Open Source Software*. O'Reilly Media, Sebastopol, CA, USA, 2005. Available at: `http://producingoss.com/`.

[34] N. Franke and S. Shah. How Communities Support Innovative Activities: An Exploration of Assistance and Sharing Among End-Users. *Research Policy*, 32(1):157–178, 2003.

[35] Free Software Movement. `http://www.gnu.org/philosophy/free-software-intro.html`. Last visited December 2013.

[36] S. Freeman. *Constructing a Community : Myths and Realities of the Open Development Model*. PhD thesis, University of Helsinki, Helsinki, Finland, 2011. Available at `https://helda.helsinki.fi/handle/10138/28432`.

[37] C. Gacek and B. Arief. The many meanings of open source. *Software, IEEE*, 21(1):34–40, 2004.

[38] R. Gardler and G. Hanganu. Governance models. `http://www.oss-watch.ac.uk/resources/governanceModels`, 2010. Last visited December 2013.

[39] D. M. German. Experiences Teaching a Graduate Course in Open Source Software Engineering. In *Proceedings of the First International Conference on Open Source Systems*, OSS '05, pages 326–328, 2005. Available at `oss2005.case.unibz.it/Papers/OEs/Es1.pdf`.

[40] GNU.org. `http://www.gnu.org/philosophy/free-sw.html`. Last visited December 2013.

[41] V. Goduguluri, T. Kilamo, and I. Hammouda. KommGame: A Reputation Environment for Teaching Open Source Software. In *Proceedings of the 7th International IFIP WG 2.13 Conference on Open Source Systems*, OSS '11, pages 312–315. Springer, 2011.

[42] B. Golden. *Succeeding with Open Source*. Addison-Wesley, Boston, MA, USA, 2004.

[43] R. Goldman and R. P. Gabriel. *Innovation Happens Elsewhere*. Morgan Kaufmann Publishers, San Fransisco, CA, USA, 2005.

[44] R. A. Gosh. *Perspectives On Free And Open Source Software*, chapter Understanding Free Software Developers: Findings from the FLOSS Study. MIT Press, Cambridge, MA, USA, 2005.

[45] I. Hadar, S. Sherman, and O. Hazzan. Learning Human Aspects of Collaborative Software Development. *Journal of Information Systems Education*, 2008.

[46] S. Hase and C. Kenyon. From Andragogy to Heutagogy. *uliBASE Journal*, 2001.

[47] J. D. Herbsleb. Global Software Engineering: The Future of Socio-technical Coordination. In *2007 Future of Software Engineering*, FOSE '07, pages 188–198, Washington, DC, USA, 2007. IEEE Computer Society.

[48] J. Howison, K. Inoue, and K. Crowston. Social Dynamics of Free and Open Source Team Communications. In E. Damiani, B. Fitzgerald, W. Scacchi, M. Scotto, and G. Succi, editors, *Open Source Systems*, volume 203 of *IFIP Advances in Information and Communication Technology*, pages 319–330, 2006.

[49] M. Iansiti and R. Levien. Keynotes and Dominators: Framing the Operational Dynamics of Business Ecosystems. *Harvard Business School Working Paper*, (03-061), 2002.

[50] A. Iriberri and G. Leroy. A life-cycle perspective on online community success. *ACM Computing Surveys (CSUR)*, 41(2):11:1–11:29, 2009.

[51] D. Jonassen, M. Davidson, M. Collins, J. Campbell, and B. B. Haag. Constructivism and Computer-Mediated Communication in Distance Education. *American Journal of Distance Education*, 9(2):7–26, 1995.

[52] K. Kiili. Digital game-based learning: Towards an experiential gaming model. *The Internet and Higher Education*, 8(1):13–24, 2005.

[53] T. Kilamo, I. Hammouda, T. Mikkonen, and T. Aaltonen. *Open Source Ecosystems: a Tale of Two Cases*, chapter 13, pages 276–306. Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry. Edward Elgar Publishing, Cheltenham, UK and Northampton, MA, USA, 2013.

[54] A. J. Kim. *Community Building on the Web: Secret Strategies for Successful Online Communities*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 2000.

[55] B. A. Kitchenham, S. L. Pfleeger, D. C. Hoaglin, K. E. Emam, and J. Rosenberg. Preliminary Guidelines for Empirical Research in

Software Engineering. *IEEE Transactions on Software Engineering*, 28(8):721–734, 2002.

[56] R. Kline and T. Pinch. Users as Agents of Technological Change: The Social Construction of the Automobile in the Rural United States. *Technology and Culture*, 37(4):763–795, 1996.

[57] J. Kotlarsky and I. Oshri. Social Ties, Knowledge Sharing and Successful Collaboration in Globally Distributed System Development Projects. *European Journal of Information Systems*, 14(1):37–48, 2005.

[58] K. R. Lakhani and R. G. Wolf. *Perspectives on Free and Open Source Software*, chapter Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects, pages 3–21. The MIT Press, Cambridge, MA, USA, 2005.

[59] J. Lave. Situating Learning in Communities of Practice. *Perspectives on Socially Shared Cognition*, pages 63–68, 1991.

[60] J. Lave and E. Wenger. *Situated Learning: Legitimate Peripheral Participation*. Cambridge University Press, Cambridge, UK, 1991.

[61] F. S. Lee, D. Vogel, and M. Limayem. Virtual community informatics: A review and research agenda. *Journal of Information Technology Theory and Application (JITTA)*, 5(1):47–61, 2003.

[62] J. Lerner and J. Tirole. The scope of open source licensing. *Journal of Law, Economics, and Organization*, 21(1):20–56, 2005.

[63] Y. Levy and T. J. Ellis. A systems approach to conduct an effective literature review in support of information systems research. *Informing Science: International Journal of an Emerging Transdiscipline*, 9:181–212, 2006.

[64] B. Lundell, A. Persson, and B. Lings. Learning Through Practical Involvement in the OSS Ecosystem: Experiences from a Masters Assignment. In J. Feller, B. Fitzgerald, W. Sacchi, and A. Sillitti, editors, *Open Source Development, Adoption and Innovation*, volume 234 of *IFIP International Federation for Information Processing*, pages 289–294. Springer, 2007.

[65] 7 Things You Should Know About MOOCs II. Available at `http://www.educause.edu/library/resources/7-things-you-should-know-about-moocs-ii`, 2013. Last visited December 2013.

[66] L. Mathiassen. Collaborative Practise Research. *Information Technology & People*, 15(4):321–345, 2002.

[67] J. McKay and P. Marshall. The Dual Imperatives of Action Research. *Information Technology & People*, 14(1):46–59, 2001.

[68] A. Meiszner, K. Moustaka, and I. Stamelos. A Hybrid Approach to Computer Science Education — A Case Study: Software Engineering at Aristotle University. In *Proceedings of the First International Conference on Computer Supported Education*, volume 1 of *CSEDU '09*, pages 39–46. INSTICC Press, 2009.

[69] D. G. Messerschmitt and C. Szyperski. *Software Ecosystem: Understanding an Indispensable Technology and Industry*. MIT Press, Cambridge, MA, USA, 2003.

[70] T. Mikkonen and T. Vadén. The Anatomy of Sustainable Open Source Community Building: the Cultural Point of View. In T. A. Imed Hammouda and A. Capiluppi, editors, *Proceedings of the First International Workshop on Building Sustainable Open Source Communities*, volume 3 of *Department of Software Systems Report*, pages 14–20. Tampere University of Technology, 2009.

[71] J. Moilanen. Emerging Hackerspaces – Peer-Production Generation. In *Open Source Systems: Long-Term Sustainability*, volume 378 of *IFIP Advances in Information and Communication Technology*, pages 94–111. Springer, 2012.

[72] J. J. Moon and L. Sproull. Essence of Distributed Work: The Case of the Linux Kernel. *First Monday*, 5(11), 2000.

[73] J. F. Moore. Predators and Prey: a new ecology of competition. *Harvard Business Review*, 71:75–86, 1993.

[74] K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y. Ye. Evolution Pattern of Open-Source Software Systems and Communities. In *Proceedings of the International Workshop on Principles of Software Evolution*, IWPSE '02, pages 76–85. ACM Press, 2002.

[75] Open Definition for Data and Content. `http://opendefinition.org/okd/`. Last visited December 2013.

[76] Open Government Partnership. `http://www.opengovpartnership.org/`. Last visited December 2013.

[77] Definition of Open Source. `http://opensource.org/osd`. Last visited December 2013.

[78] T. O'Reilly. What is web 2.0: Design patterns and business models for the next generation of software. *Communications & Strategies*, 1(65):17 – 37, 2005.

[79] Pair Programming. `http://www.extremeprogramming.org/rules/pair.html`. Last visited December 2013.

[80] J. Piaget. *The Child's Conception of the World*. Rowman and Allenheld, New York, 1960.

[81] J. Preece. *Online Communities: Designing Usability, Supporting Sociability*. John Wiley & Sons, West Sussex, UK, 2000.

[82] J. Preece. Sociability and usability in online communities: determining and measuring success. *Behaviour & Information Technology*, 20(5):347–356, 2001.

[83] Qualification and Selection of Open Source Software. `http://www.qsos.org/`, 2006. Last visited December 2013.

[84] R. N. Rapoport. Three Dilemmas of Action Research. *Human Relations*, 23(6):499–513, 1970.

[85] E. S. Raymond. *The Cathedral and the Bazaar*. O'Reilly Media, Sebastopol, CA, USA, 1999.

[86] P. Resnick, K. Kuwabara, R. Zeckhauser, and E. Friedman. Reputation Systems. *Communications of the ACM*, 43(12):45–48, 2000.

[87] L. Rosen. *Open Source Licensing Software Freedom and Intellectual Property Law*. Prentice Hall, New Jersey, USA, 2004.

[88] W. Rubens, B. Emans, T. Leinonen, A. G. Skarmeta, and R.-J. Simons. Design of web-based collaborative learning environments. translating the pedagogical learning principles to human computer interface. *Computers & Education*, 45(3):276–294, 2005.

[89] P. Runeson and M. Höst. Guidelines for Conducting and Reporting Case Study Research in Software Engineering. *Empirical Software Engineering*, 14(2):131–164, 2009.

[90] R. M. Ryan and E. L. Deci. Intrinsic and Extrinsic Motivations: Classic Definitions and New Directions. *Contemporary Educational Psychology*, 25(1):54 – 67, 2000.

[91] A. Senyard and M. Michlmayr. How to Have a Successful Free Software Project. In *Proceedings of the 11th Asia-Pacific Software Engineering Conference*, APSEC '04, pages 84–91. IEEE Computer Society, 2004.

[92] F. R. Shah, I. Hammouda, and T. Aaltonen. Open Source Engineering of Proprietary Software: the Role of Community Practices. In *Proceedings of the OSCOMM 2009 workshop, Skövde Sweden*, 2009. Available at `http://tutopen.cs.tut.fi/oscomm09/papers/cr5.pdf`.

[93] S. K. Shah. Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development. *Management Science*, 52(7):1000–1014, 2006.

[94] G. Siemens. Connectivism: A learning theory for the digital age. *International Journal of Instructional Technology and Distance Learning*, 2(1), 2005.

[95] T. Simcoe. *Open Innovation: Researching a New Paradigm*, chapter Open Standards and Intellectual Property Rights. Oxford University Press, Oxford, UK, 2006.

[96] D. I. Sjoeberg, J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanovic, N.-K. Liborg, and A. C. Rekdal. A Survey of Controlled Experiments in Software Engineering. *IEEE Transactions on Software Engineering*, 31(9):733–753, 2005.

[97] I. G. Stamelos. Teaching Software Engineering with Free/Libre Open Source Projects. *International Journal of Open Source Software & Process*, 1(1):72–90, 2009.

[98] A. Sterbini and M. Temperini. Social Exchange and Collaboration in a Reputation-Based Educational System. In *Proceedings of 9th International Conference of Information Technology Based Higher Education and Training*, ITHET '10, pages 201–207, 2010.

[99] M. Stürmer. Open Source Community Building. Licentiate thesis, 2005. University of Bern, Bern, Switzerland.

[100] G. I. Susman and R. D. Evered. An Assessment of the Scientific Merits of Action Research. *Administrative Science Quarterly*, 23(4):582–603, 1978.

[101] M. Temperini and A. Sterbini. Learning from Peers: Motivating students through reputation systems. In *Proceedings of the International Symposium on Applications and the Internet*, pages 305–308, 2008.

[102] J. Tuya and J. Garcia-Fanjul. Teaching Requirements Analysis by Means of Student Collaboration. In *Proceedings of the 29th Annual Frontiers in Education Conference*, FIE '99. IEEE Computer Society, 1999.

[103] I. van den Berk, S. Jansen, and L. Luinenburg. Software Ecosystems: A Software Ecosystem Strategy Assesment Model. In *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, ECSA '10, pages 127–134. ACM, 2010.

[104] E. von Glasersfeld. *International Encyclopedia of Education*, chapter Constructivism in Education. Pergamon Press, Oxford, UK, 1989.

[105] L. S. Vygotsky. *Mind in Society: The Development of Higher Psychological Processes*. Harvard University Press, Cambridge, MA, USA, 1978.

[106] P. Wallace. *The Psychology of the Internet*. Cambridge University Press, Cambridge, UK, 2001.

[107] S. Wang. Study on E-Learning System Reputation Service, Wireless Communications, Networking and Mobile Computing. In *Proceedings of the 4th International Conference on Wireless Communications, Networking and Mobile Computing*, WiCOM '08, pages 1–4. IEEE Computer Society, 2008.

[108] M. M. Wasko and S. Faraj. "It is what one does": why people participate and help others in electronic communities of practice. *The Journal of Strategic Information Systems*, 9(2-3):155–173, 2000.

[109] M. M. Wasko and S. Faraj. Why Should I Share? Examining Social Capital and Knowledge Contribution in Electronic Networks of Practice. *MIS Quarterly Special Issue on Information Technologies and Knowledge Management*, 29(1):35–57, 2005.

[110] J. Webster and R. T. Watson. Analyzing the past to prepare for the future: Writing a literature review. *MIS Quarterly*, 26(2):xiii–xxiii, 2002.

[111] J. West and S. Gallagher. Challenges of Open Innovation: The Paradox of Firm Investment in Open-Source Software. *R&D Management*, 36(3):319–331, 2006.

[112] J. West and S. O'Mahony. Contrasting Community Building in Sponsored and Community Founded Open Source Projects. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, HICSS '05, pages 196–196. IEEE Computer Society, 2005.

[113] A collaborative film production platform: Wreck-a-Movie. `http://www.wreckamovie.com/`. Last visited December 2013.

[114] R. K. Yin. *Case Study Research. Design and Methods*. Sage, London, UK, 3rd edition, 2003.

I

Case I: Publication I

Terhi Kilamo, Imed Hammouda, Tommi Mikkonen, and Timo Aaltonen. From Proprietary to Open Source – Growing an Open Source Ecosystem. In *The Journal of Systems and Software* (JSS). Volume 85, Issue 7, pages 1467-1478. July, 2012, Elsevier Science Inc.

From Proprietary to Open Source – Growing an Open Source Ecosystem

Terhi Kilamo[a], Imed Hammouda[a], Tommi Mikkonen[a], Timo Aaltonen[b]

[a] *firstname.lastname@tut.fi*
[b] *timo.ta.aaltonen@nokia.com*

Abstract

In today's business and software arena, Free/Libre/Open Source Software has emerged as a promising platform for software ecosystems. Following this trend, more and more companies are releasing their proprietary software as open source, forming a software ecosystem of related development projects complemented with a social ecosystem of community members. Since the trend is relatively recent, there are few guidelines on how to create and maintain a sustainable open source ecosystem for a proprietary software. This paper studies the problem of building open source communities for industrial software that was originally developed as closed source. Supporting processes, guidelines and best practices are discussed and illustrated through an industrial case study. The research is paving the road for new directions in growing a thriving open source ecosystem.

Keywords:
open source, software ecosystem, opening proprietary software, open source engineering

1. Introduction

The term ecosystem has emerged as a commonly used notion in software economy [1]. In a nutshell, a software ecosystem comprises a set of businesses that function as a single unit, instead of each participating enterprise acting individually. The ecosystem often relies on a shared platform on top of which different parties contribute their own, company-specific innovations [2]. Then, the cost of developing the platform is shared by a number of companies, each of which is free to extend it with their own modules. Similarly, all the participants gain the benefits of joint investment in the platform.

A platform used for establishing an ecosystem comprises numerous facets [3]. From the engineering perspective, a software ecosystem provides the technology for implementation, environment for the overall software project infrastucture and a development methodology. Additionally, for the ecosystem to foster, social, legal and business aspects must also be considered in addition to the technological. The ecosystem can be viewed as a business and governance model with marketing as one of the strategical advantages.

One source of such platform for ecosystems is to use Free/Libre/Open Source Software (FLOSS). Open source provides solutions for each of the above aspects needed in a fostering ecosystem. Despite being addressed with a single term 'open source', there are actually numerous flavors of open source, defined by principles, practices, culture, and licenses that differ from each other in various ways [4]. For example, some communities are geared towards companies and have long-term plans, whereas some others are geared towards individual contributors, whose innovative ideas make the community foster. Similarly, some licenses are liberal and introduce only slight obligations to the user/modifier of the system, but in contrast some other licenses introduce details such as strong copyleft[1]. All these details contribute to the characteristics of an ecosystem that can be established on top of the community.

The generally common aspects of open source software include transparency of development and the freedom to build more complex systems out of readily available building blocks. These provide a unique manner to rapidly construct ecosystems without large initial investment [5]– indeed, to a great extent ecosystems simply seem to emerge on top of successful open source projects. However, we do acknowledge that a lot of effort have been invested in the creation of associated ecosystems when considering the most famous open source communities.

Although it is difficult to measure to what extent many existing open source ecosystems have been planned ahead, it is a feasible option to build an ecosystem by releasing a piece of proprietary software as open source. Companies such as Sun Microsystems and Nokia have open sourced systems such as Java and Symbian operating system, both of which have enormous

[1]Copyleft is a wordplay derived from copyright. Copyleft licensing gives the right to modify and redistribute the software but requires that any modified versions also contain the same rights.

potential to attract other companies and individual developers who now can join the development effort. We take this as evidence that leveraging an existing proprietary system as the basis of an open source ecosystem promises huge business opportunities that just wait to be harnessed.

Despite the opportunities, the generic ability of open source to act as the basis for new business and systems has however gained relatively little research interest. Moreover, since the trend to release proprietary software as open source is relatively recent, there are few guidelines on how to create and maintain a sustainable open source ecosystem for a proprietary software [6]. Consequently, there is little evidence on how the different facets of an open source platform should be taken into account when establishing a tried and true ecosystem that is viable.

In this paper we provide an insight to our long-term research on establishing and maintaining open source communities, which have been created by companies in effort to use them as cornerstones for their future ecosystems. Even though open source provides a viable platform for growing a software ecosystem from several angles – implementation technology, development methodology, business model, organization structure, governance and legality – we feel that it has not been widely studied from the point of view of a company entering the ecosystem. Some early results of this work has been partially published in a piecemeal fashion ([7, 8, 9, 10]), but no single nor complete representation of the work has been previously published. The framework as a whole is presented here for the first time. Furthermore, the case study applying the framework to a single industry case has not been published before.

The rest of this paper is structured as follows. Section 2 discusses the background information behind forging an open source ecosystem. Section 3 introduces our framework for managing the process by considering various differecnt aspects. Section 4 presents some case studies we have conducted, and Section 5 provides an extended discussion on lessons learned and related approaches. In Section 6 the future research focuses are given. Last, Section 7 draws some final conclusions.

2. Background

The process of forging and nurturing a thriving community is complex with several aspects to be considered. The open source community is often modeled through a layered, hiearchical structure. The factors that affect the

process of growing and maintaining such a community are discussed in the following. The mehodology used in researching this is also explained.

2.1. Onion Model of Communities

The open source community is a social ecosystem on its own and in junction with other open source communities. However, it differs from other social networks in its hierarchical structure that is brought into the community by the software engineering context. Where a social ecosystem more commonly is flat an open source community is layered according to the level on engagement to development and the role in the project in general.

An often-referred model of an open source community structure is presented in Nakakoji et al. [11]. The model describes the community as an onion (see Figure 1), where the most influential roles are in the center, and each outer layer consists of less and less influential ones.

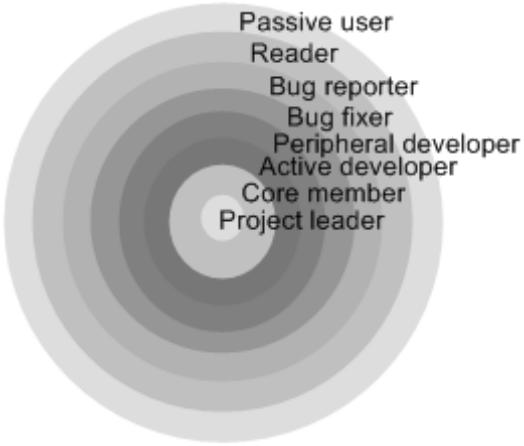


Figure 1: Onion Model of Open Source Communities

In the heart of the onion is the project leader, who is typically the person who has started the project, and is or at least has been the most active developer. A typical open source project has a small group of core members who have participated long time in the project. They coordinate the project and contribute much code. In the work of Gloor [12] it is noted that a typical core group starts out with three to seven members, and grows up to 10 to 15 members when the community is established. The passive users on the outermost level form usually the largest group and just use the software.

Even though on the outermost rims on the onion model, passive users and readers are essential for a sustainable community.

The open source community model does not depict the entire ecosystem. There are naturally business partners, industrial partners and similar interest groups participating outside the range of the model that are an integral part of an open source ecosystem. The community may also look different given a different viewpoint. An ecosystem seen through one community onion can contain underlying communities concentrated on a given subproject.

2.2. Dimensions of Opening Proprietary Software

Creating and maintaining a sustainable open source community [13] for proprietary software can be considered as a multifaceted challenge like with any other online community. It is a complex process affected and driven by various kinds of factors, which in turn can be grouped along six dimensions shown in Figure 2.

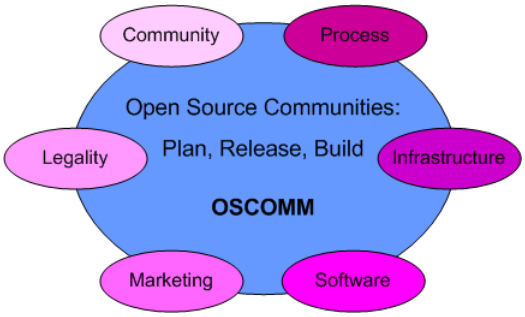


Figure 2: Community Building Dimensions

Software: When opening proprietary software, the question is: how approachable the software is as a whole. Improved software quality may increase the success rate of community building. There are several ways of enhancing the quality ranging from code refactoring to documentation. Downloading, installing, deploying and using the software should be made as easy as possible. The source code can for example be accompanied with user manuals or architecture descriptions.

Infrastructure: The community and the project repository are the two key elements in any open source project. Tools and technologies that enable communication between the community members, coordination of the project and easy access and management of the project repository are vital. Such tools are, for example, developer discussion forums and mailing lists.

Process: Open source development can be regarded as an open maintenance process. A process that balances the community practices and the needs of the company when making decision on the evolution of the software, maintenance actions, and release management is necessary. Governance of both the project and it in relation to the ecosystem is a part of this process.

Legality: The releasing company needs to choose a license type and a licensing scheme for the product. The choice may be, for example, to choose between GPL and LGPL or a single licensing stragedy over multi licensing. In addition, source code should be legally cleared against IPR and copyright issues. Also, the availability of trademarks and names used in the software should be checked.

Marketing: Building an open source community is also a marketing challenge. Effective marketing strategies are needed in order to reach potential users and developers. Selecting an existing open source community as a target customer can be a substantial success factor.

Community: The releasing company should be prepared to support the project community. Company developers who are participating in the community should be trained for their new roles and should be given clear responsibilities. Community members should be provided with clear guidelines for example for how and when contributions are made. When the software is opened, it is vital that all development related information that the company wants to share is made public. In addition, there should be trust among community members. Trust is built over time but having zero tolerance of rudeness and derogatory language both in the software artifacts and the communication among the members helps in establishing it. The relationship the community has or should have with other communities should also be considered.

2.3. Research Approach

The goal of this research is to identify best practices and tools for opening proprietary software in an industrial setting. Our study was therefore driven by a number of industrial case studies. Accordingly, our research approach has been two-fold: a constructive phase where we have built methods and tools and a participatory phase where we worked together with four companies to apply the OSCOMM framework in the business setting. The two phases are closely intertwined with each other.

To this end, our study has been guided by the following research questions:

- What kind of evaluation criteria could be used to assess software readiness for open source development?
- How the evaluation should be planned and which stakeholders are involved?
- How to obtain data for the evaluation process?
- How to exploit the results of the evaluation process?
- How to measure the current state of a newly born community?

To address the first question, we have organized a number of workshops with the industrial partners. We have combined input from the partners with existing related literature. The second point has been addressed in an action research setting [14], where the evaluation has been carried out by company representatives as internal stakeholders and the authors of this work as external stakeholders. The two perspectives have been later cross-examined. It was deemed necessary to develop an evaluation framework in cooperation with the companies. In order to carry out the evaluation process, data has been manually collected from the software products, development teams, and publishing companies. Our action research role has as well been to use the evaluation results to build an action plan for the release process. The action plan has been mostly implemented by the companies itself. Finally, the last question required the development of software tools to monitor the evolution of the community. For this, data is obtained from companies in the form of activity logs, access statistics, and community contributions.

3. The OSCOMM Framework

The OSCOMM framework for growing an open source ecosystem focuses on opening previously proprietary software. It consists of three phases that extend over the opening process to keeping a nurturing eye on the newborn community. The framework together with the stakeholders involved in the opening process is discussed in the following.

3.1. Overview

In Figure 3 a three phased process of building sustainable open source ecosystems called the OSCOMM framework is shown. The phases in the

framework are: *evaluating the readiness of the project* for being opened, *open source engineering* the product based on the findings of the readiness evaluation and *measuring* the ecosystem once the project is open. The arrows depict the flow of the work from phase to phase. Phase 2 can only be entered if phase 1 is completed first and similarly phase 3 must be preceeded with phases 1 and 2. In this paper, *release readiness* always refers to how well-equipped a piece of proprietary software is to be released as open source.

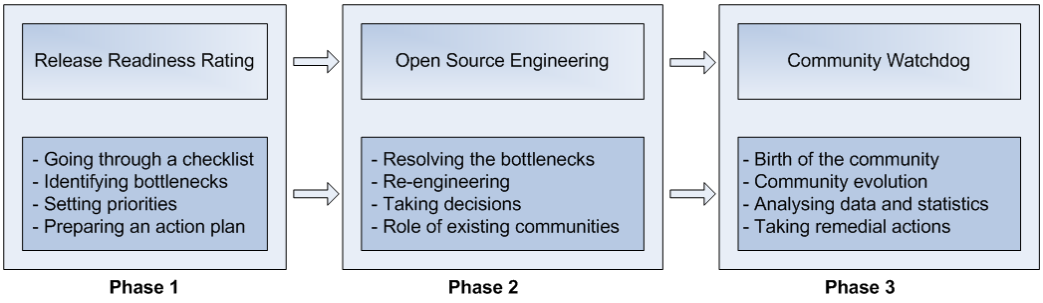


Figure 3: OSCOMM Framework

Evaluating the release readiness is done in order to identify possible bottlenecks of the opening process. Prioritizing things to be solved and setting up an action plan for the release are two essential actions to perform. Once an action plan is set, the second phase can be entered. The bottlenecks are solved by open source engineering and the role of the communities already existing within the interest range of the new-born is decided. Once the establishing community feels the product is ready to be released the process steps into the final phase – the community is born. In phase three the evolution of the new-born is kept an eye on by the community watchdog. The role of the company is to steer the community based on information gathered in phase 3. It is possible that even drastic decisions from the company may also be needed here. The framework does not decide these, it only provides information to support decision making. Each of these phases and the stakeholders involved in the releasing process are discussed next.

3.2. Stakeholders

Releasing an industrial software as open source involves multiple stakeholders with possibly different objectives and conflicting interests. It is vital that the building process balances the forces of the stakeholders so that at

least minimal satisfaction level is reached by all parties. Also issues like societal norms (e.g. seniority, way of communication) and legal matters (e.g. IPR issues, licenses used) should be considered. Any action not considering these forces could compromise the success and viability of the community.

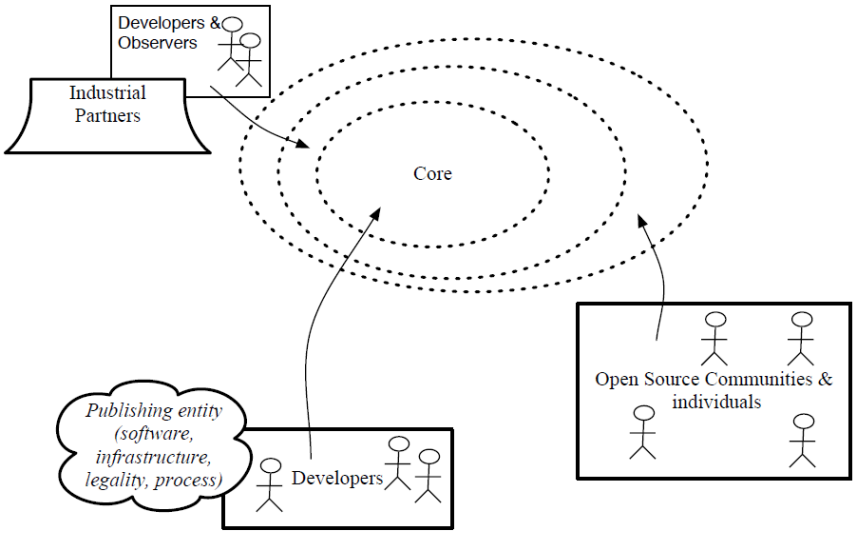


Figure 4: Ecosystem Stakeholders

Figure 4 depicts the main *stakeholders* involved during the early phases of the building process. There are three main groups of stakeholders: the *publishing entity* with its allocated resources for the project, *the industrial partners* and theirs developers, and finally existing *open source communities and other individuals*. As discussed earlier, FLOSS communities tend to take the shape of an onion with the inner layers taking more leading and contributing roles than the outer layers. The position of each of three groups of stakeholders in the onion structure is determinant by factors like familiarity with the project, objectives, and availability of skilled resources. Compared to the other two groups, the publishing entity is most familiar with the software to be released and most willing to invest in the building process. Thus, the role of this entity is essential both in preparation of the release and once opened. In this paper, our main focus is on releasing a proprietary software as open source. How the community evolves and works once the software has been opened is beyond the scope of the paper.

In addition to the software to be released, a number of *skilled developers* from the original development team should be allocated to form the core of the new-born community. The role of the developers is to lead the development of the software and to interface with the rest of the community members as they join. Furthermore, the entity should provide a proper project infrastructure, legality-related decisions such as the used license, and a process for managing the development of the software.

If the publishing entity has industrial partners associated with the software to be released, those partners might also be interested to get involved in the community. Typically, the partners are either enthusiastic or conservative. *Enthusiastic partners* are in favor of releasing the software as they see it as an opportunity for their business. Betting on the success of the community, these partners usually participate with developers to contribute to the development of the software and stay close to its evolution. Therefore, those developers are closer to the core team and might have key roles in the community.

Conservative partners however are reluctant for the software to go open source as this may change their mode of operations and business. Still, those kinds of partners would like to observe the evolution of the software and the outcome of the community by having own members in the community. Though familiar with the software, those observers are closer to outer layers of the onion structure as they do not assume any key development role in the early phases. It is of course possible that there are no industrial partners involved when releasing a software. In this case, the publishing entity should replace the void in the middle layer(s) of the onion structure by allocating more of their own resources and get other companies interested to the project.

The other vital element in the community building process is the *existing open source communities* and *other individuals*. The project released as open source will coexist with other development projects as part of the software ecosystem they create. The individuals could as well represent the interests of companies that are not partnering with the publishing entity. The software to be released should be carefully introduced to this group because existing communities and other individuals represent a pool of potential contributors who could join the project if they get interested and motivated. The individuals participating in the projects in the ecosystem form a social ecosystem of their own – the developer community. It is possible that individuals joining the new-born community do not have any earlier experience with open source projects. In any case, the newcomers typically join the community as passive

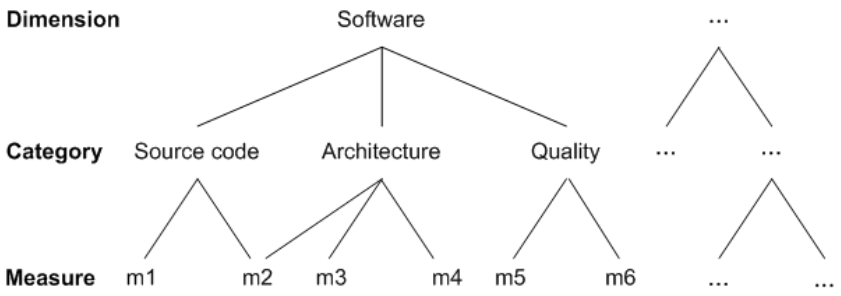


Figure 5: The Framework Model for Release Readiness, the software dimension

users and then may take key roles as they show commitment and value, thus penetrating the onion structure inward from outer layers. For this to happen, a well-defined developer promotion policy needs to be in place.

When releasing a proprietary industrial software as open source decisions need to be made on the role and type of the developer community for the software. Both the software and the social ecosystem influence the decision. The publishing entity needs to evaluate which of the alternatives company-based, volunteer or mixed type of community would suit the project and the ecosystem it will live in best. In addition attention needs to be paied to the dynamics of the onion model structure (closed or open core) and the place of the new-born in the ecosystem.

3.3. Release Readiness Rating

The aim of the release readiness rating framework for evaluating the readiness of a proprietary software is to help identifying possible bottlenecks and to eliminate them in the so-called pre-bazaar phase, where the goal is to prepare the software to be released and the releasing company to the continuation of the life of the system in the open source ecosystem.

An overview of the rating framework is shown in Figure 5. The criteria under evaluation for the readiness for release consist of four different dimensions: the software itself, the intended community and its roles, legality issues, and the releasing authority. In Figure 5 one of the criteria (software) has been opened to its subcategories and the measures used in each. Each of the criteria are similarly divided into categories and have different amounts of measures in each category.

3.3.1. Software

The **software** itself forms a substantial aspect for any open source project for obvious reasons. Based on our experience, at least the following issues must be taken into account. The role of the rating is to aid decision making and give supportive information. It is up to the company to decide whether to, for example, improve the quality of the software in case of a poor rating or to go open and allow the community to improve the code base. In either case the rating has given the company valuable information on what kind of software they are releasing.

Source code. Source code is a fundamental part of any open source project. When releasing a new software system as open source, there are numerous properties of the system related to code that can help other developers to participate in coding. Among these are *quality of code, integrity, and coding conventions*. An expressed *code of conduct* and the *documentation* of the system are also necessary.

Architecture. Developers must be able to easily understand the software system for it to be easily approachable. This in turn calls for an architecture that can be easily understood and communicated – and preferably documented. In addition to this, in the context of an open source ecosystem one should pay special attention to the design motivators of the architecture. If the architecture has been designed with changes in mind, it is often easier for other developers to adapt and create various types of new systems.

Quality attributes. The perceived quality of the software being released as open source is an noteworthy factor. The quality properties considered are not just the quality of the architecture or the actual implementation but the software as a whole is at focus.

3.3.2. Community

When releasing a software system one generally has an idea on what kinds of developers should get involved. Careful planning of the **intended community participants** can have an impact on what to release and how.

Purpose. As already stated in the book by Raymond [4], good work on software commonly starts by developers scratching their own itch. Therefore, the released system should be of practical importance and relevant for the developers. The goal of the community is probably the most significant single issue when releasing a piece of software as open source. However, since there commonly are numerous similar ongoing projects, an adequate mission is only a prerequisite for a successful launch and does not guarantee a thriving

community into the ecosystem will evolve.

User community. In addition to partners developing software, we feel that the potential for the user community is vital. Provided with an active user community, development resources can be invested in actual development, and the user community can provide support for other activities, such as peer user guidance, documentation, and testing.

Partners. The definition of partners that join in the community and thus forging an ecosystem can be rather straightforward. For instance, if a releasing company has been subcontracting from another company, the latter may be automatically involved in the newly formed community. However if the releasing company has no partners that would share the interest in the development, there should be a clear plan to motivate others to join in the development effort.

3.3.3. Legalities

In the context of companies, one of the most commonly considered aspects of releasing software as open source are the **legalities**. This is a wide topic to cover, and there can be several subtle differences in different contexts. Here, we assume a straightforward view where different concerns are discussed independently.

Copyright and intellectual property rights (IPR). Most commonly, companies release software whose copyright and IPR they own. Things are more complex, if the company does not own the copyright but sometimes it can be obtained via different transactions.

Licensing. The choice of license (or licenses) has an effect on how others perceive the community which can affect the willingness of developers to participate in the community and hence also the projects place in the intended ecosystem. Therefore strong copyleft licences such as GPL [15] have some advantage over more liberal licenses with no copyleft, such as MIT [15] and BSD [15]. However, one should also take the mission of the community into account when defining the license. From the point of view of the open source ecosystem license compatibility with related systems should also be considered.

Branding. Availability of brand names is an issue for any project looking for a good name that can be used in public. Since an open source project can be a long lasting one, selecting suitable brand names is essential. The same applies to hosting the project, since in many cases it would be practical to reflect the name of the project also in the domain.

3.3.4. Releasing Authority

The final element we address in our framework is the **releasing authority**, most commonly a company in the scope of the framework, which is aiming to release an in-house software as open source. However, Other parties such as universities, non-profit organizations, and individuals can act as the releasing authority,

Mindset, culture, and motivation. Sometimes the mindset of developers working in a company is somehow biased – either positively or negatively – towards open source development. In order to benefit from an open source community, the releasing authority should be motivated and mentally and culturally ready for dealing with developers outside the company under fair terms such as equal access to code, similar guidelines and conventions, as well as mutual respect.

Process, organization, and support. In order to gain benefits from open sourcing a system, the releasing authority should have a system in place that provides support for users and developers. This requires planning of a process that is to be followed, and putting the process in practice by the support organization and that is available from the very beginning.

Infrastructure. In order to establish an open source project, the releasing authority sometimes must be prepared to provide infrastructure. While some systems do not need such support as such, companies often wish to gain visibility through offering at least the download opportunity.

3.4. R3 Evaluation Framework Model

The release readiness rating framework discussed above is a general model for evaluating a proprietary software set out to be released as open source. Our model – the R3 evaluation framework model – is an implementation of it. The R3 model is organized into four main levels – dimensions. These levels represent the software itself, the intended community, legal issues and the releasing authority of the software. For each dimension there are a number of categories. Each category is then associated with a number of measures (i.e. questions). The R3 model, its dimension and the questions in each have been developed in co-operation with four companies (Nokia, Sesca Mobile, IT Mill, Gurux) all involved in opening their proprietary software as open source. The output of the evaluation can be considered as a vector that determines the relative values of these different elements. Each dimension is equal in weight but the questions in them are associated with varying weights based on their effect on the dimension.

The diversity of software products and additionally the different goals companies have made evaluating all software in a similar fashion impossible. The evaluation model itself must be tuned into taking into account the characteristics of the product under release. Some criteria may not make sense in some particular case and there may be a crucial criterion for the case missing altogether. Hence the R3 model should be considered as a template which needs to be instantiated to suit each case separately. When instantiating R3 all aspects of the model are gone over and validated as appropriate to the case in hand.

At the concrete level the evaluation process means taking R3 Spread Sheet Template and instantiating it to the case evaluated. Instantiating the template requires adding new dimensions and criteria to the spread sheet and possibly also removing some. The evaluation weights also require attention from the evaluator. The pracical application of the spreadsheet is discussed in Section 3.7

3.5. Open Source Engineering

The release readiness assessment of proprietary software does not just give a pass/fail result for opening the product. The evaluation yields a set of recommendations based on which the software under evaluation and its development environment then undergoes an open source engineering process that needs to be carried out before the software can be released. The process focuses on eliminating the problems and shortcomings that were identified in the evaluation. The success rate of building and sustaining the community will improve. The open source engineering process itself is driven by different kinds of influential factors that follow the same criteria as we used in the R3 framework.

Software. Any considerable rework on software requires extensive investment which puts numerous restrictions on what can be accomplished during the pre-bazaar phase. Still, it is possible to clean up and refactor the code to a certain extent, e.g. removing all company-specific comments. Since the code may already be in use, special attention must be paid to determine what to do with comments that indicate faulty or incomplete features. Some documentation can also be composed in the pre-bazaar phase, or simply be included in the comments of the code.

Community. Adding purpose as an afterthought can be difficult. Assuming that a system has been developed with only business interests in mind, it

can be difficult to introduce attractions for an independent developer. However, a mission for a community can be defined in pre-bazaar phase, provided that the software to be released enables a number of possible uses. Unfortunately companies can be somewhat biased towards supporting their own plans regarding the released system only, which in turn sometimes hinders the outside participation in the development for reaching some other goals, especially if the missions are conflicting. For instance, the releasing authority may not be willing to incorporate a community contribution for free, if the same feature can be sold for a commercial customer by the company.

Legalities. Legalities most commonly form the most straightforward category of items. Provided with copyrights, which can be difficult to obtain in pre-bazaar phase only, there are a lot of freedom to define the other aspects.

Releasing authority. We believe that the seeds of building cooperative relations between the releasing authority and the actual community should somehow be sewn at latest when entering the pre-bazaar phase. This can already be evidenced by existing ways of working and infrastructure, but they can also be introduced later on.

A conceptual framework for carrying out open source engineering can be seen in respect to the role of different community practices. In an earlier paper [9] the process is described and applied to an industrial case – the Wringer software platform originally developed by Sesca Mobile Ltd. A target community is selected as a starting point. The selection process may be based on software-related factors such as the programming environment or the application domain but any other company motives like business and strategy can be the driving forces. The selected community can be assumed to have norms and practices of it own. It is also possible that the company does not go for any existing community but prefers past experience or favoring popular solutions instead. Once a target community is selected the proprietary software and its environment is transformed and established according to the relevant practices the target community is found to have, i.e. licensing scheme, release management, communication within the community, documentation, coding style and convention, bugs reporting and tracking and source code management. Then the software can be released hopefully attracting some of the target community's members in order to form an embryo community which in turn can either join the target community or proceed to evolve as an independent one. Selecting a target community may not be an easy task in practice. The releasing company may also want to go for several communities complicating things further. If this is the case some of the community

practices, e.g. licensing and naming conventions differ or even conflict each other. Resolving such conflicts is then a part of the open source engineering process.

3.6. Community Watchdog

Once the newborn community is out in the open source world it needs to be nurtured, supported, and kept an eye on. Having up-to-date information on the developer community often plays a key role in making business decisions. Instead of relying on simple, single measures such as the number of downloads or the amount of messages on the community mailing list when evaluating the success and evolution of the community, a wider set of data sources should be considered. The community watchdog phase focuses on gathering and analyzing data. Both facets of the ecosystem – the software and the social model – are taken into account.

3.6.1. Implementing Community Watchdog

There are three different types of things to assess: the *community*, the *software* and *how well the objectives of the company are met*. The development of the first two can be seen based on the measures set up for them. The third requires assessment of whether the measures are moving to the right direction or if remedial actions are needed.

In order to collect information on the community the number of contributors to the project and the number of users who have subscribed the project mailing list should be measured. The amount of requests, feedback or inquiries received are interesting as is access to the web site, the number of hits the project gets in the media and blogs and what is the amount of activity on the project's visibility in the social media, e.g. Twitter, Facebook. The geographical distribution of the community members can be kept an eye on. The number of people participating in project events and meetings gives direct information on the activity in the community. Even the number of scientific publications mentioning the community or the amount of job advertisements can be monitored.

From the software viewpoint the number of downloads is a key piece of data. Similarly, the amount of reported bugs and feature requests are valuable. The number and impact of the contributions received tells a lot on the amount of development done outside the core of the community. In addition whether the contributions are corrective, adaptive, perfective or

preventive in nature could be analyzed. One interesting point to keep an eye on is the number of projects built on top of the platform.

Measuring and evaluating the community continuously gives valuable insight on the evolution of the ecosystem. The trends seen in the changes of the measures tell a lot on the state of the community and point out possible issues that require attention.

3.6.2. The BULB Model

The community watchdog is a framework for analysis of the community. It needs to be instantiated into a model suitable for the community in question. It may not be reasonable to measure everything and similarly a interesting metric may need to be added.

Firstly, we have instantiated the measuring of a open source community – a measurement model called BULB [10] – into an industrial case, the Vaadin community [16]. The data was gathered from the project's version control system, the bug tracker, Google Analytics and Alerts [17, 18], the database of the community's discussion forum and the projects web server logs. Things measured ranged from software specific things like amount of authors committing and changes in code to webhits on relevant sites. Based on the measures the activity in the community and the changes in the size of the community on the different layers were calculated. The evolution of the size of the different layers of the community is shown in Figure 6. The `IT Mill` layer represents the amount of people working on the software in the company behind it, `Act.dev` is the amount of active developers and `Per.dev.` represents the peripheral developers. `Bug fixer` layer is the amount of bug fixers in the community. `Reader` and `Pass.user` show the amount of readers and passive users in the community respectively. A similar picture can be created based on the changes in activity of the different layers. The changes in the activity of the Vaadin community over the same timeframe is visible in Figure 7.

Other aspects of the community, the amount of communication or the level of commitment for example, could be similarly monitored.

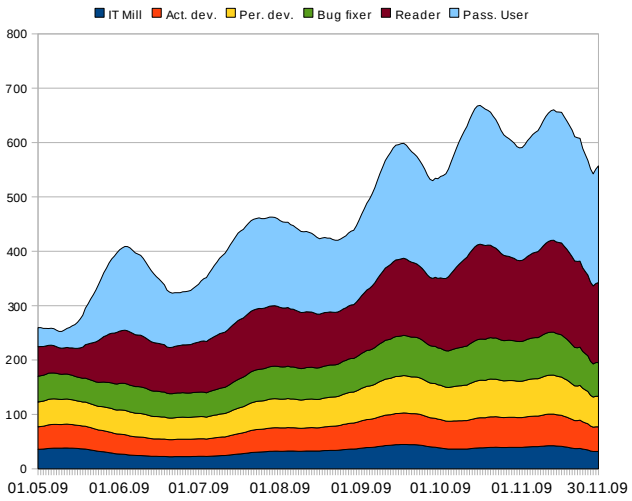


Figure 6: The Evolution of the Vaadin Community

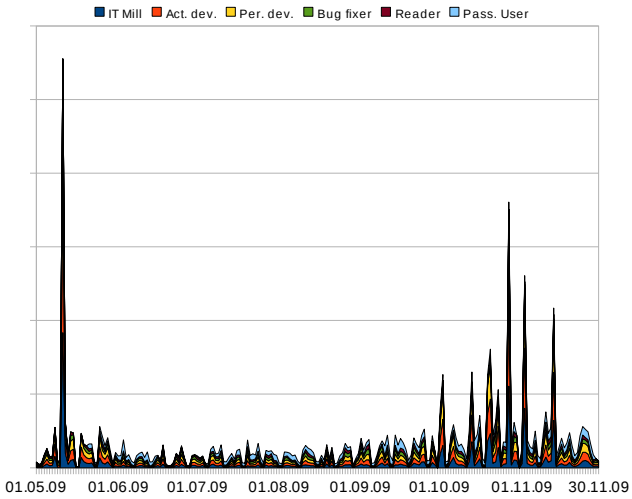


Figure 7: Activity in the Vaadin Community

3.7. Tool Support

With the framework tools to support the work in phases one and three were developed. Phase two, that concerns the open source engineering, contains tasks that can benefit from already available tools for refactoring, testing and architecture analysis for instance.

The tool for working on release readiness evaluation can be downloaded online.[2] The template needs to be assessed by the releasing authority as not all software is alike. Possibly new points need to be added or existing ones removed. The weights may also be adjusted to fit the situation tkaing into account the software and the needs of the people doing the assesment. In the example case at focus in this paper, all dimensions were seen equal and thus their respective weights set accordingly. The current dimensions, the items in each dimension and their weights are given in Table 1.

Table 1: R3 dimensions, items and relative weights

Dimension	**Item**	**Weight**
Software		*0.25*
	Source code	0.5
	Architecture	0.4
	Quality attributes	0.1
Community		*0.25*
	Purpose and mission	0.4
	User community	0.4
	Partners	0.2
Legalitites		*0.25*
	Copyright	0.6
	Licensing	0.3
	Branding	0.1
Releasing authority		*0.25*
	Mindset, culture and motivation	0.5
	Process, organization and support	0.3
	Infrastructure	0.2

The evaluation process starts from deciding in which order the criteria are going to be evaluated. The starting point should be the most crucial ones to the opening of the software under evaluation. Then the evaluation is carried out in the given order by answering concrete questions under each

[2]The excel sheet tool is downloadable on: `http://tutopen.cs.tut.fi/R3/R3_Template.xls`

dimension and item. The spreadsheet produces as a final result numerical information, a four-dimensional array, on the dimensions. The result gives indication on which dimensions need to be focused on in the next phase.

For phase two, the tools needed depend heavily on the company practises and the software product itself. Tools for code refactoring, architectural analysis, testing and documentation are used here but selection of the tools is a matter of taste and company practises and thus not fixed in the model.

For the last phase – community watchdog – a set of scripts were developed. The idea is that a variety of data sources can be used and new ones added if needed with no changes made to the current measuring. In Figure 8 the idea of using and extending the community watchdog is shown. New data sources can be added by processing their data into a fixed format in the data repository. The desired aspects, for example the size of the community and how active it is, are then collected by the scripts in the back-end and evolution of the community over time is shown.

4. Example Case Study: Gurux Software Ecosystem

The OSCOMM framework or parts of it have been applied in four separate industrial cases. Three of these have been published in separate publications. These cases were Notava [19], ITMill [10] and Sesca [9]. The latest case, Gurux, is presented here as an example of applying the framework.

4.1. Gurux Software Platform

The Gurux software platform [20] is a device communication solution originally developed as a closed source system by Gurux Ltd. The platform

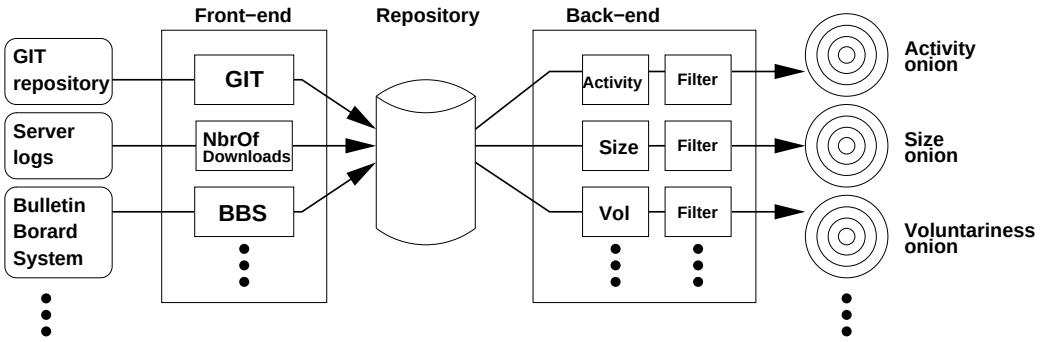


Figure 8: The intended functionality of BULB framework.

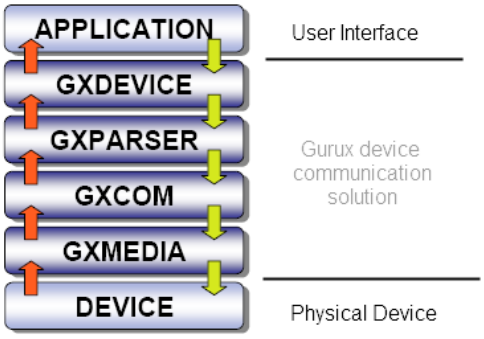


Figure 9: Gurux Platform Architecture

offers a set of communication products that are protocol, device type, and data connection independent. Figure 9 depicts the component architecture of the platform in order to give an overview of the product in question. The Gurux platform has components ranging from the user interface to those working at hardware level. The developer community of the released product has groups of developers concentrating on a single part of the entire platform. A device communication application is built using the services provided by the various components, supporting various programming languages and development environments. Example supported communication medias include SMS, GPRS, and SNMP. An example application would be to control an SMS-enabled device.

4.2. Goals

Inspired by the success of recent open source software ecosystems such as Meego [21], Gurux Ltd. took the strategic move to go open in mid 2009. The goal has been to grow and expand by reaching international partners and customers as well as reducing development cost of the software platform. Strategically, the objective has been to boost the field of technology of the products developed by the company and to gain reputation as an open source solution provider. Business-wise, the company decided to switch from selling software licenses to other business models such as software-as-a-service (SAAS), dual licensing, and offering consulting services.

The company anticipates that in a number of years the Gurux software ecosystem should grow by itself and should not anymore be heavily driven

by Gurux. The ecosystem would consist of companies and individuals with various business and technological interests in the platform.

4.3. Applying the OSCOMM Framework

In the following we describe the three phases of the OSCOMM framework applied to the Gurux software platform.

4.3.1. Phase 1

Four people from the releasing authority carried out the R3 evaluation for the Gurux platform before it was released as open source in November 2009. The overall impression was positive and R3 was considered a valuable and useful tool. The evaluation process acted as a good checklist for things that need to be considered when planning the release and showed well the items where most improvement is required. In addition, those items where improvement would be most beneficial were easily identified with R3, i.e. the releasing authority knew where to focus most of the effort.

Some items were not seen relevant in the case of the Gurux platform. However, this was not a significant problem for the process as irrelevant items were simply skipped by the people doing the evaluation. Sometimes choosing the correct grading was found hard. Grades like "well" and "reasonably" may mean different things to different people as these types of grades depend on how things are seen by individuals doing the evaluation and what they value.

The R3 evaluation data is confidential to the company and cannot be published here as is.

4.3.2. Phase 2

Having identified the bottlenecks in phase 1, Gurux allocated 6 full time workers for a period of six months to open source engineer their products. The company has worked on tasks preparing their software product on all six different dimensions of the OSCOMM approach. Listed by their order of importance, the most significant dimension has been the community aspect, then comes the process and software dimensions. To a lesser degree, the company took open source engineering related decisions related to marketing, infrastructure and legality dimensions.

The company planned to build an ecosystem of smaller communities, each focusing on a small part of platform. Contribution was expected at different levels of the component stack shown in Figure 9. It was decided that the core of the intended community would be open. It is also expected that

the community would be a mixture of volunteers and company-paid workers, provided both by Gurux and industrial partners. In order to attract users and contributors, Gurux has contacted existing related communities as a marketing strategy.

For supporting the community, Gurux established an infrastructure consisting of a bug tracker, forum, and documentation of components. Documentation includes sample code, animated tutorials and quick starts. Community registration has been made straightforward and the company plans to develop features to increase the size of the community. One of the latest feature implemented is a system where active users can gain points to reach higher levels and get more user rights. As for the software and process dimensions, it took considerable effort refactoring and cleaning up the software as well as putting up a process to manage the platform in an open source fashion.

4.3.3. Phase 3

Few months after releasing the software platform, Gurux has managed to build a community of 100 users (individuals and companies). After six months, the number increased to 200, most of which are abroad. The number of daily downloads is increasing all the time and is estimated to reach 200 by mid 2010. In addition to enthusiasts, the community consists of a number of small business partners.

The BULB model was instantiated to the Gurux community for monitoring the evolution of the community and analyzing the healthiness of the ecosystem. For monitoring Gurux has chosen to use the data sources: number of product downloads, number of visits to the webpage, volume of email communication and kind of feedback received and mentions of the product online through Google alerts. Data from these sources are collected continuously. Once these measurements are analyzed, remedial decisions and actions are taken regarding community support and software maintenance. Visually the analysis is analogous to the Figures 6 and 7.

4.4. Evaluation

Going open source has not been an easy decision or task for Gurux. A lot of effort was needed to plan and select the co-operation partners and to decide what, how, and when to do things. Eventually, going open source has caused the company to lose few customers because the software can be

downloaded and used for free. However, thanks to increased visibility, new customers have been obtained.

The role of partner organizations in the development of the Gurux ecosystem has been promising but still minor. Most of the contributions have been bug reports and feature requests but less effort has been spent on platform development. However, it is noticed that the bug reports has become well detailed which makes it easier for Gurux to fix the bugs. Content provided by the community falls short of company expectations, Gurux hope for a real social network sharing knowledge around the platform.

Most of the partner companies buy services and products from Gurux. However, some of the companies contribute with support money as the Gurux software platform is vital for their business. It is however still early to see companies build and sell own products built on top of the software platform.

Going open source helped the company to strategically view the software platform as an ecosystem. Other organizations are no more viewed as clients only but also co-developers and business partners, which may help maintaining the vitality of the platform.

Overall, building an open source ecosystem around the Gurux platform has gone slower than anticipated due underestimating the effort required. For instance, considerable time (almost the double of what is planned) has been spent on writing support documentation, rewriting code and example applications. Also it became a challenge to maintain different versions to meet the needs of different users. Nevertheless, the effort has led to better code quality and documentation. Also the company has now a better process in place for maintaining the software platform. For example, the company reported making architectural reviews on the software based on the evaluation which led to improvements in packaging. This in part helped in making the software more approachable to the community which can be seen as better quality. Maintenance of the repackaged product is also easier.

When asked about the open source engineering (phase 2), the company mentioned that marketing should have been stressed and taken more aggressively. In addition, more effort should have been spent on refactoring the software for community development. Finally, migration of infrastructure has also been underestimated and should have been considered as a more meaningful dimension. As future plans, the company intended to invest more in marketing aiming at increasing the size of the community to reach a self-sustainability level. Also more users of the software means more Gurux customers.

In what concerns the role of the research team in the Gurux case study, the company feedback has been positive. As described earlier, the research team did not just act as an observer but assumed an active part of the process, in an action research setting. Still, some aspects of action research were not applicable either. One aspect of action research is that it is an iterative process but in this study it was impossible to iterate within one case. A software product can only be released once. Nevertheless, the experiences of a single case could be utilized in releasing another product.

Furthermore, our action research role came with a number of inherent challenges. In order to avoid researcher bias and to increase internal validity of the results, we have conducted frequent reviews and syntheses of the findings with the company. Also, our role was beyond pure consulting activities by constructing concrete tools and methods in cooperation with the company. In order to mitigate threats to the external validity of our results, we have engaged four companies from different application domains. In several occasions, all the industrial partners were brought to the same discussion table. Our active involvement in the study may have exposed some of company's sensitive data. In order to address this possible source of conflict, the company has been consulted before any public use of data. Also we have avoided discussing the details of the case in the presence of other companies. On the other hand, we believe that we were not given access to some data which is classified by the company as sensitive. A wider access to the case would have allowed better and more detailed analysis.

There are several other validity threats to the design and implementation of this study. During the interviews with the companies, we may have somewhat mixed descriptions up with explanations of facts. This has influence on the kind of actions we have considered. Also, not all the data we are given represented real evidence. We had to filter out data that was not backed with clear facts. Also we were given data at different levels of detail. For instance, data related to the Community Watchdog phase has been raw and fine-grained whereas data related to the R3 phase has been brief and abstract. Our interpretation of data may have been influenced by its granularity level.

During data collection, we mostly used a single representative from the company. Although we tried to mitigate this threat by considering different perspectives such as technical, business, and legal, there is still a risk that a single company contact can be biased and provide a narrow view of the company. Similarly, the study has been carried out in a setting where

each company case has been allocated to a single researcher. In order to coordinate between all the cases, regular review meetings involving all the researchers have been organized. Finally, our understanding of the matters being investigated as researchers may have been different from that of the industry partners. In order to mitigate the associated risks, such issues were raised and consequently resolved during visits by company representatives to the research site.

5. Related Work

A main goal of our work is growing a sustainable open source community and furthermore a healthy actor in the open source ecosystem. This is directly tied to ecosystem health research – an ecosystem must be healthy to have longevity. In their paper [22] Iansiti and Levien define three measures for ecosystem healt: productivity, robustness and niche creation. Usable measures for ecosystem health at the company level are given in [23]. Although directly related to our work, the measures provided by den Hartigh et al. are strictly business oriented and on a general level. Our approach focuses on going open source with a proprietary software and thus addresses the open source and software specific issues. In [24] the authors present a model to describe the key characteristics of a software ecosystem. It investigates software ecosystems with a wide view while our approach is more indepth to a specific setting of opening propietary software.

Building a sustainable open source ecosystem, and thus the applicability of the OSCOMM approach, is best for software systems that are viewed as platforms. These kinds of systems have better chances for attracting community members as they offer higher levels of customizability through configuration and systematic adaptation to different usage contexts. A typical example of such scenario is where a software product line company decides to externalize the development of its software platform by including external developers and partner organizations [2].

Going open source with a proprietary software system is at the heart of our work. Thus the special traits of open source give an important angle to it. Open source maturity models such as OSMM^{TM}(Open Source Maturity Model) [25], QSOS (Qualification and Selection of Open Source Software) [26], and BRR^{TM}(Business Readiness Rating) [27] have been developed to aid companies in their adoption of open source. Whereas these

models evaluate existing open source software the viewpoint of our study is the opposite: opening a previously proprietary software product.

In their paper "How to have a successful free software project." [28] the authors argue that FLOSS projects typically undergo a number of activities that can be grouped into three phases: a cathedral phase, a transition phase, and finally a bazaar phase. However, in contrast to the metaphor of the "cathedral and the bazaar" [4], the authors show that the phases are complementary and represent common evolution phases of most open source projects. In a follow-up research study the framework has empirically been illustrated using a number of case studies [29]. The OSCOMM approach fits well into this three-phase classification. Release Readiness Rating can be seen as an activity that occurs at a late stage of the cathedral phase, where a decision to open up the proprietary software has been taken. Open Source Engineering, on the other hand, corresponds mostly to the transition phase, where certain aspects of the software and its environment are improved before the actual open source release. Finally, the Community Watchdog phase of the OSCOMM approach can be associated with the bazaar phase where project evolution is driven by the new born community.

When it comes to documented practices, the work of Fogel [30] is the closest to our approach. The author presents a cookbook-like guide for starting and running an open source community in general. The guide, which takes a very low-level and detailed view to the phenomenon of open source, considers different perspectives ranging from technical infrastructure to political issues in open source. Compared to Fogel's [30], our approach focuses on the specific problem of opening industrial software.

On a much larger scale, the best practices we have discussed in the OSCOMM approach have been partly considered in the so-called incubation programs. These programs aim to establish fully functioning open source communities for promising project ideas and initiatives. For example Eclipse and Apache both have their own incubation programs [31, 32]. During the incubation process, a project goes through rigorous review phases where the outcome can be either termination, continuation or promotion of the project. Incubation programs and the OSCOMM approach share the same aim: how to have the project accepted by a community. In both approaches, the evolution of the community should be monitored carefully.

The problem of open source community building has also been researched in the licentiate thesis by Stürmer [33]. The work describes a qualitative research of eight successful open source projects. The study is based on

interviews with a representative of each project. The interviews go through the life cycles of the projects from the beginning to the current state. The main contribution of the study is in describing how to initialize an open source project and how the project is promoted. Compared to our approach, the main difference is that we do not base our study to interviews made after the fact, but participate in the actual release process from the beginning. Moreover, we are not passive observers, but we attempt to impact building the project in various ways in order to improve its likelihood to succeed.

The OSCOMM approach is not a one-size-fits-all solution but rather an umbrella guiding framework that should be tailored to individual needs. Indeed, we have applied the approach differently to a number of case studies other than the Gurux software platform. In each of the case studies, one of the three phases has particularly been stressed. For instance, an intense open source engineering process has been applied to a JavaScript binding platform originally developed by Sesca Mobile Ltd. [9]. The process has been driven by the practices of a selected open source community. In another case study platform developed by IT Mill Ltd., the community watchdog phase has been the most useful among the three phases of the OSCOMM approach. The company wanted to have concrete software tools to monitor the evolution of the community around its Vaadin software platform [10].

6. Future Work

Growing an ecosystem for any software product is a long and continuously evolving process. The work presented in this paper concentrates on establishing an open source ecosystem that is viable on the basis of a previously proprietary software. The focus is on building a framework for opening the software and it does not go too far into the life of the established community.

The framework does not give strict guidelines for the day-to-day working of the community and the ecosystem. Any community however needs such to govern the work and the interaction between the participants. In an open source community there is a need for practices for decision making, communication methods, handling patches and deciding on releases to name a few. The governance framework needed for handling such things is a tempting issue for a future study.

Our focus so far has been the birth of the ecosystem and how to ensure a possibility for growth. A long term study on an established ecosystem should be conducted. A period of five to ten years in the life of the ecosystem could

be monitored either from the start onwards or from today back into the history of one of the most established open source ecosystems. Especially the latter might bring light on how large a role chance plays in success and the former more on how key business decisions and a governance organization can support the community and the ecosystem to grow and thrive.

The software business is sometimes fickle and follows trends as any business does. It is plausible that a need to close the software would also arise. To what extend the same six dimensions considered here in the opening process need to be addressed when going back to proprietary is an interesting question for future research. Additionally how choosing a multilicensing scheme instead affects the ecosystem is a possible research path.

7. Conclusions

Free/Libre/Open Source Software (FLOSS) has emerged as a promising platform – including technical, methodological, legal, commercial, and numerous other facets – for software ecosystems. Consequently, more and more companies are releasing their proprietary software as open source, forming a software ecosystem of related development projects complemented with a social ecosystem of community members.

This paper studies the problem of building open source communities for industrial software that was originally developed as closed source. Supporting processes, guidelines and best practices are discussed and illustrated through an industrial case. We conclude that while the multi-faceted nature of using open source as a platform introduces inevitable complexities, there are existing tools and techniques that can be used to deal with them. Moreover, some data from real cases is available, although not to an extent that would provide conclusive evidence.

One of the main contributions of the OSCOMM approach is making explicit and documenting the elements required for building and sustaining an open source ecosystem. We believe that such practices, at least in some form, do exist in major software companies. These practices however are very seldom documented and shared with other organizations.

Finally, getting experiences from a number of further real-life cases is what is eventually required to make it possible to turn the task of building an open source ecosystem into an engineering and business discipline. With this research, we are paving the road for future experiments conducted in different contexts and with different business settings in mind.

References

[1] D. G. Messerschmitt, C. Szyperski, Software Ecosystem: Understanding an Indispensable Technology and Industry, MIT Press, Cambridge, MA, USA, 2003.

[2] J. Bosch, From software product lines to software ecosystems, in: SPLC '09: Proceedings of the 13th International Software Product Line Conference, Carnegie Mellon University, Pittsburgh, PA, USA, 2009, pp. 111–119.

[3] S. Jansen, A. Finkelstein, S. Brinkkemper, Sense of community: A research agenda for software ecosystems, in: In proceedings of the 31st Internatinal conference on software engineering, companion volume, IEEE, 2009, pp. 187 – 190.

[4] E. S. Raymond, The Cathedral and the Bazaar, O'Reilly Media, 1999.

[5] C. Walton, The open source software ecosystem, Published in: Carles Sierra and Jaume Augusti editors, IIIA Communications, Institut d'Investigacion en Intel.ligencia Artificial, IIIA, Barcelona (2002).

[6] B. Lundell, B. Forssten, J. Gamalielsson, H. Gustavsson, R. Karlsson, C. Lennerholt, B. Lings, A. Mattsson, E. Olsson, Exploring health within oss ecosystems., in: In Proceedings of OSCOMM 2009, Sweden, 2009.

[7] T. Kilamo, T. Aaltonen, I. Hammouda, T. J. Heinimäki, T. Mikkonen, "Evaluating the Readiness of Proprietary Software for Open Source Development", in: OSS2010, Vol. 319 of IFIP Advances in Information and Communication Technology, Springer, 2010, pp. 143–155.

[8] P. Sirkkala, T. Aaltonen, I. Hammouda, "Opening Industrial Software: Planting an Onion", in: OSS2009, Vol. 299 of IFIP Advances in Information and Communication Technology, Springer, 2009, pp. 57–69.

[9] F. R. Shah, I. Hammouda, T. Aaltonen, Open Source Engineering of Proprietary Software: the Role of Community Practices, In proceedings of the OSCOMM 2009 workshop, Skövde Sweden (2009).

[10] T. Kilamo, T. Aaltonen, T. J. Heinimäki, BULB: Onion-Based Measuring of OSS Communities, in: OSS2010, no. 319 in IFIP Advances in Information and Communication Technology, Springer, 2010.

[11] K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida, Y. Ye, Evolution Pattern of Open-Source Software Systems and Communities, in: IWPSE '02: Proceedings of the International Workshop on Principles of Software Evolution (2002), ACM Press, 2002, pp. 76–85.

[12] P. A. Gloor, Swarm Creativity : Competitive Advantage through Collaborative Innovation Networks, Oxford University Press, USA, 2006.

[13] J. Preece, Online Communities: Designing Usability, Supporting Sociability, Wiley, 2000.

[14] D. Avison, F. Lau, M. Myers, P. A. Nielsen, Action research, Communications of the ACM 42 (1) (1999) 94–97.

[15] Licences, `http://www.opensource.org/licenses`, last visited February 2009.

[16] M. Grönroos, Book of Vaadin: Vaadin 6, Oy IT Mill Ltd, 2009.

[17] Google analytics service, `http://www.google.com/analytics/`, last visited April 2010.

[18] Google alerts, `http://www.google.com/alerts/`, last visited April 2010.

[19] I. Hammouda, T. Aaltonen, P. Sirkkala, Exploiting social software to build open source communities, in: In Proceedings of SoSEA 2008, IEEE Computer Society, L'Aquila, Italy, 42-25, p. 2008.

[20] Gurux Software Platform, `http://www.gurux.fi/`, last visited April 2010.

[21] The MeeGoCommunity, `http://meego.com`, last visited January 2011.

[22] M. Iansiti, R. Levien, Keystones and dominators: Framing the operational dynamics of business ecosystems (2002).

[23] E. den Hartigh, M. Tol, W. Visscher, The health measurement of a business ecosystem, in: Paper presented for the ECCON 2006 Annual meeting Organisations as Chaordic Panarchies, 2006.

[24] I. van den Berk, S. Jansen, L. Luinenburg, Software ecosystems: A software ecosystem stragedy assesment model, in: ECSA '10 Proceedings of the Fourth European Conference on Software Architecture: Companion Volume, ACM, 2010.

[25] B. Golden, Succeeding with Open Source, Addison-Wesley, 2004.

[26] QSOS, `http://www.qsos.org/`, last visited December 2010.

[27] BRR, `http://www.openbrr.org/`, last visited December 2010.

[28] A. Senyard, M. Michlmayr, How to have a successful free software project., in: In Proceedings of the 11th Asia-Pacific Software Engineering Conference, IEEE Computer Society, Busan, Korea, 2004, pp. 84–91.

[29] A. Capiluppi, M. Michlmayr, From the cathedral to the bazaar: An empirical study of the lifecycle of volunteer community projects, in: In Proceedings of the 3rd International Conference on Open Source Systems, Springer, Limerick, Ireland, 2007, pp. 31–44.

[30] K. Fogel, Producing Open Source Software, O'Reilly Media, 2005, available at: `http://producingoss.com/`, Last visited April 2010.

[31] The Eclipse Project Incubator, `http://www.eclipse.org/eclipse/incubator/`, last visited April 2010.

[32] The Apache Incubator, `http://incubator.apache.org/`, last visited April 2010.

[33] M. Stürmer, Open Source Community Building, licentiate thesis (2005).

II

Case I: Publication II

Terhi Kilamo, Timo Aaltonen, and Teemu J. Heinimäki. BULB: Onion-Based Measuring of OSS Communities. In *Proceedings of the 6th International IFIP WG 2.13 Conference on Open Source Systems (OSS'10)*, pages 342–347. Notre Dame, IN, USA, May 30 – June 2, 2010, Springer.

BULB: Onion-Based Measuring of OSS Communities

Terhi Kilamo[1], Timo Aaltonen[1], and Teemu J. Heinimäki[1]

Tampere University of Technology
firstname.lastname@tut.fi

Abstract. Up to date information on the associated developer community plays a key role when a company working with open source software makes business decisions. Although methods for getting such information have been developed, decisions are often based on scarce information. In this paper a measuring model for open source communities, BULB, is introduced. BULB provides a way of collecting relevant information and relates it to the well-known onion model of open source communities.

1 Introduction

A company working with open source software (OSS) is often dependent on the developing community. Especially, when a product or service is sold the connection is obvious. In order to make business decisions, the need for up to date information about the community is clear. For example, the size of the community should be known, the activity of developers is interesting, and how easy penetrating into the community is should be found out.

Currently getting this kind of information is hard. Precise models for such have been developed. For example, *social network analysis* (*SNA*) [3] has been suggested to get a strict view to the community. SNA analyses a mathematical graph, where nodes are the members of the community and arcs model relationships between them. Different kinds of surveys can be given as another example of community information digging.

The industry does not seem to use such advanced methods today. On the contrary, to our knowlegde the business decisions are often based on simple models, and, it is not unusual that the only two sources of information are the number of messages in discussion forums and the number of downloads.

In order to be adopted in the industry, metrics need to be instantly meaningful. We propose is a measuring model *BULB*, which relates the measurements to the well-known onion model [7] of OSS communities. The onion model is commonly accepted and it is easy to grasp, so BULB conforms to the prerequisites. The measurements are based on robots digging continuously information from various sources, like the discussion forum, the version control and the bug repository, i.e. the framework conforms to the rest of the conditions.

The rest of the paper is structured as follows. In Section 2 ways to measure open source communities are discussed. The BULB model for measuring open source communities is introduced Section 3 and applying it to an industrial community is given Section 4. The paper in concluded in section 5

2 Measuring Open Source Communities

The community behind an open source project is a key component that effects the success of the project. Information on the nature of the community is needed in order to make informed decisions on adopting open source software and to aid running a successful business based on open source components. Several different approaches have been suggested to provide support in the decision making ranging from easy to get to more extensive analysis.

Social Network Analysis: Any open source project can be seen as a social network of developers. The developers are linked to each other through different kinds of relationships that are created and maintained in OSS projects mainly by computer-enabled channels. The OSS community is thus seen as a graph with the developers as the nodes and the social relationships between developers as the edges. Social network analysis (SNA) [3, 10] can be used to study the community and its structure.

Business Readiness Rating: Business Readiness Rating (BRR) [2] proposes a method for assessing open source software. The goal is to get a rating on the open source software through four steps (1. quick assessment, 2. target usage assessment, 3. data collection and processing and 4. data translation) As BRR itself admits that phase three is the most time consuming and yields best results for mature projects, its value is somewhat limited to eliminating bad candidates. It also seems apparent that BRR is no longer being developed further at the moment.

Simple metrics: One way to evaluate the open source project is to measure some publicly available data that are easy to access and measure. Very simple metrics, such as the amount of downloads or the daily amount of discussion on email lists or the project discussion forum are used. Naturally the level of activity in the community shows if the community is still alive, but says very little on the product or the sustainability of the community on the whole.

Software use: Software use is naturally a popular metric albeit one that is difficult to measure reliably in the case of open source software. Numbers of downloads alone do not reliably tell about adoption [1]. Moreover from the business decision point of view, the community as a whole is interesting, not just the usage.

Surveys: Surveying community members is a suitable method for gaining information from different interest groups within the community [9, 4, 11]. However, surveys don't necessarily reach people, whose input would be most valuable. In addition, surveys cannot be used as a mean of continuous analysis as the likelihood of people answering them decreases over time.

Appearance: A common method of comparing projects and making decisions on the project to use is not based on any kind of measuring as there often is no time to undergo a vast analysis. The appearance of and the feeling one gets from the community are the driving factors instead of a more formal approach. Results beyond a blatant guess are needed, and therefore the need to evaluate the community further is apparent.

Onion-Based Measuring: Open source communities can be modeled with an onion model introduced in [7]. In the model each member of the community has a distinct role. The community is seen as an onion-like structure, where the most influential community members occupy the core layers, while the outer layers hold the less influential ones.

3 Constructing an Onion-Based model: BULB

In this section a measuring model for onion-based measuring of open source communities, BULB, is given. The structure of the community and how the community members fall on layers in the onion is valuable information about the community and its current state in making business decisions. BULB has been developed for this very purpose. The theoretical base of the model has already been introduced in [6]. In it, the traditional onion model for open source communities is substituted with two onions, one for the size of the community and another for the amount of activity on the onion layers. The traditional size

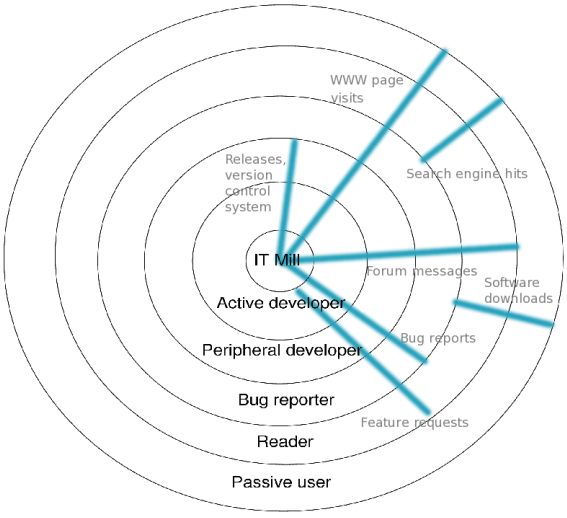


Fig. 1. The data sources on the onion

onion is produced by simply assessing the number of people on each layer of the onion. Data from several data sources is combined to get a picture of the

structure of the community according to the onion model. Data used at this point range from the version control system information to the number of web hits for relevant sites. Some of the data sources have more effect on the onion built out of the data than others. This is taken into account when constructing the onion model for the community. The data sources used and the onion layers affected by each source are depicted in Figure 1.

The onion is seen as a vector, where each element contains the relevant information about the layer, for example in the size onion the number of people on the corresponding onion layer. Each metric used is measured daily and a vector representing its distribution on the onion layers is created by multiplying it with a coefficient vector that indicates how influential the metric is on each layer. The coefficient values of the layers add up to 1.0. If we denote the set of metrics used with M the distribution vector $\mathbf{d_i}$ of each metric is calculated:

$$\forall m_i \in M : \mathbf{d_i} = m_i \mathbf{v_i} \tag{1}$$

where $\mathbf{v_i}$ is the coefficient vector of m_i. The example vectors used in the case studyfor distributing the numbers of bug reports and feature requests over the onion are shown in Figure 2. The different data sources can have significant

Bug reports:

0.2	0.3	0.3	0.2	0.0	0.0

Feature requests:

0.0	0.25	0.25	0.25	0.25	0.0

Fig. 2. Example coefficient vectors

differences in their relative values as one can be very large while the other occurs more rarely and is thus smaller. To compensate this the distribution vector of each metric is multiplied with a balancing coefficient b_i. We get a partial onion vector:

$$\mathbf{p_i} = b_i \mathbf{d_i} \tag{2}$$

The significance of the metric on the onion can also be scaled through this coefficient. In the example measurements, the balancing coefficients used were 7.0 for bug reports and 9.0 for feature requests.

After the balancing the complete onion is created by simply adding the values on each onion layer in the partial onion vectors together in order to create the final onion, i.e.

$$\mathbf{o} = \sum \mathbf{p_i} \tag{3}$$

The traditional onion alone is not able to accurately depict how active the community members on the different layers of the onion are but is simply focused on the size and structure of the community. The activity may vary over time although the size of the community has not changed. Thus the variation in activity on the different layers should be taken into account in addition to the development of the size of the community. As some of the metrics used may give information about the current level of activity on a given layer BULB suggests a second onion similar to the size onion to be used for depicting layer

activity. The activity onion is built like the size onion only based on the metrics that measure activity. The distribution vectors and the balancing coefficients are naturally adjusted suitably. In the example case, the balancing coefficients change to 100.0 for bug reports and 140.0 for feature requests as they are clear indications on activity.

4 Experimenting BULB with the Vaadin community

The BULB model is in fact a generic method of depicting the evolution of an open source community. The data values in the onion vectors can be changed to a new community characteristic and the model is still applicable. To study the applicability of the framework, we have experimented it with the developer community of Vaadin [8]. The measurements were carried out from May 1 2009 to Nov 30 2009. The onion model was instantiated to the Vaadin community as shown in Figure 1.

Vaadin is a server-side AJAX web application development framework developed by Oy IT Mill Ltd [5]. The framework is used for developing rich Internet applications with the Java programming language. Vaadin framework was released as open source in December 2007. The business model of the company is based on consulting services and the development of Vaadin. As Vaadin is open source, IT Mill needs up to date information about the Vaadin community. So far the main source of information has been the number of downloads and the number of messages post to the discussion forum. Figure 3 illustrates the

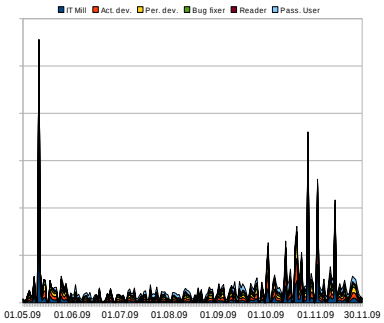


Fig. 3. The activity onion of the Vaadin community

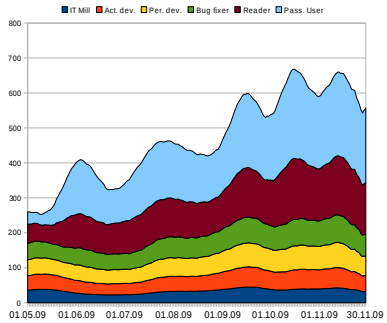


Fig. 4. Size onion of the Vaadin community.

measured activity in the Vaadin community over the measurement window. It visualizes the effect of events that have impact in the community. The size data however needs to be filtered to lessen the weekly variation in the raw measures. The size onion of the Vaadin community after filtering the data with a Gaussian filter is shown in Figure 4. The window size of the filter was 31, $\mu = 0$, $\sigma = 31/4 = 7.75$.

5 Conclusions

We have developed a new onion-based model, BULB, for measuring open source communities. We applied the model in an industrial case to measure the layer

sizes in the Vaadin community onion and the activity on the layers. The model was developed in cooperation with industry to make an easy-to-use and fast way for digging valuable information on an open source community out of the available data.

The board and other stakeholders of an open-source company or of companies thinking of adopting an open source product often base their decision to a limited amount of information. With BULB these decisions can be based on more fresh and divergent information than before. We have shown that the described model works and produces sufficiently precise information fast enough to be useful and support decision making.

References

1. Wiggins A., Howison J., and Crowston K. "Heartbeat: Measuring Active User Base and Potential User Interes in FLOSS Projects. In *Open Source Ecosystems: Diverse Communities Interacting*, volume 299/2009 of *IFIP Advances in Information and Communication Technology*, pages 94–104. Springer, 2009.
2. BRR. `http://www.openbrr.org/`. Last visited December 2009.
3. Del Rosso C. Comprehend and analyze knowledge networks to improve softaware evolution. *Journal of Software Maintenance and Evolution: Research and Practice*, 21(3):189–215, 2009.
4. Capra E., Fancalanci C., Merlo F., and Rossi Lamastra C. A Survey on Firms' Participation in Open Source Community Projects. In *Open Source Ecosystems: Diverse Communities Interacting*, volume 299/2009 of *IFIP Advances in Information and Communication Technology*, pages 225–236. Springer, 2009.
5. Oy IT Mill Ltd. web page: `http://http://www.itmill.com/`. Last visited December 2009.
6. Heinimäki T. J. and Aaltonen T. An onion is not enough: Living in the multi-onion world. In *Proceedings of the Open Source Workshop - OSW 2009 In conjunction with the 4th IEEE Systems and Software Week (SASW 2009)*, Skövde, October 2009.
7. Nakakoji K., Yamamoto Y., Nishinaka Y., Kishida K., and Ye Y. Evolution Pattern of Open-Source Software Systems and Communities. In *IWPSE '02: Proceedings of the International Workshop on Principles of Software Evolution (2002)*, pages 76–85. ACM Press, 2002.
8. Grönroos M. *Book of Vaadin: Vaadin 6*. Oy IT Mill Ltd, 2009.
9. Ghosh RA, Glott R., Krieger B., and Robles G. Free/Libre and Open Source Software: Survey and Study. International Institute of Infonomics, 2002.
10. Wasserman S. and Faust K. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.
11. Mikkonen T., Vainio N., and Vadén T. Survey on four oss communities: description, analysis and typology, 2006. Empirical insights on open source software business.

Case I: Publication III

Terhi Kilamo, Timo Aaltonen, Imed Hammouda, Teemu J. Heinimäki., and Tommi Mikkonen. Evaluating the Readiness of Proprietary Software for Open Source Development, In *Proceedings of the 6th International IFIP WG 2.13 Conference on Open Source Systems (OSS'10)*, pages 143–155. Notre Dame, IN, USA, May 30 – June 2, 2010, Springer.

III

Evaluating the Readiness of Proprietary Software for Open Source Development

Timo Aaltonen[1], Imed Hammouda[1], Teemu J. Heinimäki[1], Terhi Kilamo[1] and Tommi Mikkonen[1]

Department of Software Systems, Tampere University of Technology
Korkeakoulunkatu 1, FI-33720 Tampere, Finland
{firstname.lastname}@tut.fi

Abstract. As more and more companies are releasing their proprietary software as open source, the need for supporting guidelines and best practices is becoming evident. This paper presents a framework called R3 (Release Readiness Rating) to evaluate the readiness of proprietary software for open source development. The framework represents a checklist for the elements required to ensure a better open source experience. The framework has been applied to an industrial proprietary software planned to be released as open source. The evaluation has been carried out by both external and internal stakeholders. The early experiences of the case study suggest that the R3 framework can help in identifying possible bottlenecks before evangelizing the software to the open source community.

1 Introduction

Companies are getting more and more interested in releasing their closed source software products to open source communities. The two large scale examples of this are Sun Microsystems' opening of its Java platform during 2006 and 2007, and Nokia's actions to open the Symbian operating system during 2009-2010 [11]. As the trend is relatively recent, the phenomenon of opening industrial software is not well understood despite of the existence of general guidelines such as in [2, 10]. In this paper we tackle the problematic of releasing industrial software.

Most often companies are not used to release the source code of their products. Their standard ways of behavior tend to be more biased to hiding than to releasing information. The processes, tools and infrastructure used by companies might turn out to be an obstacle for a successful release. The software itself might have been written so that open source developers run into troubles when trying to contribute. This suggests that there is a need for proper methodologies to evaluate the readiness of proprietary software for open source development. Such methodologies would help identifying possible bottlenecks before taking the software to the open. The bottlenecks are then resolved to in order to increase the success rate of community building around the software.

Given the above observations, the research questions we would like to explore include the following:

- What kind of evaluation criteria could be used to assess software readiness for open source development?
- How the evaluation should be planned and which stakeholders are involved?
- How to obtain data for the evaluation process?
- How to exploit the results of the evaluation process?

We argue that these issues have not been studied enough by the open source research community. The closest works to our study are the open source maturity models such as OSMM (Open Source Maturity Model™) [3], QSOS (Qualification and Selection of Open Source Software) [8], and BRR (Business Readiness Rating™) [1]. These models are typically used by companies that plan to use open source. The context of our research problem in this paper is just the opposite: taking software out from companies to open source communities.

The main contribution of the paper is two fold. First, we discuss the specificities of the problem of opening proprietary software. Second, we present a framework called R3 (*Release Readiness Rating*) to evaluate the readiness of proprietary software for open source development. In order to demonstrate our approach, we have applied the framework to an industrial proprietary software planned to be released as open source.

The rest of this paper is structured as follows. In Section 2 we discuss related work and the challenges of opening proprietary software. The details of the R3 framework and the overall evaluation process are presented in Section 3. In Section 4, we evaluate the R3 framework in the context of two industrial case studies. Future work is discussed in Section 5 and finally, we conclude in Section 6.

2 Background

2.1 Open Source Maturity Models

Several methods have been developed assessing the maturity of open source software. For instance, Open Source Maturity Model™(OSMM) enables a quick assessment of the maturity level of an open source product. Products are ranked according to OSMM scores, which are evaluated in a three-phase process: 1) assess each product element's maturity and assign maturity score; 2) define weighting for each element based on the company's requirements and 3) calculate the score.

The Qualification and Selection of Open Source Software (QSOS) maturity model is a four-step iterative process: 1) define (and organize criteria), 2) asses (against the criteria), 3) qualify (define weighted scores, new and mandatory criteria) and 4) select (asses using the weights, and select). Another evaluation framework called Business Readiness Rating™(BRR) was proposed as a

new standard model for rating open source software. The model consists of a four-phase process: 1) quick assessment filter (for quickly abandon bad candidates), 2) target usage assessment (for inputting the needs of the company), 3) data collection & processing (for collecting the actual information) and 4) data translation (which leads to one outcome: the rating).

Compared to the method we propose in this paper these maturity models take a totally different direction. Whereas our model attempts to study one software product which is going out from a company, these maturity models attempt to study a set of software products, one of which is selected to come in to the company. However, some ideas are still quite similar. For example, in both cases the architecture of the software plays an important role, and it can be evaluated similarly. On the other hand the infrastructure of an open source project might not be so important when evaluating open source software to be used, however it is a crucial element when building an open source community.

2.2 Opening Proprietary Software

Like any other online community [7], creating and maintaining a sustainable open source community for proprietary software can be considered as a multi-facet challenge. It is a complex process that is driven by various kinds of factors, which in turn can be grouped along six dimensions.

Software. Improved software quality may increase the success rate of community building. Quality can be enhanced by incorporating best practices, documentation, code cleanup, coding standards and convention. Furthermore, in order to support the community, source code may be accompanied with user manuals, API documentation, and architecture descriptions. It is vital to have the first experience with downloading, installing, deploying and using the software as easiest as possible.

Infrastructure. There are two key elements in any open source project: community and project repository. An open source engineering process should provide enabling tools and technologies to facilitate the planning, coordination, and communication between the community members. In addition, efficient mechanisms and tools are needed to facilitate the access and management of the project repository.

Process. Open source development can be regarded as an open maintenance process. A process needs to be established in order to handle decisions regarding the evolution of the software, maintenance actions, and release management. The process needs to balance between the practices of communities and the needs of the company.

Legality. The releasing company needs to select a license type (e.g. GPL versus LGPL) and a licensing scheme (e.g. single or multi licensing). In addition, source code should be legally cleared against IPR and copyright issues. Also, the availability of trademarks and names used in the software should be checked.

Marketing. Building an open source community can be regarded as a marketing challenge. Effective marketing strategies are needed to market the open

source project to potential users and developers. Selecting an existing open source community as a target customer can be an important success factor.

Community. The releasing company should be ready to support the project community. For instance, community members should be provided with clear guidelines on how and what to contribute. Furthermore, company developers who are participating in the community should be trained for their new roles and should be given clear responsibilities. When the software is opened, it is vital that all information are made public and that private discussions are avoided. In addition, there should be trust among community members, zero tolerance of rudeness and no use of bad language both in the software artifacts and the communication among the members.

In the next section, we present a framework that addresses these questions by evaluating software in a pre-bazaar phase. The pre-bazaar phase helps in getting early feedback and experiences on using and evolving the software outside its original development environment. This may need the involvement of external stakeholders.

3 The Release Readiness Rating Framework

The Release Readiness Rating (R3) framework is a tool for planning the open sourcing of a software system. The goal of the framework is to help identifying possible bottlenecks and to eliminate them in so-called pre-bazaar phase, whose goal is to prepare the software to be released and the releasing company to the continuation of the life of the system as open source.

3.1 Framework Overview

The evaluation criteria for R3 consists of four different dimensions, including software itself, intended community and its roles, legality issues, and the releasing author. The output of the evaluation can be considered as a vector that determines the relative values of these different elements. The dimensions are further decomposed as indicated in Table 1. The table also lists the relative importance of the item.

The current weights are based on our experience on previous case studies. All dimensions are equal in weights when the individual items are associated with different weights.

3.2 Evaluation Criteria

In the following, we discuss the different views that should be considered when deciding the value set for the items representing different dimensions.

Table 1. R3 dimensions, items and relative weights

Dimension	**Item**	**Weight**
Software		*0.25*
	Source code	0.5
	Architecture	0.4
	Quality attributes	0.1
Community		*0.25*
	Purpose and mission	0.4
	User community	0.4
	Partners	0.2
Legalitites		*0.25*
	Copyright	0.6
	Licensing	0.3
	Branding	0.1
Releasing authority		*0.25*
	Mindset, culture and motivation	0.5
	Process, organization and support	0.3

Software The software itself forms an important aspect for any open source project for obvious reasons. Based on our experience, at least the following issues must be taken into account.

Source code. Fundamentally, any open source project deals with source code. When releasing a new software system to open source, there are numerous properties that the system itself, manifested in its code, should contain. These in particular include *quality of code, integrity, and coding conventions* hat help other developers to participate in coding. The importance of coding conventions is highlighted, since introducing coding conventions as an afterthought can turn out to be impossible. Moreover, the source code should express *code of conduct*, which is a necessity for making the code public. Finally, *documentation* of the system is a practical necessity for attracting other developers.

Architecture. In order to make a software system easily approachable, it must be easy to understand by the developers. This in turn calls for an architecture that can be easily understood and communicated - and preferably documented. In addition to this, one should pay special attention to the design drivers of the architecture: Is the system designed as a monolithic system that solves a particular problem, or has the design taken into account extensibility, modifiability, and the use of the system as a subsystem in another system. Provided that the architecture has been designed with changes in mind, it is often easier for other developers to alter certain parts to create various types of new systems.

Quality attributes. In software, quality attributes are commonly associated with architectures. Indeed, many qualities, such as performance, scalability,

and memory footprint, are often dictated by the architecture. However, when considering a completed software system, quality properties are often considered separately from the actual implementation, which makes the perceived quality of software being released as open source an important factor.

Community In order to release a software system in open source, one generally has an idea on what kinds of developers should get involved. Careful planning of the intended community participants can have an impact on what to release and how.

Purpose. As already stated in [9], good work on software commonly starts by developers scratching their own itch. Therefore, we feel that in order to become attractive for developers, the released system should be of practical importance and relevant for the developers. This in turn enables one to solve their own problems, not further developing some random software for companies who seek profit in maintenance and creativity of others. Based on the above, the goal of the community is probably the most important single issue when releasing a piece of software as open source. Provided with a mission statement welcomed by developers, companies, and other organizations, a community can obtain support from numerous sources. The goal must be practical enough to be meaningful for the developers, as well as clear enough to manifest itself in the development. Unfortunately, estimating the attractiveness of a certain purpose is difficult, and therefore it is sometimes difficult to make assumptions in this respect. Moreover, since there commonly are numerous similar ongoing projects, the adequacy of the mission is only a prerequisite for a successful launch, not an automata for succeeding in community building.

User community. In addition to partners developing software, we feel that the potential for the user community is important. Based on recent findings, it seems that a community of 100 users can support one full-time developer. In contrast, provided with an active user community, development resources can be invested in actual development, and the user community can provide support for other activities, such as peer user guidance, documentation, and testing.

Partners. The definition of partners that join in the community can be straightforward. For instance, if a releasing company has been subcontracting from another company, the latter may be automatically involved in the newly formed community. Moreover, the use of subcontracting may also imply that at least some documentation exists, which in turn simplifies introducing the system to other partners. In general, getting partners involved in a community guides the authoring company towards processes that liberate the development from company specific tools and practices. In contrast, if the company that is about to release a piece of software as open source has no partners that would share the interest in the development, there should be a clear plan to motivate others to join in the development effort.

Legalities In the context of companies, one of the most commonly considered aspects of releasing software as open source is legalities. This is a wide topic to cover, and there can be several subtle differences in different contexts.

Here, we assume a straightforward view where different concerns are discussed independently.

Copyright and intellectual property rights (IPR). Most commonly, companies release software whose copyright and IPR they own. However, things can be more complex, if subcontractors or open source communities have provided pieces of the system that is being released. If the company does not own the copyright, it can sometimes be obtained via different transactions.

Licensing. Provided with the copyright of the system to be released, the company is in principle somewhat free to determine its licensing scheme. However, the choice of license (or licenses if several alternatives are offered) also has an effect on how others perceive the community. This in turn potentially affects the willingness of developers to participate in the community effort. Therefore, since licenses that have strong copyleft, such as GPL [6], can be considered safe for community building, they bear some advantage over other licenses in this respect. However, since licenses with no copyleft, such as MIT [6] and BSD [6], are favored due to their liberal flavor in some other contexts, one should also take the mission of the community into account when defining the license. Moreover, license compatibility with related systems should also be considered. Furthermore, one can even compose a list of accepted open source licenses, which can be used during the development.

Branding. Availability of brand names is an issue for any project looking for a good name that can be used in public. While issues such as trademarks may bear little significance when developing an in-house product, once the product is released, branding becomes an issue. Moreover, since an open source project can be a long lasting one, selecting suitable brand names is important. The same applies to hosting the project, since in many cases it would be practical to reflect the name of the project also in the domain.

Releasing Authority The final element we address in our framework is the releasing authority, most commonly a company in the scope of the framework, which is targeted to releasing in-house software as open source. However, also other parties can act as the releasing authority, including universities, non-profit organizations, and individuals.

Mindset, culture, and motivation. Sometimes the mindset of developers working in a company is somehow biased - either positively or negatively - towards open source development. This is particularly true when their pet project is about to be open sourced. In order to benefit from an open source community, the releasing authority should be mentally and culturally ready for dealing with developers outside the company under fair terms. The terms include equal access to code, similar guidelines and conventions, as well as mutual respect. We believe that the seeds of building such cooperative relation should somehow be sewn well before entering the pre-bazaar phase. Therefore, evaluating the readiness of the company to go for open source is fundamentally dependent on mindset, culture, and motivation.

Process, organization, and support. In order to gain benefits from open sourcing a system, the releasing authority should have a system in place that provides support for users and developers. This requires planning of a process that is to be followed, and putting the process in practice by the support organization. Establishing such support organization is a natural step to take towards the end of the pre-bazaar phase. However, it should be in place before the actual release, since support should be available from the very beginning.

Infrastructure. In order to establish an open source project, the releasing authority sometimes must be prepared to provide infrastructure. For instance, the company that releases a piece of software may provide web servers for hosting the system, as well as maintain a build system needed for compiling the code on top of certain reference hardware. While some systems do not need such support as such - it would be perfectly reasonable solution to release a vanilla Linux program in SourceForge - companies often wish to gain visibility through offering the download opportunity. Moreover, if a company is releasing a system targeted for the development of embedded systems, it is only reasonable to assume that also tools for composing builds are offered from the very beginning in open source.

The R3 evaluation model is organized into three main levels. For each dimension there are a number of categories. Each category is then associated with a number of measures (i.e. questions). This is illustrated in Figure 1 taking the software dimension as example.

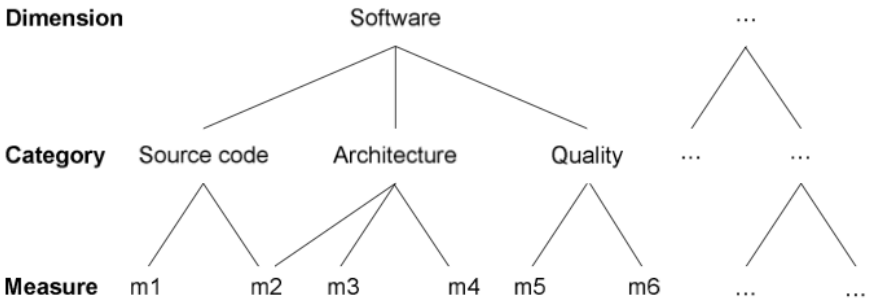


Fig. 1. The R3 framework model

3.3 Evaluation Process

The diversity of software products (and different goals of companies) makes it impossible to evaluate all software in similar fashion. The evaluation model itself must be tuned to take into account the characteristics of the product under release. Not all criteria make sense to all cases, and some crucial criteria

might be missing. The proposed R3 model should be considered as a template which has to be instantiated to each case. Instantiation R3 means going through all aspects of the model and validating that they are appropriate to the case in hand.

At the concrete level the evaluation process starts with downloading R3 Spread Sheet Template from `http://tutopen.cs.tut.fi/R3/R3_Template.xls`. Instantiating the template requires removing and adding dimensions and criteria to the spread sheet. Also the evaluation weights require attention from the evaluator.

On the Criteria The evaluation criteria form a continuum from a criterion that can stop the release process to others that can be easily fixed. Examples of the former are some legality issues, like possible copyright and IPR violations, and probably mindset of partners. Changing these is hard or even impossible. The latter group can be worked on during the releasing process: usability, and quality can be improved; infrastructure and process can be organized. An example continuum is depicted in Figure 2.

Fig. 2. The continuum of criteria

Actual Process The evaluation process consists of three phases depicted in Figure 3

Phase 1: Deciding the evaluation order. The evaluation is carried out according to continuum of criterion. This allows early no-go decisions. If, for example, the company does not own copyright of the product, it makes no sense to continue the process. However, there is no one unique and universal order of the criteria, but the order is fixed in the beginning of the process.

Phase 2: Data collection and processing. Most of the work is done in this phase. The criteria are evaluated one by one in the order fixed in the first phase. This activity is shaped by the framework model presented in Figure 1. First a dimension (e.g. software) is picked, then a specific category (e.g. source code) is selected, and finally concrete questions are answered. After each evaluation the decision of continuing the release process is made. Each measurement is filled to a standard spreadsheet with justification of the evaluation.

Measures require expertise from different fields: engineers, marketing people, legal experts and external open source experts. For example, the legal department of the company is often contacted in the beginning of the evaluation

to verify the copyright and possible IPR issues of the release. Engineers take care of technically-oriented criteria. External open source experts have probably the best understanding of the whole release process, and they know how open source communities operate. The releasing process reminds much of marketing challenges, therefore, marketing people are valuable for the process.

Phase 3: Data translation. In the data translation phase the evaluation of the criteria is transformed to an array of scalars with respect to the dimensions of the framework. The final result of the process is a four-dimensional array of evaluations of each dimension: software, community, legalities and releasing authority.

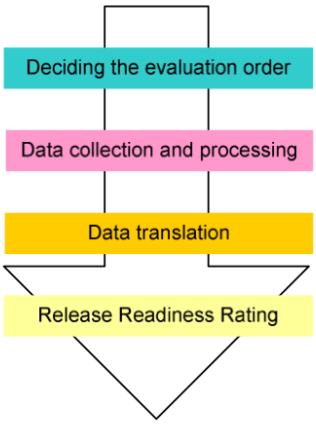


Fig. 3. Evaluation process

3.4 Open Source Engineering

R3 assessment of proprietary software is not just a pass-or-fail process. The outcome of R3 evaluation is a set of recommendations based on which the software under evaluation and its development environment undergoes an open source engineering process. This process needs to be carried out before evangelizing the software to the open source community. The aim is to eliminate the problems and shortcomings identified during the assessment process. This will increase the success rate of community building and sustaining. The open source engineering process itself is driven by different kinds of influential factors that follow the same criteria as we used in the R3 framework.

In the case of the *software* itself, any considerable rework requires an extensive investment. Therefore there are numerous restrictions on what can be accomplished during the pre-bazaar phase. Still, it is possible to clean up the code, if there are some company specific remarks in comments. Since the code

may already be in use in products, special attention must be paid to determine what to do with comments that indicate faulty or incomplete features. To some extend, documentation can also be composed in pre-bazaar phase, or simply included in the comments in the code.

Adding purpose to a *community* as an afterthought can be difficult. Assuming that a system has been developed with only business interests in mind, it can be difficult to introduce attractions for an independent developer. However, a mission for a community can be defined in pre-bazaar phase, provided that the software to be released enables a number of possible uses. Unfortunately companies can be somewhat biased towards supporting their own plans regarding the released system only, which in turn sometimes hinders the outside participation in the development for reaching some other goals, especially if the missions are conflicting. For instance, the releasing authority may not be willing to incorporate a community contribution for free, if the same feature can be sold for a commercial customer by the company.

Legalities most commonly form the most straightforward category of items. There is a lot of freedom to define the other legal aspects once the copyrights have been provided. However, copyrights can be difficult to obtain in pre-bazaar phase only.

The seeds of building cooperative relation between the *releasing authority* and the actual community should in our opinion be somehow sewn at the latest when entering the pre-bazaar phase. This can already be evidenced by existing ways of working and infrastructure, but they can also be introduced later on.

4 Case Study

In order to demonstrate our approach, we have applied the R3 framework to measure the open source readiness of an industrial software platforms: Wringer and Gurux. The Wringer software is a JavaScript binding platform for GNOME/GTK+ [4] using V8 [12] as JavaScript engine. It was originally developed by Sesca Mobile Oy. The Gurux software [5] is a platform for developing device communication systems.

4.1 The Wringer case

The R3 evaluation of the Wringer platform has been carried out separately by the releasing authority as an internal stakeholder and by us as external open source experts. It was observed that we agree on most answers. However there are still a number of differences. For instance, in the software dimension, there were few differences with respect to rating the technology used, the use of well-known design principles, rating of bugs and warnings, and scalability level. We received more pessimistic answers from the releasing authority. A partial reason for this could be that the company is assessing Wringer, which is a prototype software, relative to other high quality products developed inside the company.

On the opposite, the answers with respect to the community perspective for instance have been mostly identical. This probably shows that both parties are aware and honest about the community-related properties of the software. Also this shows that both parties are fairly aware of related user and developer communities.

Taking a numerical perspective, we noticed that the software and legality dimensions received the best scores compared to the community and releasing authority dimensions. This confirms our hypothesis that companies are generally dealing with open source from legality point of view. We could also infer that software-related properties are considered important irrespective of whether the software is supposed to be used and developed as closed source or as open source.

The low rating of the community dimension suggests that the company have to work on making the software more attractive to open source communities, involve business partners if possible, and look for potential users of the software. As for the releasing authority dimension, the low rating suggests that the company is not fully ready for open source operations. Concrete remedial actions include training internal developers, setting clear open source related processes, and building an infrastructure for the project. As mentioned earlier, these remedial actions are to be carried out in the context of an R3 evaluation post activity called the open source engineering process.

4.2 The Gurux case

Four people from the releasing authority carried out the R3 evaluation for the Gurux platform before it was released as open source in November 2009. The overall impression was positive and R3 was considered a valuable and useful tool in the pre-bazaar phase. The evaluation process acted as a good checklist for things that need to be considered when planning the release and showed well the items where most improvement is required. In addition, those items where improvement would be most beneficial were easily identified with R3, i.e. the releasing authority knew where to focus most of the effort.

Some items were not seen relevant in the case of the Gurux platform. However, this was not a significant problem for the process as irrelevant items were simply skipped by the people doing the evaluation.

Sometimes choosing the correct grading was found hard. Grades like "well" and "reasonably" may mean different things to different people as these types of grades depend on how things are seen by individuals doing the evaluation and what they value.

5 Future Work

We are in the process of improving the R3 framework based on our experiences with the case projects. This includes adjusting the weights, proposing new metrics, and covering other dimensions if found necessary. Currently the weights

are chosen based on experience gained forn earlier similar case studies. As the framework is applied to further cases, enough data will be gathered to enable us to better finetune the weights.

Furthermore, the case studies confirmed that it is difficult to come up with a one R3 framework template for all software projects. For instance, usability as a quality metric should be considered for end user software but was found less relevant in the case of software platforms like Wringer and Gurux. We plan to provide different templates for different kinds of software. Still, it is highly probable that the R3 framework template needs to be adapted to the needs of the subject software on a case by case basis.

6 Conclusions

Similarly to other major steps in software development, aiming at releasing a piece of in-house software as open source requires an engineering effort. Moreover, in order to estimate the outcome of the release, tools and techniques are needed for evaluating the potential of the emerging community as well as the attractiveness the system for external developers.

In this paper, we have introduced the concept of Release Readiness Rating Framework to determine how complete an in-house piece of software is for releasing in open source, and what its potential to attract external developers is - in essence evaluating how easily a cathedral could be transformed into a bazaar. We also discussed potential engineering actions that can be taken as a part of this transformation process, and provided a summary an industrial care we have studied.

References

1. BRR. `http://www.openbrr.org/`. Last visited March 2009.
2. Karl Fogel. *How to Run a Successful Free Software Project*. O'Reilly Media, Inc., Oct 2005.
3. Bernard Golden. *Succeeding with Open Source*. Addison-Wesley, 2004.
4. GTK+. `http://www.gtk.org/`. Last visited March 2009.
5. Gurux/open source. `http://www.gurux.fi/index.php?q=OpenSource`. Last visited December 2009.
6. Licences. `http://www.opensource.org/licenses`. Last visited February 2009.
7. Jenny Preece. *Online Communities: Designing Usability, Supporting Sociability*. Wiley, 2000.
8. QSOS. `http://www.qsos.org/`. Last visited March 2009.
9. Eric S. Raymond. *The Cathedral and the Bazaar*. O'Reilly Media, 1999.
10. Matthias Stürmer. Open source community building. licentiate thesis, 2005.
11. the Symbian Foundation. `http://www.symbian.org/`. Last visited December 2009.
12. V8 JavaScript Engine. `http://code.google.com/p/v8/`. Last visited February 2009.

Case II: Publication IV

Terhi Kilamo, Imed Hammouda, and Mohamed Amine Chatti. Teaching Collaborative Software Development: A Case Study. In *Proceedings of the 34th International Conference on Software Engineering (ICSE'12)*, pages 1165–1174. Zurich, Switzerland, June 2 – 9, 2012, Institute of Electrical and Electronics Engineers (IEEE).

IV

Teaching Collaborative Software Development: A Case Study

Terhi Kilamo, Imed Hammouda
Department of Software Systems
Tampere University of Technology
Tampere, Finland
firstname.lastname@tut.fi

Mohamed Amine Chatti
Informatik 9 (Learning Technologies)
RWTH Aachen University
Aachen, Germany
chatti@cs.rwth-aachen.de

***Abstract*—Software development is today done in teams of software developers who may be distributed all over the world. Software development has also become to contain more social aspects and the need for collaboration has become more evident. The importance of teaching development methods used in collaborative development is of importance, as skills beyond traditional software development are needed in this modern setting. A novel, student centric approach was tried out at Tampere University of Technology where a new environment called KommGame was introduced. This environment includes a reputation system to support the social aspect of the environment and thus supporting the learners collaboration with each other. In this paper, we present the KommGame environment and how it was applied on a course for practical results.**

***Keywords*-collaborative software development; SE education; case study**

I. Introduction

Software is created by people, with people and for people. These people may work in varying environments, may have their particular backgrounds and act under different conditions [3], yet sharing in many cases the same activities and interests. This is exhibited in recent software development settings such as pair programming, global software engineering and open source software, where individuals –from developers to users– join together to work on common software engineering activities. In many such settings, developers are expected to collaborate with other members with limited face-to-face contact. This leads to new kinds of software engineering problems including technical, organizational, and social challenges.

Understanding team work in collaborative software engineering is crucial to understanding how methods and tools are used, and thereby improving the creation and maintenance of software systems as well as the management of software projects. A natural starting point would be to introduce those concepts to software engineering graduates. In this regard, a number of approaches have been proposed [8], [14], [16], [25]. However, many software engineering curricula still lack education on collaborative software development. In particular, the software engineering education community is continuously seeking innovative ideas, effective tools, and valuable experience.

Driven by social learning theories, this paper introduces an approach to teaching collaborative software development where students collaborate collectively to achieve a common goal. The approach is based on adapting reputation systems [11] and using them to compute and publish members contribution in a team. The approach allows students to generate new knowledge through interaction with the teams past experience and contribution with new ideas. The idea is also inspired by the experiences of using reputation systems to reward and recognize developers in open source communities such as Qt [23].

We present an example reputation model and a concrete reputation environment known as KommGame that mimics real team projects. The environment has successfully been tested at Tampere University of Technology (TUT) to introduce community driven development to software engineering students.

The remainder of this paper is structured as follows: Section 2 gives the background of our work, including social learning theories, personal learning environments, teaching collaborative software development and reputation systems. Section 3 presents an example reputation model and environment for collaborative software development trams. Section 4 presents a concrete course setting applying the proposed approach. An evaluation of the course and the underlying approach is presented in Section 5. Finally we conclude in Section 6.

II. Background

In modern pedagogical approaches, the learner plays a central role. Learning is seen as an active effort of the learner with intrinsic motivation towards learning and it builds new knowledge based on the learner's previous experience. The role of the instructor is to assist the learner throughout the learning process by applying the right teaching methods and by providing a suitable learning environment. In collaborative software development, the software product gets created by a team, or a community, of developers, who keep in

close communication throughout development. Collaborative software development is utilized within companies both in-house and to run multi-site development. Open source software is also developed through collaboration of globally distributed developers. As learning is a social process as well as an individual effort, teaching collaborative software development benefits from adding social aspects into teaching methods to reach the best learning outcome.

A. Learning is Social

Learning is social in nature, as has been emphasized by many researchers. Lave and Wenger [18]., for instance, introduce communities of practice (CoP) as ideal vehicles for leveraging tacit knowledge and learning and explore the participation metaphor of learning. More recent research also view learning as a social process. Most of the computer-supported collaborative learning (CSCL) literature relies on the socio-cultural theory of learning, such as social constructivism [22], [32] and activity theory [32], [10]. Recently, Siemens [26] stresses that knowledge resides in networks and points out that the challenge today is not what you know but who you know, and introduces connectivism as a new learning theory, which presents learning as a connection/network-forming process. Chatti et al. [5] discuss the Learning as a Network (LaaN) theory, which puts the learner at the center and represents a knowledge ecological approach to learning. LaaN starts from the individual learner and views learning, for an individual, as an issue of continuously building, maintaining, and extending her personal knowledge network (PKN). A PKN is comprised of a myriad of explicit knowledge nodes (i.e. information) and tacit knowledge nodes (i.e. people) with complex connections. In order to learn, we build, maintain, and extend our PKN with new explicit/tacit knowledge nodes and when needed we activate the nodes that we believe are able to help us in mastering a learning situation. The process of developing a PKN is driven by the learning demands of the individual learner. In LaaN, participation suggests permanent listening, active networking, and genuine knowledge sharing with others, thus enabling each participant to build and extend her PKN, and in so doing learn.

Recognizing the social aspect of learning, learning initiatives need to place a strong emphasis on knowledge networking and community building to leverage, create, sustain and share knowledge in a collaborative way, through participation, dialogue, discussion, observation and imitation.

B. Personal Learning Environments

Learning is self-directed in nature. Among others, Hase and Kenyon [31] argue that this rapid rate of change suggests that we should now be looking at a learning approach where it is the learner who determines what and how learning should take place, and point out that self-organized learning may well provide the optimal approach to learning in the twenty-first century. In recent years, self-organized learning is increasingly supported by personal learning environments (PLE), where the learner is in control of her own development and learning. The PLE concept translates the principles of self-organized learning into actual practice. From a pedagogical viewpoint, a PLE-driven approach puts the learner at the center and gives them control over the learning experience [4].

C. Reputation Systems in Teaching

Reputation systems are used to measure the contribution of individuals in many online communities. They are also applied in versatile other fields such as e-commerce, search engines and social news. As reputation systems are applied for measuring online activities one can argue that they can be applied in the e-learning educational context where most of the activities happen online in order to support the learning outcomes and activities. Students today are already children of the web 2.0 era and thus well accustomed to existing reputation models. Furthermore, those who are active in online communities on their free time are used to gaining reputation through their activities.

The reputation for any entity, such as a person, an activity or a product, can be decided in several ways: by giving feedback to the entity, by rating the entity with a points system, by marking the entity as a favorite or by voting for the entity. All of the models mentioned above are simple, but simple does not mean less powerful, less useful or that they do not produce desirable results. A more versatile reputation model suited for a more complex setting such as a collaborative software development environment can be built by creating an aggregation or a fitting combination of the more simple models. In [11] Farmer discussed different reputation models, which are listed in Table I with a description of their reputation values.

Table I
LIST OF REPUTATION MODELS

Model	**Reputation value**
Rating	The average of all ratings that the entity has been given.
Favorites and Flags	The number of times the entity has been marked.
This-or-That Voting	The number of votes to the entity. Voting for a particular entity within a bounded set of possibilities.
Points	The sum of points for different actions an active entity has engaged in.
Reviews	The number of normal ratings or freeform text comments.
Karma	The aggregate of all online activities of interest.

In a *rating model* the entity is rated by giving it a value from a given range. There are different types of rating models in use, e.g. a points scale rating, a star rating, or the so called HotOrNot rating where only two rating values are used. Through rating it is quick and easy for the users to provide reputation feedback, for example www.imdb.com provides this kind of a model for users to rate movies.

The *favorites and flags* model gives control to the community for identifying entities of exceptionally high or low quality. There are three variants of the model: vote to promote, favorite, and report abuse. Vote to promote model is for users to vote for the supporting a particular item in a pool of choices for it to gain a step up. A favorite model keeps track of the number of times a particular entity has been bookmarked as someones favorite, thus increasing the reputation of that entity. Report Abuse is a negative reputation model. This model is used to avoid bad content or to identify it so it can be removed if necessary. In this model total reputation is the number of times particular item is flagged as abuse and the higher the reputation gets, the more likely the item is abusive.

The *this-or-that voting* model is used when the best option within a bounded set of options available needs to be chosen. An example of this type of model is when someone answers a question, the other users can vote for the answer as helpful or useful. As the votes accumulate the option most perceive as useful rises above the others.

A *points* model is used when a very specific value of user activity is desired. When a user is engaged in various actions, these actions are recorded, weighted and summed to calculate a value representing the total reputation. The reviews model is also a combination of rating models or freeform text comments on a particular entity. For example in www.amazon.com users can write review comments about the product they bought. The reputation increases or decreases based on the review comments.

The term karma is used to refer to the results humans get due of their past actions. A *karma* model of reputation is used in particular when the entity to which is the reputation is subjected to is human. There are two primitive forms of karma: participatory and quality. Participatory karma is used on representing the amount of contribution the user has made. Quality karma on the other hand focuses to represent the quality of user contribution over simply the amount of it.

D. Related Work

Collaboration has been used in teaching different aspects of software development to university students. In [31] requirements analysis is taught by using collaborative students teams. Collaborative software development itself is in focus in [12] where students from different countries collaborate on a software learning project. Similarly in [3] a multidisciplinary environment ranging several universities was tried out to create a more realistic setting for learning collaborative development. Teaching collaborative aspects of software engineering is also reported in [16], [14] and [25]. In [8] a tools perspective is taken.

While global software development is not necessarily always collaborative in nature, multi-site projects can also work on a joint software project in collaboration with each other. In [7] the authors describe an environment to teach practical software engineering in a setting mimicking a global multi-site project and with a collaborative approach. Similar endeavors that address the global aspects of software development are reported in [15] and [7].

As open source software (OSS) had gained importance also a software business model, the need for OSS professionals has become apparent. The collaborative nature and the key principles of open source development are discussed in [24]. In the recent past collaborative software development has already been taught from the OSS perspective [19], [20], [4]. Also, some universities have tried out teaching software engineering by using OSS as a collaborative learning environment [27] [13], to help students learn basic and advanced software engineering topics. In [2] open source evolution is taught in software engineering courses to give students a more realistic experience. On a similar note Google and The Finnish Center for Open Source Solutions (COSS) have conducted summer code camps to encourage students collaboration and participation in real life open source project. There are some communities like Teaching Open Source [29] and OpenSE [21] dedicated for promoting and researching open, online and collaborative learning.

Reputation systems have been a popular method of motivating the participants of online communities. Even though only a few OSS communities have adopted the reputation systems for motivating developers, it is discussed in [30] that reputation systems suit small groups of young participants; they have a high competitive sprit, which makes learning more active and motivated which supports the suitability to teaching environments. A Reputation system can be applied in the CoP [6] context. In [33] it is showed that reputation systems are applicable in the context of e-learning. A web based reputation system called SocialX [28] was designed to support collaboration and social aspects of e-learning. SocialX lets the students to sharing and exchange of solutions, to discuss solutions of exercises through a forum, and to participate in project activities. Combined with our findings these support the use of reputation systems in educational context in combination of a collaborative development environment built for educational purposes.

III. Reputation Environment for Teaching Collaborative Software Development

In collaborative software development the software developers can be widely distributed over sites separated by geographical distance and time zones. The consequence of

this is that most of activities and communication are carried out online. Hence, collaborative development requires an infrastructure where the developers can perform their project related activities. For example such an infrastructure can include a bug tracking system, a wiki system, a discussion forum, a version controlling system, an IRC channel and mailing lists. The most common activities in software development relate to bugs, features, improvements, and discussions on development details between developers via different methods and among variant groups of people.

A reputation model for online software development education that incorporates collaborative aspects of the development work should be designed so that most common activities are taken in to consideration in order to give the learners a realistic experience and thus promoting apprehension of key skills. From a developer community and software development perspective all kinds of contributions are of importance and there is no one good metric with which to compare or quantify different types of contributions over each other. In an educational context, however, the course moderator or tutor can decide which types of contributions promote key skills and should thus be emphasized. Taking this viewpoint, we can design a reputation model which supports learning the right skills and in the same time supports collaboration.

We argue that the karma reputation model fits well with the activities and the nature of collaborative software development to enforce social learning and uphold motivation among students in an online learning environment at the same time. Both forms of the karma model, i.e. participatory and quality, can be used to measure both the amount and the quality of contributions.

Table II
ACTIVITIES THAT CAN CONTRIBUTE TO PARTICIPATORY KARMA

Category	**Activity**
Bug	Reporting new bug
	Commenting on bugs
	Closing bugs
Feature	Request new feature
	Commenting on feature request
	Closing new feature
Improvement	Request Improvement
	Commenting on Improvement
	Closing Improvement
Wiki pages	Creating/Editing wiki pages
Code repository	Apply a patch
	Add a new code file
	Removing a code file
Forum	Starting new discussion
	Commenting discussion
IRC	Communicating through IRC
Mailing List	Send email to mailing list

A. Concrete Quantities of Karma

As said, all kinds of online activities can contribute to the reputation of an entity. The activities that were considered for the participatory karma values in a software development environment are listed in Table II.

In practice all these activities need to be evaluated and a single karma value be formed. Let us take a code bug handling process as an example. In the process of contributing to the development of a piece of software anyone may find a bug in the software. Then the bug is reported vie a bug tracking system used. Other developers verify, comment on and discuss the bug and eventually someone writes the code to fix the bug and the bug report gets closed. These all add up to the karma values of the contributor. The feature requests, and the requests for improvements, are handled in the same manner. Essentially all developer activities from adding or editing open content on the wiki pages or participation to discussions with other developers via the available channels to adding code or applying patches to the source code in the code repository, contribute to the developers participatory part of the karma value. The developers can also grant quality karma to others in the model by liking their activities. The number of these bookmarks, likes, will then contribute to the quality part of the karma value of the author of the wiki page. At regular intervals of time, a best quality contributor can be selected based on their activity, or quality of their activity, or be given a hat. The hat is considered as token for best quality contribution. The number of hats any contributor receives contributes to their quality karma also.

B. Karma Model for Teaching

The karma model we suggest is composed of the concrete contributions and the karma attributes given in Section A. Each contribution is given a particular weight relative to their size or importance to the developer community. As in our case the model is applied in an educational setting with pedagogical goals the values are chosen by the course moderator so that the highest weights support the key learning outcomes. The final karma value of each learner is the sum of weight times of each contribution. The universal karma model can be written as

$$Karma = \sum_{k=1}^{n}(f_k(Contribution) + f(Favorites) + g(WeeklyQualityTokens) \quad (1)$$

Here n corresponds to the total number of contributions. *fk* is the weight function corresponding to contribution type. Favorites is the number of like bookmarks a content author gets. Weekly Quality Tokens corresponds to the number of time the particular participant was selected as the best quality contributor of the week by the rest of the members of community. The karma model in Equation 1 is composed of two types of karma, participatory karma and quality karma.

The sum of all contributions gives the participatory karma. The quality karma is composed of two parts, favorites, and weekly quality tokens.

For example, a sample karma equation, which covers activities related to bugs, features, improvements and wiki edits, is given in Equation 2. In the formula each activity is multiplied with its associated weight. Total karma is the sum of all karma values gained from each activity. The weights were set based on the relative importance of the contributions to the example project. However, what we used here is just a framework that could be adapted to different project and course contexts.

$$\begin{aligned} Karma = 6 * \sqrt{nr.\ of\ bugs\ reported} + \\ 2 * (nr.\ of\ bugs\ closed + 4 * \sqrt{nr.\ of\ feature\ reqs.} + \\ 4 * \sqrt{nr.\ of\ bug\ comments} + 2 * (nr.\ of\ closed\ new\ features) + \\ 4 * \sqrt{nr.\ of\ requests} + 3 * \sqrt{nr.\ of\ improvements\ comments} + \\ 2 * \sqrt{nr.\ of\ closed\ improvements} + 4 * \sqrt{nr.\ of\ edits} + \\ 4 * \sqrt{nr.\ of\ likes} + 4 * \sqrt{nr.\ of\ weekly\ qual.\ tokens} \end{aligned} \quad (2)$$

There is mapping between the first and the second equation The sum of all the contributions that belongs to bugs, features, improvements and wiki edits is the participatory part of total karma, this corresponds to *fk(contributions)* in Eq. 1 and quality karma is calculated from the number of likes. This corresponds to *f(Favorites)* in Eq. 1 and lastly the number of weekly quality tokens corresponds to *g(WeeklyQualityTokens)*. The weight function of each contribution is chosen based on the priority of contribution, in this example bug reports is given the highest priority.

C. KommGame Environment

We have developed an OSS learning environment based on the reputation model presented earlier. The learning environment, called KommGame [17], maintains karma values to support collaboration through gaining karma via collaborative actions. The gained reputation acts also as a motivational factor for the learners as the higher karma values get more visibility. Healthy, positive competition between learners can support learning. The KommGame environment forms an infrastructure required for collaborative and student centric learning.

The KommGame infrastructure has been developed to mimic the infrastructure of a real online software development environment. It has features to add and edit open content, a user management system to manage users of the community, a system to track user activities, a communications channel, a bug management system, a source code base to maintain source code of the project, a reputation system to calculate the karma of each community member and an user interface to publish karma values.

From the architectural point of view the learning environment can be divided into the following modules:

- Content management module
- Blogging module
- Bug reputation system
- Karma Engine
- User management module
- Version control system

All the modules share a common learner database. The Karma module interacts with all other modules to collect the learner contribution information and calculates the user karma based on learner contribution and makes the karma value publicly visible to every other learner.

IV. Case Study: A TUT Sofware Engineering Course

A course on OSS was organized at Tampere University of Technology for the second time during the academic year 2009-2010. KommGame with an open source development community infrastructure was setup for the course and a the reputation model was set to match the educational goals. The main goal of the course was to give a practical experience of OSS development in an actual open source infrastructure, i.e. give the learners the possibility to learn collaborative software development, in this case in the context of open course, in a realistic, but still educational setting.

A. Course Setup

The duration of the course was 16 weeks and its extent was 3 to 6 credits depending on learner performance. Every week students participated in a two hour classroom session. The first part of the session was dedicated to discussing a specific open source topic. The second part is to discuss student contributions to an example open source project. Both parts use the KommGame environment to record student contributions. Attendance was not mandatory. However, participating in the example project and contributing to OSS topics was an absolute requirement to pass the course. The course had two main coordinators and two TAs, who were responsible for the KommGame environment and the community, for the course.

A total number of 24 students registered to the course. Initially two students who had the prior knowledge of open source development and infrastructure were assigned the committer role and the rest of the students played the role of developers, testers, and users. The KommGame environment was introduced to the students during the first class session. As the environment is accessible from the Internet, like in real open source projects, students were allowed to contribute to the project at any time they wanted to do so.

B. Course Activities

Course activities were divided in to three parts. In the first part every student had to select one topic from a given set of topics related to OSS. Then each student contributed with their own part of content to the wiki so that information could be accessed by everyone and would be available also for the upcoming instances of the course. Every week a

number of students presented their topic to the rest of the class. During the presentation, another group of students acted as topic discussants providing peer review on the content.

he second part was to participate in the development of an example open source project called aKro. Initially aKro was developed with very basic features and made available in the version control system. In this exercise students were asked to add new features, fix existing bugs and maintain the project wiki. Whenever students observed a bug, improvement, or a new feature request they reported those issues on the bug tracking system. Any student interested in the reported issue could take up the report and submit a corresponding patch file. Once the patch file was submitted committers would test the patched version and apply the patch. During the learning process students used IRC as a communication channel. Every student maintained their own blog page where they reported their activities in the project. Every week in the classroom the course coordinators inspected the contributions of the past week and rewarded the student with the best quality contribution.

The karma model used in the course covers the following activities:

- Creating or editing Wiki pages.
- Reporting bugs, new feature and improvement requests.
- Comment on bugs, new feature and improvement requests.
- Closing bugs, new feature and improvement requests.
- Thumbs up to wiki content page.
- Adding the hats for best quality contributor of the week.

The third part of the course was to contribute to a real open source project after getting familiar with the OSS principles and practices covered by the first two course parts. As possible open source projects, students were introduced to the Apache Software Foundation programme [1] and the Demola open innovation platform [9]. However, they were free to choose whichever project they liked.

C. Example Scenarios

Learners can do different activities in the KommGame infrastructure in order to contribute to project and open content. Some example scenarios of activities that users can do with the infrastructure include accessing the environment, contributing open content, viewing the karma report and viewing individuals profile. Details of each are discussed in the following.

1) Accessing the KommGame Environment: The KommGame environment is accessible from the Internet at [17] by anyone interested. The user has to provide login credentials in order to access the environment. Once the user is logged in the user interface looks like the one shown in Figure 1. Different parts of KommGame are accessible through links that also are visible in Figure 1. The View issues link gives the view to all reported. The Report Issue link is used to report a new issue as the name suggests. The FLOSSpedia link accesses the open content generated by participants. Through the Project wiki the wiki content created by the participants can be accessed. Finally the Karma Ranks link allows the karma value of all the students to be investigated.

2) Adding Content: The system used to share the open content with community is built as a wiki. Every participant can add open content to the wiki system by providing the authentication credentials. Figure 2 shows a wiki edit feature where the user is adding content to the wiki. Once the content edited is saved, any other user can start editing the page. If any of the participants likes the content of the wiki page, they can bookmark the page as a favorite. When any of the wiki pages is marked as a favorite then the author of the page gains quality karma.

3) Viewing the Karma Report: All participants can view the karma values of all the participants in a graph using the Karma reporting system. Figure 3 shows a sample view of the karma graph. The names in Figure 3 are blurred out to protect user privacy. The graph shows the different categories of users, which are reported in different colors. Each vertical bar in the graph represents the score of each of the users in the system. Each vertical bar has two parts with different colors; the bottom part indicates the score of the previous week while the upper part indicates the score of the current week. The hat like icon on some of the bars indicates that those users were the best contributors for some week in the past. From the karma report there are links to access the user

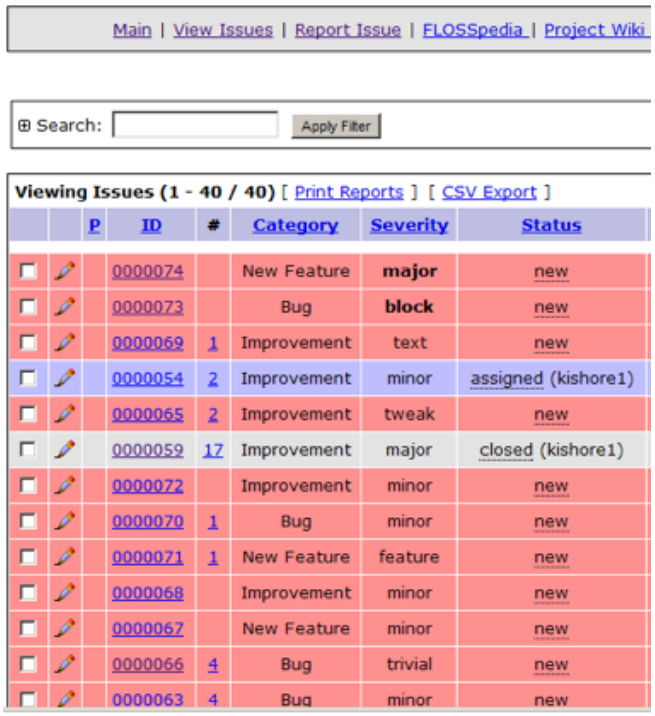


Figure 1. System user interface

profile, which provides the details of the users contributions and is discussed in next section.

4) Viewing Individuals Profile: All participants can have a detailed view of the activities done by a participant by using the user profile system. Figure 4 shows a sample view of the user profile.

The left side table shows numerical figures on the participants contribution, on the right side we can see the blog that is maintained by the participant about his work. This view is publicly available to anyone. When the course coordinator logs in they can see two more buttons Add a hat and Remove a hat. The notion of a hat is used as a token for the quality contributions here. If the current user is the best quality contributor of the week then he can be given a hat through the interface. There is a remove button to eliminate human errors.

V. Discussion

Collaboration is an important aspect in modern software development. Reputation systems are is use in many online environments to enforce the social aspects of the environment and give the users the feel of belonging to a group. Focusing on the course case, we set out to teach collaborative software development with an online environment incorporating a reputation system. Our focus on the course was in teaching open source software but aspects of collaborative development in general are covered. Neither the environment nor the learning outcomes are limited to the OSS context.

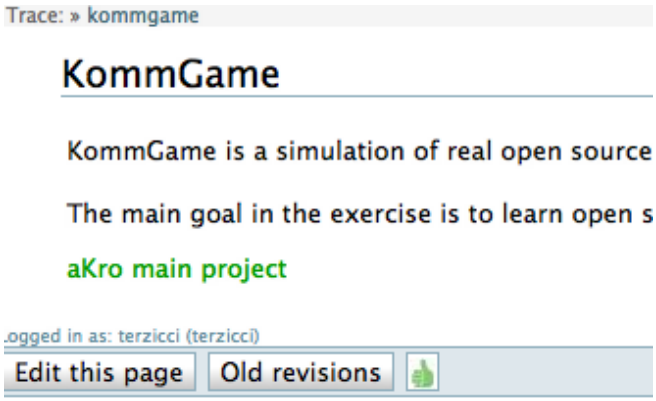


Figure 2. Add content to wiki and favorites bookmark

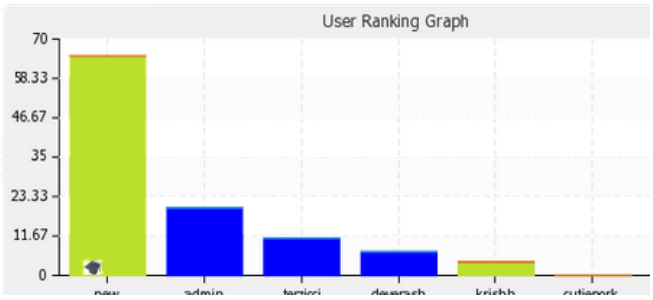


Figure 3. Karma reporting interface

Out of the 24 registered students 15 participated actively on the course. The details of student participation are shown in Table III.

During the first part of the course where students were to create wiki content on the chosen topics, participants got to know KommGame environment. Working on the wiki also acted as an introduction to working online and the need for collaboration. In the second part of the course, the students were to actually develop a small software project called aKro. During the project it took some time for the students to understand the key concepts: bugs, features, requests etc. in the online collaboration context. Throughout both phases the emphasis was on social learning, peer support and learning by doing. The course coordinators were not the ones teaching but the learners collaborated together in reaching the learning outcomes.

Initially there was a slow start in the project as students were not used to, not only collaborative and OSS development, but also to a social learning course setting. As the project progressed there were more contributions from the students and the online work was more active. This trend can be observed from the statistics. There were totally 63 issues reported in the bug tracking system, on average each student posted 4 issues. In the first week there were only 3 issues reported and the second week 12 issues were reported. This count kept fluctuating with the range from 3 issues upwards. While the overall number of unique issues kept rather steady, the increased activity can be seen in total numbers of issues reported. The activity statistics per week are shown in Tables IV and V and in Figure 5. While observing all the contributions we found that there were 6 students who were more active than rest of the community.

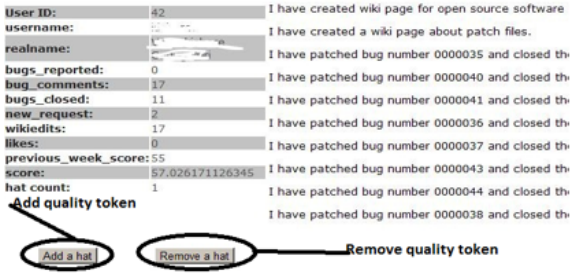


Figure 4. Detailed user profile interface

Table III
Details about number of students participated

Students	Total Number
Registered	24
hline Participated	15
hline Active	6

This kind of scenario is also very common in real open source project where very few members of the project create 90% of the code. On an average each student contributed 40 wiki edits and what shows the collaborative aspect the best on the 7 bugs reported a total of 97 comments were made.

Initially, it was planned that all the communication during the development process happens online, but in some situations there was verbal communication between groups of students. This verbal communication aroused when students had to make a critical decision about the project. This kind of physical interaction is also common in real world software projects when handling the most important decisions. As the project development was progressing, the amount of students contribution also kept increasing. All the students played the key roles: user, developer, and bug fixer and reporter. Most often the person who reported them also fixed the bug, with only a few cases where someone else fixed the reported bug. When the participants were asked for the reasons why, they answered that coding is fun.

Every week, after the seminar session the course coordinators along with the participants verified the contributions made to the project to decide who would be the best quality contributor of the week and thus to add a hat in their profile. The top contributor of the week is given a small gift to motivate others. This practice of verifying all users contribution and select the best quality and top contributors appeared to motive the others to work. It also enforced collaboration and gave a sense of a community to the course. Every week a new student was rewarded.

Table IV
TOTAL NUMBER OF DIFFERENT ACTIVITIES DONE BY STUDENTS

Different Activities	Total Number
Bugs Reported	7
Bug comments	97
Comments on issues	40
Bugs closed	2
New features title	10
New features closed	4
Improvements	23
Improvements closed	12
Wiki edits	40 (Average)
Likes	9
Hats	7

Table V
NUMBER OF DIFFERENT AND TOTAL ISSUES CARRIED ON

Number Of Week	Different Issues	Total Issues
First	3	4
Second	12	25
Third	2	6
Fourth	2	7
Fifth	3	5
Sixth	8	20
Seventh	3	5
Eighth	3	6
Ninth	4	13
Tenth	4	6

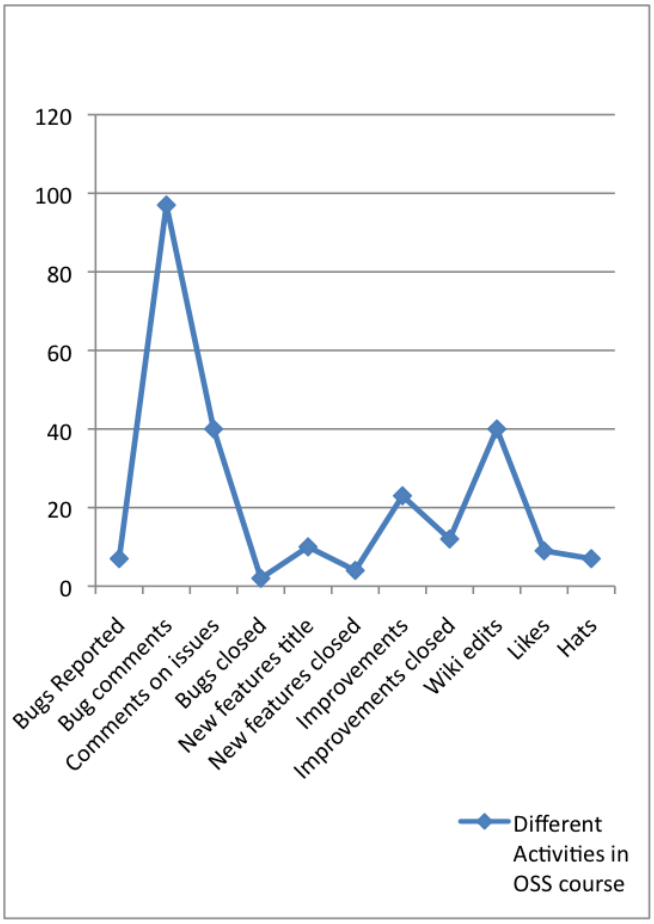


Figure 5. Activities completed

The weekly activity over the week KommGame was running is shown in Figure 6. There are two significant slumps in weekly student activity. The drop in initial enthusiasm explains the first slump. Furthermore, it occurs when the students needed to make an adjustment to their learning practices as they had to put in a more continuous work effort to what the students were accustomed to be required. The second is due to a break in the teaching for an examination period. The last peak can be contributed to the course coming to an end and thus participants working hard to close the loose threads.

The student feedback after the course supports the notion that the participants feel they reached the intended learning outcomes and found the KommGame to have given them a sufficient head start to collaborative development methods.

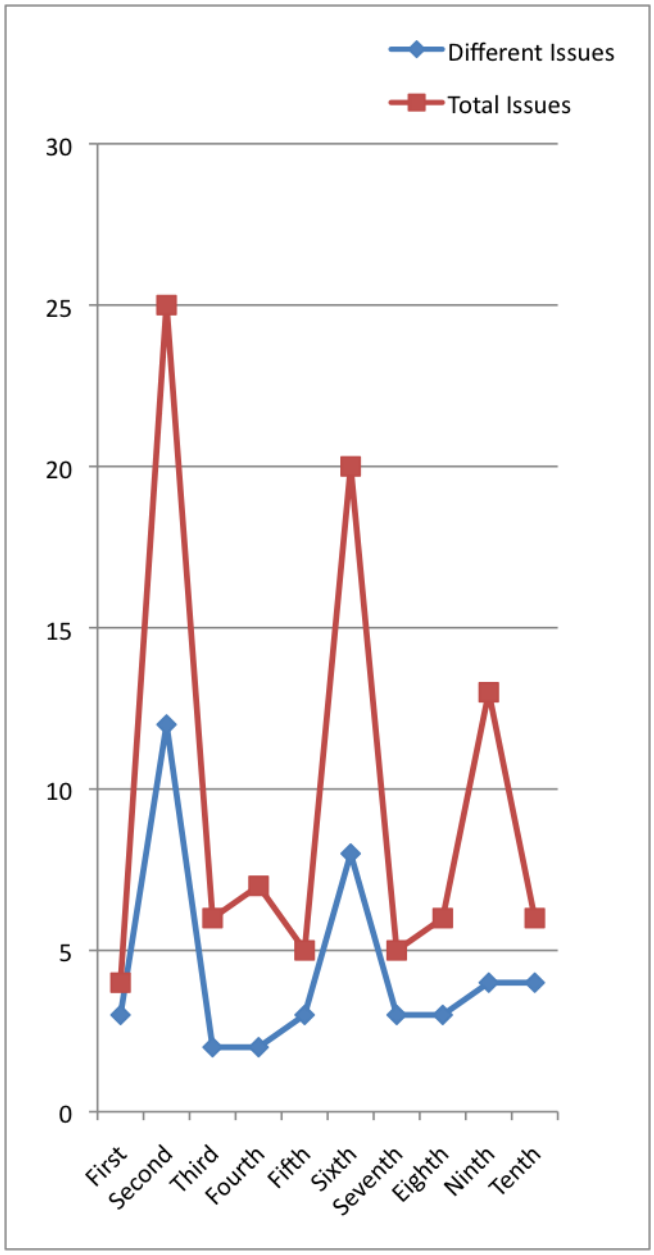


Figure 6. Activity per week

A. Conclusions

The approach of KommGame for collaborative software development education allows students to practice development details and characteristics in a realistic yet safe setting. The KommGame enforces collaboration through the reputation aspects and thus gives the participants valuable software project experience. The students are free to learn from their own past and present experience as well as the others. Based on the experiences gathered by the pilot course in teaching OSS development we consider that using the KommGame setting for collaborative development education adds to the learning experience and paves the way for working as a developer in the modern software industry.

The future plans for KommGame are to research how this can be applied in traditional programming courses where students have to collaborate and participate in programming exercises. Research should also be done to know how to use KommGame as standard system to issue certificate of OSS beginner to any participant around the world who participate and contribute.

References

[1] The Apache Software Foundation. http://www.apache.org/foundation/. last visited on March, 2011.

[2] J. Buchta, M. Petrenko, D. Poshyvanyk, and V. Rajlich. Teaching evolution of open-source projects in software engineering courses. In *Proceedings of 22nd IEEE International Conference on Software Maintenance (ICSM'06)*, pages 136–144, 2006.

[3] L. J. Burnell, J. W. Priest, and J. B. Durrett. Teaching Distributed Multidisciplinary Software Development. *IEEE Software*, 19(5):86–93, 2002.

[4] M. A. Chatti, M. R: Agustiawan, M. Jarke, and M. Specht. Toward a personal learning environment framework. *International Journal of Virtual and Personal Learning Environments*, 1(4):71–82, 2010.

[5] M. A. Chatti, M. Jarke, and M. Specht. The 3p learning model. *Journal of Educational Technology & Society*, 13(4):74–85, 2010.

[6] C. C. P. Cruz, M. T. A. Gouvêa, C. L. R Motta, and F. M. Santoro. Towards reputation systems applied to communities of practice. In *Proceedings of 11th International Conference on Computer Supported Cooperative Work in Design*, pages 74–79, 2007.

[7] D. Damian, A. Hadwin, and B. Al-Ani. Instructional design and assessment strategies for teaching global software development: a framework. In *Proceedings of the 28th international conference on Software engineering (ICSE '06)*. ACM, 2006.

[8] J. DeFranco-Tommarello and F. P. Deek. Collaborative software development: A discussion of problem solving models and groupware technologies. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*. IEEE, 2002.

[9] Demola: Open innovation platform for students and companies. http://www.demola.fi/what-demola-new-factory. last visited on March, 2011.

[10] Y. Engeström. *Learning by Expanding: An Activity - Theoretical Approach to Developmental Research*. Helsinki: Orienta-Konsultit. Retrieved from http://lchc.ucsd.edu/MCA/Paper/Engestrom/expanding/toc.htm.

[11] R. Farmer and B. Glass. *Building web based reputation systems*, page 72. O'Reilly Media / Yahoo Press, 2010.

[12] J. Favela and F. Pena-Mora. An experience in collaborative software engineering education. *IEE Software*, 18(2):47–53, 2001.

[13] M. D. German. Experience teaching a graduate course in open source software engineering. In *Proceedings of the First International Conference on Open Source Systems (OSS 2005)*, pages 326–328, July 11-15 2005.

[14] F. D. Giraldo, C. A. Collazos, S. F. Ochoa, S. Zapata, and G.T. de Clunie. Teaching software engineering from a collaborative perspective: Some latin-american experiences. In *Workshop on Database and Expert Systems Applications (DEXA)*, pages 97–101, 2010.

[15] O. Gotel, C. Scharff, and S. Seng. Preparing computer science students for global software development. In *36th annual Frontiers in Education Conference*. IEEE, 2006.

[16] I. Hadar, S. Sherman, and O. Hazzan. Learning human aspects of collaborative software development. *Journal of Information Systems Education*, 2008.

[17] OSS Learning environment at Tampere University of Technology. http://osscourse.cs.tut.fi/mantis/login_page.php. last visited on March, 2011.

[18] J. Lave and E. Wenger. *Situated Learning: Legitimate Peripheral Participation*. Cambridge University Press, 1991.

[19] B. Lundell, A. Persson, and B. Lings. Learning through practical involvement in the oss ecosystem: Experiences from a masters assignment. *Open Source Development, Adoption and Innovation*, 234:289294, 2007. IFIP International Federation for Information Processing, Springer.

[20] D. Megas, J. Serra, and R. Macau. An international master programme in free software in the european higher education space. In *Proceedings of the First International Conference on Open Source Systems*, page 349352, July 2005.

[21] Open software engineering. http://opense.net. last visited 20 March 2011.

[22] J. Piaget. *The Childs Conception of the World*. Rowman and Allenheld, New York, 1960.

[23] Qt developers network reputation system. http://developer.qt.nokia.com/ranks, last visited on March 2011.

[24] E. S. Raymond. *The Cathedral and the Bazaar*. OReilly Media, 1999.

[25] C. Y. Shim, M. Choi, and J. Y. Kim. Promoting collaborative learning in software engineering by adapting the pbl strategy. *World Academy of Science, Engineering and Technology*, 53, 2009.

[26] G. Siemens. Connectivism: A learning theory for the digital age. *International Journal of Instructional Technology and Distance Learning*, 2(1), 2005. Retrieved from http://www.itdl.org/Journal/Jan_05/article01.htm.

[27] I. Stamelos. Teaching software engineering with free/libre open source projects. *IJOSSP*, 1(1):7290, 2009.

[28] A. Sterbini and M. Temperini. Social exchange and collaboration in a reputation-based educational system. In *Proceedings of 9th International Conference of Information Technology Based Higher Education and Training (ITHET)*, pages 201–207, April 29 -May 1 2010.

[29] Online community for open source software educaton. http://www.teachingopensource.org, last visited March 2011.

[30] M. Temperini and A. Sterbini. Learning from peers, motivating students through reputation systems. In *International Symposium on Applications and the Internet*, pages 305–308, 2008.

[31] J. Tuya and J. Garca-Fanjul. Teaching requirements analysis by means of student collaboration. In *the 29th ASEE/IEEE Conference in Education*, 1999.

[32] L. S. Vygotsky. *Mind in Society: The Development of Higher Psychological Processes*. Harvard University Press, 1978.

[33] S. Wang. Study on e-learning system reputation service, wireless communications, networking and mobile computing. In *WiCOM '08*, pages 1–4, October 12-14 2008.

Case II: Publication V

Terhi Kilamo, Imed Hammouda, Ville Kairamo, Petri Räsänen, and Jukka P. Saarinen. Applying Open Source Practices and Principles in Open Innovation: the Case of Demola Platform. In *Proceedings of the 7th International IFIP WG 2.13 Conference on Open Source Systems (OSS'11)*, pages 307–311, Salvador, Brazil, October 6 – 7, 2011, Springer.

V

Applying Open Source Practices and Principles in Open Innovation: the Case of the Demola Platform

Terhi Kilamo[1], Imed Hammouda[1], Ville Kairamo[2], Petri Räsänen[2], and Jukka P. Saarinen[3]

[1] Tampere University of Technology firstname.lastname@tut.fi
[2] Uusi Tehdas/New Factory firstname.lastname@hermia.fi
[3] Nokia Research Center jukka.p.saarinen@nokia.com

Abstract. In numerous fields, businesses have to rely on rapid development and release cycles. Variant new ideas and concepts can emerge through open innovation as the participants are not limited to the company scope. This makes open innovation an increasingly appealing option for the industry. One such open innovation platform, Demola, allows university students to work on real life industrial cases of their own interest. We have identified similarities with its way of operation to open source software development and find that it offers a viable motivational, organizational and collaborative solution to open innovation.

1 Introduction

Constant, lightning-fast innovation is becoming an essential element to companies in software business. Innovation can lie in any commodity it being something novel that can be put to actual use. Many companies rely on innovation on a daily basis to create better products and to improve their internal processes [2]. Traditionally such advantages have been kept within the company.

Opening up the option to innovate to a wider group of partners can enforce and expand the scope of the innovation process, which becomes free of the boundaries of the company and what knowledge is available within. Open innovation helps in identifying the best ideas by combining internal and external ideas into architectures and systems [11, 2]. The open innovation process typically involves proof of concepts, trials, and, perhaps most importantly, the right people to identify what should (or must) be focused on.

Open innovation, however, comes with a number of challenges such as motivation, integration and exploitation of innovation [5]. It needs a governance framework [4] that enables organizational alignment of the different partners, proper handling of intellectual property rights issues, and the emergence of new kinds of business opportunities. These challenges have to be taken into account when building any open innovation platform with the goal of driving future development and solutions.

In this paper, we argue that the open source model of development and knowledge creation brings a set of principles of practices that could be adapted to the context of open innovation, in the same way as observed in [10]. We focus on open innovation in the context of academia-industry co-operation. In order to support our arguments, we have analyzed an open innovation platform for students called Demola [12].

We identify characteristics of open source software development in the motivational, organizational, and collaboration aspects of open innovation. The main research question answered is: How Demola's approach shares similarities with other community driven development methods, mainly open source? In Section 2 we give background on the Demola organization and discuss the practices of open source within the open innovation context of Demola in Section 3. Section 4 then concludes the paper with some discussion and final remarks.

2 Platform for Open Innovation and Learning

There is a real need for increased opportunities for innovation projects that can lead to new business ideas. Open innovation environments allow businesses to reach beyond the company scope in the search for new concepts and ideas. A governance framework is needed with practices and working principles to bring innovation partners together and to ensure ongoing innovation work.

Demola is one such open innovation platform intended for students. It aims to multidiciplinary and agile development of innovative products and product demos. The project ideas come from the industry and public organisations and

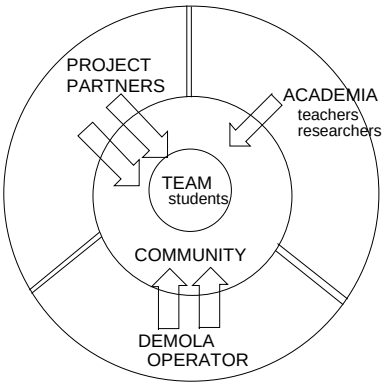


Fig. 1. Demola Partners

thus concepts that have practical business importance are developed. The student work is supported by both the industrial and the academia partners that provide guidance throughout the project. Demola offers a governance framework that facilitates team building and supports emerging business ideas. It also incorporates a model for managing immaterial rights that supports startups and

respects the authors. On a practical level, Demola provides workspaces that support team work and co-creation. Demola is a modern and actual learning environment to students from different universities.

Figure 1 shows the partners in Demola innovation and the flow of communication and support for the project work. The team is at the heart of development while others direct, aid and facilitate the work. In terms of numbers, there are currently 35 companies involved in Demola as project partners. During 2011 the aim is to reach a yearly level of around 100 projects running. The Demola operator itself employs three people: one manager and two assistants.

3 Adopting Open Source

Demola was built on the basis of openness. There are different aspects and challenges that need to be addessed in making the platform open and functional. A set of principles and practices of free/libre open source (FLOSS) can been identified in Demola.

Motivation A famous quote from Raymond [1] claims that "Every good work of software starts by scratching a developer's personal itch." This is commonly seen as one of the driving forces behind open source software quality and success. The participant's motivation is also one of the main characteristics of Demola team building and work. Similarly Raymond's restatement of the itch: "To solve an interesting problem, start by finding a problem that is interesting to you" is a major driving force in the Demola way of doing.

Internal motivation as driver: The participant's internal motivation is the main driving factor for the Demola team work. Participation is fueled by their own background, motivation and goals that range beyond normal school work.

Participant chooses the project: The way teams are formed in Demola is similar to how open source communities come into being. Students search for project topics that are meaningful and interesting to them and apply for participation in it. The reasons for choosing a project are personal to the applicant with widely varied factors behind the selection. The applicants have no knowledge of the possible other team members in advance and it is not possible to choose the people you form the team with.

Collaboration Jukka Saarinen, one of the key people behind the the foundation of Demola, has said about the platform: *"What is special about Demola is the way of doing things: anyone and everyone can contribute ideas to a demo which is then built together. The let's do it attitude without bureaucracy and formal processes makes the atmosphere fruitful"* [8]. This reflects the philosophical standpoint of open source software development. Those with the interest and skill can contribute their work to the community.

Co-creation: Demola has been built on the notion of bringing the right people together and to enabling collaboration between participants. Demola

itself is a developer community where anyone can contribute their work based on their own interest and skill. Similarly the development of the project concepts and demos in Demola is done through collaborative teams. Each team member brings his or her own knowledge and expertise into the team and each team is different.

Community spirit: The student teams, active academia members and the project partners form an innovation ecosystem where all participants benefit from the Demola platform. Demola acts like a community of developers where the teams share ideas and work and where the project partners benefit from the work done in teams for other partners.

Legal Concerns What is special about open source is its philosophy on intellectual property rights (IPR). The approach chosen for managing the IPR of the project teams in Demola is akin to the idea of licensing in open source. The open innovation approach in Demola respects the IPR of the teams: the students own the rights to the project results. The originator of the project idea can buy wide and parallel usage rights to the results by paying the project team an agreed reward, i.e. the team licences their work to the industrial partner.

4 Discussion and Conclusions

We have identified the best practices and principles of FLOSS development within an open innovation platform, Demola. When its way of doing is juxtaposed with the FLOSS principles and practices the common factors are identifiable. FLOSS also enables a better and wider exploitation of the results as the teams hold the rights to their work. Demola not only provides support for the project partners to buy rights to the work but also for the students themselves to start new businesses on top of the results.

Traditional open source principles and practices, however, may fall short in other aspects such as timely delivery, communication, and quality. Such challenges in the daily workflow of the project development need futher management methods on top of FLOSS. How these challenges are met is a focus of future research. Our findings suggest that the open source model offers a viable solution to open innovation in terms of motivational, organizational, and collaborational aspects.

References

1. Raymond E.S. *The Cathedral and the Bazaar*. O'Reilly Media, 1999.
2. Chesbrough H. *Open Innovation: Researching a New Paradigm*, chapter Open Innovation: A New Paradigm for Understanding Industrial Innovation. Oxford University Press, 2006.
3. Takeuchi H. and Nonaka I. The New New Product Development Game. *Harvard Business Review*, pages 137–146, January-February 1986.

4. Feller J., Finnegan P., Hayes J., and O'Reilly P. Institutionalising information asymmetry: governance structures for open innovation. *Information Technology & People*, 22(4):297 – 316, 2009.
5. West J. and Gallagher S. Challenges of Open Innovation: The Paradox of Firm Investment in Open-Source Software. *R&D Management*, 36(3):319–331, 2006.
6. Beck K. Embracing Change With Extreme Programming. *Computer*, 32(10):70–77, October 1999.
7. Beck K., Beedle M., van Bennekum A., Cockburn A., Cunningham W., Fowler M., Grenning J., Highsmith J., Hunt A., Jeffries R., Kern J., Marick B., Martin R.C., Mellor S., Schwaber K., Sutherland J., and Thomas D. Manifesto for Agile Software Development. Available at: `http://agilemanifesto.org/`, March 2002. Last visited March 2011.
8. Facilitating Innovation at Demola. Open Threads: Open Innovation Newsletter, April 2009.
9. Abrahamsson P., Salo O., Ronkainen J., and Warsta J. Agile Software Development Methods Review and Analysis. VTT Publications 478, 2002.
10. Goldman R. and Gabriel R.P. *Innovation Happens Elsewhere: open source as business strategy*. Morgan Kaufmann, 2005.
11. Davis S. How to Make Open Innovation Work in Your Company. *Visions Magazine*, December 2006.
12. Demola Innovation Platform. `http://www.demola.fi`. Last visited March 2011.

Case II: Publication VI

Terhi Kilamo. The Community Game: Learning Open Source Development Through Participatory Exercise. In *Proceedings of the 14th International Academic MindTrek Conference: Envisioning Future Media Environments (MindTrek'10)*, pages 55–60. Tampere, Finland, 2010, Association for Computing Machinery (ACM).

VI

The Community Game: Learning Open Source Development Through Participatory Exercise

Terhi Kilamo
Tampere University of Technology
Department of Software Systems
Korkeakoulunkatu 1
FI-33101 Tampere, Finland
terhi.kilamo@tut.fi

ABSTRACT

As open source software has gained more foothold in the software industry, teaching open source development to the future software professionals has become a practical necessity. A pioneering course was arranged during the academic year 2009-2010 at Tampere University of Technology to teach software engineering students how open source is developed and what makes open source ecosystems special. To allow students to learn the practical side of open source software development, an exercise called the community game was organised. The game is a part of the courses and focuses on community-centric software development. It allows the students to initiate and nurture a small developer community and practise open sourcce development in a safe setting. The game ensures that the students have a good enough understanding on the open source development practises in order to make contributions to actual open source projects.

General Terms

Human Factors, Experimentation

Keywords

Open source, software engineering education, participatory learning

1. INTRODUCTION

Open source software engineering has its basis in the developer community. As Eric S. Raymond states in [6], "every good work of software starts by scratching a developer's personal itch". This statement holds especially true in open source development where developers commonly participate because they want to, not because they have to.

The rise of open source development in the software industry has encouraged teaching open source software engineering to university students. The traditional lecture-heavy course format cannot fully convey the special traits of an open source ecosystem. An approach that engages the students more is needed instead. Choosing student-centric teaching methods is further supported by modern learning theories that emphasise the active role of the student and thus naturally guide teaching away from the lecture environment.

Tampere University of Technology has not previously had a course dedicated to the development of open source software, but a pioneering course was developed and arranged during the academic year 2009-2010 to remedy this.

In this paper we focus on a participatory exercise called *the community game* where students set out to learn open source software development and methods via an exercise mimicking actual open source projects. The game is a compulsory part of the entire course, and its goal was to engage students and let them learn the practical side of open source development in the safety of the computer classroom.

In Section 2 the course on open source software is described and the motivation behind it is given. In Section 3 how the community game was organised and played is explained. Section 4 describes the experiences from the game. In Section 5 different aspects of the game are discussed and the paper is concluded in Section 6.

2. TEACHING OPEN SOURCE

The role of the learner is at the heart of modern pedagogical learning theories. It is the learner who takes the responsibility for learning driven by their own motivation and influenced by their earlier knowledge and experiences. The instructor's role is to help the learner in gaining more understanding and knowledge. When contrasting different constructivist pedagogical models such as problem-based and project-based learning with open source development one can see that these can promote learning open source software engineering.

Teaching methods that borrow practises from open source have been tried out in other universities. An open source ecosystem can be used in teaching software engineering in general as suggested by [7]. In [4] a hybrid approach blending open source community principles with education was used. The approach shares similarities with the ideas behind the community game.

Several courses teaching open source development have also been arranged in universities worldwide [1], [3]. The requirements of the courses contain a project asking for participation in and contribution to an open source project. On a similar note, Google has organised the Google Sum-

MindTrek 2010 October 6th-8th 2010 Tampere, Finland

mer of Code (http://socghop.appspot.com/) since 2005 to promote open source development to students and to provide them with an opportunity to work on things related to their studies while engaging them in real-life software development. The Finnish Centre for Open Source Solutions (COSS) has also arranged a similar annual summer code event in Finland (http://www.coss.fi/en/summercode) to support participation in open source projects and thus allow participants to gain a better competence in open source software development.

While the aforementioned ways of learning through participation in an actual project are desirable, it requires a certain amount of skills and time to get into an open source community and to contribute to it. Open source projects may not act as an ideal learning environment for the basic open source software engineering practises as they are ongoing and active projects building software, and are not a playground for newbie developers.

Our class, Introduction to Open Source Software (http://tutopen.cs.tut.fi/course10/) was arranged in the form of a course for the first time during the spring 2010. As the course was new, the course staff was liberated to choose the teaching methods freely to best reach the intended learning outcomes. The course staff also set out to learn from the chosen teaching methods in order to develop the course in the future. Due to the developers and the community being the driving forces of open source development, the course was organised similarly, with the students in the key role and the staff just supporting the learning.

The general idea behind all course arrangements was self-organization and participation, which are also essential in open source software. The goal was to keep the course as open and open-source-like as possible without compromising the educational goals. The material and content produced during the course were visible to everyone and editable by all course participants. The main bulk of the course material was created by the students in collaboration with each other. A book [2] was recommended as a basis for research in various open source related topics but the students were to collect information from other sources as well.

The course was offered to both undergraduate and postgraduate students and it turned out to be rather popular with 32 registered students, most of whom where undergraduates. It is normal that not all students end up taking the course and that was also the case here, with approximately 22 active participants.

The course consisted of three parts: getting to know and reporting a key topic of open source software, taking part in a community game, which is the focus of this paper, and focusing on making a contribution to an existing open source project.

3. PLAYING THE GAME

In the course, the students were given a safe pond to hone their open source development skills before diving in the deep end of open source development in an exercise called "the community game". In the game the students set out to build and nourish a small open source project and thus learn how open source development is organized and run in vitro before moving on to rampage free in the open source ecosystem. The game was organised as a single three-hour session in a computer class.

3.1 Game setup

The students were provided with an initial start community. However, as the entire course and thus also the game were based on the notion of self-organization of the students, i.e., the students make the decisions themselves, this principle was naturally applied in setting up the game. The embryo community for the game was therefore almost skeletal in nature and required the students to decide on most of the practises.

The skeleton of the project was available in the course wiki in the beginning. Similarly, the basic project infrastructure was ready. Subversion (svn, http://subversion.apache.org/) was used as the version control system and in the beginning it contained the initial code. The bug reports and feature requests were given in the Mantis bug tracking system (http://www.mantisbt.org/).

The program being developed was a simple hyphenation system. The program was small on purpose as the focus was not the program itself but how it was being developed. Initially, the program could only handle the most basic hyphenation rule of the Finnish language: inside a word a hyphen preceeds consonants that are followed by a vowel. Even the initial revision had known reported bugs in it: the program added a hyphen at the beginning of the word and it did not recognise the character 'å'. There were naturally also unreported and unknown bugs. The initial code was 80 lines written in C++ including comments.

In the beginning, there were feature requests for multi-language support, other hyphenation rules, wrapping words and reading input from a file given as a command line parameter in Mantis. Out of these the input from a file proved to be rather interesting, as it turned out that the core of the community ended up deciding it was not needed.

The students were expected to make a contribution every 15 minutes. The goal was to ensure that people contribute early and often although such timeboxing is foreign in real open source communities. The reasons for timeboxing the game were explained to the students at the beginning.

The participants were able to discuss with each other in an irc (http://www.irc.org/) channel set up for the course called. An additional mailing list and/or a discussion forum would have been a little more realistic, but setting up one for the purpose of the exercise seemed overkill.

In order to cheer up the students and keep up the good work, free coffee and tea was served after two hours of work.

3.2 Student roles

In total 22 students – in other words, all the students actively participating in the course – registered to Mantis and thus participated in playing the game. The students were assigned individual roles in the beginning of the game. There were three possible roles: committer, community member and forker. The committers played the owners of the product and had the right to make commits to the project code base, decide over releases and had the uppermost control of the project. The forkers were given the instruction on forking the project right in the beginning and later try to merge back. The community members focused on the development of the main project. The students were free to decide what to work on in their assigned role. However, each role was a developer role in nature with no roles around using the product.

The role of the committers was delegated to three pres-

elected students with enough prior experience. The rest of the roles were arbitrarily assigned through giving the students a piece of cardboard of a certain color as they came into the classroom. The team's colour codes were as follows: yellow for the committers, blue for forkers and green for the community members.

3.3 Game Flow

The students started with the game after a brief introduction at 14:15 and the first bug was reported at 14:18 in irc. Self-organization worked smoothly as people immediately got to work, and the committers gave instructions on submitting patches and reporting bugs within 15 minutes in the irc discussion[1]. Mantis was used for both bug reports and feature requests.

```
14:29 < committer1> on behalf of the yel-
low team:  can you please assign an issue from
the mantis to yourself and then post patches
under that issue as a note

14:31 < committer1> furthermore, if you find
a bug that is not in the mantis, please add
it in there as a new bug or feture request
```

The first new release was set to be released at 15:30, but it was first postponed by fifteen minutes. The actual release missed even the postponed time by about five minutes. When asked about it after the session nobody admitted getting upset about the delay. The first release made had four issues for the community to still fix. These issues, numbers 1, 2, 16 and 17, are visible in Figure 1.

			0000013	1	[Hyphenation] Feature request	minor	closed
			0000019	1	[Hyphenation] Bug	major	closed
			0000009	2	[Hyphenation] Feature request	minor	closed (to
			0000005	1	[Hyphenation] Feature request	feature	closed (dar
			0000001	2	[Hyphenation] Bug	major	closed (kork
			0000017	2	[Hyphenation] Feature request	major	closed
			0000016	2	[Hyphenation] Bug	crash	closed (kork
			0000003	5	[Hyphenation] Feature request	feature	closed (kork
			0000002	4	[Hyphenation] Bug	major	closed (tu

Figure 1: Issues resolved between the two releases.

The second stable release was made towards the end of the session at 16:40. In it the four issues shown in Figure 1 had been covered but one major issue, number 12, remained unresolved. The status of the project in Mantis is visible in Figure 2 which also shows the additional minor issues that still remained. Figure 3 shows a new bug, issue 16: crash, being found. In Figure 4 the issue 16 is seen as resolved.

The final code in the version control system is still in one file and it is 98 lines long in total. In addition to the code being fixed and developed a Makefile has been written for the project. A README file was also written, but at the end of the game it was still only available in Mantis.

[1]The identities of the participants have been changed in the quotes from the irc discussion

Viewing Issues (1 - 15 / 15) [Print Reports] [CSV Export]

P	ID	#	Category	Severity	Status	Updated	
	0000018	1	[Hyphenation] Feature request	minor	confirmed	2010-03-18	README file
	0000012	3	[Hyphenation] Bug	major	assigned (halle)	2010-03-18	Doesn't recognice
	0000007	6	[Hyphenation] Feature request	feature	acknowledged	2010-03-18	Other hyphenatior
	0000008	11	[Hyphenation] Bug	minor	resolved (Anssi)	2010-03-18	Double hyphenatic
	0000024		[Hyphenation] Feature request	minor	resolved (ranta5)	2010-03-18	Exit command
	0000025	1	[Hyphenation] Feature request	minor	new	2010-03-18	GUI
	0000022	1	[Hyphenation] Feature request	minor	resolved (amatoori)	2010-03-18	Command prompt
	0000014		[Hyphenation] Feature request	feature	resolved (jlautamaki)	2010-03-18	cleanup of tavuta
	0000023		[Hyphenation] Bug	major	confirmed	2010-03-18	Compuond word
	0000021		[Hyphenation] Bug	minor	confirmed	2010-03-18	No license inform
	0000011		[Hyphenation] Bug	trivial	resolved (dantas)	2010-03-18	Accepts numbers
	0000006		[Hyphenation] Feature request	feature	confirmed	2010-03-18	Word wrap

Figure 2: Mantis after second release

Viewing Issues (1 - 16 / 16) [Print Reports] [CSV Export]

P	ID	#	Category	Severity	Status
	0000006		[Hyphenation] Feature request	feature	confirme
	0000016		[Hyphenation] Bug	crash	new
	0000011		[Hyphenation] Bug	trivial	assigned (da
	0000003	5	[Hyphenation] Feature request	feature	assigned (kor
	0000008	5	[Hyphenation] Bug	minor	confirme

Figure 3: A bug reported in Mantis

3.4 Forking the Project

The project was forked at the beginning of the game. Four people set out to take the original program and develop it independently as a separate project. They decided to implement English hyphenation and thus had little in common with the main branch. The forkers were not given an infrastructure for the fork project but they were expected to build the fork from scratch. This took a lot of their time, and they ended up using Google Docs (`http://docs.google.com`).

The forkers had to spend a significant amount of time in

Viewing Issues (1 - 17 / 17) [Print Reports] [CSV Export]

P	ID	#	Category	Severity	Status
	0000008	6	[Hyphenation] Bug	minor	confirmed
	0000016	2	[Hyphenation] Bug	crash	resolved (kork
	0000017		[Hyphenation] Feature request	major	new
	0000006		[Hyphenation] Feature request	feature	confirmed
	0000011		[Hyphenation] Bug	trivial	assigned (dan

Figure 4: Reported bug fixed

connecting with each other and setting up the fork. The solutions for developing the fork were ad hoc and decisions were made in the heat of the moment. Hence the fork project ended up using tools out of the range of normal open source development.

The fork project produced a program that could hyphenate basic English. The code had less of a complete feel to it then the orginal project's and had 186 lines in total. Merging the fork was not done by the end of the game, as the programs were both in separate files.

4. EXPERIENCES

We consider that the first community game implementation was successful. The general observation was that all students got to work in the community and the game gave everyone practical knowledge on what open source software development entails. The group self-organised even better than expected. Feedback from the students was altogether positive and constructive in nature. The feedback also supported the course staff's views on how the game could be improved in the future.

4.1 Student Feedback

The initial feedback that came up during the discussion directly after the game was positive. Additional feedback was collected when the third part, making the contribution, was about to start. The motivation for collecting the feedback here was to collect not only experiences and opinions from the game once the dust had settled but also to map how well the students felt prepared for working on making contributions to actual open source projects having played the game. In total 15 students gave feedback.

There were five claims about the game in the feedback form. They were answered on scale from one to five based on how well the respondent agreed with the claim. One was complete disagreement and five complete agreement. The claims were:

1. The community game was a good exercise.
2. The community game focused on the key practises of OS software engineering.
3. The size of the project was suitable for the game.
4. You got to work on things you wanted during the game.
5. The community game helps with working on the contribution required for the course credits.

In addition "which role did you play?" was asked together with the fourth claim. The statistics on the feedback is given in Table 1.

	1.	2.	3.	4.	5.
Average	4,2	4,07	3,87	3,2	3,64
Median	4	4	4	4	4

Table 1: Student feedback statistics

The participants thought the game was a good exercise and that the focus was correct. The average of the answers to the first claim was 4,2 with only one answer below "mostly agree". The median answer was "mostly agree". Five answered "completely agree" that the game was a good exercise. The second claim got similar result with no answer lower than "do not agree or disagree" with a median answer "mostly agree" and 4,07 as the answer average.

People found the size of the game mostly suitable. The average here was 3.87 and the median answer was "mostly agree". Two respondents disagreed somewhat and found the project smallish. Otherwise the answers were distributed between three to five. The additional written feedback given about the size of the project was on the lines that the project could be bigger.

The average of the answers to getting to contentrate on things they wanted was 3,2. The median answer was still "mostly agree". The forkers were happy with their role with 4,25 average and with two respondants answering "agree completely", one answered "mostly agree" and one "do not agree or disagree". The handpicked committers were less satisfied with their role with two "do not agree or disagree" answers and one "mostly agree". Among the community members two respondents felt they didn't get to do what they wanted. One was focused on technical problems and there were none. The other didn't state what they did or would have wanted to do. The average of the answers of the community members was 3,1. This gives reason to think if there should be more roles to assign to people. In addition instead of randomly assigning the roles the students could choose their roles from a given pool providing them with more power in what they do and thus emphasising the self-organisation.

The average of the answers to the last claim was 3,64 with again a median of "mostly agree" and one complete disagreement and no other answers below "do not agree or disagree". One respondent didn't know if the game helped or not and answered '?' as their own choise. The importance of the claim can be judged better once the course is completed and the contributions done. Based on the feedback here, the students felt that the game was helpful in getting into actual open source community work.

At the end of the feedback the respondents were asked an additional question: "How would you improve the game?" and given an opportunity to give free feedback. The main improvement to which everything boiled down was that the game needed more time to run. The timeframe of three hours was too short for the game to get beyond the first steps. After two releases the community had found a common tune and suitable development practices and then they ran out of time. With a longer session the project could be made larger and more realistic, more roles could be given and a community would have more time to form making the game more interesting and more educational. Some suggested more instruction on the tools, but the longer timeframe would remove the need for extra tool tutorials by leaving enough time into the game for the community to evolve and members to work on the product after the tool setup and basic organisation. Outside tutorials would be an artificial part of the game and therefore the group learning together and teaching each other is preferred.

4.2 Staff Perspective

Based on the initial observations by the course staff the community game reached the intended learning objectives. The students got to know the practical side of open source development and gained experience that would help them in

getting into actual open source community work. However, it is difficult to evaluate the learning of individuals. Self-organization can lead to people concentrating more on what they already know instead of challenging themselves. However, the point is not to force learners but to let their own internal motivation to drive them. This appeared to work well. The overall learning can be better evaluated once the entire course is completed but based on the present evidence the game seems to be a good exercise.

There is an obvious need for more fixed roles than the three used this time in the community game. There is no real need to assign all of the roles from the onion model for open source communities [5] as in the game everyone needs an active role. However, members acting as bug reporters only would have been valuable. When the game community was let to form freely among the participants assigned a community member role, everyone assumed a developer role. Leaving out only the passive, outermost roles would have brought more discussion into the community and the community would have more likely adapted more realistic practises for example for handling the bugreports. This observation is again supported by student feedback.

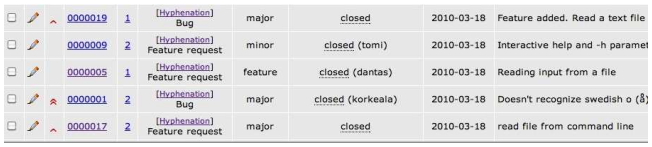
It was obvious as the game drew towards an end that the timeframe was too short. This was further highlighted by the fact that one of the committers had to leave early as real life intervened with the game. In addition, there is no real need to timebox the contributions. The student feedback heavily supports this notion.

5. DISCUSSION

Even though most of the students had prior knowledge of all the tools used in open source development, using them in a development setting showed that, in addition, practical knowledge is essential. Especially how to create and send in a patch, what a patch is, and how issues were to be handled were significantly clarified by the exercise. The Mantis bug reporting system was found confusing, and the students felt it should have been explained. Getting to know the bug tracking system and finding out how to use it was one of the goals of the game. The game environment allowed the students to learn through their own work, participation and collaboration, how the bug tracking system worked which led to a better learning outcome. Instead of giving students the answers, they were given an environment to be active and learn things through their own insight.

Although there was an initial requirement that all discussion was to be held on the irc channel, the channel backlog is only 247 lines long and some verbal communication between smaller subgroups did emerge. The closer the game came to the end, the more face to face discussion emerged. This was to be expected and was not considered to be harmful to the information flow within the community as the key issues were raised into discussions in irc. In order to track the interaction that went on outside the communication media the entire session should have been videotaped, which may be an option for the next implementation of the course and the game.

In order to provide options in the working environments, two computer classrooms had been reserved for the last two hours of the game. However, nobody wanted to geographically distance themselves to another classroom and even the fork team remained physically in the same room as the main community. There was feedback directly after the session that the computers should have been more modern and efficient but still nobody had switched classrooms. It is apparent that the sense of being together and working as a group was a stronger motivator than better computers. The three hours reserved for the game was not enough to establish a strong enough online community for the students to distanse themselves to another classroom. The social aspect of developing software together was enforced by the shared space and participants were reluctant to break that. This does go against the actuality of open source development. There was no real need to force more solitarity as the development was carried out without much face to face discussion despite the shared space.



0000019	1	[Hyphenation] Bug	major	closed	2010-03-18	Feature added. Read a text file
0000009	2	[Hyphenation] Feature request	minor	closed (tomi)	2010-03-18	Interactive help and -h paramet
0000005	1	[Hyphenation] Feature request	feature	closed (dantas)	2010-03-18	Reading input from a file
0000001	2	[Hyphenation] Bug	major	closed (korkeala)	2010-03-18	Doesn't recognize swedish o (å)
0000017	2	[Hyphenation] Feature request	major	closed	2010-03-18	read file from command line

Figure 5: Duplicate requests in Mantis

The community members adopted a pattern of handling bugs which cut a few corners on the way. The reporters of the bugs often also fixed it themselves immediately without waiting for confirmation. This pattern can be seen in bug 16 visible in Figures 4 and 3. The person who reported it also fixed it in a timely fashion. The time between the reporting and fixing the bug was 12 minutes, but naturally this left no time for the core members to confirm the bug. Adding a role for bug reporters would help to avoid this pattern.

The yellow committer team was handpicked to consist of students the course staff knew had enough programming experience to assume this role. This ensured a smooth start for the game and mimicked real life. The core members in a real open source project are also skilled and experienced. Similarly like in open source communities in general all participants were given an opportunity to get promoted to a committer based on their activity and progress during the game. Surprisingly, nobody wanted to get promoted to a higher status. The main reason given when asked in a free discussion after the game was: “coding is fun”. Despite that, the original committers did give one member the commit rights towards the end of the game. The choise was based on the member’s irc channel chat activity and contribution to the community. The core members saw no reason not to give the promotion. Adding more possible roles might ensure more movement between the roles, mostly from the outer roles towards the more influential ones.

The tight timeframe left no real time to test thoroughly. The committers made only quick test runs and visual verification of the patches before accepting them. This way they could accept patches rather quickly. Patches were rejected only if they were not applicable or didn’t implement a new feature which meant that very few commits got rejected. Some features were even left out of a release as there was no time to go through them sufficiently. Furthermore the game ended when it had started to run smoothly and more fun and interesting things could have developed given more time.

A feature request that came up several times and as duplicates was a request for a possibility of handling input from a file. This can be seen in Figure 5. Duplicate entries were

closed and the original rejected due to none of the commiters seeing a motivation for the feature. There was some discussion of the duplicates and of the feature on the irc channel, but nobody supported it being taken on so it was disgarded:

```
15:54 < committer1> so, can you remind me why
is there a need for this feature?  can't we
just use redirection?
i.e.  ./tavutus < file.txt
15:55 < committer2> I allready closed that
one
...
```

```
15:56 < committer2> I think we are not go-
ing to get feature ./tavutus file.txt
15:57 < committer2> except if there is some-
thing like ./tavutus file1.txt file2.txt
15:57 < committer2> or something else like
that
15:57 < member1> committer2:  you mean that
is not necessary?
15:58 < committer1> i just wanted to know
what's the motivation for the file option?
as opposed to using < ?
15:58 < member2> what is 0000017 then
```

The shell redirection of the input was able to sufficiently fill in the need for the feature and the issue was closed. Here the students took control of the product as the feature request was one the originals given in Mantis as a starting point for the game.

The members of the fork project were not really interested in the main branch but acted as a small project of their own. They were not provided with a similar initial project as the main branch, which obviously set them back. In the end the fork was built on top of ad hoc tools instead of tools used in actual open source development. In future implementations of the game this needs to be taken into account and a better starting point provided. The fork was for the first time asked to remerge with main branch at 16:00. They however did not proceed to do so, and in fact, the fork remained unmerged also at the end of the game. There were bugs in the fork code that would have broken the main branch and thus the merge was not even attempted. The overlap between the main and the fork was small as the fork was about English hyphenation whereas the main branch focused on Finnish text. The forkers estimated that completing the merge should not require too much work even though they were at that point not prepared to merge. Feedback supports keeping the forking as a part of the game as the forkers were really satisfied with the game. However, a better infrastructure needs to be ensured so that the fork can reach the intended learning outcomes. Now the goal of learning the tools and practises was not fully reached in the fork's case.

The initial code was given as a single code file. The main branch didn't see a need for changing this as patches needed to be applied anyway. The fork saw it as a problem. However, the fork did not report it to the main community even though a more modular structure would have helped them to merge back. The single code file should be kept as the starting point in the future as refactoring it acts as a good feature request.

6. CONCLUSIONS

Based on our experiences, the community game teaches open source software development in a safe but realistic setting giving the students a good starting point for work in real open source projects. The students are in no need for strict boundaries or outside direction. Instead they can learn by doing and together as a group through exercises such as the game that take their incentive from real life but still keep the scope small enough to further enforce learning.

There obviously are points for improvement and for the course to learn and adapt but the community game did turn out to be a good practise for real world and in addition – and maybe more importantly – fun.

Acknowledgements

This research is a part of the openSE project supported through the European Union's Lifelong Learning Programme (LLP).

7. REFERENCES

[1] D. M. German. Experiences teaching a graduate course in open source software engineering. In *Proceedings of the First International Conference on Open Source Systems*, pages 326–328, July 2005. `oss2005.case.unibz.it/Papers/OEs/Es1.pdf`.

[2] J. M. González-Barahona, J. S. Pascual, and G. Robles. Introduction to free software. GNU Free Documentation License, Creative Commons Attribute ShareAlike License. `http://ftacademy.org/materials/fsm/1#1`.

[3] B. Lundell, A. Persson, and B. Lings. Learning Through Practical Involvement in the OSS Ecosystem: Experiences from a Masters Assignment. In J. Feller, B. Fitzgerald, W. Sacchi, and A. Sillitti, editors, *Open Source Development, Adoption and Innovation*, volume 234 of *IFIP International Federation for Information Processing*, pages 289–294. Springer, 2007.

[4] A. Meiszner, K. Moustaka, and I. Stamelos. A Hybrid Approach to Computer Science Education — A Case Study: Software Engineering at Aristotle University. In *CSEDU 2009 - Proceedings of the First International Conference on Computer Supported Education*, volume 1, pages 39–46. INSTICC Press, 2009.

[5] K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y. Ye. Evolution Pattern of Open-Source Software Systems and Communities. In *IWPSE '02: Proceedings of the International Workshop on Principles of Software Evolution (2002)*, pages 76–85. ACM Press, 2002.

[6] E. S. Raymond. *The Cathedral and the Bazaar*. O'Reilly Media, 1999.

[7] I. Stamelos. Teaching Software Engineering with Free/Libre Open Source Projects. *IJOSSP*, 1(1):72–90, 2009.

Tampereen teknillinen yliopisto
PL 527
33101 Tampere

Tampere University of Technology
P.O.B. 527
FI-33101 Tampere, Finland

ISBN 978-952-15-3237-5
ISSN 1459-2045