Kevin Crowston, Imed Hammouda, Juho Lindman,
Björn Lundell and Gregorio Robles (Eds.)

Proceedings of the Doctoral Consortium at the 12th International Conference on Open Source Systems

Gothenburg, Sweden, 30 May 2016

Skövde University Studies in Informatics 2016:1

UNIVERSITY
OF SKÖVDE

Crowston, K., Hammouda, I., Lindman, J., Lundell, B., and Robles, G. (Eds.) Proceedings of the Doctoral Consortium at the 12th International Conference on Open Source Systems, Skövde University Studies in Informatics 2016:1, ISSN 1653-2325, ISBN: 978-91-978513-9-8, University of Skövde, Skövde, Sweden.

Copyright of the papers contained in this proceedings remains with the respective authors.

Skövde University Studies in Informatics 2016:1
ISSN 1653-2325
ISBN 978-91-978513-9-8

www.his.se

UNIVERSITY
OF SKÖVDE

Proceedings of the Doctoral Consortium at the 12th International Conference on Open Source Systems, 2016

Edited by:

Kevin Crowston
Syracuse University, Syracuse, NY, USA

Imed Hammouda
Chalmers and University of Gothenburg, Gothenburg, Sweden

Juho Lindman
Chalmers and University of Gothenburg, Gothenburg, Sweden

Björn Lundell
University of Skövde, Skövde, Sweden

Gregorio Robles
Universidad Rey Juan Carlos, Madrid, Spain

Preface

The last two decades have witnessed a tremendous growth in the interest and diffusion of Free/libre and Open Source Software (FLOSS) technologies, which has transformed the way organisations and individuals create, acquire and distribute software and software-based services. The Open Source Systems conference as its premier publication venue has reached its twelfth edition this year.

To facilitate new researchers with an arena to present and receive feedback on their research, the Open Source Systems conference has had a Doctoral Consortium for several years. The principle objective of the consortium is to provide doctoral students the opportunity to present their research at various stages of production – from early drafts of their research design to near completion of their dissertation – in a forum where they can receive constructive feedback from a community of interested scholars and other students as they work to finish their degree.

This volume contains the eight papers, each of which was reviewed by members of the program committee. After the reviews, authors were given the opportunity to revise their paper based on the input they received from the reviewers and participants who provided feedback during the event. This volume contains the revised versions of the papers, which were presented and discussed at the Doctoral Consortium at the Twelfth International Conference on Open Source Systems, in Gothenburg, Sweden in May 2016.

We wish to thank the reviewers and members of the Program Committee of the Doctoral Consortium who have provided valuable feedback on the papers. We also thank all Ph.D. students and senior researchers for their participation. Finally, we are grateful for the support provided by Chalmers and University of Gothenburg, and the financial support (award number 1639136) provided by the U.S. National Science Foundation (NSF).

Kevin Crowston
Imed Hammouda
Juho Lindman
Björn Lundell
Gregorio Robles

Program Committee

Kevin Crowston	Syracuse University, Syracuse, NY, USA
Joseph Feller	University College Cork, Ireland
Jonas Gamalielsson	University of Skövde, Sweden
Imed Hammouda	Chalmers & University of Gothenburg, Sweden
Juho Lindman	Chalmers & University of Gothenburg, Sweden
Björn Lundell	University of Skövde, Sweden
Gregorio Robles	Universidad Rey Juan Carlos, Spain

Table of Contents

Requirement Engineering in Open Source Software – The Role of the External Environment 1
Author & presented by Deepa Gopal

A Quantitative Analysis of Performance of the Key Parameters in Code Review – Individuation of defects 11
Author & presented by Dorealda Dalipaj

Analysing on how the bugs are injected into the source code 25
Author & presented by Gema Rodríguez Pérez

Internet of Things and Web Squared: Open for Inclusive Development? 37
Author & presented by Katja Henttonen

Predicting Faults in Open Source Software: Trends and Challenges 55
Authors: Malanga K. Ndenga, Jean Mehat, Ivaylo Ganchev, and Wabwoba Franklin
Presented by Malanga K. Ndenga

Competing on a Common Platform 75
Authors: Siobhan O'Mahony and Rebecca Karp
Presented by Rebecca Karp

The Quest for UML in Open Source Projects Initial findings from GitHub 95
Authors: Regina Hebig, Truong Ho-Quang, Gregorio Robles, and Michel R. V. Chaudron
Presented by Truong Ho-Quang

Evolution and Influence of Sub-groups on Group Productivity and Success 105
Author & presented by Pinar Ozturk

Requirements Engineering in Open Source Software – The Role of the External Environment

Deepa Gopal
Case Western Reserve University, Weatherhead School of Management,
Department of Design & Innovation, 11119 Bellflower Rd, Cleveland, Ohio
WWW home page: http://www.weatherhead.case.edu

Abstract. Popularity of open source software (OSS) projects has spiked an interest in requirements engineering (RE) practices of such communities that are starkly different from those of traditional software development projects. Past work has focused on characterizing this difference while this work centers around the difference in RE activity across OSS projects based on how OSS RE activity has been conceptualized as a socio-technical distributed cognitive (DCog) activity where heterogeneous actors deploy artifacts to 'compute' requirements. To explore how the attributes of the DCog configuration within the projects respond to the attributes of environment housing the OSS projects and subsequently affect the attributes of software requirements produced by such communities, a comparative analysis of successful OSS projects will be undertaken with an instrument developed to measure various requirement attributes.

Keywords: Requirements quality, distributed cognition, open source software, external environment, social network analysis, complexity.

1 Introduction

The determination and management of system requirements continues to be one of the major challenges of contemporary software development (Cheng and Atlee, 2009). One conundrum that recently has confronted researchers is how to characterize the determination of requirements in non-traditional contexts, such as Open Source Software (OSS). Past work has mainly focused on delineating the features which make OSS RE distinct from RE in traditional forms of software development. It has also been argued that RE in OSS is a high level distributed cognitive (DCog) process spread over time and space comprising of multiple stakeholders and heterogeneous artifacts (Hansen et.al, 2012). However the extant literature exposes a number of variations in practices and structures of OSS projects (Crowston et al. 2012) such as the social structures of OSS communities differing substantially (Mockus et al. 2002) and the codebases growing along different trajectories (Darcy et al., 2010). Given these observed variations it is unlikely that requirements are determined in a unitary fashion across all OSS projects.

To characterize differences in RE practices across various OSS projects a DCog view of OSS (Hansen et.al, 2012) is deployed that is sensitive to the dynamic and distributed nature of practices in the OSS context, and assumes that multiple actors deploy heterogeneous artifacts to compute requirements. Further, drawing upon the Information Processing View (IPV) (Galbraith; 1973, 1974), it is conceptualized that the various ways in which an OSS community organizes its cognitive activities socially and structurally is a response to the RE environment or more specifically, to requirements emanating within the environment (Jarke and Lyytinen, 2014). These diverse DCog configurations in turn, affect the quality of requirements produced internally in such communities. Gopal et.al (2016) study the reciprocal relationship between the varying attributes of requirements that are addressed by different configurations of social and structural distributions for 'computing' requirements, in four successful OSS projects and opine that these varying DCog configurations have an effect on the quality of requirements computed by the OSS projects. The current study investigates this relationship further to unveil the exact effect of the requirements emanating from the environment on the DCog mode of OSS communities measured via network centrality constructs which in turn affects the quality of requirements produced by the OSS communities expressed in its degree of vagueness and veracity. The rest of the paper is structured as follows: a review of the literature on OSS RE and DCog and how they are influenced by the environment and in turn influence the quality of internal requirements produced to make explicit the appropriateness of deploying DCog in distributed RE, followed by the theoretical model of the proposed study and the proposed research design of the study.

2 Literature Background

2.1 Requirements Engineering in Open Source Software

So far only a sparse amount of studies have shed light on the RE activities of OSS groups (Vlas and Vlas 2011). They have established that RE processes in OSS communities are starkly different from those in traditional software development. due to the voluntary nature of participation in OSS development (Crowston et.al, 2007) and the use of informal web-based documentation practices which replace formal specifications and other design documents (Scacchi 2002, 2009). The requirements in OSS projects are made explicit through a wide range of 'informalisms' such as threaded discussion forums, web pages, e-mail communications, and external publications (Scacchi 2002). Accordingly, OSS RE is considered to be less formal and dependent on online documentation and communication tools (Ernst and Murphy 2012; Noll and Liu 2010). The requirements emerge from developers' experience and domain knowledge (Noll and Liu 2010). Though this research provides detailed explanations of how distributed artifacts support RE, it does not consider the flow of requirements computation

through interaction of actors and artifacts. A model of this interaction is suggested by Thummadi et.al (2011) who opine that the quality of RE is related to the structural distribution of the OSS project and the use of diverse artifacts through which requirements knowledge is disseminated. A recent study (Xiao et.al, 2013) suggests moreover that OSS RE is a socio-technical DCog activity where multiple actors deploy multiple artifacts to compute requirements to reach a common understanding of what the software is going to do. The organization of developer communities demonstrates significant variation around the generic core-periphery model (Mockus et al., 2002). This suggests that OSS projects exhibit considerable diversity in their social and structural distributions. However the reason behind this diversity and how it is reflected in RE activity remains an unexplored area.

2.2 Distributed Cognition

To accommodate the distributed nature of OSS wherein requirements knowledge is distributed through multiple actors, artifacts and their interaction the DCog theory (Hutchins, 2000; Hutchins and Lintern, 1996) is used as the theoretical lens of inquiry. The theory postulates that cognition is not limited to mental states in the skull of an individual but rather it is deeply distributed among the social actors and artifacts which together constitute a system. Cognition is perceived as a socially and structurally distributed phenomenon where cognitive workload is shared among the members of a team and its artifacts (Hutchins and Klausen 1996; Hutchins 1995). This view is fit to examine RE in OSS as it involves multiple actors and heterogeneous artifacts and complex cognitive processes. This collective effort of 'requirements computation' ends in a set of feasible requirements- a closure, as it has been referred to (Xiao et.al, 2013).

Cognitive processes are distributed across the members of a social group; they are also distributed in the sense that the operations of the cognitive system involve coordination between internal and external (material or environmental) structure. Finally, processes can be distributed over time in such a way that the products of earlier events will transform the nature of later events (Hutchins, 2000). These three forms of distribution have been identified as 'social distribution' (the distribution of cognition among actors), 'structural distribution' (the distribution of cognition across artifacts), and 'temporal distribution' (the distribution of cognitive process and tasks over time) (Hansen et al. 2012; Thummadi et al. 2011).

Social distribution is relevant to OSS development, because multiple actors with diverse skills volunteer to play different roles in the project (Crowston and Howison, 2005). Structural distribution of DCog activity refers to the distribution of cognitive workload achieved through the use of a collection of artifacts (Xiao et.al, 2013). Temporal distribution in RE is manifested by the use of computational heuristics – rules of thumb that the social actors deploy in OSS RE and which state what to do when.

2.3 Influence of Requirements Emanating From the External Environment

To uncover how the different DCog modes can be explained by the environmental characteristics housing the OSS groups, the IPV theory is appropriate. IPV posits that managers use organizational mechanisms, such as communication flows and work processes, to address the information processing needs of the organizational tasks. Alternative organizational mechanisms are geared towards either reducing information processing needs or increasing capacity for processing information (Galbraith; 1973, 1974). The choice of the mechanisms is dependent on the amount of information that needs to be processed. The information processing needs in itself stem from the level of environmental uncertainty. Thus, the environmental characteristics of various OSS groups are likely to invoke varying DCog mechanisms depending on the type of RE task they need to address.

The information processing needs are also related to the complexity associated with RE. The perception of this complexity has shifted from managing inner and static complexity (the set of requirements remains stable since its inception) to a dynamic external form of complexity (the set of requirements is dynamic and has high level of dependencies) (Jarke and Lyytinen, 2014). The requirements thus emanating from the RE environment can be studied in terms of six V's – volume, veracity, vagueness, velocity, variance and volatility.

The design complexity in RE is manifested in how RE deals with software and its components and how they interact with the socio-technical components of the RE environment (Jarke and Lyytinen, 2014). This comes explicit by looking at which types of DCog configurations the OSS project chooses to carry out its developmental efforts. In this regard the design task is approached as an effort to improve the environmental 'fit' of the software system by adapting it into a growing number of technical, social and organizational subsystems (Hanseth and Lyytinen, 2010). Thus the DCog configurations 'chosen' by an OSS project can be seen as a direct response to the specific environmental factors it is subjected and a study of different OSS groups subject to diverse environments will manifest various social and structural cognitive distributional modes that are conducive for a particular technological environment (Gopal et.al, 2016). The authors use a comparative analysis of four OSS projects to unveil this relationship though the exact nature and results of the same and the mechanisms through which this relationship is produced has not been explored in depth.

2.4 Factors Affecting Quality of Requirements Produced

RE success is conceived as comprising of three different dimensions - cost effectiveness of RE process, quality of RE product and quality of RE service (El Emam et.al, 1996). Quality of RE service is perceived to be the most important dimension of success and cost effectiveness of RE process, the least important

dimension. Quality of requirements in general can be studied in terms of the atomicity, precision, completeness, consistency, understandability, unambiguity, traceability, abstraction, validability, verifiability and modifiability of requirements (Génova et.al, 2013). Given the emphasis placed on quality of RE phase in information systems development, it is interesting to look at the factors that ensure higher requirements quality. Though a rapidly changing environment is detrimental to the quality of RE, it has been found that user participation alleviates some of its negative effects (El Emam et.al, 1996). However this beneficial effect of user participation diminishes as the external environment stabilizes and the uncertainty is reduced. The finding has been reinforced by a later study (Kujala et.al, 2005) that shows that user involvement is the key concept in the development of useful and usable systems and has positive effects on system success and user satisfaction. This insight is very valuable in determining the extent to which the stakeholders must be included in the RE phase of a project and especially in the OSS context, where participants are both producers and users of the end software product. The above findings emphasize the influence of social structure of development teams on the ensuing RE activity and in the OSS context, resounds in the manner in which the social distribution of DCog activities affect the RE process.

3 Theoretical Model

To address the effects between the environmental characteristics and the DCog configuration of an OSS community, Crowston and Howison's (2005) work on social structures of OSS projects helps shed some light. It has been established that OSS projects with a wider scope often take on a modular social structure and are decentralized (Crowston and Howison, 2005). It can be assumed that the scope of the projects increases with its functionality and changes in the technology it is based upon. The amount of functionality offered by the end OSS product is manifested in the volume of requirements it faces and the rate of change in technology it is based upon affects the focus of development activity which is explicit in the velocity of change in the requirements it faces. It can also can be argued that communication decentralization in a social network increases as the social modularity of the network increases. Thus forming the final hypothesis of the study the first three hypotheses of the study are stated as follows:

H1: OSS communities facing a lower volume of requirements have a lower value of social network modularity than those facing a larger volume of requirements.

H2: OSS communities facing a lower velocity of change in requirements have a lower value of social network modularity than those facing a higher velocity of requirement changes.

H3: OSS communities with a higher degree of social network modularity experience a lower degree of communication centralization compared to OSS communities with a lower degree of social modularity.

To address the effects of various social structural DCog configurations on the quality of requirements produced, the 6-V model attributes are compared to Génova

et.al's (2013) list of desirable attributes in requirements. Veracity, volatility, vagueness and variance as stated in the 6-V model has parallels in the list of attributes stated by Génova et.al (2013) and can thus be used to measure requirement quality. As seen in the literature review above, stakeholder participation and involvement in RE activities help increase mutual understanding and thus the quality of requirements produced resulting in more unambiguous, concise, well-understood requirements. In an OSS context, communities exhibiting a lower degree of communication centralization point towards increased stakeholder participation and thus higher quality in ensuing requirements. The next two hypotheses of the study can thus be stated as:

H4: OSS communities with a higher degree of communication centralization produce requirements that are less veracious than those produced by communities with a lower degree of communication centralization.

H5: OSS communities with a higher degree of communication centralization produce requirements that are vaguer than those produced by communities with a lower degree of communication centralization.

The variance and volatility in requirements during a release cycle can be directly traced to the scope of the OSS project in terms of the functionalities offered and change in technology, with projects offering higher functionality and facing rapid changes in technology, deploying more diverse artifacts during the development process and thus more heterogeneous design components, and more requirement changes. Thus the final two hypotheses of the study can be stated as:

H6: OSS projects facing a larger volume of requirements and higher velocity of change in requirements exhibit higher variance in requirements than those facing a lower volume of requirements and lower velocity of change in requirements.

H7: OSS projects facing a larger volume of requirements and higher velocity of change in requirements exhibit higher volatility in requirements in a given release cycle than those facing a lower volume of requirements and lower velocity of change in requirements.

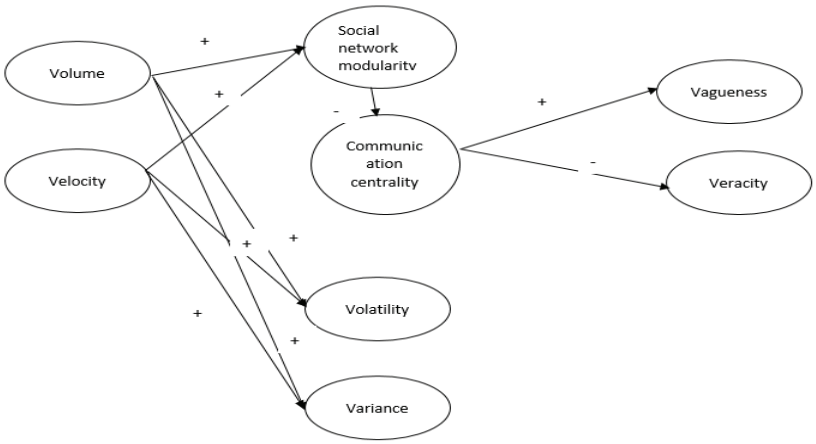


Fig I: Theoretical model of the study

The theoretical model of the study is illustrated above in Figure I.

4 Research Design

The objective of this study is to compare various DCog configurations that an OSS community deploys in response to its external environment and its effect on the quality of its internal requirements. Therefore a multiyear multisite study of successful OSS projects housed in a common repository like Github is proposed that will be compared in terms of the varying environmental factors affecting their DCog configurations (social and structural). Github houses 38 million OSS projects. Out of these projects of diverse size, scope and technologies used only those projects that are production stable and have at least three members in its community will be considered in the study.

4.1. Measurement of Constructs

Eight theoretical constructs are under the radar of investigation in this study – six relating to requirements (volume, velocity, veracity, vagueness, variance and volatility) and two relating to network centrality (social network modularity and communication centrality). The network measures are not discussed in detail here as they are well established in social network studies and have been used in OSS contexts in previous scholarly work as in the Crowston and Howison (2005) study of variations in organization of and communication between social actors in OSS development projects. The operationalization of the six requirement constructs are discussed below:

1. Volume of requirements

Volume is defined in the study as 'the size of requirements pool influencing the scope of the work' (Jarke and Lyytinen, 2014). This can be inferred from the lines of code in each project as well as the number of commits made to the project code repository.

2. Velocity of change

Velocity of change is perceived of as 'the rate at which requirements are changing over time' (Jarke and Lyytinen, 2014). Since our focus is on the velocity of change induced by technological changes in the environment, a qualitative inquiry into the nature of the projects will yield this information. A quantitative measure of the same can be attained by adapting Zowghi et.al's (2002) requirements volatility measure to reflect instability in requirements and change in business environment over multiple release cycles that accrue due to technological changes. Thus the scale can be adapted from Zowghi et.al (2002).

3. Veracity and vagueness of requirements

Veracity of requirements produced is 'the extent to which requirements express the needs of the stakeholders and are consistent' (Jarke and Lyytinen, 2014) while vagueness is defined as 'what extent designers and other stakeholders understand the content and consequences of the requirement' (Jarke and Lyytinen, 2014). These two attributes have been measured using a design science approach formulated by Génova et.al (2013) that involves a textual analysis of requirements and using lexical indicators that allude to the preciseness, consistency, unambiguity, and understandability of requirements. The huge number of projects in our sample prevents such a mode of inquiry and an alternate measurement will have to be developed following the trail of Vlas and Robinson (2015).

4. Volatility of requirements

The volatility of requirements is defined as the 'rate at which requirements change over a given period of time' (Jarke and Lyytinen, 2014). In the context of this study, the time period is the release cycle of the end OSS product. The most commonly used measure of volatility is the percentage change in code via additions, deletions, and modifications. This change in can be inferred from the OSS project's code frequency graph on Github.

5. Variance of requirements

Variance is defined as "The variation in the design scope and consequences of the requirement pool and the heterogeneity of design components involved" (Jarke and Lyytinen, 2014). Lindberg (2015) opines that the variety in design scope and heterogeneity of design components in an OSS project is reflected in the variety of routines prevalent in the OSS project. A higher routine variety signals variety in design scope and heterogeneity in design components which in turn allude to variance in requirements. Lindberg (2015) constructs a measure of routine variety that consists of entropy and routine heterogeneity and uses sequence analysis of pull-requests of an OSS project to measure the same and these measures can be used to operationalize the variance of requirements.

4.2 Data Collection and Analysis

Quantitative data for the study will be collected from survey questionnaires as well as digital traces of the projects on Github. The survey questionnaire will be quantitatively analyzed to measure requirement constructs such as vagueness, veracity and volatility. Scripts developed in the data mining toolkit by Gousios & Spinellis (2012) can be used to capture every activity related to each pull request during the given time period. The activity sequences can be analyzed computationally using sequence analysis as well as qualitatively as texts of bug reports, discussions around how to fix bugs, and how the eventual code fixes were done to measure variance in requirements. Social network analysis will be done on the network data of each project (available on Github) to deduce social and

communication centrality of each project. A logit regression will be done to test hypotheses 1-7.

5 References

Cheng, B. H., and Atlee, J. M. 2009. "Current and Future Research Directions in Requirements Engineering," In *Design Requirements Engineering: A Ten-Year Perspective*, K. J. Lyytinen, P. Loucopoulos, J. Mylopoulos, and W. N. Robinson (eds.), Berlin, Germany: Springer, pp. 11–43.

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

Crowston, K., Li, Q., Wei, K., Eseryel, Y. U., and Howison, J. 2007. "Self-Organization of Teams for Free/Libre Open Source Software Development," *Information and Software Technology* (49:6), pp. 564–575.

Crowston, K., Wei, K., Howison, J., & Wiggins, A. 2012. "Free/Libre Open-source Software Development: What We Know and What We Do Not Know," *ACM Computing Surveys (CSUR)* (*44*:2), pp. 7-43.

El Emam, K., Quintin, S., & Madhavji, N. H. 1996. "User participation in the requirements engineering process: An empirical study," *Requirements engineering*, *1*(1), 4-26.

Ernst, N. A., and Murphy, G. C. 2012. "Case Studies in Just-In-Time Requirements Analysis," In *IEEE 2nd International Workshop on Empirical Requirements Engineering*, pp. 25–32.

Galbraith, J. R. 1973. *Designing complex organizations* (p. 150). Addison-Wesley Pub. Co.

Galbraith, J. R. 1974. Organization Design: An Information Processing View. *Interfaces*, (*4*:3), pp. 28–36. doi:10.1287/inte.4.3.28

Génova, G., Fuentes, J. M., Llorens, J., Hurtado, O., & Moreno, V. 2013. "A framework to measure and improve the quality of textual requirements," *Requirements Engineering*, *18*(1), 25-41.

Gopal, D., Lindberg, A., & Lyytinen, K. 2016. "Attributes of Open Source Software Requirements--The Effect of the External Environment and Internal Social Structure," In *2016 49th Hawaii International Conference on System Sciences (HICSS)*, pp. 4982-4991. IEEE

Gousios, G., & Spinellis, D. 2012. "GHTorrent: Github's data from a firehose," In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on* IEEE, pp. 12-21.

Hansen, S. W., Robinson, W. N., and Lyytinen, K. J. 2012. "Computing Requirements: Cognitive Approaches to Distributed Requirements Engineering," In *2012 45th Hawaii International Conference on System Sciences*, pp. 5224–5233.

Hanseth, O., Lyytinen, K. 2010. "Design Theory for Adaptive Complexity in Information Infrastructures," *Journal of Information Technology* (25:1), pp. 1-19.

Hutchins, E. 1995. *Cognition in the Wild,* MIT Press, Cambridge, MA, pp. 408.

Hutchins, E. 2000. "Distributed Cognition." *Internacional Enciclopedia of the Social and Behavioral Sciences*.

Hutchins, E. and Klausen, T. 1996. "Distributed Cognition in an Airline Cockpit," In *Cognition and Communication at Work,* Y. Engestrom and D. Middleton (eds.), Cambridge University Press, New York, pp. 15-34.

Hutchins, E., and Lintern, G. 1996. *Cognition in the Wild*, MIT Press, Cambridge, MA.

Jarke, M., and Lyytinen, K. 2014. "Special Issue on Complexity of Systems Evolution: Requirements Engineering Perspective," *ACM Transcations on Management Information Systems*

Kujala, S., Kauppinen, M., Lehtola, L., & Kojo, T. 2005. "The role of user involvement in requirements quality and project success," In *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on* (pp. 75-84). IEEE.

Lindberg, A. 2015. "The Origin, Evolution, and Variation of Routine Structures in Open Source Software Development: Three Mixed Computational-Qualitative Studies*,"* Doctoral dissertation, Case Western Reserve University.

Mockus, A., Fielding, R. T., and Herbsleb, J. D. 2002. "Two Case Studies of Open Source Software Development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology* (11:3), pp. 309–346.

Noll, J., and Liu, W.-M. 2010. "Requirements Elicitation in Open Source Software Development," In *Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development,* ACM Press, pp. 35–40.

Robinson, W., & Vlas, R. 2015. "Requirements Evolution and Project Success: An Analysis of SourceForge Projects."

Scacchi, W. 2002. "Understanding the Requirements for Developing Open Source Software Systems," *IEE Proceedings Software* (149:1), pp. 24–39.

Scacchi, W. 2009. "Understanding Requirements for Open Source Software," In *Design Requirements Engineering: A Ten-Year Perspective*, K. Lyytinen, P. Loucopoulos, J. Mylopoulos, and B. Robinson (eds.), Berlin, Germany: Springer, pp. 467–494.

Thummadi, B. V., Lyytinen, K., and Hansen, S. 2011. "Quality in Requirements Engineering (RE) Explained Using Distributed Cognition: A Case of Open Source Development," *Sprouts: Working Papers on Information Systems* (11).

Vlas, R., and Vlas, C. 2011. "A Requirements-Based Analysis of Success in Open-Source Software Development Projects," In *Proceedings of the 17th Americas Conference on Information Systems,* Detroit, Michigan.

Xiao, X., Lindberg, A., Hansen, S., and Lyytinen, K. 2013. "'Computing' Requirements in Open Source Software Projects," In *The 34th International Conference on Information Systems* (ICIS 2013).

Zowghi, Didar, and Nur Nurmuliani. 2002. "A study of the impact of requirements volatility on software project performance." *Software Engineering Conference, Ninth Asia-Pacific*. IEEE, 2002.

A Quantitative Analysis of Performance of the Key Parameters in Code Review - Individuation of Defects.

Dorealda Dalipaj[1]

Universidad Rey Juan Carlos, Madrid. dorealda.dalipaj@urjc.es

Abstract - Finding and removing defects close to their point of injection remains the main motivation and a key parameter for review.

Yet, little is known on how this process affects to important parameters of development and deployment processes. Different studies have shown that code review is not performing as expected. They argue that the performance of the process is low and that the actual outcome of code reviews in finding errors is less than the expected one. Furthermore, another study argues that as many software programs rely on issue reports to correct software errors during maintenance, developers spend too much time in identifying bug[1] reports[2] due to duplicated reports.

By analyzing code review repositories, and other repositories containing information about software processes, we expect to better understand how code review affects to the whole development and deployment process, when bottlenecks are caused, when unnecessary delays are found, how expensive code review is in terms of impact on different metrics, and to which extent it has an overall positive impact.

Therefore the main focus of this study is to find proof of the rate at which the defects are discovered and fixed during the code review process. Additionally analysing its overall impact on different metrics and understanding the performance of the process, we will be able to identify if there are any development practices providing more value than the others.

The first step in this study is focused on the analysis of the time that developers need to identify bug reports in the bugtracking repository, the review time and the bugs fixed during the code review process. These are the most fundamental parameters for characterization of performance of the code review process, as pointed out by previous studies. They are also the most important metrics having a positive increasing relation with the benefits of the same process, as pointed out by industry.

[1] In this paper bug and defect refer to the same object.
[2] In this paper bug report, report and ticket refer to the same object.

1 Introduction

Code review, sometimes referred as peer review, employed both in industrial and open source contexts, is an activity in which people, other than the author of a code snippet, examine it for defects and improvement opportunities. Code review is characterized *as a systematic approach to examine a product in detail, using a predefined sequence of steps to determine if the product is fit for its intended use* [8].

There have been different ways of performing defect detection since its beginning up to nowadays. The formal review or inspection according to Fanagan's [9] approach required the conduction of an inspection meeting to actually find defects. Different controlled experiments showed that there were no significant differences in the total number of defects found when comparing meeting-based with meetingless-based inspections [10, 11]. Other studies [12] were carried out. They proved that more defects were identified with meetingless-based approaches. As a result a wide range of mechanisms and techniques of code review were developed. From static analysis [15, 16, 17], which examines the code in the absence of input data and without running the code and is tool based, to modern code review [18, 19, 20], which aligned with the distributed nature of many projects is asynchronous and frequently supporting geographically distributed reviewers. Because of their many uses and benefits, code reviews are a standard part of the modern software engineering workflow.

It is generally accepted that quality in software remains a challenge due to defects presence. A major quality issue with software is that defects are a byproduct of the complex development process, and the ability to develop *defect free* software remains a big challenge for the software community. It is possible to improve the quality of software product by injecting fewer defects or by identifying and removing injected defects.
It is also generally accepted that the performance of software reviews is affected by several factors of the defect detection process. So, code review performance is associated with the effort spent to carry out the process and the number of defects found.

Most empirical studies try to assess the impact of specific process settings on performance. Sources of process variability range from structure (how steps of the inspection are organized), inspection inputs (reviewer ability and product quality), techniques applied to defect identification that define how each step is carried out), context and tool support [13]. A controlled experiment by Johnson and Tjahjono [10] showed that *total defects identified, effort spent in the process, false positive defects, and duplicates* are fundamental variables to analyse when controlling the performance of code review.

2 Discussion

Although code review is used in software engineering primarily for finding defects, several studies argue that the outcome of code reviews in finding errors is less than expected.

Over the past years, a common tool for code review, `Code Flow`, has achieved wide-spread adoption at Microsoft. The functionality of `CodeFlow` is similar to other review tools such as `Mondrian` [18] (adopted at Google), `Phabricator` [19] (adopted at Facebook) or the open-source `Gerrit` [20]. Two studies have been conducted at Microsoft on code review process, with `Code Flow` as case study.

The first study [2] took place with professional developers, testers, and managers. The results show that, although the top motivation driving code reviews is finding defects, the practice and the actual outcomes are less about finding errors than expected: Defect related comments comprise a small proportion, only 14%, and mainly cover small logical low-level issues. The second study [3] stated that code reviews do not find bugs. They found that only about 15% of the comments provided by reviewers indicate a possible defect, much less functionality issues that should block a code submission.

Another empirical study on the effectiveness of security code review [1], conducted an experiment on 30 developers. They conducted manual code review of a small web application. The web application supplied to the developers had seven known vulnerabilities. Their findings concluded that none of the subjects found all confirmed vulnerabilities (they were able to find only 5 out of 7) and that reports of false vulnerabilities were significantly correlated with reports of valid vulnerabilities.

A different experiment argued that in large scale software programs where bug tracking systems are used, developers spend much time to identify the bug reports (mainly due to the excessive number of duplicate reports).

Keeping in mind the above discussion, as a first step towards our scope, we decided to investigate the following:

- Q 1. *What amount of time developers need to identify bug reports?*
- Q 2. *What amount of time developers spend to carry out the review process?*
- Q 3. *What influences the time to review and the time to identify bug reports?*
- Q 4. *Are all the code review processes performing the same in detecting and fixing a low number of defects?*

The analysis of the first and second questions will bring evidence on metrics that do not involve subjective context but are material facts and are very important metrics for the industry. As such, they can be recorded to trace the efficiency and effectiveness of the code review function. The third question serves to individuate the bottlenecks or delays in the whole process leading to further studies on what is the cause of these fenomenas and try to find a solution for them.

Whereas the forth question addresses the results arised by previous studies. They argue that code review is not finding and fixing defects as we expected. Those previous studies conducted on the number and type of defects fixed during code review are performed, to the best of our knowledge, on proprietary projects. We need to verify if these results hold in open source projects. This is relevant for both academy and industry. For academy it is important to further investigate the reasons that cause this effect, while industry needs a lower and upper bound of this parameter as it directly expresses the benefits of the code review process.

With the abundance of data coming from the engineering systems and having a diverse set of projects to observe [6, 7], we ask if there is *any* code review process that provides more value than the others?

To provide an answer to the above question, we are performing a large empirical study on the 213 active projects of OpenStack. For the purpose of our study, we analyse them divided by the *9 core projects* of OpenStack category (see 3), and group the rest in the *Other Projects* category.

OpenStack is a large project that has adopted code reviews on a large scale. It has a reasonable traceability between commits, reviews and defects reports. It uses Launchpad, a bugtracking system for tracking the issue reports, and Gerrit, a lightweight code review tool. Additionally, being an open source cloud computing software, it is backed by a global collaboration of developers. It has other flavors worthy of additional benefits which influences the outcome, and can bring a different picture from the one found in previous literature [1, 2, 3, 4].

In the remainder of this paper, we first describe the necessary background notions for our work (section 3). Next, we describe the case study setup (section 4), then present the results of our questions (section 5). After threats to validity and future work (section 6 and section 7), we discuss some conclusions (section 8).

3 Background

This section provides background information about the bug-tracking and code review environments of OpenStack and the tools for obtaining data from their repositories.

OpenStack is a free and open source set of software tools for building and managing cloud computing platforms. OpenStack is made up of many different moving parts. Because of its open nature, anyone can add additional components to OpenStack to help it to meet their needs. This is why, actually, in OpenStack there are 213 active projects. But the OpenStack community has collaboratively identified 9 key components that are a part of the *core* of OpenStack. These components are distributed as a part of any OpenStack system and officially maintained by the OpenStack community: Nova, Swift, Cinder, Neutron, Horizon, Keystone, Glance, Ceilometer, and Heat. Therefore we will

expose the results grouped by the 9 core components of OpenStack and categorise the rest as Other Projects.

OpenStack uses Launchpad as the issue tracking system. Launchpad is a repository that enables users and developers to report defects and feature requests. It allows such a reported issue to be triaged and (if deemed important) assigned to team members, to discuss the issue with any interested team member and to track the history of all work on the issue. During these issue discussions, team members can ask questions, share their opinions and help other team members. OpenStack uses a dedicated reviewing environment, Gerrit, to review patches and bug fixes. It supports lightweight processes for reviewing code changes, i.e., to decide whether a developer's change is safe to integrate into the official Version Control System (VCS). During this process, assigned reviewers make comments on a code change or ask questions that can lead to a discussion of the change and/or different revisions of the code change, before a final decision is made about the code change. If accepted, the most recent revision of the code change can enter the VCS, otherwise the change is abandoned and the developer will move on to something else.

To obtain the issue reports and code review data of these ecosystems, we used the data set provided by González-Barahona et al. [21]. They developed the MetricsGrimoire tool to mine the repositories of OpenStack, then store the corresponding data into a relational database. We make use of their issue report and code review data sets [22] to perform our study.

4 Case Study Setup

This section explains the methodology used to address our questions. In this paper, we are interested in quantifying and analysing:

- (Q 1) the time that developers need to identify bug reports,
- (Q 2) the time that developers spend to carry out the code review process,
- (Q 3) what influences the time to review and time to identify bug reports,
- (Q 4) the bugs (and possibly their type) that where fixed in the code changes successfully merged to the code base.

Next we discuss the methodology applied for carrying out our study:

1. the selection of the case study system,
2. how we individuated which reports (from Launchpad) were classified as bug reports and how we extracted them for measuring the *time to identify bug reports*,

3. how we linked the issues (bug reports from Launchpad) to their review in Gerrit for measuring the *time to review*,
4. as this is the starting of the PhD, Q.3 and Q.4 are work in progress, thus we will discuss how we intend to carry it out in Future Work (section 7).

4.1 Selection of Case Study System

The case study system choice is OpenStack because for achieving our aims we require projects with a substantial number of commits linked to issue reports and code review. And it is readily done in OpenStack. Furthermore, thanks to MetricsGrimoire tool, we can mine the repositories of Launchpad and Gerrit, which are being systematically updated. What we need to do is to identify and extract the issue reports classified as bugs, link them the respective review and then extract the patterns we need to carry out our results.

4.2 Identifying classified Bug Reports

In Launchpad, besides bugs reports, the developers work with *specifications* (approved design specifications for additions and changes to the project teams code repositories) and *blueprints* (lightweight feature specifications). Identifying which of the reports have been classified as bugs is not a trivial task. Tickets usually are commented. Reviewers do discuss about bugs found in the reports.

But, analysing the comments of a ticket is not the most efficient way for extracting its classification. Not only because we will not identify 100% of the tickets but we risk false positives too.

Manually analysing a number of randomly selected tickets and studying the Launchpad work flow and structure, we found a pattern in the evolution of a report states (which is how new bugs are confirmed):

a) when a ticket, stating a possible bug, is opened in Launchpad, its status is set to *New*;
b) if the problem described in the ticket is reproduced, the bug is confirmed as genuine and the ticket status changes from *New* to *Confirmed*;
c) only when a bug is confirmed, the status then changes from *Confirmed* to *In Progress* the moment when an issue is opened for review in Gerrit.

Thus, we analysed the Launchpad repository searching for tickets that match this pattern. Those are the tickets that have been classified as bugs. Once identified, we extracted them in a new repository for further inspection.

Our results showed that, in Launchpad, 57,720 tickets out of 88,421 have been

classified as bugs[3]. Hence 65.3% of the total tickets in Launchpad are confirmed bugs. For each of these bugs, an issue for fixing has been opened in Gerrit.

At this point we are able to quantify the time that developers spend on identifying bug reports as the distance in time between the moment when the ticket is first inserted in Launchpad up to the moment it is *Confirmed* as a genuine bug.

You can see the numbers and percentages of the extracted bug reports from OpenStack divided by the 9 core projects, the Other Projects category and over all OpenStack, in fig. 1 below.

The percentages of reported issues (tickets) classified as bugs in OpenStack, grouped by projects and last the percentage on the total number of tickets.

Project	Total Number Tickets	Bug Classified Tickets	Percentage of Bug Tickets
Nova	13018	8073	63%
Swift	1681	974	58%
Cinder	4064	2691	66%
Horizon	5349	3526	66%
Keystone	4101	2517	61%
Glance	2895	1826	63%
Ceilometer	1880	1302	69%
Heat	3125	2218	71%
Other Projects	53245	34593	65%
Total OS	88421	57720	65%

Fig. 1. The percentages of reported bugs in OS - From July, 2010 - January, 2016.

4.3 Linking the Issue Reports to the Reviews

The next step is to link the bug reports we already extracted with their respective review in the code review system. To detect the links between ticket and reviews, we first referred to the name of the branch on which a code change had been executed, since some of them follow the naming convention "bug/989868" with "989868" being a ticket identifier.

After the extraction, we manually analysed a random number of reviews and their respective tickets. We discovered that some reviews were matched to a ticket (meaning they were fixing whatever bug of that ticket). But in reality

[3] These results can be looked up in a Python notebook at `http://github.com/ddalipaj/CR\_Defect\_Individuation\_Rate/blob/master/finding\_bugs.ipynb`

the review was merging the fixing in some version of a project (in some cases in the same project, while in other cases in a project different from the one which originated the defect). The merges done in versions of the same (or even different) project, for the preservation of compatibility, clearly are not elements for measuring the time to review.

To quantify the time that developers need to carry out the review process, we must be sure to take into consideration only merges into the master branch of the projects. Thus this selection was clearly erroneous.

We tried another approach. We linked tickets to reviews using the information that we find in the comments of the tickets. Whenever a review receives a proposal for a fix, or a merge for a fix, it is reported in the comments of the respective ticket.

> Precisely, a merge comment looks like the following:
> Reviewed: https://review.openstack.org/100018
> Committed: https://git.openstack.org/cgit/openstack/nova/commit
> /?id=be58dd8432a8d12484f5553d79a02e720e2c0435
> Submitter: Jenkins
> Branch: master ...

The first line, clearly, provides us with the link to the review in Gerrit.

The first problem that arises in analysing the comments is that, for some ticket, they are a summary of a commit history. In these cases, we find more than one match with the pattern we are looking for within the body of the comment, while the commit itself is not a merge in the master branch of the project that originated the defect, consequently not the correct result.

However there is a fixed format of the comments that report a merge (which is the one you can see in the example above). In this format, the information related to the review is stated at the very beginning of the comment. Manually analysing the tickets in Launchpad, we have seen that they are found in the first 6 rows of the comment.

Thus trunking the comments we extracted only the first 6 lines from every one of them. Doing this, we are sure we will identify the right review[4].

At this point, we are able to quantify the time to review in OpenStack as the distance in time between the moments the first patch is uploaded in Gerrit up to when a fix change is merged to the code base.

[4] The steps of this process are contained in a Python notebook `https://github.com/ddalipaj/Analysis\_Tickets\_Issues/blob/master/master\_merge.ipynb`

The table below (fig. 2) shows: on the left, the number and percentage of tickets from Launchpad linked with the reviews in Gerrit that are fixing them; and on the right, the number and percentage of reviews from Gerrit linked with the tickets in Launchpad that they are reviewing.

Tickets from Launchpad		**Issues from Gerrit**	
Number of distinct tickets merged	**Number of distinct tickets linked**	**Number of distinct fixes merged**	**Number of distinct fixes linked**
44799	37080	211207	42200
Percentage of distinct tickets linked to its review	**82,8%**	**Percentage of distinct reviews linked to the tickets**	**20%**
Number of all tickets merged	**Number of all tickets linked**	**Number of all fixes merged**	**Number of all fixes linked**
70587	63739	211207	63739
Percentage of all tickets linked to its review	**90,2%**	**Percentage of all reviews linked to the tickets**	**30,2%**

Fig. 2. The percentages of tickets and issues linked with its counterparts in OS - From July, 2010 - January, 2016.

Our approach was able to link 90.2% of the tickets from Launchpad to their corresponding issue, and 30.2% of the issues from Gerrit to the corresponding ticket (the reason behind the results in Gerrit is that we are not selecting every merge, but only the ones into the master branches).

5 Case Study Preliminary Results

In this section we expose the results that we have obtained for the Q 1 and 2.

As we mentioned before, this is the initial phase of the PhD, thus Q 3 and 4 are currently work in progress.

5.1 Q 1. What amount of time developers need to identify bug reports?

We computed the time for identifying the bug reports as discussed in 4.2. Afterwards, we calculated the median effect size across all OpenStack projects in order to globally rank the metrics from most extreme effect size, and last the quantiles.

We discovered that the median time for identifying a bug report in OpenStack

(Launchpad) is 1.96 hours.

Additionally, we can say that the 1st quartile is less than 5 minutes, the 2nd quartile is 1.96 hours, the 3rd quartile is 71.6 hours (less than 3 days), and the interquartile range (IQR) is 71.4 hours (less than 3 days).

The results are shown in the table below:

Median time for classifying a bug report:		1.96 hours	< 1 days	
Quantiles:				
0.25	273.0 seconds	4.6 minutes		
0.50	7059.5 seconds	117.7 minutes	1.96 hours	
0.75	257626.0 seconds	4293.8 minutes	71.56 hours	2.9 days

Fig. 3. The median time to classify a bug report across all projects in OpenStack - From July, 2010 - January, 2016.

5.2 Q 2. What amount of time developers spend to carry out the review process?

We computed the time to carry out the review process as discussed in 4.3. Again, we calculated the median across all OpenStack projects.

We discovered that the median time for reviewing is 52.17 hours (2.2 days). Additionally, we can say that the 1st quartile is 8.21 hours (0.3 days), the 2nd quartile is 52.17 hours (2.2 days), the 3rd quartile is 213.75 hours (less than 9 days), and the IQR is 205.54 hours (8.6 days).

The results in the table below:

Median time for closing a review:	52.17 hours	< 3 days	
Quantiles:			
0.25	29587.3 seconds	8.21 hours	< 1 days
0.50	187816.0 seconds	52.17 hours	< 3 days
0.75	769489.5 seconds	213.75 hours	< 9 days

Fig. 4. The median time to carry out the review process across all projects in OpenStack - From July, 2010 - January, 2016.

The table in Fig. 5 exposes the median of the time to review during various years (from 2011 up to 2015) for the 9 core projects of OpenStack, the "Other Projects" category of OpenStack, and over all OpenStack (last row). Additionally the last column exposes the median of the time to review over all the history (from July, 2010 to January, 2016) of the above mentioned categories.

We can conclude from the results that in OpenStack and the Other Projects category the time to merge is under control. But we can not declare the same for some of the core projects. See the trend of the median time in Nova, Cinder, Neutron, Keystone and Glance projects.

Project	Year 2011	Year 2012	Year 2013	Year 2014	Year 2015	All History
Nova	0.9	1.0	**5.5**	**11.0**	10.7	5.2
Swift	1.0	0.9	2.5	3.2	4.1	2.8
Cinder	0	1.1	1.9	**5.6**	**5.8**	3.9
Neutron	0.2	1.1	1.5	**6.3**	4.1	3.6
Horizon	0.02	0.3	2.9	4.8	3.4	2,8
Keystone	0.1	1.6	**5.9**	**6.2**	4.9	4.3
Glance	0.4	1.01	**6.7**	**6.7**	**7.1**	4.2
Ceilometer	0	0.9	2.2	5.1	3.1	3.0
Heat	0	0.03	1.3	7.2	4.1	3.1
Other Projects	0.5	0.5	0.9	2.3	1.9	1.8
Open Stack	0.2	0.7	1.5	3.3	2.4	2.2

Fig. 5. The median time to carry out the review process in OpenStack.

5.3 Q 3. What influences the time to review and the time to identify bug reports?

5.4 Q 4. How many bugs (and possibly of what type) are fixed during code review?

Q 3 and 4 are the topic of the work in progress.
We are currently analysing several technical and non-technical factors that may influence the metrics in Q.3, like patch size, priority, review queue, patch writer experience, level of agreement ecc. We are working with two elements of the review process that we dispose, the comments and the commit analysing both the human discussion and the changes in the code to address Q.4.

6 Threats to Validity

Threats to internal validity concern confounding factors that might influence the results. There are likely unknown factors that impact defect-detection that we have not analysed and measured yet.
Due to the elaborate filtering that we performed in order to link two repositories (bug repository, and code review), the heuristics used to find the relations

between them are not 100% accurate, however we used the state-of-the-practice linking algorithms at our disposal. Recent features in Gerrit show that clean traceability between version control and review repositories is now within reach of each project, hence the available data for future of this study will only grow in volume.

7 Future Work

Our immediately future work is to identify the factors that influences the performance of code review process and quantify the rate at which bugs are discovered during this process. We are analysing comments and commits not only to identify the changes in the code that are actually fixing bugs, but also to find patterns that we can use to automate the process of individuating as precise as possible the number of bugs solved during a review. Finally, we can build a tool for monitoring the lower and upper bounds of bugs fixed during code review along with other metrics of performance. There are several factors technical and non-technical that we think influences the performance of code review process (like what influences the time to review (fig. 5)) and that we are investigating.

8 Conclusion

In this paper we empirically studied the impact of the time developers spend to identify bug reports and to carry out the code review process. We are conducting a study to quantify the number of bugs fixed during a code review and individuate the factors that influence the performance of such process.
From the preliminary results that we bring into evidence and the future results that we hope to have, we believe that our study will open up a variety of research opportunities to continue investigating the impact of collaborative characteristics on performance assurance in code review.

9 Aknowledgement

We would like to thank the SENECA EID project, which is funding this research under Marie-Skodowska Curie Actions and Bitergia for providing the tools to mine the repositories used in this study [5].

[5] The preliminary results exposed in this study can be found in three python notebook available online: 1. https://github.com/ddalipaj/CR_Defects_Individuation_Rate/blob/master/finding_bugs.ipynb; 2. https://github.com/ddalipaj/Analysis_Tickets_Issues/blob/master/master_merge.ipynb; 3. https://github.com/ddalipaj/Reviewing_Time_Gerrit/blob/master/reviewing_time_Gerrit.ipynb

References

1. Edmundson, A., Holtkamp, B., Rivera, E., Finifter, M., Mettler, A., Wagner, D. (2013). An empirical study on the effectiveness of security code review. In Engineering Secure Software and Systems (pp. 197-212). Springer Berlin Heidelberg.
2. Bacchelli Alberto, and Christian Bird. "Expectations, outcomes, and challenges of modern code review." Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, 2013.
3. Czerwonka Jacek, Michaela Greiler and Jack Tilford. Code Reviews Do Not Find Bugs. How the Current Code Review Best Practice Slows Us Down. Proceedings of the 2015 International Conference on Software Engineering. IEEE Publisher, 2015.
4. Tao Zhang; Byungjeong Lee "A Bug Rule Based Technique with Feedback for Classifying Bug Reports", Computer and Information Technology (CIT), 2011 IEEE 11th International Conference on, On page(s): 336 - 343
5. IaaS cloud computing of Rackspace Cloud and NASA: https://it.wikipedia.org/wiki/OpenStack and https://www.openstack.org
6. Bug Tracking for OpenStack. https://it.wikipedia.org/wiki/Launchpad
7. Gerrit Code Review for OpenStack. //review.openstack.org/Documentation/intro-quick.html
8. D. L. Parnas and M. Lawford. Inspection's role in software quality assurance. In Software, IEEE, vol. 20, 2003.
9. M. E. Fagan. Design and Code inspections to reduce errors in program development. In IBM Systems Journal 15 pp. 182-211, 1976.
10. P. M. Johnson, and D. Tjahjono. Does Every Inspection Really Need a Meeting? In Empirical Software Engineering, vol. 3, no. 1, pp. 9-35, 1998.
11. P. McCarthy, A. Porter, H. Siy et al. An experiment to assess cost-benefits of inspection meetings and their alternatives: a pilot study. In Proceedings of the 3rd International Symposium on Software Metrics: From Measurement to Empirical Results, 1996.
12. A. Porter, H. Siy, C. A. Toman et al. An experiment to assess the cost-benefits of code inspections in large scale software development. In SIGSOFT Softw. Eng. Notes, vol. 20, no. 4, pp. 92-103, 1995.
13. D. E. Perry, A. Porter, M. W. Wade. Reducing inspection interval in large-scale software development. In Software Engineering, I EEE Transactions on, vol. 28, no. 7, pp. 695-705, 2002.
14. Broy M (2002) Ayewah, N., Hovemeyer, D., Morgenthaler, J. D., Penix, J., Pugh, W. (2008). Using static analysis to find bugs. Software, IEEE, 25(5), 22-29.
15. W. R. Bush, J. D. Pincus, D. J. Sielaff. A static analyzer for finding dynamic programming errors, Softw. Pract. Exper. , vol. 30, no. 7, pp. 775802, 2000.
16. Hallem, D. Park, and D. Engler, Uprooting software defects at the source, Queue, vol. 1, no. 8, pp. 6471, 2003.
17. B. Chess and J. West, Secure Programming with Static Analysis, 1st ed. Addison-Wesley Professional, Jul. 2007.
18. N. Kennedy. How google does web-based code reviews with mondrian. http://www.test.org/doe/, Dec. 2006.
19. A. Tsotsis. Meet phabricator, the witty code review tool built inside facebook. http://techcrunch.com/2011/08/07/oh-what-noble-scribe-hath- penned-these-words/, Aug. 2006.

20. Gerrit code review - https://www.gerritcodereview.com/
21. J. M. Gonzalez-Barahona, G. Robles, and D. Izquierdo-Cortazar. The metrics-grimoire database collection. In 12th Working Conference on Mining Software Repositories (MSR) , pages 478481, May 2015
22. http://activity.openstack.org/dash/browser/data/db/

Analyzing how the bugs are injected into the source code.

Gema Rodríguez-Pérez[1]

gerope@libresoft.es, LibreSoft, Universidad Rey Juan Carlos

Summary. There is an ample research in the software engineering literature on software defects. In the field of mining software repositories, it's very important understand how the bugs are injected into the source code to prevent the system to fail. In the currently literature many studies on bug seeding start with an implicit assumption: the bug fixed had been introduced in the previous modification (i.e., in the previous commit) of those same lines of the source code. However, we have conducted and observational study that proved the assumption that bugs have been introduced in the previous commit and the results, showed this assumption does not hold for a large fraction of the bugs analyzed.

Our objective is shed some light on bug seeding topic by analyzing how the bug are inserted into the source code and understanding why the bug appears in the source whereas the developers are at their disposal code reviews and automatic inspections. We pretend conducted an large observational study that involved bug notifications from a free and open-source cloud computing software platform in order to find some pattern that can help us preventing the bugs.

Key words: Bug introduction, bug seeding, SZZ algorithm, previous commit

1 Introduction

Many efforts on how and why bugs are introduced in the software source code are underway in the software engineering research community. Software source code is affected by many changes, many of them due to failure of the software because of emergent bugs. Developers try to fix them by locating and modifying the line(s) of source code in which the bug is. Concepts such as *bug seeding* help us to find how and where a bug was inserted in the source code, and should be reasonable to assume that last modification, or *previous commit*, of this line or these lines injected the bug.

In spite of the many studies in the area of mining software repositories that are based on this implicit assumption, it is not a trivial task to find

when and where a bug has been introduced in the source code, and thus to identify who introduced the bug. There are some reasons to assume that in some cases the bug may not have been introduced in the previous commit, being other actions such as change in the API that is being called or an older modification the cause for a bug. But in fact, this has been largely ignored in the related work; as anecdotal evidence in papers of different areas of research the following statements can be found:

- in bug seeding studies, e.g., *"This earlier change is the one that caused the later fixed"* [20] or *"The lines affected in the process of fixing a bug are the same one that originated or seeded that bug"* [9],
- in bug fix patterns, e.g., *"The version before the bug fix revision is the bug version"* [15],
- in tools that prevent future bugs, e.g., *"We assume that a change/commit is buggy if its modifications has been later altered by a bug-fix commit"* [4].

While performing research on the topic, the unique empirical evidence found in the literature that supports this assumption is based on a manual verification of 25 random bug-fix commits with some improvements in the use of the SZZ algorithm, concluding that the SZZ intuition in which the change previous to a bug fix introduces the bug is fulfilled [20]. But this empirical evidence is not enough, it only takes a small population of bug-fix commits, and we need more empirical evidence because of the assumption can be found frequently in the literature. That is the reason why we decided to investigate its validity in the case of a large project, such as the OpenStack, pinpointing the origin of a bug in the source code and devoting significant effort to understand the causes.

The many changes to the code in this project enables us to identify bug reports in which the bug had not been introduced in the previous commit. As we have mentioned before, an example of this could be a change in the API that is being called. For instance, the code presented below shows a real code extracted from OpenStack in which a certain volume doesn't work with multiple backend enable, due to in the current design it was not necessary, see red lines in *Bug-Insertion (V2)*, but in a certain moment and for some reasons, the community need it. So, the fixed bug added a new value in API call, see green lines in *Bug-Fix (V5):*.

The Figure 1 shows an example of the history of commits done in a file, we can see the current version, *V6*, the commit that fix the bug, *V5*, the commit that injected the bug according with SZZ intuition, *V2*. The commit *V1* is the first time that the function involved in the bug fix appear in the file, and the commits *V3,V4* are different states of the file.

Before Bug-Insertion (V1):

```
dbb854635 xioaxi 2013-07-11        def _check_backup_service {self,volume}:
afd69a95b victor 2013-08-28        """Check if there us an backup service available."""
dbb854635 xioaxi 2013-07-11        topic = CONF.backup_topic
```

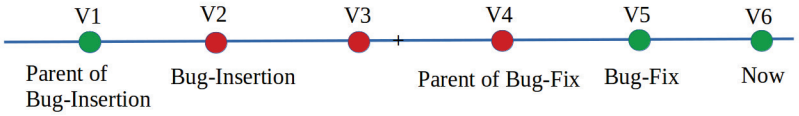


Fig. 1. Control version history with the bug-insertion commit and the bug-fix commit

```
        ...                                    ...
dbb854635 xioaxi 2013-07-11                    srv['host']==volume['host'] and not srv['disable']
```

Bug-Insertion (V2):

```
d6dd5cdfa keniche 2013-09-05        def _is_backup_service_enabled {self,volume}:
afd69a95b victor 2013-08-28         """Check if there us an backup service available."""
dbb854635 xioaxi 2013-07-11         topic = CONF.backup_topic
        ...                                    ...
dbb854635 xioaxi 2013-07-11                    srv['host']==volume['host'] and not srv['disable']
```

Bug-Fix (V5):

```
bd5c3f5a0 Edward 2013-09-26         def _is_backup_service_enabled{self,volume,volume_host}:
afd69a95b victor 2013-08-28         """Check if there us an backup service available."""
dbb854635 xioaxi 2013-07-11         topic = CONF.backup_topic
        ...                                    ...
bd5c3f5a0 Edward 2013-09-26                    srv['host']==volume_host and not srv['disable']
```

In this example, according to SZZ algorithm the previous commits *d6dd5cdfa and dbb854635* inserted the bug; due to some drawbacks in this algorithm, the approach carried on [13] added some improvements removing some false positives. So according with this approach, the commit *dbb854635* was who inserted the bug, because the other commit, *d6dd5cdfa*, only modified the name of the API call and it is a false positive. But this intuition in this example in not fulfilled, because of the two lines involved in the bug-fix which were inserted in the *V1* haven't any bug, this function works fine at these moment.

The goal of my PhD, at this moment, is to find out to which extent the cause of bugs can be attributed to the previous commit, and shed some light to understand why and what the bug appeared. This is the reason why we conducted an observational study that involved issue notifications from the most active components in the OpenStack project. From the issue notifications we have to analyze only those that are bug notifications, so before analyzing who and what caused a bug, we have to be able to identify the bugs reports. We think that this observational study is the first step to identify and understand how, when and why the bug was injected into the source code.

In detail, we would like to answer our main research question, *How the bugs are seeding in the code?*. But first, we attempt to address the following research questions, some of them are methodological questions:

- RQ1: Which reports in the issue tracking system are (real) bug reports?

- RQ2: How often is the previous commit the cause of the bug?

The results from the above research questions, give cause for attempting address the following research questions in a future study;

- RQ3: How does researching this help us improve software code quality?

The remainder of this paper is structured as follows; first, we present the related work in Section 2, in Section 3 we detail the two stages carried out in our methodology. Results from the two stages are then shown in Section 4. Finally, we present the future work and the possible applications of our findings in Section 5 .

2 Related Work

The most used algorithm to locate automatically bug-introducing code changes by linking information from version control system to a issue tracking system repository has been proposed by Sliwerski et al. [19]. This algorithm is an approach of previous works [2, 5, 6] and it is a well-known algorithm, called SZZ.

The SZZ algorithm has some limitations linking bug fixing commit and to bug-introducing commits, because of it only looks for some special keywords in commit messages (e.g., "Bug" or "Fix" [11]). If a bug-fix commit message does not contain the keywords, this bug-fix commit will be ignored. This is the reason why there are articles that suggest improvements on SZZ algorithm such as the Kim et al. [13], it uses annotation graphs instead of CVS annotation to locate, in the previous versions, the lines involved in a modification or a deletion. Also, they discarding some false positives as blank spaces, changes in the format or changes in the comments.

There are studies based on SZZ, Williams et al. have revisited the SZZ algorithm to track bug-inducing changes and identify types of changes [20]. Yang et al. apply SZZ to find what kind of bug-inducing changes are likely to become a great threat after being marked as bug-fix changes [21]. Bavota et al. reported an empirical study about what extent refactoring activities induce faults obtaining that a few percentage of bug fixes ere introduced by a refactoring [1]. Kim et al. show how to classify file changes as buggy or clean using change information features and source code terms [12].

Furthermore, other articles have used the SZZ algorithm to measure quality assurance in source code. Matsumoto et al. study the effect of developer features on software reliability resulting that the modules touch by more developer contained more faults [14]. Also bug prediction models, that help allocate quality assurance efforts, used the SZZ in their approach such as Kamei et al. who apply it to validate effort-aware the two common bug-prediction models; (1) Process metrics LOC, (2) package level predictions and file level predictions [10]. Eyolfson use it to study if time of the day and developer experience

affect the probability of a commit to introduce a bug [3]. Izquierdo et al. use the SZZ algorithm to see if developers are fixing their own bugs [9].

Other approach to identify the origins of a bug is describes by Sinha et al. [18]. Their technique is not a text-based technique, as the SZZ algorithm, the authors analyzed the effects of bug-fix changes on program dependencies. Taking into account the semantic of the source code they achieved more accuracy identifying the origins of a bug. These two approaches have the similar ideas: (1) find the differences between the bug-fix version of and the previous version of the file to recognize those changes done by bug-fix commit, (2) look back in the code revision history until identify which version touched the lines affected in the bug-fix for the last time.

3 Methodology

Until now, we have carried out an empirical study where 459 issues reports were analyzed. These issues reports are called tickets in OpenStack, and were taken randomly from four of the more actives repositories of OpenStack project, `Nova, Cinder, Neutron and Horizon`. The OpenStack project was particularly of interest because of its highest scope and heterogeneous nature with hundreds of developers (more than 5,000 developers) contributing to provide an infrastructure for the worlds largest brands such as Ericsson, Dell, Nokia or EBAY. OpenStack has about 184,000 tickets, of which more than 144,000 have been closed by more than 6,000 developers in all its history, and it has more than 233,000 commits with more than 2 million lines of code[1]. All its history is saved and available in a version control system[2], as well as its issue tracking system (Launchpad[3]) and the source code review system (Gerrit[4]). These statistics have been extracted from the publicly available OpenStack database[5].

The study consists of two stages in which at the end we obtain a classification. In the first, three researchers with programming knowledge have worked in parallel analyzing and classifying tickets using a double blind review process. In the second, only one researcher analyzed the cause of the bug.

3.1 First Stage

At this stage, we have to identify what issues found in the Launchpad of each repository are bug reports. This is not a trivial task due to we have

[1] `http://activity.openstack.org/dash/browser/`
[2] `https://wiki.openstack.org/wiki/Getting_The_Code`
[3] `https://launchpad.net/openstack`
[4] `https://review.openstack.org/`
[5] `http://activity.openstack.org/dash/browser/data/db/`

to extract information from different places such as Launchpad, Gerrit or Git; and it has to be done manually to ensure a useful classification. But, as the process is repetitive, we developed a web-based tool[6] that helps in this classification process. The tool offers all relevant information required to decide if an issue corresponds to a bug report or not. The tool extracts automatically the information from the project repositories, and offers a web-based interface which allows for collaboration, traceability and transparency in the identification of bug reports.

During the analysis of the issues, we have to take into account the next parameters for each ticket:

- The title of the issue report
- The description of the issue report
- The description of the fix commit
- The changes to the source code, as sometimes neither the descriptions nor the comments by developers and reviewers in the Launchpad and Gerrit of each ticket, clarified the underlying ticket.

Each ticket was then categorized into one of three following groups:

1. Group 1 (*Bug Report*): The ticket describes a bug report.
2. Group 2 (*Not Bug Report*): The ticket describes a feature, an optimization code, changes in test files or other not bug reports.
3. Group 3 (*Undecided*): The ticket presents a vague description and cannot be classified without doubts or because of the complexity of the issue .

3.2 Second Stage

In this second stage, we only focused on analyzing the previous commit for those tickets classified in the *Bug Report* group. Therefore we had to locate the buggy line or lines, find out the reason of the software failure, and gathering additional information on the context of the project.

To locate the bug seeding moment, we analyzed the lines involved in the bug fix in two different versions of the file or files where they were; first in the bug fix commit and second, in the *parent* commit of the bug fix commit, in the figure 1 *V4*. We refer to *parent* commit as the commit that modified any line of code in the file before the fix-bug commit was done. And this commit usually is different from the *previous* commit in which the modified lines were the same than in the fix-bug commit. The lines modified in the parent commit do not have to be the ones that have been modified in the bug-fix commit, could be independent commits.

This process ensures us that we are looking the correct change, the change in where the bug was inserted. Because sometimes although the bug seeding commit added many lines, if you look the code before this commit you can

[6] `bugtracking.libresoft.es`

check that some of the lines added was there, meaning that came from older commits, and in that case, it is a false positive where the previous commit did not cause the bug.

The analysis was done manually. We used *git blame* to see the previous commit for each line of the involved file. Also, we used *diff* to see the differences between the two files, in our case as the file is going to be the same, between the file in two different moments in the control version system.

We had to discard some *noise* present in our results, some changes in the previous commit could not have caused the bug and we deleted this previous commit such as blank lines, changes in the format, copied lines, changes in source code comments or updates in the version number of a file/software. Finally, we got a list with all the previous commit and we have to look in all those of them whether the bug was inserted in the previous commit and caused the failure.

4 Preliminary Results

A total of 459 different tickets, from the Launchpad of the four main projects in OpenStack: 125 tickets from Nova, 125 tickets from cinder, 125 tickets from Horizon and 84 tickets from Neutron.

4.1 Fist Stage

Three researchers including me, were involved in the first stage, classifying a total of 459 tickets using the tool and carried out a double bind, meaning that each ticket were analyzed by two researchers. The Table 1 shows the classification obtained by each developer after analyzed the tickets; not all the developers analyzed the same number of tickets. As a result, researchers identified 292 tickets in the same group, that is, their results matched in over 70% of the cases. Of those, 209 tickets had been classified in the *Bug report* group, 74 in the *Not Bug Report* group, and 9 tickets classified in the *Undecided* group.

	Bug Report	Not Bug Report	Undecided	Total
R1	(184) 55%	(115) 34%	(35) 11%	334 (100%)
R2	(188) 76%	(54) 22%	(7) 3%	249 (100%)
R3	(188) 56%	(116) 35%	(30) 9%	334 (100%)
Agree	(209) 72%	(74) 25%	(9) 3%	292 (100%)

Table 1. Statistics for each researcher as a result of the classification process. For each researcher R, the number of tickets (and percentages) classified into the three groups is given. The *Agree* row gives the number of tickets (and percentages) where two researchers agreed.

After this, we can answer the first research question because at this moment we have all the data necessary and all the knowledge to can distinguish bug reports from others reports. We have obtained that in at least 72% of the tickets analyzed the bug-fixes were real bug reports.

4.2 Second Stage

Only 189 of the tickets classified as bug reports by both researchers were considered in this stage; At this point we have to analyze whether the previous commit caused the bug or not, in some cases we were unable to decide if this previous commit caused or not the bug.

We have in mind that that a bug-fix could lead changes in many different lines, which could be inserted in different moments by different previous commits. But in fact, not all them may have caused the bug. Hence, the cause could be found in a single previous commit, in many or even in none. But, we need remove some false positives such as blank spaces/lines, changes in the comments or copied lines from previous commits.

At the end, we were be able to identified a total of 348 previous commits, that according with the current assumption caused the 189 bug reports. After discarding the false positives we had to analyze 308 previous commits; Resulting that 152 (49%) of them caused the bug and 114(37%) didn't cause the bug, whereas we have been unable to decide in 42 (14%) cases.

To understand the complexity of identifying the cause of a bug, we focused on measure how many previous commits had each bug reports analyzed. The results are showed in Table 2; obtaining that 131 only had a previous commit, whereas 58 had more than one previous commit.

	One previous commit	More than one previous commit
Cause	(65) 50%	(86) 48%
Not cause	(30) 23%	(82) 46%
Undecided	(36) 27%	(11) 6%

Table 2. Probability of being the cause of a bug depending on if just one previous commit or more than one previous commits are identified.

Finally, answering the second research question we can demonstrate that only 50% of the previous commits analyzed caused the failure in the system, whereas the 37% of them did not introduce the bug in the code source

5 Possible applications and further research

In the results we have observed that the current assumption, based on the SZZ algorithm, does not hold in all cases. Around half of the previous commits analyzed have been identified as the cause for inserting the line with the bug. At this moment we know that exist two types of bug seedings, the ones where the previous commit inserted the bug and those in which the bug were inserted by other actions such as changes in the API. Furthermore, one of the challenges in my PhD is going in depth to understand how the bug is inserted in the source code, and what we can do to prevent the bug. Focusing in find an approach that could be automatize and alert to developers in code review process or in automatic inspection process, in these cases where a clean line at some point in the evolution of the code, could be buggy at other point of the source code.

The next researches attempt look for some pattern in this behaviour, classifying all the bugs which have not been prevented using either machine learning classifiers or support vector machine [17, 16] based on SZZ algorithm to find a pattern. We could provide a classification of all these bugs which were not prevented, as other previous studies have done [12, 10, 14, 8, 7], according to the following variables to avoid fault prone bug fixes:

- The date of the changes.
- The type of changes done in the bug seeding such as changes in a function, changes in an If/Else condition, changes in a name variable and so on.
- The number of lines modified in the bug-fix commit.
- The experience of the developer that fixed the bug.

This classification could contribute to understand if one of the new variables studied is more prone to appear in these kind of bugs. Meaning that in all these cases where the previous commit didn't injected the bug; is there a pattern repeating?. Currently and after the study presented in this paper, our hypothesis supports the idea of the changes in the API are one of the most prone bugs which are not been inserted by the previous commit, and we could try to prevent these type of bugs in a further study. But the difficulty of this next study reside in how we can prevent the bugs inserted in a change whereas we cannot recognize if this change will be a bug in the future.

Also, we extend this study to other projects and other languages such as Java or C due to the compiler interpreter is different in these languages. That way we can ascertain that what we have found occurs regardless of the project analyzed, because the OpenStack may be a special project, where the code is evolving continuously, also the language used is Python. In fact, we can access a huge database of other projects such as Eclipse or Mozilla that will allow us analyze this assumption, and where we will use statistical analysis to compare the results obtained in the different projects.

To answer RQ3 we should reproduce some research works that measure the quality in the software, but applying our hypothesis. Thus, we could compare both results and recognize who is the one that fits better.

In addition, we want to continue developing the tool, and we will try to automatizes second stage. In addition, could be a good idea develop an automatic classifier based on keywords extracted from the issues tracking systems and code review systems that can distinguish bug reports from other issues. An article about the tool has been accepted in the OSS 2016 conference.

References

1. G. Bavota, B. De Carluccio, A. De Lucia, M. D. Penta, R. Oliveto, and O. Strollo. When does a refactoring induce bugs? an empirical study. In *Source Code Analysis and Manipulation (SCAM), 2012 IEEE 12th International Working Conference on*, pages 104–113. IEEE, 2012.
2. D. Čubranic and G. C. Murphy. Hipikat: Recommending pertinent software development artifacts. In *Software Engineering, 2003. Proceedings. 25th International Conference on*, pages 408–418. IEEE, 2003.
3. J. Eyolfson, L. Tan, and P. Lam. Do time of day and developer experience affect commit bugginess? In *Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 153–162. ACM, 2011.
4. M. Fejzer, M. Wojtyna, M. Burzańska, P. Wiśniewski, and K. Stencel. Supporting code review by automatic detection of potentially buggy changes. In *Beyond Databases, Architectures and Structures*, pages 473–482. Springer, 2015.
5. M. Fischer, M. Pinzger, and H. Gall. *Analyzing and relating bug report data for feature tracking*. IEEE, 2003.
6. M. Fischer, M. Pinzger, and H. Gall. Populating a release history database from version control and bug tracking systems. In *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on*, pages 23–32. IEEE, 2003.
7. K. Fujiwara, K. Fushida, N. Yoshida, and H. Iida. Assessing refactoring instances and the maintainability benefits of them from version archives. In *Product-Focused Software Process Improvement*, pages 313–323. Springer, 2013.
8. H. Hata, O. Mizuno, and T. Kikuno. Bug prediction based on fine-grained module histories. In *Proceedings of the 34th International Conference on Software Engineering*, pages 200–210. IEEE Press, 2012.
9. D. Izquierdo-Cortazar, A. Capiluppi, and J. M. Gonzalez-Barahona. Are developers fixing their own bugs?: Tracing bug-fixing and bug-seeding committers. *International Journal of Open Source Software and Processes (IJOSSP)*, 3(2):23–42, 2011.
10. Y. Kamei, S. Matsumoto, A. Monden, K.-i. Matsumoto, B. Adams, and A. E. Hassan. Revisiting common bug prediction findings using effort-aware models. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1–10. IEEE, 2010.
11. Y. Kamei, A. Monden, S. Matsumoto, T. Kakimoto, and K.-i. Matsumoto. The effects of over and under sampling on fault-prone module detection. In *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, pages 196–204. IEEE, 2007.

12. S. Kim, E. J. Whitehead Jr, and Y. Zhang. Classifying software changes: Clean or buggy? *Software Engineering, IEEE Transactions on*, 34(2):181–196, 2008.
13. S. Kim, T. Zimmermann, K. Pan, and E. J. Whitehead Jr. Automatic identification of bug-introducing changes. In *Automated Software Engineering, 2006. ASE'06. 21st IEEE/ACM International Conference on*, pages 81–90. IEEE, 2006.
14. S. Matsumoto, Y. Kamei, A. Monden, K.-i. Matsumoto, and M. Nakamura. An analysis of developer metrics for fault prediction. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, page 18. ACM, 2010.
15. K. Pan, S. Kim, and E. J. Whitehead Jr. Toward an understanding of bug fix patterns. *Empirical Software Engineering*, 14(3):286–315, 2009.
16. S. Shivaji, E. J. Whitehead, R. Akella, and S. Kim. Reducing features to improve code change-based bug prediction. *Software Engineering, IEEE Transactions on*, 39(4):552–569, 2013.
17. S. Shivaji, J. E. J. Whitehead, R. Akella, and S. Kim. Reducing features to improve bug prediction. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, pages 600–604. IEEE Computer Society, 2009.
18. V. S. Sinha, S. Sinha, and S. Rao. Buginnings: identifying the origins of a bug. In *Proceedings of the 3rd India software engineering conference*, pages 3–12. ACM, 2010.
19. J. Śliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? *ACM sigsoft software engineering notes*, 30(4):1–5, 2005.
20. C. Williams and J. Spacco. Szz revisited: verifying when changes induce fixes. In *Proceedings of the 2008 workshop on Defects in large software systems*, pages 32–36. ACM, 2008.
21. H. Yang, C. Wang, Q. Shi, Y. Feng, and Z. Chen. Bug inducing analysis to prevent fault prone bug fixes. In *Proceedings of the Twenty-Sixth International Conference on Software Engineering and Knowledge Engineering (SEKE 2014)*, pages 620–625, 2014.

Internet of Things and Web Squared: Open for Inclusive Development?

Katja Henttonen
VTT Technical Research Centre of Finland, Espoo
WWW home page: www.vtt.fi

Abstract. Future Internet developments, particularly a phenomenon labeled as "Internet of Things" and "Web Squared", have been predicted to change lives globally. The nature of that change will eventually depend on who participates in "shaping and making" of the future Internet, who has most influence on its evolution and what are their interests. This article discusses 'openness' of the third generation Internet and its relationship with globally inclusive development. This is proposed as a topic of my Phd work. After introducing the topic area, I articulate the research questions, briefly explain the motivation of the research and suggest a methodological approach.

Keywords: Internet of Things, Open source systems, Open development, Web 3.0, Inclusive development, Inclusive innovation, Web Squared

1. Introduction

As information and communication have a central role in the coordination of social activities, it is not surprising that ICTs can play a major role in transforming social life [1]. Until today, successive "waves" of ICTs (e.g. printing press, Internet) have made significant impact on the dynamics of societies worldwide. Recently, the phenomenon labelled as "Internet of Things" (IoT) has been predicted to become the next big ICT wave and change lives globally [2]. This essentially refers to expanding of the Internet into physical objects and organisms, resulting in the merging of physical and digital worlds. A related trend has been described as "Web Squared" [3]: unprecedentedly large amounts of real-time data uploaded by physical objects and/or masses of people can now be stored and effectively analyzed.

These future internet developments have inspired both utopian and dystopian visions. For example, Mahaus and Mavin (cited in [4]) imagine the "Open Revolution" where large masses of people engage in participatory and sustainable innovation. They vision that people become "smart prosumers" instead of "dumb consumers" and switch from competition to collaboration worldwide, which raises levels of respect for both people and the planet. To exemplify a very different thinking, the government of China has promoted an idea of the "Sensing Planet" where all natural processes would be captured digitally (by distributed sensor grids), stored and analyzed, arguably giving people an unprecedented power to control them [5]. This raises an obvious concern on who is in command. Similarly, US military supported research on Smart Dust, tiny microelectromechanical systems,

which could be distributed over a large area to monitor everything from movement to brainwaves, easily sound like a post Orwellian fantasy [6].

The social and developmental impact of the third generations Internet (3GI) will eventually depend on decisions by people who are engaged in its development, uptake and use. This paper proposes "openness" of the third generation internet as a topic of my Phd work. The openness is not understood solely in terms of property rights, but much more widely, as possibilities of different stakeholders to participate in the creation on the future Internet - its technologies, ecosystems and governance processes. The 3GI openness is studied in relation to inclusive development, i.e. the ability of marginalized groups, particularly those living under poverty line, to take part in these developments as agents and/or beneficiaries.

The rest of the paper is structured as follows. The next section gives basic background information on the main topics: Internet of Things and Web Squared, open models to organize social activities and inclusive development. The third section articulates research questions and fourth presents motivation for the study. The fifth and sixth sections propose a research approach, explaining theoretical underpinnings and methodological choices.

2. Background

2.1. Internet of Things and Web Squared

"Internet of Things" is essentially about augmenting the "things" of the physical world (objects, places and even organisms) with embedded computing, sensing and networking capabilities and connecting them with the virtual world of websites and digital services [2] [7]. The "things" can also be connected to social media, allowing masses of people to interact with them [8]. Thanks to recent innovations in cloud computing, sematic technologies and big data management, large amounts of real-time data collected from interconnected "things" and/or crowds of people can be effectively stored, managed and analysed [9]. As the amounts of data uploaded online grow exponentially and mechanism to process that data improve dramatically, the web is becoming more and more "intelligent" [10, 11] O'Reilly and Battelle [3] labelled these developments as "Web Squared". The terms "Web 3.0" or "third generation Internet" (3GI) or "Internet of Everything" has also been used to describe the same developments.

Figure 1 illustrates some building blocks of the future Internet. Digital ecosystems are large networks of digitally connected services, where each participating node adds value to what an end-user will consume or experience [12]. The ecosystems have properties of self-organisation, scalability and adaptability [12] Digital service is any service that can be delivered through an information infrastructure, e.g. web or mobile devices [13]. The "politics" of the Internet (governance and regulation) and dominant business models influence the evolution

of digital ecosystems [14]. Herein, the focus is on human-centred solutions which engage ordinary people as end-users and/or objects being monitored.[1]

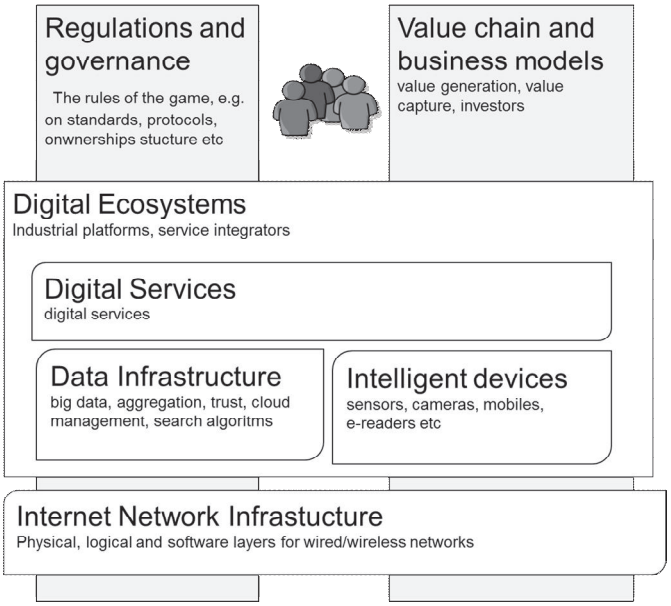


Fig.1. Some building blocks of the third generation Internet (adapted from [14])

It has been suggested that 3rd generation Internet (3GI) solutions could play a role in addressing global development challenges. For example, they have been proposed to enhance supply chain transparency for ethical consumption [11], improve the agricultural yields in poverty-stricken areas [15] and allow people to prepare for natural disasters with new effectiveness [16]. 3GI can also enable radically new models of collaborative consumption and production such as peer-to-peer traffic [17] and peer-to-peer energy trading [18]. However, such developmental applications are still a small niche within mainstream IoT agenda dominated by big companies and military [6]. Further, especially when dealing with the public sector, ICT companies have been critiqued of selling IoT technologies under the cloak of social and environmental sustainability despite evidence of the opposite [19].

One also needs to note that new technologies have only limited capacity to change the status quo, as evident from the developments of the current Internet [20]. Even though Internet of Things and Web Squared enable unprecedented amounts of

[1] Iot-based machine-to-machine (M2M) communication in manufacturing, i.e. "Industrial Internet" is not in the focus directly but obviously M2M forms the basis for human-centred IoT services.

data to be collected and analysed, the existing power relationships still serve to make some information visible over other [11]. Further, over 90% of the people from the 48 poorest countries lack internet connection [21] and an important consideration is what 3GI offers for them.

2.2 Openness, Development and ICTs

According to Relly and Smith [22], social (or socio-technical) systems can be thought to be "open" along three dimensions: content, people and process (see **Figure 2**). The dimension of content mirrors the two freedoms embedded in many commons-based licenses (e.g. open source licences): the freedom to access and the freedom to manipulate the content (reuse, revise, remix, redistribute) [22]. The second dimension of people refers to who can actively participate and collaborate within the system [22]. This extends much beyond technical collaboration tools to social issues such as individual and organizational capabilities for collaboration. The third dimension of openness is one of processes and divides into two elements: openness as transparency and openness as contingency [22]. The former tells that information about a process is accessible while the latter is related to participation, i.e. the results of the participatory activity emerge contingently from inputs of the participants.

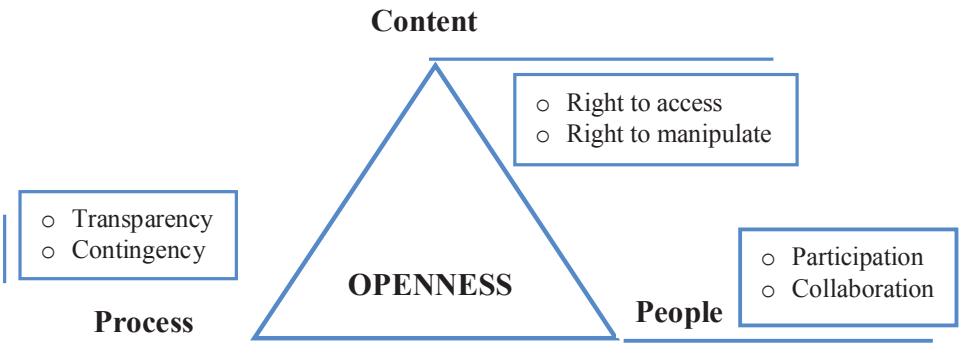


Fig. 2. Dimensions of 'openness' as a concept (illustration based on [22])

The term *open development* has recently been introduced by scholars interested in the role of ICTs in global developments. According to Smith et al. [23], open development refers to catalyzing "positive bottom-up change through open information-networked activities in international development". Openness is herein defined as a way of organizing social activities that favours (1) universal over restricted access, (2) universal over restricted participation and (3) collaborative over centralized production [1]. These three elements of openness are seen as a continuum, where each prior element is a pre-requirement for the following one. For example, open co-creation of e-government services requires citizens, not only to

have access to relevant information and technologies, but also to have sufficient motivation and capacities to participate in public institutions.

As open development theory studies the role of new ICTs in open processes for development, there are two different but inter-related research interests [24]:

1. openness of ICT systems with development goals (e.g. open development data, open co-creation of e-government services, open source software for humanitarian aid etc)
2. ICTs enabling other open process for development (e.g. participatory budgeting, open educational content online, peer-to-peer information sharing for farmers etc)

The two issues are interconnected because, arguably, genuinely open and participative processes can only be facilitated by ICTs developed with (some degree of) openness [24].

Beyond development studies, the openness in ICTs has been widely studied by researchers of open source software (e.g. [25–27]) and technology innovation (e.g.[28–30]), for example. The "people dimension" of openness has been studied in-depth on the field of participatory design and co-creation [31, 32]. **Table 1** gives examples on how openness (or lack of it) has been understood in relation to few common ICT-related artefacts and processes. Even though the examples are simplified and debatable, the information illustrates that openness is not a dichotomy but there are several degrees of openness. It also (implicitly) demonstrates that there are several different forms of openness and there may be trade-offs between them.

Table 1: Examples of less or more open ICT-activities (loosely based on [1, 22, 26, 33] etc)

Artefact	Less	← Openness →	More
Devices	proprietary/patented hardware	devices with open source software	collaborative open hardware development
Software	traditional proprietary software development	e.g. open APIs, open source spin-outs, inner/controlled source development	collaborative open source software development
User Interface	top-down design	e.g. user-centric design, personalized UI	co-creation of UI with users, users as innovators
Standards	monopolistic proprietary standards	open standards with proprietary extensions	collaboratively maintained open standards
Data	closed data (only available to its owner)	shared data, commercially licenced data	open data

ICT policy	closed, non-transparent policy processes	transparent and consultative policy making	open and participative policy-making
Online content	restricted content	free access content	collaboratively created and openly available content

The aforementioned model by Smith et al. [24], where the three 'progressive' levels of openness (access, participation and collaboration) are placed on a continuum, is also suited for conceptualizing 'openness' of ICT systems development. **Figure 3** exemplifies how openness can be understood in case of a single technical artefact like software or hardware.

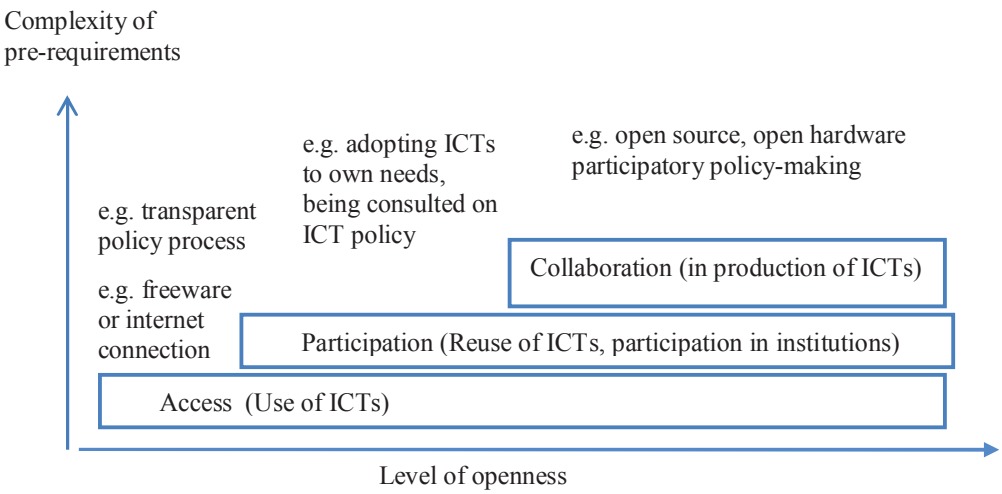


Fig. 2. Elements of openness in ICT development (adapted from [24])

The first level of *access* is about availability of an artefact, the second level of *participation* is about ability to adapt the artefact (reuse/modification) and the third level of *collaboration* relates to peer-production models such as open source development. In case of ICT policy making, access is about transparency of a top-down policy process, participation is about being consulted and collaboration is about fully participatory policy making. The amount of technical and organizational pre-requirements increases when moving from access to participation or from participation to collaborative production [24][2]

[2] There are scenarios suggestions that the relationship between the three elements is more complex. For example, in some cases, collaboration might be pre-requirement to accessing the resulting artefacts resources; in another cases it may not grant access to the results at all [44].

2.3 Inclusive vs. open development?

The term *inclusive development* has been used to describe socio-economic development which emphasizes inclusion of previously disadvantaged and excluded groups as agents and beneficiaries in development processes [34]. While a related term, inclusive growth, refers to equity of economic opportunities, inclusive development has a broader definition and is interested in "equity of well-being" [34]. Inclusive development is pro-poor development by definition [34] and it is also socially sustainable develoment [35]. This is because highly exclusive systems do not tend to prioritize the needs of the poorest, leave alone encourage equal access to opportunities and resources [34]. Inclusion can also be regarded important for ecological aspect of sustainable development, at least if we suppose that most promising environmental innovations emerge bottom-up [35].

Where new technology meets development, the term *inclusive innovation* is often used [36, 37]. While innovation means any invention that creates public or private value, inclusive innovation aims to create value for previously disadvantaged groups [36] . The target group to be included can be any (e.g. women, disabled) but most often it refers to the globally poor, i.e. the billions living on lowest incomes [37]. Like openness, also inclusion can be understood as a continuum from totally exclusive to inclusive [37]. For example, innovation can be understood to be "more" inclusive if the poor are included as innovators, not merely as beneficiaries of innovations [37].

At first glance, the open development theory (see 2.2.) may seem to be in perfect harmony with the ideas on inclusive development and inclusive innovation. By definition, openness is about processes which place little or no restrictions on who can access information or technology, participate in particular institutions or engage in co-production. For example, open technologies (e.g. open source software, open hardware etc) allow for local adaptation and innovation without centralized control [38]. Similarly, open government data and open co-creation can potentially shift e-government design priorities towards real needs of citizens [39]. Von Hippel [40] wrote about "democratizing innovation" and May [41] likens open ICT processes to Mumford's [42] "democratic technics", ie. utilization of technology in a manner that empowers local groups and enables bottom-up innovation and societal change.

However, "open" is not at all synonymous with equal. As many commentators have pointed out, existing social-economic power structures and individual/organizational capabilities influence who is able to exploit an open resource/process and how [11, 22]. For example, the digitalization and opening of land records in an Indian province reportedly lead to increased corruption and hastened the loss of lands by small-scale farmers [43]. For another example, some experiences suggest that open policy processes can be very vulnerable to corporate capture and, therefore, non-transparent and centralized governmental efforts may lead to more equitable policy outcomes [44]. The examples illustrate that open processes do not operate in a 'power vacuum' [45]. Much of the debate on open

development is focused on the questions of inclusion and exclusion, i.e. does openness "empower the empowered" or does it open up new opportunities for the previously disadvantaged [45, 46].

3. Research questions

The goal of the research could be expressed as follows:

To understand and conceptualize the role of openness in 'making' the 3rd generation Internet (3GI) to serve inclusive development

To achieve the goal, I would essentially need to answer questions on what, why and with what results, i.e. what the openness means in the context of 3GI exactly, why do these patterns of 3GI openness emerge and how do they contribute to inclusive development. To correspond with this logic, three preliminary research questions were formulated as follows:

1. In what ways and to what extent can 3GI infrastructure and applications be open?
2. What factors drive or hinder the construction of 3GI towards a degree of 'openness'?
3. How does the 'openness' of 3GI influence the inclusivity of development outcomes?

It is worth noting that 3GI initiatives are in an early experimentation phase, especially in the developing countries, and it might be a decade or more before their socio-economic impacts can be meaningfully analysed. The focus of the research is consequently biased towards "upstream" (pre-requirements and production) instead of downstream (uptake, impact) processes associated with 3GI solutions. The answer to the third question is therefore partial and focused on immediate development outcomes of 3GI construction. The inclusivity of development outcomes of uptake/use are also analysed if sufficient data is available.

The logic of reasoning herein leans strongly towards a socially determinist position: the outcome and impacts of technologies are not perceived to be dependent on their "inherent" properties but on people developing and using the technologies [47]. Thus, the starting point is on how and by whom the technologies, services and ecosystems of the 3rd generation Internet are constructed. These constructions are assumed to influence how the new ICTs can be used by people, which in turn influences their development outcome and impacts.

4. Research justification

The selection of the topic is grounded in my deep interest in how the third generation internet (3GI) will change societies globally. The key question is essentially about who can participate in the 'shaping and making' of the third generation internet, with what motives and with what results for the inclusivity of

international development. If we believe those who associate 3GI with fundamental societal changes [2, 16], these questions are of obvious importance. They are essentially related to sustainable development, which is about inter- and intra-generational justice in the distribution of resources and opportunities.

The subject is topical now when many consider the openness of the Internet to be at cross-roads [11, 48, 49]. There is a vast body of research [1, 24, 50–53] on Web 2.0 technologies enabling participatory governance and peer-to-peer activities for development. These participatory solutions have emerged largely due to an open nature of the current Internet [24]. Technology-wise, 3GI could offer many opportunities in this regard (see 2.1) but it is not yet known whether the creation of such solutions for development will be equally easy within the Internet of Things framework [1, 4]. Openness and inclusions are key issues here, especially if one appreciates bottom-up approaches to international development.

The 3GI openness has already been studied on the level of Internet governance, particularly from the transparency viewpoint, but it tends to be focused on multi-stakeholder institutions (e.g. ITU, IETF) whose role is being weakened by the privatization of Internet [48]. There is very little research on what openness means on the level of digital ecosystems and integration platforms, except for comparisons between the supposedly 'open' mobile ecosystem by Google and the gate-walled one by Apple [54]. One dimension of 3GI openness has been studied on the field of participative design, e.g. by investigating "Maker Spaces" and other groups co-producing IoT devices and services [55, 56]. However, there seems to be a need for more holistic understanding and conceptualization on how and to what extent 3GI infrastructure and applications can be 'open' and why do these patterns of openness emerge.

Open development theory has been critiqued for lacking means to facilitate inclusion of economically and socially marginalized groups into 'open' processes (see section 2.3). Researchers have been debating on whether open processes tend to entrench those who are powerful and well-resourced already or might they actually support transformation towards more equal distribution of power, resources and opportunities [22, 44, 45]. Except for the aforementioned techno-political account by May [41], the debate bases mostly on positive and negative experiences derived from 'random' case studies. There seems to be a need for more in-depth analysis on the relationships between openness and inclusive development.

To summarize, the Phd work can fill a genuine research gap and benefit fellow researches on the following areas: a) conceptualizing openness in the context of third generation Internet and understanding how that openness emerges and b) understanding and conceptualizing relationship between openness and inclusive development (particularly in the context of 3GI but results could be applicable beyond). The results can also support ICT4D practitioners in their decision-making, e.g. in assessing the claims of ICT companies who readily market all of their 3GI solutions as 'open', 'participative and 'empowering' [33, 57].

5. Philosophical and theoretical underpinnings

As to epistemology, I am sympathetic towards the *interpretivist* tradition. I strongly subscribe to the general idea that the study of the social world needs a very different outlook of research procedures from the study of the natural world (see e.g. [58]). However, my interest in the relationship between action and knowledge attracts me towards a pragmatic stance. Consequently, the ontological and epistemological footing of the work might be better described as *critical realism* [59, 60] This position combines ontological realism, a belief that some mind-independent reality is 'out there', with epistemological relativism, a position that our knowledge of the reality is always conceptually mediated and thus approximate or probabilistic at its very best [61]. Thus, the study aims to understand (and even explain to a degree) social structures and mechanisms which are assumed to exist beyond mere perceptions of research participants. This understanding of empirical reality is taken as a basis for assessing theoretical abstractions.

The study employs both deductive and inductive circles of data collection; with emphasis on the latter. In practice, this means that the data is examined for undiscovered patterns and emergent understanding but also for relevance in connection to selected theories [62]. A literature-derived sensitizing conceptual framework will be constructed and used to guide the analysis. The research will draw on studies of open ICT activities (open source etc) and open/inclusive development (see sections 2.2 and 2.3). In addition, the conceptual framework could build on one of the following theoretical perspectives:

- Political economy of technology, with a viewpoint of how political and economic power structures shape (and are shaped by) 3GI developments (e.g. [41, 63]).
- New institutionalism [64, 65], with a viewpoint of how design and evolution of 3GI influences and is influenced by conflicting institutional logics
- Socio-technical transitions [66, 67], with a viewpoint of how 3GI as an innovation challenges existing socio-technical regimes

A pilot case study is conducted to build the conceptual framework and refine research questions (read more in the next section). During the entire research process, it is considered important to remain open to discovering concepts, processes and relationships that are not included in the preliminary framework.

6. Methodological approach

The Phd work is planned to consist of multiple qualitative case studies [68, 69]. The case study approach was selected because it offers access to a wealth of detailed information and an opportunity to stay close to real-life [70]. This helps to develop a nuanced view of reality and contributes to the own learning process due to the close interaction with the subjects of study [70]. In social sciences, knowledge is always context-dependent to a degree. Therefore, to study any interaction between technologies and society, one must typically start with a particular place and

accumulate understanding of research questions in a situated context [7]. Through analytical generalization [71], it is still possible to produce knowledge which is relevant beyond boundaries of the cases.

The exact number of the case studies is to be determined later, but it is likely to be between three and six. The cases are selected strategically to maximize the amount and utility of the information collected. The objective is to find "paradigmatic cases" [70], ie. cases which have prototypical value and are therefore particularly helpful in understanding the target phenomenon. Recognising such cases is not easy, but at least the selected cases should be 3GI initiatives that are of some public importance, set development goals and are preferably undertaken in developing countries.

In the beginning of the study, an exploratory pilot case study is conducted. The primary goals of the pilot case study are to a) decide on a conceptual framework to be used in the rest of the work b) refine research questions and c) readjust methodological choices if needed. The pilot case study is not understood herein as a predictive experiment (as in [72]), but rather as a means to gain preliminary experimental understanding on the topic and thereby improve research design and conceptualization (as in [73]).

After the pilot study, multiple case studies are conducted. Multiple data collection methods are to be employed, the most important ones being *semi-structured interviews* and *participant observation*[3]. In addition, analysis of relevant technical or organizational documents may have a secondary role. Interviews will be recorded and transcribed where possible. As the study focuses on inclusive development, particular care must be taken to ensure that the research process itself does not become exclusive of some stakeholder groups.

Qualitative content analysis [74] is planned to be used as an approach to data analysis (on textual data such as interview transcripts, field notes and documents). More precisely, the method is *directed content analysis* where all codes are not derived directly from data but instead the selected conceptual framework provides guidance for initial codes [75]. Any data that cannot be categorized with the initial coding scheme is given a new code [75]. Following a type of directed content analysis, also called "Template Analysis" [76], a coding template is developed iteratively whilst the analytical process moves forwards.

Figure 4 illustrates the planned research process. The work starts with a first literature review and familiarization with relevant theoretical perspectives. Then, a pilot study is conducted to refine research design and construct a conceptual framework as explained afore. After the pilot study, multiple case studies are conducted in an iterative cycle. The results of each case study feed insights into the selection, design and analysis of the next case. Because research requires staying up-

[3] Because many issues of openness (especially technical ones) are such that it requires expertise to notice them, participant observation might be equally or even more important source of information than the interviews. If the role of participant observation gets emphasized during the pilot study, it could be justified to draw also on ethnographic methods for data collection and analysis.

to-date with the latest 3GI developments, the continuous study of literature and practice is integrated into the research cycle. Eventually, the results are written up, discussed in the light of existing literature/theory and evaluated.

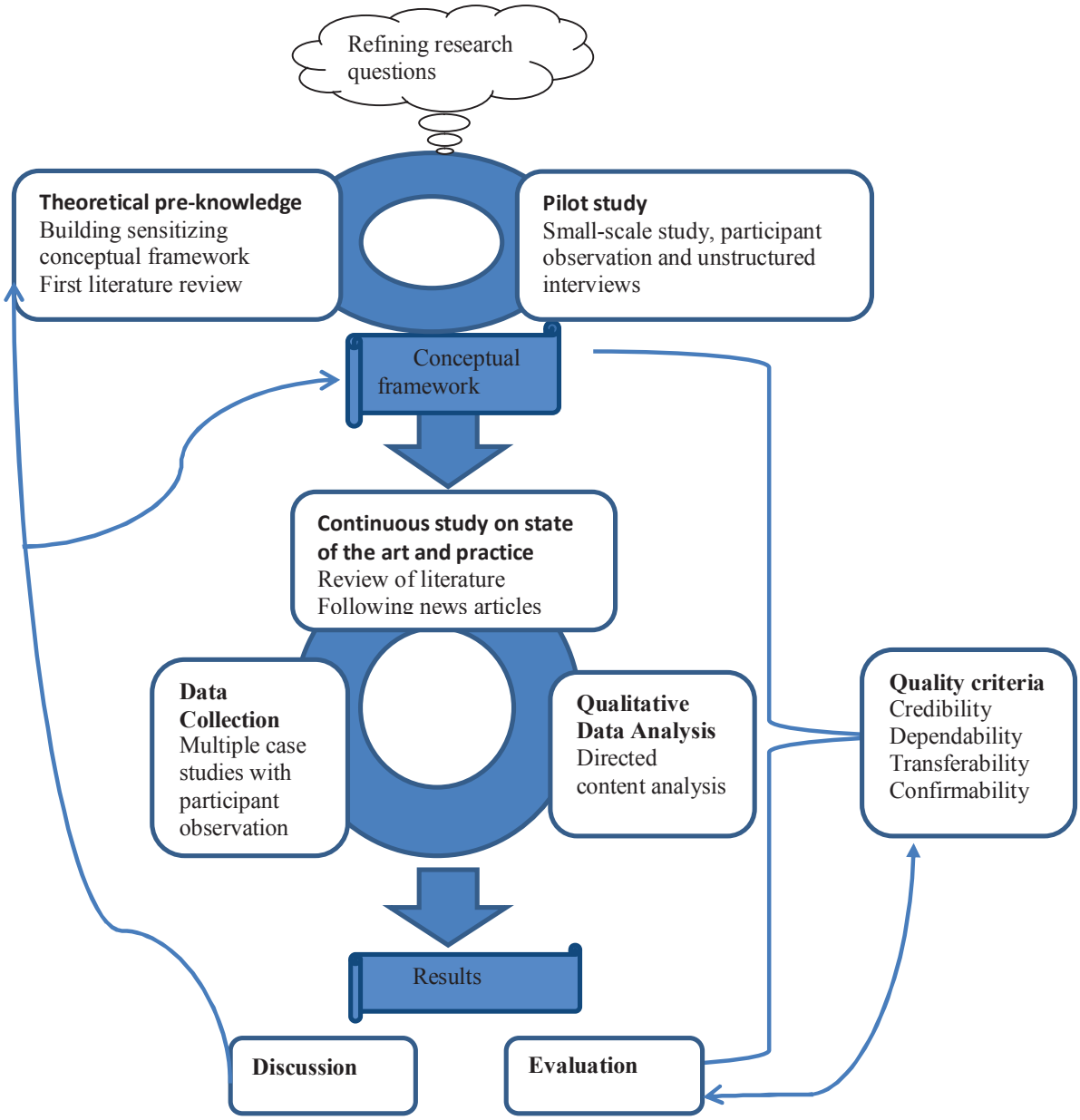


Fig 4. Research process illustrated

The evaluation of the results is based on quality criteria to be decided later, e.g. the one suggested by Guba and Lincoln [77, 78]: credibility (a parallel of internal validity) dependability (a parallel of reliability), transferability (a parallel of external validity) and confirmability (a parallel of objectivity). Herein, *credibility* is concerned on establishing that the findings and interpretations are credible from the perspective of the participants in the research [78]. *Dependability* emphasizes "trackable variance", ie. the task of describing the changes that occur in the research setting and analyzing how these changes affect the study [78]. *Transferability* refers to the degree to which the results can be transferred to other contexts [78] and is thus concerned on the soundness of analytical generalizations made (cf. [71]). *Confirmability* refers to the degree to which the results could be confirmed or refuted by others [78].

Acknowledgements

The publication of this paper was supported by ACCELERATE project funded by VTT and Tekes. Feedback from and discussions with prof. Pasi Tyrväinen (University of Jyväskylä), prof. Richard Heeks (University of Manchester) and prof. Gregorio Robles (Universidad Rey Juan Carlos) were very valuable.

Bibliography

1. Smith, M., Elder, L.: Open ICT ecosystems transforming the developing world. Information Technologies & International Development 6,1 (2010)
2. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of Things (IoT): A vision, architectural elements, and future directions. Futur. Gener. Comput. Syst. 29, 1645–1660 (2013).
3. O'Reilly, T., Battelle, J.: Web Squared: Web 2.0 Five Years On. O'Reilly Media Inc (2009).
4. Penzenstadler, B., Tomlinson, B., Baumer, E.P.S., Pufal, M., Raturi, A.: What will be the Systems Supporting Sustainability in 15 Years? Proceedings of the 2014 conference ICT for Sustainability. Vol. 2. Atlantis Press (2014).
5. van Kranenburg, R., Bassi, A.: IoT Challenges. Commun. Mob. Comput. 1, 9 (2012).
6. Nold, C., Kranenburg, R. van: The Internet of People for a Post-oil World. Vol. 8. Lulu. com (2011).
7. Shin, D.: A socio-technical framework for Internet-of-Things design: A human-centered design for the Internet of Things, Telematics and Informatics 31, 519-531. (2014).
8. Formo, J., Laaksolahti, J., Gårdman, M.: Internet of things marries social media. Mob. HCI 2011 - 13th Int. Conf. Human-Computer Interact. with Mob. Devices Serv. 753–755 (2011).

9. Mayer-Schönberger, V., Cukier, K.: Big Data: A Revolution that Will Transform how We Live, Work, and Think. Houghton Mifflin Harcourt (2013).
10. Calma, G., Palazzi, C.E., Bujari, A.: Web squared: paradigms and opportunities. Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques, 256–261 (2012).
11. Graham, M., Haarstad, H.: Transparency and Development: Ethical Consumption through Web 2.0 and the Internet of Things, Information Technologies & International Development (2011)
12. Li, W., Badr, Y., Biennier, F.: Digital ecosystems. In: Proceedings of the International Conference on Management of Emergent Digital EcoSystems - MEDES '12. p. 117. ACM Press, New York, New York, USA (2012).
13. Immonen, A., Ovaska, E., Kalaoja, J., Pakkala, D.: A service requirements engineering method for a digital service ecosystem. Serv. Oriented Comput. Appl. (2015).
14. Oreste et al: The evolution of the Internet: Towards common capture and surveillance of the collective intelligence? In: The Internet Identity Ecosystem Seminar. W3, Roma (2013).
15. Dlodlo, N., Kalezhi, J.: The internet of things in agriculture for sustainable rural development. In: 2015 International Conference on Emerging Trends in Networks and Computer Communications (ETNCC). pp. 13–18. IEEE (2015).
16. Soler, L.S., Saito, S., Gregorio, L.T., Leal, P., Gonçalves, D., Londe, L., Soriano, É., Cardoso, J., Coutinho, M., Santos, L.B.L.: Challenges and perspectives of innovative digital ecosystems designed to monitor and warn natural disasters in Brazil. In: Proceedings of the Fifth International Conference on Management of Emergent Digital EcoSystems - MEDES '13. pp. 254–261. ACM Press, New York, New York, USA (2013).
17. Big Data and Transport: Understanding and assessing options. Corporate Partnership Board Report, OECD/ITF (2015).
18. Giotitsas, C., Pazaitis, A., Kostakis, V.: A peer-to-peer approach to energy production. Technol. Soc. 42, 28–38 (2015).
19. Viitanen, J., Kingston, R.: Smart cities and green growth: outsourcing democratic and environmental resilience to the global technology sector. Environ. Plan. A. 46, 803–819 (2014).
20. Warf, B.: Segueways into cyberspace: Multiple geographies of the digital divide. Environ. Plan. B Plan. Des. 28, 3–19 (2001).
21. UNESCO, ITU: The state of the broadband 2015: The broadband as a foundation for sustainable development. Geneva (2015).
22. Relly, K., Smith, M.: The emergence of open development in a network society. In: Smith, M. and Relly, K. (eds.) Open Development: Networked Innovations in International Development. The MIT Press, Cambridge (2014).
23. Smith, M.L., Elder, L., Emdon, H.: Open Development: A New Theory for ICT4D. Inf. Technol. Int. Dev. 7, iii–ix (2011).
24. Smith, M., Elder, Gideon, Christian, G., Diga, K., Rashid, A., Flynn-Dapaah, K.: Open ICT4D. Working paper, Ottawa, IDRC (2008).
25. Damiani, E., Frati, F., Riehle, D., Wasserman, A.I. eds: Open Source Systems: Adoption and Impact. In: 11th IFIP WG 2.13 International Conference of Open

Source Systems OSS 2015. , Florence, Italy (2015).
26. Crowston, K., Howison, J., Wiggins, A.: Free/Libre Open Source Software Development: What We Know and What We Do Not Know. ACM Comput. Surv. 40, 1–37 (2010).
27. Fisher, R., Ledwaba, L., Hancke, G., Kruger, C.: Open hardware: a role to play in wireless sensor networks? Sensors (Basel). 15, 6818–44 (2015).
28. Dahlander, L., Gann, D.M.: How open is innovation? Res. Policy. 39, 699–709 (2010).
29. West, J., Gallagher, S.: Patterns of Open Innovation in Open Source Software. Open Innovation: researching a new paradigm (2006)
30. Lerner, J., Tirole, J.: The Economics of Technology Sharing: Open Source and Beyond. J. Econ. Perspect. 19, 99–120 (2005).
31. Sanders, E.B.-N., Stappers, P.J.: Co-creation and the new landscapes of design. CoDesign. 4, 5–18 (2008).
32. Muller, M.J.: Participatory design: the third space in HCI. In: The human-computer interaction handbook: Fundamentals, evolving technologies and emerging applications. pp. 1051– 1068 (2003).
33. Woert, Schuwer, Owerhand: Connecting Various Forms of Openness: Seeking a stronger value proposition. (2015).
34. Kanbur, R., Rauniyar, G.: Conceptualizing inclusive development: with applications to rural infrastructure and development assistance. J. Asia Pacific Econ. 15, 437–454 (2010).
35. Gupta, J., Baud, I., Bekkers, R., Bernstein, S., Boas, I., Cornelissen, V., Iguchi, M., Kanie, N., Kim, R.E., Lima, M.B., Obani, P., Schoof, P., Stevens, C., Zoomeren, D. Van: Sustainable Development Goals and Inclusive Development. (2014).
36. Cozzens, S., Sutz, J.: Innovation in informal settings: reflections and proposals for a research agenda. Innov. Dev. 4, 5–31 (2014).
37. Heeks, R., Amalia, M., Kintu, R., Shah, N.: Inclusive Innovation : Definition , Conceptualisation and Future. Acad. Innov. Entrep. 1–24 (2013).
38. May, C.: Escaping the TRIPs' trap: The political economy of free and open source software in Africa, (2006).
39. Olphert, W., Damodaran, L.: Citizen Participation and engagement in the Design of e-Government Services: The Missing Link in Effective ICT Design and Delivery. J. Assoc. Inf. Syst. 8, 491–507 (2007).
40. Von Hippel, Eric. Democratizing innovation: The evolving phenomenon of user innovation. Journal für Betriebswirtschaft 55 (2005)
41. May, C.: Opening other windows: a political economy of "openness" in a global information society. Rev. Int. Stud. 34, 69–92 (2008).
42. Mumford, L.: Technics and Civilization. (1937).
43. Benhamin, Bhuvaneswari, Manjunatha: Bhoomi: "E-governance." Or, An Anti-politics Machine Necessary to Globalize Bangalore?,". Presented at the .
44. Jeet Singh, P., Gurumurthy, A.: Establishing Public-ness in the Network: New Moorings for Development—A Critique of the Concepts of Openness and Open Development. In: Smith, M. and Reilly, K. (eds.) Open Development: Networked Innovations in International Development. MIT Press (2013).
45. Buskens, I.: The Importance of Intent: Reflecting on Open Development for

Women's Empowerment. Inf. Technol. Int. Dev. 7 (2011).
46. Gurstein, M.B.: Open data: Empowering the empowered or effective data use for everyone? First Monday. 16 (2011).
47. Bijker, W., Law, J.: Shaping Technology/building Society: Studies in Sociotechnical Change. Journal of Social History (1994)
48. DeNardis, D.L.: The Emerging Field of Internet Governance. SSRN Electron. J. (2010).
49. Open Internet of Things: Statement of the Open Internet of Things Assembly at London, United Kingdom on the 17th June, 2012, (2012).
50. Chadwick, A.: Web 2.0: New Challenges for the Study of E-Democracy in an Era of Informational Exuberance. I/S A J. Law Policy Inf. Soc. 5, (2008).
51. Evely, A. et al.: Sustainability 2.0. New Strategies for achieving behaviour change in ore connected world. In: Sustainability: Key Issues (2015).
52. Heeks, R.: ICT4D 2.0: The Next Phase of Applying ICT for International Development. Computer (Long. Beach. Calif). 41, 26–33 (2008).
53. Hamari, J., Sjöklint, M., Ukkonen, A.: The sharing economy: Why people participate in collaborative consumption. J. Assoc. Inf. Sci. Technol. n/a–n/a (2015).
54. Schultz, N., Wulf, J., Zarnekow, R., Nguyen, Q.T.: The new role of developers in the mobile ecosystem: An Apple and Google case study. In: 2011 15th International Conference on Intelligence in Next Generation Networks, ICIN 2011. pp. 103–108 (2011).
55. Hribernik, K. a, Ghrairi, Z., Hans, C., Thoben, K.: Co-creating the Internet of Things - First Experiences in the Participatory Design of Intelligent Products with Arduino. Concurr. Enterprising (ICE), 2011 17th Int. Conf. 1–9 (2011).
56. Füller, J., Mühlbacher, H., Matzler, K., Jawecki, G.: Consumer empowerment through internet-based co-creation. J. Manag. Inf. Syst. 26, 71–102 (2010).
57. Viitanen, J., Kingston, R.: Smart cities and green growth: outsourcing democratic and environmental resilience to the global technology sector. Environ. Plan. A. 46, 803–819 (2014).
58. Weber, R.: The Rhetoric of Positivism Versus Interpretivism: A Personal View. MIS Q. 28, iii–xii (2004).
59. Sayer, a: Realism and Social Science. (2000).
60. Easton, G.: Critical realism in case study research. Ind. Mark. Manag. 39, 118–128 (2010).
61. Groff, R.: Critical realism, post-positivism, and the possibility of knowledge. Routledge (2004).
62. Patton, M.Q.: Variety in Qualitative Inquire. In: Qualitative research & evaluation methods. pp. 75–142 (2002).
63. Taylor, L., Broeders, D.: In the name of Development: Power, profit and the datafication of the global South. Geoforum. 64, 229–237 (2015).
64. Powell, W.W., DiMaggio, P.J.: The New Institutionalism in Organizational Analysis. University of Chicago Press (1991).
65. Immergut, E.M.: The theoretical core of the new institutionalism. Polit. Soc. 26, 5 (1998).
66. Geels, F.W.: The Dynamics of Transitions in Socio-technical Systems: A Multi-level Analysis of the Transition Pathway from Horse-drawn Carriages to

Automobiles (1860–1930). Technol. Anal. Strateg. Manag. 17, 445–476 (2005).
67. Geels, F.W.: Ontologies, socio-technical transitions (to sustainability), and the multi-level perspective. Res. Policy. 39, 495–510 (2010).
68. Walsham, G.: Interpretive case studies in IS research: nature and method. Eur. J. Inf. Syst. 4, 74–81 (1995).
69. Simons, H.: Case Study Research in Practice. In: Planning, Designing, Gaining Access. pp. 28–43 (2009).
70. Flyvbjerg, B.: Five Misunderstandings About Case-Study Research. Qual. Inq. 12, 219–245 (2006).
71. Halkier, B.: Methodological Practicalities in Analytical Generalization. Qual. Inq. 17, 787–797 (2011).
72. Yin, R.K.: Case study methodology R.K. Yin (2003, 3rd edition). Case Study Research design and methods. Sage, Thousand Oaks (CA)..pdf. In: Case Study Research: design and methods. pp. 19–39; 96–106 (2003).
73. Kezar, A.: The Importance of Pilot Studies: Beginning the Hermeneutic Circle. Res. High. Educ. 41, 385–400 (2000).
74. Kohlbacher, F.: The use of qualitative content analysis in case study research. Forum Qual. Soc. Res. 7, Art. 21 (2006).
75. Hsieh, H.-F., Shannon, S.E.: Three approaches to qualitative content analysis. Qual. Health Res. 15, 1277–1288 (2005).
76. King, N.: Template analysis. Qualitative methods and analysis in organizational research: A practical guide. . In: Qualitative methods and analysis in organizational research: A practical guide. pp. 118–134 (1998).
77. Lincoln, Y.S., Guba, E.G.: But Is It Rigorous? Trustworthiness and Authenticity in Naturalistic Evaluation. New Dir. Eval. 15–25 (2007).
78. Guba, E.G.: Criteria for assessing the trustworthiness of naturalistic inquiries. Educ. Commun. Technol. 29, 75–91 (1981).

Predicting Faults in Open Source Software: Trends and Challenges

Malanga K. Ndenga[1,2], Jean Mehat[1], Ivaylo Ganchev[1] and Wabwoba Franklin[3]

[1] University of Paris 8 - LIASD, Saint-Denis, France.
[2] Dedan Kimathi University of Technology, P.O. Box 657-10100 Nyeri, Kenya.
malangalanga@dkut.ac.ke
[3] Kibabii University, Bungoma, Kenya.

Abstract. The ability to accurately identify faulty software components early enough can mitigate risks associated with software failures at a lower cost. A measure of software faults can be used to estimate quality of Open Source Software. The purpose of this study is to identify current trends and challenges in post-release software fault-prediction in relation to: choice of reliable software metrics for predicting post-release software faults; modeling techniques for predicting post-release software faults; and measuring performance of software fault-prediction models. The study reviews 31 empirical studies on software fault-prediction that were published between 2005 and 2015. The review finds out that: software process metrics are better predictors of post-release software faults; statistical and machine-learning techniques are commonly used to develop software fault prediction models; and performance of software fault prediction models is determined using Confusion Matrix, Receiver Operator Curve and Cost-Effective (CE) measures. In conclusion, there is need for replication of software fault-prediction studies so as to generate statistically significant results that can guide software engineering decisions.

Key words: Open Source Software (OSS); software metrics; software faults; prediction modeling; performance measures; cost measures.

1 Background

Software faults in deployed software products have expensive consequences. For example they may result in: massive loss of revenue, huge software maintenance costs or even loss of life for cases of critical mission systems. Software developers should therefore be able to identify and correct software defects prior to releasing software so as to save clients from increased software maintenance costs and customer dissatisfaction. Organizations intending to adopt software products should also be able to determine the quality of the products in terms of reliability and maintenance effort early enough before settling on the software. Stol and Babar [1] believe that one of the main challenge users of OSS

face is evaluation and selection of OSS products. One significant way through which quality of a software product can be estimated is by determining its fault proneness. Fault prediction is possible for a majority of Open Source Software products since the development data for most of these projects is publicly available. Moser, Pedrycz and Succi [2] point out that defect prediction is needed for three main reasons: final product quality assessment, estimating contractual quality standards and making decisions regarding resource allocation for quality assurance activities. Several other researchers concur with Moser, Pedrycz and Succi. For instance, Fenton and Neil [3] argue that organizations are driven to predict the number of defects in software systems before they are deployed so as to gauge the likely delivered quality and maintenance effort of the software. Malhotra and Jain [4] and Xu, Ho, and Capretz [5] believe that a measure of number of defects or fault-proneness of software can be treated as a measure of software quality.

Software quality assurance activities in many organizations are in most cases resource constrained in terms of time, finances and developers [6], [7]. Despite these constraints, Arisholm, Briand and Johannessen [14] assert that software development companies end up spending between 50% and 80% of software development effort on testing. Okutan and Yildiz [8] and Bird et al. [9] maintain that correctly predicting and fixing software defects prior to software project deployment results in a decrease in the total cost of the project and an increase in overall project success rate in the market.

The goal of this study is to identify current trends and challenges in post-release software fault-prediction in relation to: choice of reliable software metrics for predicting post-release software faults, modeling techniques for predicting post-release software faults, and measuring performance of software fault-prediction models. We make the following contributions in this paper;

(a) We put into perspective reliable software metrics for software fault prediction.
(b) We present techniques for developing fault-prediction models.
(c) Finally, we present various measures for measuring performance of software fault-prediction models.

The rest of the paper is organized as follows: Section 2 describes the research method adopted in this study and links the study with previous similar studies. Results are presented in section 3 , while section 4 wraps up the results in a discussion. Finally section 5 concludes the study and gives a highlight of future work.

2 Research Method

This study employs Kitchenham's method for performing literature reviews in software engineering [55].

2.1 Research Questions

The study seeks to address the following research questions as shown in table 1:

Table 1. Research Questions

	Research Questions	Motivation
RQ 1:	What software metrics are reliable post-release software fault predictors?	Software metrics have been known to be predictors of software faults. This question seeks to unravel software metrics that are best predictors of post-release software faults.
RQ 2:	What models are used to predict post-release software faults?	Several scientific modeling techniques exist. This question seeks to establish modeling techniques that apply to prediction of post-release software faults.
RQ 3:	How is performance of software fault prediction models determined?	Models built using data for a specific metric should be compared with models built with different metrics. This question is expected to find out how performance of different models can be measured and compared.

2.2 Selection of articles for review

Articles reviewed here were sourced from digital libraries using a query that retrieved papers related to predicting faults, predicting defects or predicting bugs in Open Source Software. Like in [12], the inclusion criteria for an article in this study were that: the study was an empirical study; focused on predicting faults in units of a software system; and finally have faults in the code as the main output. Other significant articles that were identified from references of initial papers were subsequently included. Articles that we settled on as being relevant in addressing our research questions were 31 out of 39 retrieved articles. Both reviewed[1] and rejected[2] articles have been included in the reference section of this article.

[1] Reviewed articles are:[1],[2],[3],[4],[5],[6],[7],[8],[9],[10],[11],[14],[16],[18],[22],[23],[24], [25],[26],[27],[28],[29],[30],[31],[32],[38],[39],[40],[41],[45] and [47].
[2] Rejected articles are: [17],[48],[49],[50],[51],[52],[53] and [54].

2.3 Previous literature reviews on software fault prediction

Literature reviews on software fault prediction have initially been carried out by other researchers like Hall, Beecham, Bowes, Gray and Counsell [12] and Radjenovic, Hericko, Torkar and Zivkovic [13]. This review differs from Hall et al.'s and Radjenovic et al.'s systematic literature reviews from two main perspectives: period of study and objectives of study as shown in table 2. There is an overlap in the period of study of this literature review with the reviews done by [12] and [13]. The overlap was necessary so as to ensure that important information from key papers that was missed by previous reviews on software fault prediction modeling was included.

Table 2. Research Questions

Literature Review	Period	Objectives
Hall, Beecham, Bowes, Gray and Counsell [12]	2000 - 2010	They investigated how context of models, independent variables used and modeling techniques applied influence the performance of fault prediction models.
Radjenovic, Hericko, Torkar and Zivkovic [13]	1991 - 2011	They investigated the influence of context on metrics' selection and performance.
This review	2005 - 2015	To establish current state-of-the-art in software fault prediction in terms of: metrics that can reliably predict post-release software faults, modeling techniques applied in prediction of post release software faults, and techniques of measuring performance of software fault prediction models.

3 Results

3.1 Metrics for fault-prediction modeling (Independent variable)

Fault proneness and number of defects present in a software product are the most studied dependent variables in software quality models as noted by Arisholm, Briand and Johannessen [14]. However researchers working on software fault-prediction are faced with a challenge of selecting the best independent variables that can predict defects with the highest precision. Independent variables studied in most defect prediction models have primarily focused on

software product metrics and process metrics. Product metrics are computed from source code while process metrics are derived from software repositories like source code management and bug repositories. Various researchers have studied software product and process metrics in their effort to predict software defects within different contexts and found out varying results.

Product Metrics: Software product metrics are intrinsic code measures and can be grouped into static or dynamic metrics as shown in Fig. 1. Static code metrics include software product attributes like size and complexity. Line Of Code (LOC) is a common size metric while McCabe [19] and Halstead [20] metrics are commonly studied complexity metrics. Object-Oriented metrics involve software attributes like coupling, cohesion and inheritance as described by Chidamber and Kemerer [21].

Size Metrics: Okutan and Yildiz [8] and Zimmermann, Premraj and Zeller [22] performed differerent experiments and found out that lines of code and size of files seemed to be good defect predictors.

Complexity Metrics: Zimmermann, Premraj and Zeller [22] and Nagappan, Ball and Zeller [16] showed that complexity metrics can successfully predict post release defects. However Nagappan, Ball and Zeller [16] argue that there is no single set of metrics that is applicable to all projects. Nguyen, Adams and Hassan [23] replicated a study done by Zimmermann and Nagappan [24] and found out that network measures miss more critical classes and packages than complexity metrics. This contradicts the original study [24] that found out that network measures could identify twice as many critical binaries as identified by complexity metrics.

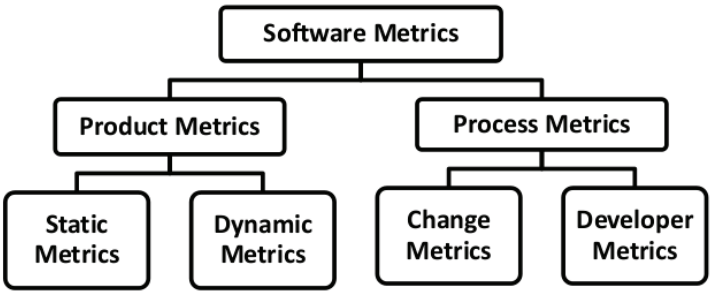


Fig. 1. Software metrics hierarchy

Process Metrics: Process metrics are a measure of the software development process and include change metrics and developer metrics – see Fig. 1. Developer metrics can further be broken down into ownership metrics, network of

developers metrics and organizational metrics. Arisholm, Briand and Johannessen [14] found out that including process measures significantly improved prediction models compared with models that only included structural measures like object oriented measures in terms of Receiver Operator Curve (ROC) area and Cost Effectiveness. A study done by Kamei et al. found that when a test is conducted on 20% of all files, product metrics could detect only 29% of all faults while process metrics could detect up to 74% of all faults [6].

Change Metrics: Predictive power of product and process metrics was performed by Moser, Pedrycz and Succi [2] who found out that their model associated with change metrics outperformed the model associated with code metrics in terms of the percentage of correctly classified instances, recall and false positive rate. However, Giger, D'Ambros, Pinzger and Gall [25] realized that including both change and source code metrics sets does not improve classification performance of a model. Using statistical regression models from Windows Server 2003 Service Pack-1, Nagappan and Ball [26] concluded that relative code churn measures are better than absolute code churn measures and can be used as efficient predictors of system defect density for discriminating between fault-prone binaries and not fault-prone binaries. Giger, Pinzger and Gall [27] argue that complexity metrics and code churn metrics are imprecise since they do not reflect all the detailed changes of particular source code entities during maintenance activities and advocate for fine-grained Source Code Changes (SCC). Their models built with logistic regression and SCC as predictor, ranked bug-prone files higher than not bug-prone files with an average probability of 90% and a better performance of Area Under Curve (AUC) compared to models built with modified lines.

However, Neuhaus, Zimmermann, Holler, and Zeller [32] investigated Mozilla's vulnerability history and revealed that components that had a single vulnerability in the past were generally not likely to have further vulnerabilities. They provided an empirical evidence that contradicts the popular notion that vulnerable components are generally more vulnerable in the future.

Developer metrics: Bird et al. [28], Nagappan, Murphy and Basili [29] and Meneely, Williams, Snipes and Osborne [30] believe that human factors in software development organizations play a significant role in the quality of software components. They all agree that ownership and organizational structure metrics are significantly better predictors for identifying failure-prone software components in terms of precision and recall compared to models built using code churn and code complexity.

Although process metrics seem to outperform product metrics in fault prediction, there still exists a gap to identify a specific set of metrics that can perform consistently well in most contexts. The conclusion validity of most of the reviewed studies is questionable as argued by Nagappan, Ball and Zeller [16] who believe that there is no single set of metric that is applicable to all projects. In connection to suitability of predictor metrics, Radjenovic, Hericko,

Torkar and Zivkovic [13] are encouraging researchers to perform more studies on large industrial software systems so as to be able to find metrics that are more relevant for the industry and suitable for specific contexts.

RQ 1: What software metrics are reliable post-release software fault predictors? In general, software process metrics outperform software product metrics when used as independent variable in fault prediction models. However the effectiveness of process or product metrics in predicting faults varies with changes in the context of studies.

3.2 Software fault prediction modeling techniques

Selecting a suitable modeling technique for software faults prediction is a challenging task since different modeling techniques perform differently in different contexts. Fenton and Neil [3] explain that defect prediction modeling studies are associated with theoretical and practical problems. First, models are weak because of their inability to cope with unknown relationship between defects and failures. Secondly, there are fundamental statistical and data quality problems that undermine model validity. In Fenton and Neil's [3] argument, many prediction models tend to model only part of the underlying problem and seriously specify it wrongly. Hall et al. [12] add that fault predictive models do not report their contexts appropriately making it difficult to know how well such models can be transferred to other systems. Statistical and machine learning modeling techniques are evident in the reviewed studies. Statistical modeling involves techniques like logistic regression, linear regression, univariate regression and multivariate regression whereas machine learning modeling involves techniques like J48 decision tree, Random forest, Naive Bayes, and Bayesian networks. Witten and Frank [33] argue that there is no clear dividing line between machine learning and statistics since there is a multi-dimensional continuum of data analysis techniques between the two.

Statistical modeling techniques: Regression is the most common statistical technique used in fault-prediction modeling. Faraway [34] explains that regression analysis is used for explaining the relationship between a single dependent variable Y, and one or more independent variables, X_1,.....,X_p. According to Faraway, when p=1, the type of regression is a simple regression but when p>1 it is a multiple regression or multivariate regression. However when there is more than one Y, then it is a multivariate multiple regression [34]. Faraway explains that the objectives of regression analyses include: prediction of future observations; assessment of the effect of, or relationship between explanatory variables on the response; and a general description of data structure. Variants of regression reported in the reviewed articles are: Linear regression that has been used by [22] and [23], Logistic regression that has been used by [22],[23] and [30], and Multiple Regression that has been used by [26],[29],[30] and [31].

Machine learning modeling techniques: In the recent past, machine learning techniques have gained popularity in software fault-prediction modeling. Malhotra and Jain [4] observes that much of previous work on fault prediction used traditional statistical methods but recently the trend is shifting to modern machine learning methods. Machine learning is a technique to optimize performance criterion of a computer using example data or past experience that is required in cases where writing a computer program directly to solve a program is difficult [35]. Software fault-prediction is basically a classification problem where unseen data instances are segmented into groups as per class label. From this review, five machine-learning classifiers are predominant as shown in table 3 and Fig. 2. Other machine learning algorithms that are not predominant but have been applied in the studies include J48 Decision trees, Adaboost, Bagging, Multilayer Perceptron and Genetic programming. Studies that applied machine learning techniques include; [2], [4], [6], [8], [11], [12], [14], [27], [38], [39] and [40].

Table 3. Prominently used machine-learning classifiers

Machine-learning algorithm	% of use
Random-Forest	18.2
Logistic Regression	15.1
Naives Bayes	12.1
Support Vector Machine	12.1
Bayesian Networks	9.1
Others	33.4

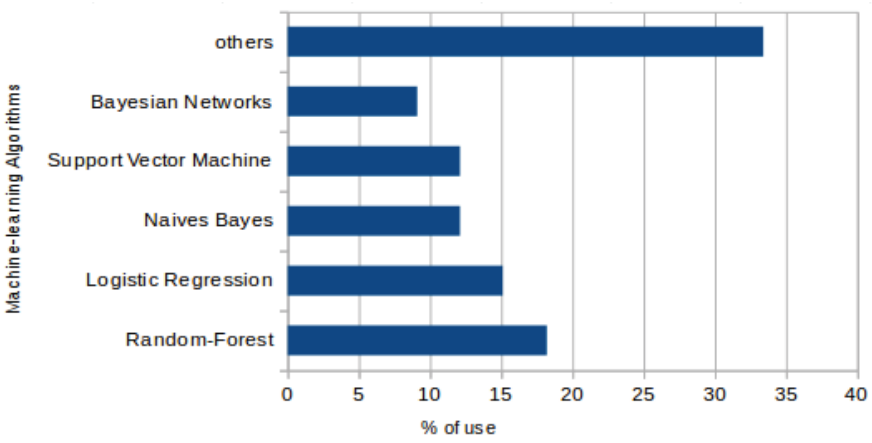


Fig. 2. Graph showing usage of machine-learning classifiers in reviewed articles

Aggarwal [37] argues that different classifiers may work more effectively with different kinds of data sets and application scenarios. This scenario is evident in this literature review where different studies report different classifiers as best performing. For instance, Shin et al. [39] found out that J48, Random forest, and Bayesian network provided similar results to Logistic regression, while Naive Bayes provided better results. According to Malhotra and Jain's [4] results, Random Forest and Bagging models were perceived to be the best models for prediction. Fukushima et al. [11] used Random Forest algorithm on the strength that Random Forest produces robust, highly accurate, stable models that are resilient to noisy data while Hall et al. [12] on the contrary report that models based on Random Forest perform poorly compared to those based on Naive Bayes and Logistic Regression.

It can be observed that different studies report different classifiers as performing the best. Therefore there still exists a gap in identifying the best performing algorithm across different experimental contexts.

RQ 2: What models are used to predict post-release software faults? Statistical and machine-learning models are commonly used to predict software faults. Regression is a popular statistical modeling technique while Random-Forest, Logistic regression, Naives Bayes, Support Vector Machine and Bayesian Networks are popular machine-learning modeling techniques for fault prediction.

3.3 Measuring performance of defect prediction models

Defect prediction models should address effectiveness of models in addition to addressing issues related to appropriateness of metrics used as defect predictors in modeling, types of algorithms used for defect prediction and cost of utilization of the models [2]. Arisholm, Briand and Johannessen [14] maintain that there is need for evaluating and comparing fault-proneness prediction models not only by considering their prediction accuracy, but also by assessing the potential cost-effectiveness of applying such models. Mende and Koschke [18] argue that evaluations of models is a challenge since identification of suitable performance measures is still a problem.

Confusion Matrix: Several studies in the reviewed literature used various Confusion Matrix criteria to evaluate their classifiers. The matrix in Fig. 3 shows a typical Confusion Matrix as depicted by Maimon and Rokach [42], where columns correspond to the predicted class while rows are the actual class. In the Confusion Matrix shown in Fig. 3, TN is the number of negative examples correctly classified (True Negatives), FP is the number of negative examples incorrectly classified as positive (False Positives), FN is the number of positive examples incorrectly classified as negative (False Negatives) and TP is the number of positive examples correctly classified (True Positives) [42]. Several performance measures have been derived from the Confusion Matrix

which include; Precision, Recall or Sensitivity, Specificity, False discovery rate, F-measure and Accuracy.

	Predicted Negative	Predicted Positive
Actual Negative	**TN**	**FP**
Actual Positive	**FN**	**TP**

Fig. 3. Confusion Matrix

$$Precision = TP/(TP + FP) \tag{1}$$

$$Recall\ \ (Sensitivity) = TP/(TP + FN) \tag{2}$$

$$Specificity = TN/(FP + TN) \tag{3}$$

$$FalseDiscoveryRate = FP/(TP + FP) \tag{4}$$

$$F - value = ((1 + \beta^2) * Recall * Precision)/(\beta^2 * Recall + Precision) \tag{5}$$

where β is usually set to 1 and corresponds to the relative importance of precision versus recall.

$$Accuracy = (TP + TN)/(P + N) \tag{6}$$

or

$$Accuracy = TP/(P + N) \tag{7}$$

In terms of defective software modules, Precision is the ratio of actually defective modules within modules predicted as defective while Recall(Sensitivity) is the ratio of detected defective modules among all defective modules [18]. Recall measures how often we find what we are looking for [43]. Mende and Koschke [18] and Cios et al. [43] advise against using the accuracy measure for two reasons: accuracy is a bad performance measure for imbalanced data thus not suitable for defect prediction and that accuracy can be defined in two different ways thus giving conflicting outcomes respectively. Of all the Confusion Matrix performance measures, Recall and Precision are the only measures that have been reported in the reviewed studies. Some of the studies that have reported using these two performance measures include [14], [9], [18], [29] and [32].

Receiver Operating Curve: Receiver Operator Characteristic (ROC) curve is a two dimensional graphical representation for evaluating classification models where True Positive (TP) is plotted on the y-axis and False Positive (FP) on the x-axis [11]. ROC analysis is performed by drawing curves on a two-dimensional space, with the y-axis representing Sensitivity = TP rate, while the x-axis represents 1 − Specificity = FP rate as shown in Fig. 4 [43]. Cios et al.[43], Arisholm, Briand and Johannessen [14], and Okutan and Yildiz [8] agree that the best way of selecting an optimal model is by determining the Area Under Curve (AUC) of the ROC whereby best performing model will have the largest AUC. Mende and Koschke [18] add that a perfect classifier would have an AUC of 1, while a random classifier is expected to achieve an AUC of 0.5.

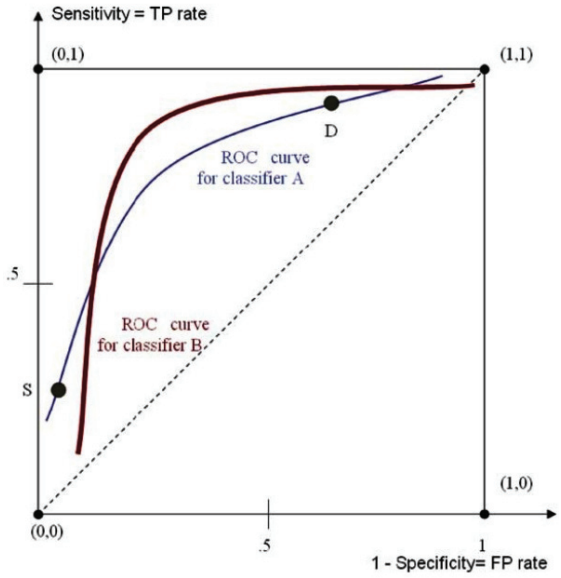


Fig. 4. Cios et al.'s ROC curves for classifiers A and B

Other performance measures for models: Sokolova, Japkowicz and Szpakowicz [44] argue that Confusion Matrix and ROC measures do not fully meet the needs of learning problems where several algorithms are compared. They suggested evaluating performance of classifiers using the Youdens index, Likelihood and Discriminant Power measures from the premise that these measures allow a comprehensive comparison of learning algorithms for problems in which; the data classes are equally important, one classifier is to be chosen from among two or more classifiers, and data gathering is extremely expensive [44]. Youden's index, γ, evaluates an algorithm's ability to avoid failure where a higher value of the index indicates better ability to avoid failure [44].

$$\gamma = sensitivity - (1 - specificity) \tag{8}$$

Likelihood is described in terms of positive likelihood ($\rho+$) and negative likelihood ($\rho-$) and accommodates both sensitivity and specificity thus enabling evaluation of a classifier's performance with respect to both classes [44];

$$\rho+ = sensitivity/(1 - specificity) \tag{9}$$

$$\rho- = (1 - sensitivity)/specificity \tag{10}$$

A higher positive likelihood and a lower negative likelihood imply better performance on positive and negative classes respectively [44]. They continue to explain that Discriminant Power (DP) is a measure that evaluates how well an algorithm distinguishes between positive and negative examples by summarizing sensitivity and specificity;

$$DP = (\sqrt{3}/\pi) * (logX + logY) \tag{11}$$

where; X = *sensitivity/(1-sensitivity)* and Y = *specificity/(1-specificity)*. They conclude that an algorithm is a poor discriminant if DP < 1, limited if DP < 2, fair if DP < 3, good in other cases.

Evaluating Cost Effectiveness of fault prediction models: It is important to include the notion of effort into evaluation of fault prediction models since there exist additional costly quality assurance activities in terms of effort that are associated with fault prediction problem [40]. Arisholm, Briand and Johannessen [14] proposed a Cost-Effectiveness (CE) measure arguing that the cost of effort required to inspect and fix a fault in a class is roughly proportional to the size of the class. Mende and Koschke [40] also used size (lines of code) as a proxy for effort similar to Arisholm et al. [14]. To assess Cost-Effectiveness of models, Arisholm, Briand and Johannessen compared the curves shown in the Fig. 5.

From Fig. 5, the solid curve is the Cost-Effective (CE) curve that represents actual percentage of faults given a percentage of lines of code (% NOS) of classes selected for further review while the dotted line is a baseline of comparison. Arisholm et al. [14] suggest that the overall cost-effectiveness of fault prediction models would be proportional to the surface area between the Cost-Effective (CE) curve and the baseline [14]. The dotted curve in Fig. 5 shows an optimum model that can be used as an upper bound to further assess a model. For purposes of comparing CE areas, Arisholm et al. [14] determined a normalized cost-effectiveness measure given by equation 12.

$$CE_\pi = (CE_\pi(model) - CE_\pi(baseline))/(CE_\pi(optimal) - CE_\pi(baseline)) \tag{12}$$

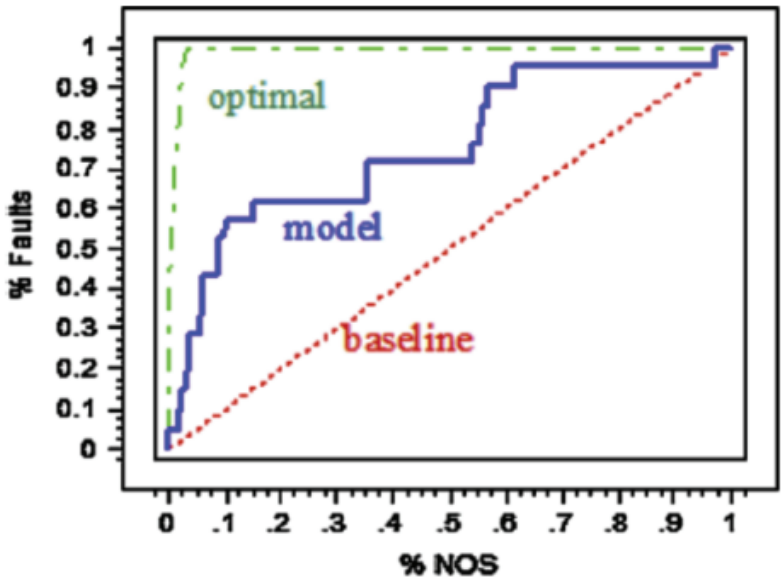


Fig. 5. Arisholm et al.'s surrogate measure of Cost Effectiveness (CE)

where $CE_\pi(x)$ is the area under the curve x (baseline, model, or optimal) for a given π% of NOS. Mende and Koschke [40] found out that inclusion of effort-awareness into defect prediction models improve their performance. Arisholm, Briand and Fuglerud [45] recommend use of a specific cost-effectiveness model in addition to standard confusion matrix criteria.

RQ 3: How is performance of software fault prediction models determined? Performance of software fault prediction models has been determined by using the Confusion Matrix, Receiver Operator Curve and establishing Cost-Effective (CE) measures for each model.

4 Discusion

4.1 Threats to validity

Conclusion validity of fault prediction models: Software fault prediction studies, like other areas of empirical software engineering, suffer from conclusion validity issues. It is very difficult to confidently accept or reject a hypothesis from a software fault prediction experiment. Basili et al. [46] assert that a wide experimental spectrum of context variables, uniqueness of each software product and human factors in software engineering studies are responsible for this situation. Basili et al. argue that: for knowledge to be created from empirical

software engineering studies, it requires a community of researchers that can replicate studies, vary context variables, and build abstract models that represent common observations about the discipline [46]. To strengthen Basili et al.'s concern, Meneely, Williams, Snipes, and Osborne [30] question the high success rates claimed by many fault prediction models – most of which cannot demonstrate that they have been replicated successfully and consistent results have been established – or they are a replication of previous studies achieving consistent results. Results from this review concur with Meneely et al.'s position. From this review, only three studies out of 31 studies demonstrate that they are a replication or related directly to other studies. Arisholm et al. [14] performed analysis related to [45]; Mende and Koschke [40] replicated Ostrand et al's. [47] study, while Nguyen et al. [23] replicated a study by Zimmerman and Nagappan [24]. It follows that conclusions made based on consistent results from such replicated studies can be relied upon.

However from this review it is not clear whether the lack of replication is as a result of the research community's unwillingness to replicate studies or poorly explained experimental research methods in published works. To enable easier replication of experiments, it is important for researchers to explicitly explain; context, research questions or hypothesis and methodology of their experiments.

Internal validity: Most of the reviewed articles have mitigated threats resulting from internal validity by employing tools that automate data collection. Such tools guarantee a deterministic data collection process.

4.2 Discussion of results

Reliable software metrics for software fault prediction: It has generally been observed that software process metrics outperform software product metrics in fault-prediction. However there is need to carryout many experiments in different contexts so as to find out which particular software process metrics are superior fault predictors.

Classification algorithms: It has also been observed that many reviewed studies have applied many classifiers in their experiments. It is still not very clear whether using many classifiers in a classification problem adds value to the prediction solution. Some researchers have found out that there is no significant variation in results from different classifiers while others report that some classifiers give better results. For example Lessman et al. single out Random Forest as the best performing classifier but at the same time they conclude that performance differences between classifiers are not necessarily significant as cited in [27]. In a different study, Shin, Meneely, Williams, and Osborne [39] report that Naives Bayes classifier provided better results than Logistic regression, J48, Random forest and Bayesian network. Arisholm, Briand and Johannessen [14] seem to agree with Lessman et al. since both of them argue that the choice of fault-proneness modeling technique has limited impact on

the resulting classification accuracy or cost-effectiveness (CE). Aggarwal [37] believes that different classifiers may work more effectively with different kinds of data sets and application scenarios. It is also observed that many studies do not give reasons as to why the modeling technique they have selected is the most appropriate for the context of their study.

Measuring performance of prediction models: Performance measures for models are important for two main reasons: they express the economic viability of a model in addition to being used as yard sticks for comparison in scenarios where models are to be compared. Therefore measuring of performance of fault prediction models is an issue researchers working on fault-prediction should emphasize. Apart from using the Confusion Matrix measures, majority of reviewed studies did not report performance of their models based on ROC and CE measures. Other performance measures for example Youdens index, Likelihood and Discriminant Power should also be tested in the context of software fault prediction.

5 Conclusion

It is necessary for researchers working on software fault prediction to collaborate and replicate experiments so that they can draw conclusions from statistically significant results thus creating knowledge. The researchers should explicitly explain the research contexts, research questions or hypotheses and methods of their experiments so as to enable easier replication. They should also measure the performance of their software fault prediction models so as to enable easier comparison of models from different studies. Finally, more research work is required to establish the most superior set of software process metrics.

Future work: We plan to replicate a fault prediction study by using datasets retrieved from different projects relative to the original study. We will build fault prediction models for a set of process metrics and compare these models in terms of Confusion Matrix, ROC and Cost Effective measures. Finally we will compare our models with models in the original study.

Acknowledgement: We sincerely thank the Advanced Computing Laboratory of Saint-Denis (LIASD) – France, the French Embassy in Nairobi and Dedan Kimathi University of Technology (DeKUT) for supporting this study.

References

1. Stol, K.-J. and Babar, M. A. (2010). A comparison framework for open source software evaluation methods. Open Source Software: New Horizons (pp. 389394). Springer.

2. Moser, R., Pedrycz, W., & Succi, G. (2008, May). A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. InSoftware Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on(pp. 181-190). IEEE.
3. Fenton, N. E., & Neil, M. (2005). A critique of software defect prediction models. SERIES ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING,16, 72.
4. Malhotra, R., & Jain, A. (2012). Fault prediction using statistical and machine learning methods for improving software quality.Journal of Information Processing Systems,8(2), 241-262.
5. Xu, J., Ho, D., & Capretz, L. F. (2015). An empirical study on the procedure to derive software quality estimation models.arXiv preprint arXiv:1507.06925.
6. Kamei, Y., Matsumoto, S., Monden, A., Matsumoto, K. I., Adams, B., & Hassan, A. E. (2010, September). Revisiting common bug prediction findings using effort-aware models. InSoftware Maintenance (ICSM), 2010 IEEE International Conference on(pp. 1-10). IEEE.
7. Kim, S., Zimmermann, T., Whitehead Jr, E. J., & Zeller, A. (2007, May). Predicting faults from cached history. InProceedings of the 29th international conference on Software Engineering(pp. 489-498). IEEE Computer Society.
8. Okutan, A., & Yildiz, O. T. (2014). Software defect prediction using Bayesian networks.Empirical Software Engineering,19(1), 154-181.
9. Bird, C., Nagappan, N., Gall, H., Murphy, B., & Devanbu, P. (2009, November). Putting it all together: Using socio-technical networks to predict failures. InSoftware Reliability Engineering, 2009. ISSRE'09. 20th International Symposium on(pp. 109-119). IEEE.
10. Couto, C., Pires, P., Valente, M. T., Bigonha, R. S., & Anquetil, N. (2014). Predicting software defects with causality tests.Journal of Systems and Software,93, 24-41.
11. Fukushima, T., Kamei, Y., McIntosh, S., Yamashita, K., & Ubayashi, N. (2014, May). An empirical study of just-in-time defect prediction using cross-project models. InProceedings of the 11th Working Conference on Mining Software Repositories(pp. 172-181). ACM.
12. Hall, T., Beecham, S., Bowes, D., Gray, D., & Counsell, S. (2012). A systematic literature review on fault prediction performance in software engineering. Software Engineering, IEEE Transactions on,38(6), 1276-1304.
13. Radjenovic, D., Hericko, M., Torkar, R., & Zivkovic, A. (2013). Software fault prediction metrics: A systematic literature review.Information and Software Technology,55(8), 1397-1418.
14. Arisholm, E., Briand, L. C., & Johannessen, E. B. (2010). A systematic and comprehensive investigation of methods to build and evaluate fault prediction models.Journal of Systems and Software,83(1), 2-17.
15. ISDW Group. (2010). 1044-2009-IEEE Standard Classification for Software Anomalies.IEEE, New York.
16. Nagappan, N., Ball, T.,& Zeller, A. (2006, May). Mining metrics to predict component failures. InProceedings of the 28th international conference on Software engineering(pp. 452-461). ACM.
17. Couto, C., Silva, C., Valente, M. T., Bigonha, R.,& Anquetil, N. (2012, March). Uncovering causal relationships between software metrics and bugs. InSoftware Maintenance and Re-engineering (CSMR), 2012 16th European Conference on(pp. 223-232). IEEE.

18. Mende, T., & Koschke, R. (2009, May). Revisiting the evaluation of defect prediction models. In Proceedings of the 5th International Conference on Predictor Models in Software Engineering(p. 7). ACM.
19. McCabe, T. J. (1976). A complexity measure.Software Engineering, IEEE Transactions on, (4), 308-320.
20. Halstead, M. H. (1977). Elements of Software Science (Operating and programming systems series). Elsevier Science Inc..
21. Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design.Software Engineering, IEEE Transactions on,20(6), 476-493.
22. Zimmermann, T., Premraj, R., & Zeller, A. (2007, May). Predicting defects for eclipse. InPredictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on(pp. 9-9). IEEE.
23. Nguyen, T. H., Adams, B., & Hassan, A. E. (2010, September). Studying the impact of dependency network measures on software quality. InSoftware Maintenance (ICSM), 2010 IEEE International Conference on(pp. 1-10). IEEE.
24. T. Zimmermann and N. Nagappan, Predicting defects using network analysis on dependency graphs, in Proc. of the 30th inter. conf. On Soft. Engi. Leipzig, Germany: ACM, 2008, pp. 531540.
25. Giger, E., D'Ambros, M., Pinzger, M., & Gall, H. C. (2012, September). Method-level bug prediction. In Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement(pp. 171-180). ACM.
26. Nagappan, N., & Ball, T. (2005, May). Use of relative code churn measures to predict system defect density. In Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on(pp. 284-292). IEEE.
27. Giger, E., Pinzger, M., & Gall, H. C. (2011, May). Comparing fine-grained source code changes and code churn for bug prediction. InProceedings of the 8th Working Conference on Mining Software Repositories(pp. 83-92). ACM.
28. Bird, C., Nagappan, N., Murphy, B., Gall, H., & Devanbu, P. (2011, September). Don't touch my code!: examining the effects of ownership on software quality. In Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering(pp. 4-14). ACM.
29. Nagappan, N., Murphy, B., & Basili, V. (2008, May). The influence of organizational structure on software quality: an empirical case study. In Proceedings of the 30th international conference on Software engineering (pp. 521-530). ACM.
30. Meneely, A., Williams, L., Snipes, W., & Osborne, J. (2008, November). Predicting failures with developer networks and social network analysis. In Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering (pp. 13-23). ACM.
31. Shin, Y., Bell, R., Ostrand, T., & Weyuker, E. (2009, May). Does calling structure information improve the accuracy of fault prediction?. In Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on (pp. 61-70). IEEE.
32. Neuhaus, S., Zimmermann, T., Holler, C., & Zeller, A. (2007, October). Predicting vulnerable software components. InProceedings of the 14th ACM conference on Computer and communications security(pp. 529-540). ACM.
33. Witten, I. H., & Frank, E. (2005). Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann.
34. Faraway, J. J. (2002). Practical regression and ANOVA using R.
35. Alpaydin, E. (2014).Introduction to machine learning. MIT press.

36. Hastie, T., Tibshirani, R., & Friedman, J., (2009).The elements of statistical learning(Second Edition). Springer, Berlin: Springer series in statistics.
37. Aggarwal, C. C. (Ed.). (2014). Data classification: algorithms and applications. CRC Press.
38. Fenton, N., Neil, M., & Marquez, D. (2008). Using Bayesian networks to predict software defects and reliability. Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability,222(4), 701-712.
39. Shin, Y., Meneely, A., Williams, L., & Osborne, J. (2011). Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities.Software Engineering, IEEE Transactions on,37(6), 772-787.
40. Mende, T., & Koschke, R. (2010, March). Effort-aware defect prediction models. In Software Maintenance and Reengineering (CSMR), 2010 14th European Conference on(pp. 107-116). IEEE.
41. Mizuno, O., Ikami, S., Nakaichi, S., & Kikuno, T. (2007, May). Spam filter based approach for finding fault-prone software modules. In Proceedings of the Fourth International Workshop on Mining Software Repositories(p. 4). IEEE Computer Society.
42. Maimon, O., & Rokach, L. (Eds.). (2005). Data mining and knowledge discovery handbook(Vol. 2). New York: Springer.
43. Cios, K. J., Pedrycz, W., Swiniarski, R. W.,& Kurgan, A.L., (2007). Data Mining: A Knowledge Discovery Approach. Springer US.
44. Sokolova, M., Japkowicz, N., & Szpakowicz, S. (2006). Beyond accuracy, F-score and ROC: a family of discriminant measures for performance evaluation. In AI 2006: Advances in Artificial Intelligence(pp. 1015-1021). Springer Berlin Heidelberg.
45. Arisholm, E., Briand, L. C., & Fuglerud, M. (2007, November). Data mining techniques for building fault-proneness models in telecom java software. In Software Reliability, 2007. ISSRE'07. The 18th IEEE International Symposium on(pp. 215-224). IEEE.
46. Basili, V. R., Shull, F., & Lanubile, F. (1999). Building knowledge through families of experiments. Software Engineering, IEEE Transactions on, 25(4), 456-473.
47. T. Ostrand, E. Weyuker, and R. Bell, Predicting the location and number of faults in large software systems, IEEE Transactions on Software Engineering, vol. 31, no. 4, pp. 340355, 2005.
48. Kumaresh, S., & Baskaran, R. (2010). Defect analysis and prevention for software process quality improvement. International Journal of Computer Applications,8(7).
49. Yadav, D. K., Chaturvedi, S. K., & Misra, R. B. (2012). Early Software Defects Prediction Using Fuzzy Logic. International Journal of Performability Engineering,8(4).
50. D'Ambros, M., Lanza, M., & Robbes, R. (2010, May). An extensive comparison of bug prediction approaches. In Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on(pp. 31-41). IEEE.
51. Peters, R., & Zaidman, A. (2012, March). Evaluating the lifespan of code smells using software repository mining. In Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on (pp. 411-416). IEEE.
52. Bijlsma, D., Ferreira, M. A., Luijten, B., & Visser, J. (2012). Faster issue resolution with higher technical quality of software.Software quality journal,20(2), 265-285.

53. Bettenburg, N., Nagappan, M., & Hassan, A. E. (2012, June). Think locally, act globally: Improving defect and effort prediction models. InProceedings of the 9th IEEE Working Conference on Mining Software Repositories(pp. 60-69). IEEE Press.
54. Liu, Y., Cheah, W. P., Kim, B. K., & Park, H. (2008). Predict software failure-prone by learning bayesian network.International Journal of Advanced Science and Technology,1(1), 35-42.
55. Kitchenham, B. (2004). Procedures for performing systematic reviews. Keele, UK, Keele University, 33(2004), 1-26.

Competiting on a Common Platform

Siobhan O'Mahony, Rebecca Karp
Boston University Questrom School of Management
595 Commonwealth Avenue, Boston MA. 02215

Abstract. Much research has studied how dominant firms can design and lead a single platform that complementary firms leverage. Often, control over the platform's architecture, rules of engagement and business model are managed by a single firm. What happens when firms transition from a dominant mode of control to one that is collectively managed? With field data from a platform founder and 17 firms, we examine how these firms responded differently to the emergence of a collectively managed platform. We examine firms' motivations for joining a common platform, and the challenges and benefits encountered. In doing so, we identify a more nuanced range of strategic responses that differ from either pure collaboration or pure competition: from competitive observation to distributed platform leadership, which prompted varying degrees of internal adaptation. Our research shows not only how platforms can be collectively managed, but also how firms guardedly engage with common technical resources in ways that both preserve and threaten their competitive advantage.

1. Introduction

Often the success of an ecosystem as a whole is at odds with that of a single firm, (Iansiti and Levien, 2004; Baldwin and Woodard, 2007). For example, an ecosystem[1] overall may benefit from open governance structures as they increase aggregate value creation, but the dominant or lead firm within a niche may no longer be able to appropriate all of the rents associated with this value creation. This paper explores the relationship between the openness of a platform's governance and competitive strategies firms use to engage with the platform and broader ecosystem. We ask then, under varying platform governance structures, how can a firm contribute to an ecosystem while also maintaining strategic advantage?

With an inductive study of a platform that transitions from closed and proprietary, to open and collectively managed, we explore the varying and dynamic strategies firms develop to both contribute to the ecosystem and guard competitive advantage. Instead of evaluating this process from the lens of the platform "owner" or "sponsor", we evaluate the decisions members of the ecosystem make vis-à-vis changes to the governance. The Eclipse Platform, an integrated development environment (IDE) written in Java, was funded and created solely by IBM but released as an open source project that allowed derivative works to be distributed

[1] The term Baldwin and Woodard 2007 utilizes is modular clusters vs. ecosystems. They define modular clusters as "disaggregated "clusters," "networks," or "ecosystems" of firms". We therefore believe ecosystem can be used interchangeably with modular clusters

royalty-free worldwide in 2001. To alleviate partner concerns and grow an independent ecosystem, IBM transitioned the project from proprietary control to an independent private foundation in 2005. As one participating firm explained, "There were some very large players that said we're not going to be part of something that IBM controls...IBM could not be seen as a controlling entity. It is like stagnate the growth or set it free. I am sure that was a very hard problem for IBM to work out internally...if you set it free, you don't really know what it is going to do."

1.1 Open Innovation

If firms cannot appropriate the full value of their innovation, why would they collaborate through open governance structures? Extant literature evaluates how and why firms benefit from open innovation (Chesbrough, Vanhaverbeke and West, 2014), which includes engaging in open innovation to 1) procure or distribute knowledge; 2) access resources that exist outside the four walls of the firm. Literature about open innovation (Chesbrough 2003; 2006; Chesbrough, Vanhaverbeke and West 2006; Laursen and Salter, 2006) demonstrates that firms can both find and monitor leading edge innovation by consuming external sources of knowledge. A mature body of scholarship also shows that no one company can produce a path-breaking innovation on its own. "In most industries, no single firm commends a majority of the resources available for research nor can any one firm respond to more than a portion of the needs or problems requiring original solutions" (Utterback, 1974). Firms in many industries collaborate within networks or communities enabled by information technology, reduced trade barriers, and globally dispersed skill sets (Sinha and Van de Ven 2005). In fields such as information technology, nanotechnology and biotechnology (Powell, Koput, and Smith- Doerr, 1996; Powell and Brantley, 1992; Grodal, Gotsopoulos and Suarez 2015), the locus of innovation may not lie within any one firm, but within the capabilities of an ecosystem (Iansiti, Levien, 2004) community (Lakhani and O'Mahony, 2011) or network of firms collaborating on developments critical to the evolution of their industry (Powell et al., 1996; Powell, White and Koput,2005; Powell and Giannella, 2010).

1.2 Platform Leadership

We also know that successful firms can strategically create platforms to achieve competitive advantage through platform dominance or leadership of an ecosystem (Gawer and Cusamano, 2002; Gawer 2011; Iansiti and Levien, 2004; Parker, Van Alystne, Choudary, 2015). In these cases, firms are the sole architects of a platform, creating modular substructures (Baldwin and Clark, 2000), which invite other firms to engage. Often, the founding firm establishes and maintains the rules by which all other firms participate and thus, the rules by which other firms can profit. Apple and Amazon are illustrative examples of this model (Parker, Van Alysne, and Choudary, 2015). While platform leaders define the rules, they depend on complementors to create value. In this manner, firms in platform leadership positions consider their platform as a foundation to an evolving ecosystem that is not valuable without the

contributions of complementary firms (Gawer and Cusumano 2002). Thus, platform leaders have strong incentives to attract complementing firms to engage with and extend their platforms (Parker et al, 2015).

Under these conditions, platform leaders encourage complementary contributors, but still extract much of the value created within the ecosystem (Baldwin amd Woodard 2007; Gawer and Cusumano 2008; Eisenmann, Parker, and Van Alstyne, 2009). Further, platform leaders can choose who to invite and accept within the ecosystem; platform leaders can invite complimentary firms and limit potential competitors from joining. Leaders also have the ability to both switch the rules as their own firm strategies necessitate and commoditize complimentary products. Firms within an ecosystem must therefore arbitrate how much of their proprietary assets they can share with the platform leader and community or network of firms (von Hippel and von Krogh 2003; Parker, Van Alstyne and Choudary, 2015). While maintaining a distinct form of competitive advantage is important, firms have much to gain from selectively revealing aspects of their innovation platforms (Henkel, 2006; von Hippel and von Krogh, 2006).

2.1 Research Design

We conducted a longitudinal study which tracked how different member firms engaged with the Eclipse platform as it transitioned from proprietary and closed to collectively managed and open from 2000 to 2016.

2.2 Research Setting

Eclipse was well suited to study how changes in the design of collectively used platforms affect firm strategies for engagement. First, the Eclipse platform was built by an open source community that is still large and thriving today. As of May 2016, 230 firms participated in the community. Over one million lines of committed code had been developed or donated to Eclipse. The platform had significant market reach in 2014, as it was utilized by nearly 50% of all software developers worldwide[2]. Second, Eclipse transitioned from a closed and proprietary IBM project to an extensible, vendor neutral, open source platform built by a pool of common resources. While Eclipse was released by IBM as open source, firms were licensed to create commercially viable, proprietary derivative works on the back of the common source code. Third, surrounding the Eclipse platform, was a strong and divergent ecosystem of firms that leveraged the platform and open source community with diverse goals in mind. Firms varied how and what they contributed to the platform. Some freely shared knowledge and others carefully guarded assets to tilt the direction and extension of the platform's capabilities. Last, Eclipse over-time ratified a non-profit Foundation which managed the distribution of IP, scope of the platform and growth of the ecosystem. Unlike other open source communities, the Foundation

[2] Zeroturnaround.com May 2015

was wholly owned by the member firms and directed by a small representative board.

Unlike previous studies (Gawer and Henderson 2007) on platform design, the study of Eclipse enabled us to delve into the unique and varied strategies firms activate within an ecosystem vs. the strategy platform owners or sponsors take to lead or manage an ecosystem. While Eclipse is an extreme case in itself, our study compares the interaction models of a set of firms that engage with Eclipse. Thus we employed an embedded case design. This enabled us to better test validity, variance and commonalities across our sample (Eisenhardt and Graebner 2007). We constructed a theoretical rather than representative sample of 18 firms, selected to enumerate the range of possible strategic choices firms took within the Eclipse ecosystem vs. possible strategies firms deployed broadly, outside of our research setting (Eisenhardt, Graebner 2007). Our research design was critical in enabling us to take the position of the member firms vs. the platform leader.

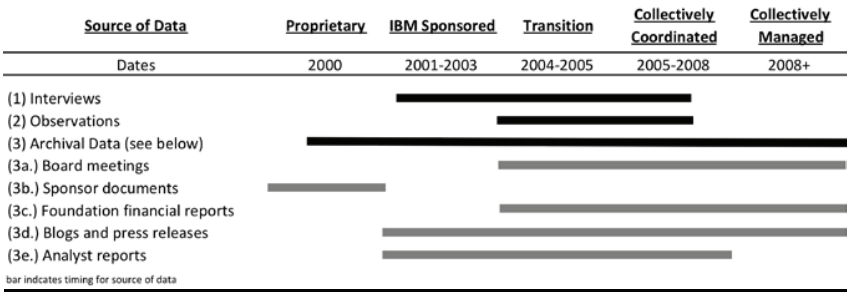
2.2 Data Collection

Our sample was constructed of 18 firms that were members of the Eclipse community at some point between 2001 and 2008. Not all of the firms in our sample became members at the same time. Varying tenure and partcipation in the community allowed us to draw distinct rationale for why and how firms changed their ecosystem strategies. To triangulate evidence for construct validity (Yin 1984), we collected several sources of data. While the firms represent a theoretical sample, they also broadly mirror the Eclipse membership at large during the time period in which interviews were conducted. For example, size of firm often dictates whether a firm will engage and reveal more readily in an open source community (Henkel 2006). We tested our sample to ensure we accurately captured the range of firm sizes present in the broader membership. Approximately one third of the firms in our sample were small firms (annual revenue less than $10 million), one third of the firms were medium sized (annual revenue between $50 million and $1 billon), and one third of the firms were large (annual revenue over $1 billion). In the broader population of firms, approximately forty-three percent were small firms and twenty-three percent are large. Further, scholars have also demonstrated that product strategy can influence and determine how a firm engages in an open source community (Henkel 2006). Approximately sixty-one percent of firms in our sample were software producers, twenty-two percent were service providers and seventeen percent were hardware producers. Amongst all members, sixty-four percent were software producers, seventeen percent were service providers and nineteen percent were hardware producers.

Over the period of time we conducted this study some firms merged, were acquired or ceased to remain active entities. Approximately twenty-eight percent of all firms in our sample merged or were acquired. All of the firms that were acquired, were bought by, at the time, members of the Eclipse community. We recognized a similar pattern amongst the broader universe of member firms. Thirty percent of all member firms were acquired during the period we studied, many by other member firms.

Data sources included the following: 1) interviews with founders of Eclipse, board members, leaders of the Foundation, committers and project managers; 2) observations of board meetings, committee sessions and two Eclipse conferences; 3) Archival data which included: 3a. over 100 board meetings, 3b. IBM sponsor documents, 3c. Foundation annual reports, 3d. blogs, and 3e. analyst reports. (Table 1)

Table 1: Sources of Data



(1) Interviews: We conducted 40 interviews which ranged from one to three hours with both IBM as the platform founder and the participating firms within the ecosystem. In total, we conducted interviews with 18 firms, and 4 interviews with employees of the foundation. We conducted a range of 1-3 interviews with each firm. Where possible, we conducted multiple interviews per firm to better account for potential variance in perspective by role (e.g., marketing managers may have a different perspective than developers). We asked IBM as the platform founder to elaborate on how they opened the platform and the process they pursued to create collective management with attention to how barriers encountered were overcome. We asked participating firms why and how they engaged with the platform; how their participation changed over time and how their related proprietary product development and strategy was affected by the transition. All participants were guaranteed anonymity and all interviews were recorded and transcribed.

(2) Observations: Observations were conducted at board meetings, committee sessions and at Eclipse membership conferences. We observed activity between the Eclipse membership for over thirty hours, often with the same individuals interviewed. Observations deepened our understanding of the strategies firms took to articulate and champion their private goals within the community.

(3) Archival data: Archival data was utilized to validate the dates and timelines presented by the interviewees. Product releases and project launches on the platform were also confirmed utilizing archival data. We supplemented our primary data sets to ensure we accurately understood the level of engagement of each firm with the platform and the overall evolution of the platform as it transitioned from IBM-owned

to collectively managed.

2.3 Analytical approach:
Because research on how design configurations affect firm engagement with a common platform was limited, we employed an inductive approach. We followed an iterative process leveraging grounded theory to develop common themes and working hypotheses. Data analysis then included three phases: 1) mapping of ecosystem phases employed to manage Eclipse; 2) identifying and comparing firm strategic choices across two continuums, contribution to the ecosystem and platform, competition with the platform; 3) synthesizing of firms into four distinct ecosystem strategies based on strategic choice.

Phase 1: Identifying Ecosystem Phases
The Eclipse platform transitioned through five distinct phases: 1) (2000-2001) Proprietary; 2) (2001-2003) IBM Sponsored; 3) (2004-2005) Transitional; 4) (2006-2008) Collectively Coordinated; 5) (2008+) Collectively Managed (Table 1). Each phase was demarcated by a significant reconstitution in the allocation of decisions rights from IBM to the member firms and the Foundation across a set of three modes of design. We defined modes of design as organizational: framework for determining who is granted membership and how membership rights are allocated; architectural: degree of inter-dependance and modularity *across* Eclipse projects including pacing of technology development, release management, inter-operability of interfaces; technical: degree of inter-dependance and modularity *within* an Eclipse project including project leader decision rights, prioritization of requirements and committer rights. We designated Eclipse as collectively managed when all modes of design converted from determined by a single firm to independent of any one firm and pluralistically managed

In 2000, IBM launched an internal project entitled Eclipse which aimed to build an integrated development environment that could be the foundation for all IBM middleware company wide. While technology developed by acquired firm OTI formed the basis for Eclipse, the platform was far from meeting the requirements of a diverse set of developers within the organization. And while IBM owned all of the design decisions around Eclipse as a closed platform, it was never completed and brought to market as a proprietary product.

IBM Sponsored
IBM released code for the Eclipse platform under CPL, an IBM published open source license, in 2001 and recruited industry partners to join a consortium tasked with building an open and technically best-in-class integrated development environment. While IBM donated the code and initiated a consortium to manage the platform and develop derivative works, IBM still retained control of all design domains, organizational, architectural and technical. For example, while consortium members all had granted a voice in determining strategic priorities and the scope of

Eclipse, because IBM had the most members actively involved on the consortium board, in strategic committees, and large numbers of developers granted rights to commit code, they were able to control the direction of the platform and subsequent projects developed as part of the community. Regardless, firms joined the open source community and became consortium members. While some firms encountered barriers to influencing the direction of the platform and appropriating value from their efforts. Other firms, figured out how to successfully maneuver around the constraints imposed by IBM.

Transition
In 2004, the consortium created an independent, non-profit Foundation, directed by a representative board, to govern Eclipse. The Foundation board was assigned control of capital management and allocation, stewardship for the Eclipse brand, rights to direct the scope and vision of Eclipse including determining how new projects were introduced and launched. While the Foundation created the premise of separation between IBM and the Eclipse platform, architectural decision making, technical production and implementation was still directed by IBM. Most Foundation staff members either transitioned directly from IBM or were former employees of IBM. Additionally, IBM committed more developers to projects than other firms. This allowed them to enacted barriers such as restricting the posting of solutions and help on common forums and newsgroups which limited the ability of other firms to contribute to the core platform. Further the open source CPL license granted IBM rights as agreement steward. Thusly, diminishing the view that Eclipse was architected for maximum flexibility, extensibility and neutrality.

Collectively Coordinated
The Foundation members drove a set of control changes across legal and technical domains which resulted in a balancing of power across the foundation members. First, the Foundation transitioned the licensing agreement from CPL to EPL, which designated the Foundation as agreement steward. Second, the Foundation established a series of councils (e.g., strategic, architectural, requirements) that guided and coordinated technical direction of Eclipse. Third, the foundation initiated the Eclipse roadmap and "release train". Both of these project requirements ensured: 1) that members were aware and understood the future scope of Eclipse; 2) coordination would need to occur across projects to synchronize release of IP.

Collectively Managed
Further changes were made by the Foundation to integrate firms at the project level. The Foundation started to enforce the rule that new projects must be pitched and committed to by multiple member firms. Further, the Foundation addressed legacy issues of transparency and access for committers and introduced training for members of the community at various levels of contribution (e.g., bug fixer to committer).

Phase 2: Defining Strategic Choices

In the second phase of analysis we identified and defined strategic choices deployed by the firms interviewed. Starting by broadly coding for strategic choices enacted by our sample of firms, we then iteratively refined the set to establish a core group that were most commonly employed. We developed a set of 16 strategic choices firms made to either contribute to the ecosystem and platform or collaborate or compete with the platform. In table 2, we provide definitions.

Phase 3: Identifying Common Ecosystem Strategies

Next, we synthesized the strategic choices firms deployed in the Eclipse ecosystem into four categories. We created these categories in response to the patterns emerging from our data set. To develop these categories we conducted the following steps. First, firms were given a "1" if they executed a strategic step and a 0 if they did not. Firms in some cases executed a strategic step and then ceased to take that action moving forward. For example, some firms joined a project in an early ecosystem phase and then stopped later on. Therefore, we also indicated when each strategic choice was committed. Third, we analyzed patterns across firms to determine if we could synthesize our data into large groups that expressed insight regarding the choices firms make in an ecosystem. Four categories emerged. The four categories were "Competing with the platform", "Contributing to the platform" "Consuming the platform", and "Integrating with the platform". Definitions can be found in table 3.

3.0 Preliminary Findings

While the success of Eclipse hinged on the ability of many firms to collectively build a pervasive technical platform and a robust ecosystem, firms entered into the community with distinct strategies regarding how to engage with the platform and the broader Eclipse ecosystem. *"We created this dual-edged or bi-polar organization that on one-side would play by Open Source rules of engagement to develop the technology, and the other side was the eco-system side, or the commercialization of the technology".* Member firms decided the extent to which they would commit resources to: 1) to build the actual open source Eclipse platform by joining a project, leading a project and developing committers; or 2) purely participate in the ecosystem by joining the community, becoming a board member of Eclipse or privately developing software on-top of the open source code.

Another distinct tension existed between strategic action taken to: 1) create value which could be shared collaboratively across the community: or 2) appropriated competitive by a member firm. This tension is different than the decision to commit resources to the ecosystem and also the platform. Firms committed resources to develop the platform with intention to collaborate or compete. As one informant described, "*Obviously when you do something like this you have to cleverly for both intellectual property reasons and other reasons carefully delineate how much you are going to help the Open Source and how much you are going to do for yourself".*

While committing resources to "help" the open source community often resulted in creating mutually appropriable value. For example, jointly developing Eclipse allowed a host of firms to spread the costs to build and amass the requisite talent necessary to generate the foundational platform technology to begin with. Contributors to the platform could also compete with it. For instance, competitors contributed down-graded code to the platform and withheld higher quality propritary software. As a partner describes, *"Some of my technology stuff I'll Open Source. I'm make sure that my cool stuff will then be proprietary."*

The combination of either contributing to the ecosystem, or the ecosystem and the platform, and competing or collaborating with the platform resulted in unique strategies for engagement. The product of these choices was four ecosystem strategies entitled "*Competing*", "*Contributing*", "*Consuming*", "*Integrating*" that ranged from highly contributory and collaborative to withdrawn and competitive.

3.1 Competing with the platform:

Firms that "*Competed with the platform*," were direct rivals with Eclipse, yet surprisingly they contributed development resources to the platform. Their product offerings included mature IDEs, developer product suites, business intelligence and reporting tools, all of which fiercely competed with the Eclipse platform. However, while they were not users of Eclipse, Competitors "joined as members", "joined the board", "joined projects", and "led projects". One informant from a Competitor remarked, *"We live in the world of two IDE's. Eclipse is one and we think ours is going to be the other for dealing with Web Logic specific tools."* However, Competitors acquiesced to customer urging and joined Eclipse. As firm 1 remarked,

> *"The obvious question is are you doing Eclipse? What does this mean? Oh you so you are capitulating. You are giving in and then you are going to pay your membership fee because you kind of have to. I wanted the powerful story, like if we are going to do this let's come out strong, and not come out like well we were forced into it. It's not just the impression, it's like what are we really going to do. Yeah we've been kind of forced into it."*

Said by another informant, *"Eclipse was beginning to get mindshare and customers. . . People kept asking what about Eclipse"?* While customers forced Competitors to join Eclipse, that does not explain why they contributed resources to develop the platform. Surprisingly most Competing firms in our sample committed over $1.5 million a year in development and financial resources. Competitors contributed resources to both informally (through direct development on the platform) and formally (through the board) shape the direction of Eclipse. As a member commented, *"Are we going to be just barely be forced into it or are we going to*

really embrace this and lead it"? By joining the Eclipse board, Competing firms could dictate how projects were enveloped into the mainstream open source community. By donating resources to develop on the platform, Competing firms determined what to reveal to the open source community, how requirements were prioritized, how extensible the technology would be and the pace of development. For example, some firms used donations to the platform as a method for directly selling their similar but more advanced products. They contributed downgraded code to the platform. They would inform downloaders of the Eclipse code that more advanced, proprietary software was available on their websites for a fee. As firm 6 informed us,

> *"So, there really is no salesperson in the company except me, but we are doing quite well, so word-of-mouth etcetera. So, my idea is that I am going to go and formulate a sales organization around what comes out of my open source stuff. Build a sales organization, a marketing campaign, etcetera. So, at least part of my drive into Open Source will be it's got to be a good enough teaser to get people to want to use it but will also adequately promote my stuff."*

While it seems counterintuitive to invite (which they are directly invited) these firms into the community, when fierce competitors joined, Eclipse gained legitimacy and share in the marketplace. As one member relied about competitors,

> *"There are all of these IDEs that we have to actually create plug-ins for. If everybody, which we wanted to have a little party when Borland announced that they were going to become a strategic developer and everything that they do is now going to be based on Eclipse. We can throw away all of that code. We don't have to—we no longer have to support the Borland IDE. So, that is why Eclipse is critical to us, because if the majority of the development tools standardize on the Eclipse platform we have one plug-in to manage and that is huge for us."*

3.2 Contributing to the platform:

Contributors committed resources to develop the open source platform and then integrated Eclipse into complementary proprietary product offerings. For example, firm number 10, a firm that led an open source project shared,

> *"We would be taking products and make them plug into eclipse. We would be releasing and supporting them, this is a significant investment to do across all of our products. 100+ man years, it is big investment – it would encompass all of our development tools across architectures and the same for performance analysis tools, libraries, debuggers, threading, cluster and performance analysis tools."*

Firms that *Contribute to the Platform* "used Eclipse", "joined as members", "joined the board", "joined projects", "led projects" that supported growth of the platform.

These firms joined to both formally (e.g., board members) and informally influence (develop on the platform) the direction of Eclipse. As one contributor remarked, *"If counting on a component and dependent on features, implementation can get… that is why we are leading the project, at least in terms of managing the product, obviously we cant go roughshod over everyone in the industry but as project leaders and board members we have a great deal of influence over development cadence and feature support"*. For Contributors, purely joining the board was not enough to ensure key components and requirements were: delivered on time; compliant with their proprietary interfaces; and prioritized for their benefit. For example, firm number 9 commented, *"At some point it became clear that becoming a developer made sense because we actually had an agenda that we would like to see in see in Eclipse that we would like to influence Eclipse, that's how we arrived at contributing."*

Beyond attempting to control the technical direction of Eclipse, firms that contributed participated to create a technical solution to a problem otherwise unsolvable by a single firm due to the inability to amass the diversity of talent necessary and prohibitive costs. Member firms wanted to guarantee that Eclipse was built flexibly and extensibly so that firms with varying requirements could leverage it. For example, Eclipse was developed to be compatible with Java, and C++; it was developed for a host of different users from developers to operations personnel.

Firm 16, an organization that ultimately became a Contributor commented,

> *"We went into it not just looking for a C++ Java development environment, but as a general purpose cockpit that would be applicable to multiple audiences that we needed to address…We wanted to use the Eclipse environment for both development and deployment tools that we would present to both developers and operation's people. We thought Eclipse was the best underlying platform that we evaluated to do just that"*.

The member firms acknowledged that talent and skill necessary to accomplish this mission would not be available within the boundaries of any given firm. As firm 10 opines, *"This is about cost savings and efficiency. I mean we could have built our own, but this is more cost effective for us"*.

Lastly, because Contributors participated to efficiently generate and control open, freely available, common technology, their participation triggered commoditization of proprietary products built off the back of what was learned in the community or through their efforts to evolve the platform. Contributors took leading edge technology built privately by other member firms and integrated it into the common, open source Eclipse platform. For instance, one informant protested about a Contributor,

> *"We invested for a year and a half and we created this incredible refactoring technology and product. Just about the time we were launching*

> *it we started to hear these rumors about refactoring. We knew that the people in Switzerland were downloading our free evaluation versions of our software real regularly, which means they knew exactly what we were doing. And when you looked at what they did they did it's like they really like our ideas which was quite flattering to us in between being pissed off. We made a big stink. They came back and said we're really sorry, and we're really embarrassed, but you know we kind of have to put it in the base product."*

As another firm, firm 3 states, *"So, as a company ever since the first company was started in '88 we have always been adding innovative, leading edge, functionality to some base set of software, usually an IDE. So, it is our way of life to have our functionality what we think of as eaten from below by the main vendors that we are partnering with."*

3.3 Consuming the platform:

Quiet parasites that guardedly assessed the state of affairs on the platform, firms that "Consume the platform," joined the community to build competitive products on-top of the platform, and before the rest of the community could move. These firms contributed to the broader ecosystem because they "commercialized around Eclipse," "joined Eclipse," and sometimes "joined the board," but they did not contribute any development resources to build the platform. Consumers typically were inventive, leading edge technology firms that constantly innovated up the product ladder away from integrated development environments, which sat at the bottom rung, into application services, middleware applications, etc.

They joined the community to first source information regarding the future direction of Eclipse and the surrounding ecosystem. *For example, firm number 5 comments, "You may learn something that you didn't learn before. You may see something happening that is a market trend that you'd want to jump on because it's something that you think is going to have merit in terms of the business investment. Being involved in these communities I think again, has lots of benefit if you are smart enough and have the wherewithal to take advantage of it".* Second, Consumers sought out quality code instantiated by the community, which could cheaply lay the groundwork for new proprietary products. Firm number 4 commented, *"What we'll do is if we want to go into a particular product area, we'll go look at the Open Source efforts that have been done in the area. When we find some with compatible license and reasonable quality that get us 50-percent of the way there, or 60 or 70 whatever it happens to be. We'll take that technology and we'll integrate it, and we'll extend."* Third, firms that were Consuming the platform, attempted to attenuate the forces of commoditization brought about by other firms in the community. For example, one firm engaged in consuming the platform stated:

> *"Truthfully this is my opinion, of course like most of it. I think most of the "Open Source" backers just want free stuff, and Eclipse is no different. It's*

a really good development tool and they want it for free, and they want it more and they want it today. So, if it commodities every commercial piece of software in the space no one cares. The user community doesn't care".

3.4 Integrating with the platform:

Integrators joined the community to learn the best methods to align their existing products with Eclipse. These firms recognized that compatibility with Eclipse was critical, but they did not typically develop new products on-top of the platform. Firms "Integrating with the platform" joined the community, and occasionally joined the board. They did not contribute development resources to the platform. One of our Integrator informants commented, *"In the past we have had plug-ins to do just about everything you can imagine. The idea of having one plug-in to write means instead of writing plug-ins, our developers can go and write more features for building. So, Eclipse is critical to us because of that reason."* Integrators strategically attempted to direct Eclipse to architect software which was both extensible and vendor neutral. As our informant mentioned, Integrators benefited when the market for IDEs consolidated. When Eclipse serviced the requirements of the broadest set of users, Integrators were required to dedicate less internal resources to comply with market alternatives.

Firms that were integrators built Eclipse market share. For example, one integrator commented, *"We offer support but only to big organizations. We offer basically service when customers want an environment that is not necessarily a complete Eclipse eco-system but they pick plug-ins that they would like to work with. There is no single source. There is no single vendor. It is not necessarily all in one schedule. So, someone needs to take care of as things integrate, new versions work with older versions of other plug-ins"*.

REFERENCES:

Baldwin, Carliss Y., and C. Jason Woodard. "The architecture of platforms: a unified view." *Harvard Business School Finance Working Paper* (2008) pp.09-034.

Chesbrough, Henry. "The logic of open innovation: managing intellectual property." *California Management Review* 45, no. 3 (2003): 33-58.

Chesbrough, Henry. "Open innovation: a new paradigm for understanding industrial innovation." *Open innovation: Researching a new paradigm* (2006): 1-12.

Chesbrough, Henry, Wim Vanhaverbeke, and Joel West, eds. *Open innovation: Researching a new paradigm*. OUP Oxford, 2006.

Chesbrough, Henry and Melissa Appleyard. Open innovation and strategy. *California management review*, *50*(1), (2007) pp.57-76.

Chesbrough, Henry, Wim Vanhaverbeke, and Joel West, eds. *New frontiers in open innovation*. OUP Oxford, 2014.

Cusumano, Michael A., and Annabelle Gawer. "The elements of platform leadership." *MIT Sloan management review* 43, no. 3 (2002): 51.

Dittrich, Koen, and Geert Duysters. "Networking as a means to strategy change: the case of open innovation in mobile telephony." *Journal of product innovation management* 24, no. 6 (2007): 510-521.

Eisenhardt, Kathleen M., and Melissa E. Graebner. "Theory building from cases: Opportunities and challenges." *Academy of management journal* 50.1 (2007): 25-32.

Eisenmann, Thomas R., Geoffrey Parker, and Marshall W. Van Alstyne. "Opening platforms: how, when and why?." *This paper has been published under the same title as* (2008): 131-162.

Gawer, Annabelle, and Michael A. Cusumano. "How companies become platform leaders." *MIT Sloan management review* 49, no. 2 (2008): 28.

Gawer, Annabelle, and Michael A. Cusumano. *Platform leadership: How Intel, Microsoft, and Cisco Drive Industry Innovation*. Boston: Harvard Business School Press, 2002. Gawer, Annabelle, ed. *Platforms, Markets and Innovation*. Edward Elgar Publishing, 2011.

Gawer, Annabelle, and Rebecca Henderson. "Platform owner entry and innovation in complementary markets: Evidence from Intel." *Journal of Economics & Management Strategy* 16.1 (2007): pp.1-34.

Grodal, Stine, Aleksios Gotsopoulos, and Fernando F. Suarez. "The Coevolution of Technologies and Categories During Industry Emergence." *Academy of Management Review* 40.3 (2015) pp. 423-445.

Eisenmann, Thomas R., Geoffrey Parker, and Marshall W. Van Alstyne. "Opening platforms: how, when and why?." *This paper has been published under the same title as* (2008): pp.131-162.

Henkel, Joachim. "Selective Revealing in Open Innovation Processes: The Case of Embedded Linux." *Research policy* 35.7 (2006) pp. 953-969.

Hippel, Eric von, and Georg von Krogh. "Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science." *Organization science* 14.2 (2003) pp. 209-223.

Iansiti, Marco, and Roy Levien. *The Keystone Advantage: What the New Dynamics of Business Ecosystems Mean for Strategy, Innovation, and Sustainability*. Harvard Business Press, 2004.

Laursen, Keld, and Ammon Salter. "Open for innovation: the role of openness in explaining innovation performance among UK manufacturing firms." *Strategic management journal* 27, no. 2 (2006): 131-150.

O'Mahony, Siobhán. "The Governance of Open Source Initiatives: What Does it Mean to be Community Managed?." *Journal of Management & Governance* 11.2 (2007) pp. 139- 150.

O'Mahony, Siobhán, and Karim R. Lakhani. "Organizations in the Shadow of Communities." *Research in the Sociology of Organizations* 33 (2011) pp. 3-36.

Parker Geoffrey G., Marshall W. Van Alstyne and Sangeet Paul Choundary. *Platform Revolution.* Forthcoming 2015

Powell, Walter W., and Peter Brantley. "Competitive Cooperation in Biotechnology: Learning through Networks." *Networks and Organizations* 7 (1992) pp. 366-394.

Powell, Walter W., Kenneth W. Koput, and Laurel Smith-Doerr. "Interorganizational Collaboration and the Locus of Innovation: Networks of Learning in Biotechnology." *Administrative science quarterly* (1996) pp. 116-145.

Powell, Walter W., Doug R. White, Kenneth W. Kopet, and Jason Owen-Smith. "Network Dynamics and Field Evolution: The Growth of Interorganizational Collaboration in the Life Sciences." *American Journal of Sociology* 110. 4 (2005) pp. 1132-1205.

Quinn, James Brian, Jordan J. Baruch, and Karen A. Zien. *Innovation Explosion: Using Intellect and Software to Revolutionize Growth Strategies*. Simon & Schuster, 1997.

Sinha, Kingshuk K., and Andrew H. Van de Ven. "Designing Work within and

between Organizations." *Organization Science* 16.4 (2005) pp. 389-408.

Utterback, James M. "Innovation in Industry and the Diffusion of Technology." *Science* 183.4125 (1974) pp. 620-626.

Von Krogh, Georg, and Eric Von Hippel. "The Promise of Research on Open Source Software." *Management Science* 52.7 (2006) pp. 975-983.

West, Joel, and Siobhán O'Mahony. "The Role of Participation Architecture in Growing Sponsored Open Source Communities." *Industry and Innovation* 15.2 (2008) pp. 145- 168.

Yin, Robert K. "Applied social research methods series Case study research: Design and methods." (1984).

Table 2: Definitions for Strategic Choices

Strategic Choice	Definition	Representative quote
Use Eclipse	Utilize Eclipse as an internal integrated development environment	Firm 7: "Once we saw Eclipse the development staff here started glomming onto it. That naturally, you know our developers started using it, they liked it."
Join Eclipse	Become a formal member of Eclipse consortium or foundation	Firm 14: "It was a real simple decision. We did not make that decision for ourselves. It was kind of decided for us by our customers. Most of our customers initially were IBM customers as well. When WebSphere Application Developer came to play they jumped on that wagon and they wanted us to support that effort, because what we are building, we are in business of application life cycle management, and so we help people to be more efficient as they build and deploy their applications. So, integration was development environment, and production environment, and the business environment that they have was crucial to the success of our business. So, that was one of the major reasons initially why we decided to join Eclipse. So, we've been there since almost the very beginning."
Become Board Member	Hold position on the board of stewards or foundation based on election or overall contribution to the community	Firm 4: "When it first became visible to me I really scratched my head and said this is the first time I ever heard of an Open Source project with a board of directors. An Open Source project that is fundamentally a consortium industry heavyweight. Prior to that it's has always been the exact opposite. Its organic amateur even when the individuals may be professions in real life active as amateur's very grassroots and then the corporations get involved to take advantage, contribute, very different evolution then this thing. So, I'm not aware of any similar technology where it started as a corporate technology. It's been immediately been created as "Open Source" with this corporate overlay with a board of directors etcet I guess if I perceived my role in general board member role, it is over side observance."
Commercialize around Eclipse	Develop a propritary product that is built on top of the Eclipse platform	Firm 1: "We need to build on Eclipse. We need to stop what we are doing now, our last year of development building on this thing, and change and move onto Eclipse, because customers are demanding for it, and we just need to do it. We need to bite the bullet and slip our schedule and do it now."
Donate Code/IP	Distribute freely and share existing IP or Code with open source community	Firm 7: "The contribute aspect where commercial quality software appears overnight is still a bit of a problem. But what has happened in reality, even though code contributions of pieces of commercial tools have shown up they've typically been, they've still gone through a due diligence process of will it be approved, and how much of it we take, and will we change it and that kind of thing? So, that has actually helped quite a bit. It's not just we ramrod. People realize that just because you have a contribution doesn't mean it's the best thing for Eclipse just because it was the best thing for the basis commercial tool. "
Join a Project	Contribute development capacity to an offical Eclipse Project	Firm 16: "It varies, and it goes up and down, probably between six and ten. It goes up and down based on what we are trying to do. You get enough of a base there that you can have a relatively small number like that but still have impact right? If we were driving our own platform with our own plug-ins, we'd probably need double or triple that number of people committed. So, it's a real leverage benefit to us."
Join Multiple Projects	Contribute development capacity to multiple projects	Firm 3: "We've got two Open Source, two Eclipse projects running right now. One we might have put in there anyway because we don't care so much about it. The other one we did because we care."
Lead a project	Architect, arbitrate and guide development of key priorities for an offical Eclipse project based on inputs from the community	Firm 11: "The complexity of integration. If counting on a component and dependent on features, implementation can get….that is why we are leading the project – at least in terms of managing the product, obviously we cant go roughshod over everyone in the industry but as project leaders and board members we have a great deal of influence over development cadence and feature support."
Develop committers	Effectuate developers to achieve status as "committers" which enables locking/publishing of code into Eclipse community	Firm 4 "They are committing to it. I think that you'll find that people who are committing to it more have a lot more committers it's just the best way to gather information."
Sync development schedule with Eclipse	Plan proprietary product releases to align with Eclipse releases (e.g., signfigantly before, after or with an Eclipse release)	Firm 5: "I think you have to be prepared to be light on your feet. You have to be adaptive, and you have to be prepared to quickly move forward with initiatives based on what is going on with Eclipse. So, I think it's time—it's a critical time to market notion."

Strategic Choice	Definition	Representative Quote
Integrate Eclipse into current product suite	Augment existing product suite to comply and integrate with Eclipse	Firm 13: "In the past we have had plug-ins to just about everything you can imagine. The idea of having one plug-in to write means instead of writing multiple plug-ins our developers , we can go and write more features for building. So, Eclipse is critical to us because of that reason."
Observe Eclipse community	Actively take notice of direction and activity within Eclipse Community	Firm 7: "Naturally I watch the areas that first affect me. At Eclipse that is mainly things like the Web tools project that is out there."
Front run Eclipse	Build propritary products that improve on Eclipse offerings and move up the stack	Firm 3: "But we are there, we are involved primarily to watch. It could be competitive. It could be competitive, or it could be very technological advantageous to us. If they ever got their fundamental basic code to be better than ours, which right now it's nowhere near as good as ours, then we'll take theirs and put it under our product, because our user functionality sits on top of a bunch of stuff. They are really working down at a lower level. If they get good we'll just take it."
Sell up from the platform	Develop code on Eclipse that is limited or inferior to existing propritary product offerings	Firm 6: "So, at least part of my drive into Open Source will be it's got to be a good enough teaser to get people to want to use it but will also adequately promote my stuff. So, I'll sit down and classically with Open Source type of stuff and this kind of thing the way you do lead generation is let somebody download a free version and then they like it, or you do like MYSQL and let them have it for free on business purposes. So, I'm probably going to use the Open Source stuff as a lead generation capability."
Compete head on with Eclipse	Sell a product that directly competes with the Eclipse platform	Firm 1: "We do have a product that we already have out on the market and we were worried that this might cannibalize the business there. That is a risk that was, that was almost a certainty. "

Table 3: Ecosystem Strategy Definitions

Group	Definition	Characteristics	Representative Quote
Competing with the platform	Compete directly with Eclipse or a member firm within the ecosystem; provide development and financial resources to the platform	**Strategic Choices:** Join Eclipse, Become Board Member, Commercialize around Eclipse, Donate Code/IP, Join a Project, Lead a project, Develop committers, Sync development schedule with Eclipse, Compete head on with Eclipse, Front run Eclipse, Sell up from the platform **Rationale:** Acquire customers/serve customers; Attenuate commoditization, Control direction of development, Increase partcipation and decision rights of partners	*It was clear from listening to our customers that they weren't going to consider our IDE in its current form. They had adopted Eclipse even though we had done nothing. That if we were going to continue to sell them technology ,it needed to be supported by the Eclipse tools and the best way to make that happen was to have us involved with Eclipse.*
Contributing to the platform	Embed Eclipse in existing and new product suites; provide financial and development resources to the platform	**Strategic Choices:** Use Eclipse, Join Eclipse, Become Board Member, Commercialize around Eclipse, Donate Code/IP, Join a Project, Join Multiple Projects, Lead a project, Develop committers, Sync development schedule with Eclipse, Integrate Eclipse into current product suite, Observe Eclipse community **Rationale:** Generate or Learn to generate best technology, Control direction of development, Spread cost to build, Achieve return on investment	*We are basically trying to build the best design tool we can and make that available on Eclipse, Open Source. But for running or in deployment we have a big commercial offering, a server offering in which we will ultimately integrate with the open source project. So, just like IBM makes money off of the Web Server stuff and let's people build applications with the tools, we are going to let people build reports with the tools and execute them if you will in our server.*
Consuming the platform	Utilize the platform as an input to build competitive propritary products or as a marketing channel for propritary products; do not develop on the platform	**Strategic Choices:** Use Eclipse, Join Eclipse, Become Board Member, Commercialize around Eclipse, Sync development schedule with Eclipse, Observe Eclipse community, Compete head on with Eclipse, Front run Eclipse **Rationale:** Access information, Attenuate commoditization, Achieve return on investment	*And a lot of it really wasn't that IBM was really trying to do it, they just weren't aware. Again, it's when the elephant moves ants get squashed. Does the elephant know? No, not unless you have one screaming ant. I was the screaming ant that kind of rose their awareness that you guys will not build a community if you insist on commoditizing everything below what a multi-billion dollar company can put out.*

Group	Definition	Characteristics	Representative Quote
Integrating with the platform	Access information to embed Eclipse in existing product suites or components; very limited development on the platform	**Strategic Choices:** Use Eclipse, Join Eclipse, Integrate Eclipse into current product suite, Observe Eclipse community **Rationale:** Access information, Ensure platform extensibility and vendor neutrality, Increase partcipation and decision rights of partners	*We had the source, so it's not like by being a contributor we got the source and other people didn't. Everyone has it. We got access to IBM and we got some materials that we might not have gotten otherwise. We didn't seem to get any preferential treatment.*

The Quest for UML in Open Source Projects Initial findings from GitHub

Regina Hebig[1], Truong Ho-Quang[1], Gregorio Robles[2], and Michel R.V. Chaudron[1]

[1] Chalmers — Göteborg University
{hebig,truongh,chaudron}@cse.gu.se
[2] GSyC/LibreSoft, Universidad Rey Juan Carlos
grex@gsyc.urjc.es

Abstract. UML is frequently used in commercial software development because of its beneficial effects of specifying, constructing, visualizing and documenting software. However, little is known about the use of UML in Free/Open Source Software (FOSS) projects. A number of quantitative studies have been conducted to discover the use of UML in FOSS projects. However these studies are limited in the number of projects and contributors. In this paper we study GitHub - one of the biggest code hosts in the world. Our first contributions is an automated process for collecting UML diagram. For this process, firstly, a list of files which potentially contain UML instances was obtained from over 1.24 million GitHub repositories. We then used a number of filters to automatically identify the files that actually contain UML diagrams. An additional contribution is that we consider a novel set of research questions related to the process of creation and use of UML and to the contributors of UML. Example questions are: when, during a project, are UML diagrams introduced, when are UML files are updated, and which developers author and change UML files.

1 Introduction

The Unified modeling language (UML) provides the facility for software engineers to specify, construct, visualize and document the artifacts of a software-intensive system and to facilitate communication of ideas [2]. For commercial software development, the use of UML has been introduced and commonly accepted to be a prescribed part of a company-wide software development process.

When it comes to Free/Open Source Software (FOSS) development, characterized by dynamism and distributed workplaces, code remains the key development artifact [1]. Little is known about the use of UML in open source. Researchers in the area of modeling in software engineering have performed some efforts to collect examples of models and of projects that use modelling. However the results are often limited [17]. For example, the Repository for Model Driven Development (ReMoDD)[5] is an initiative driven by an international consortium of leading researchers in the field of modeling. Nevertheless its

content is growing at a low rate: after 7 years (summer 2014) it contains around 60 models. Industrial projects are very reluctant to share models because they believe these reflect key intellectual property and or insight into their state of IT-affairs.

Due to the limited success in systematically identifying open source projects with UML, many researchers (including the authors themselves at the start of this study) are rather pessimistic finding much use of UML in open source projects. Furthermore, since most open source platforms, such as GitHub, do not provide facilities for model versioning, such as model mergers, we were even more pessimistic about finding examples of UML models that were updated over time.

The lack of available data is the reason why so far no answers could be given to several basic questions on the amount of UML files in open source projects that are static or updated, the time span during which models are created or updated during the open source project, or the question which of the project's contributors do create models. Thus it seems that UML is not frequently present in FOSS projects. However, there is no exact quantification of its presence.

GitHub hosts around 10M of non-forked repositories, which makes it a good starting point to obtain an estimation of the use of UML in FOSS projects. GitHub's web search is limited for this type of endeavor as it targets mainly source code searches by developers. However, there are many other ways to access GitHub data (GHTorrent or the GitHub API), but as we will show obtaining data on UML usage is not trivial.

In this paper we present our efforts to mine GitHub in order to gain a list of open source projects that include UML models. It turned out that for achieving this goal we required to join forces and expertise from different fields. The first challenge is the identification of non-forked repositories in GitHub with the help of the GHTorrent [6] in order to retrieve candidates for files that might include UML diagrams. Since these many of these diagrams are stored in formats that can also include other information than models, e.g. images or XML based files, it is further necessary to perform an automated recognition of those files that actually are UML. Therefore, it is required to perform two different checks, one for XML based formats and one for images, which is a state of the research technology that just became available in 2014 [7]. Finally, with the retrieved list of UML models, the git repositories of these projects were triggered in order to retrieve information about the repositories and further information about commit and update histories of these models.

As a result we gain out of over 1.24 million repositories a first list of 3 295 UML projects.

Our preliminary findings are are:

- **1.** A first list of 3 295 GitHub repositories including altogether including 21 316 UML models.
- **2.** Based on this data we give for the first time answers descriptive questions about the number of models that are subject to updates, the number of

contributors adding or manipulating models, and the point in a projects life time where models are created and updated.
– **3.** Furthermore, this research provides the basis to ask questions related to "when" and "by whom" UML models and other artifacts are introduced and updated together. Surely the approach has still limitations, for example we will not be able to identify how often the models are read.

However, we believe that these first descriptive results are just a starting point. They enable us and other researchers to formulated and address more advanced questions about UML usage and its impacts on a project in future work.

2 Related research

This paper builds on previous research done in two research communities: the software modelling- and the mining software repositories communities.

2.1 Use of UML in FOSS

Studies on the usage of UML are frequently done amongst in industry (mostly through surveys) [14, 18].

However, only few studies focus on freely available models, such as can be found in open source projects. Reggio et al. [14] investigated which UML diagrams are used based on diverse available resources, such as online books, university courses, tutorials, or modeling tools.

While this work was done mainly manually, Karasneh et al. use a crawling approach to automatically fill an online repository[1] with so far more than 700 model images [9]- Both works focus on the models only and do not take their project context into account. Further, they do not distinguish between models that stem from actual software development projects and models that are created for other reasons, e.g. teaching.

An index of existing model repositories can be found online [17][2]. However, in addition to their small size, these repositories seldom include other artifacts than the models, making it impossible to study the models in the environment of actual projects.

Further, there are some works addressing small numbers of case studies of modeling in open source projects. Yatani et al. studied the models usage in Ubuntu development by interviewing 9 developers. They found that models are forward designs that are rarely updated[20].

Osman et al. investigated 10 case studies of open source projects from Google-code and SourceForge that use UML. They focused on identifying the

[1] `http://models-db.com/`

[2] Index of model repositories `http://www2.compute.dtu.dk/~hsto/fmi/models.html`

ratio of classes in the diagrams compared to classes in the code. They find only seldom cases where models are updated [13].

Finally, there are three works that actually approach a quantitative investigation of models in open source projects. Chung et al. questioned 230 contributors from 40 open source projects for their use of sketches [3] and found that participants tend to not update these sketches.

A study that focuses on software architecture documentation in open source projects was performed by Ding et al. They manually studied 2 000 projects from SourceForge, Google code, GitHub, and Tigris. Amongst those projects that used such documentations they identified 19 projects that actually use UML [4].

The work that is probably closest to our study is the one of Langer et al. They searched for files conforming to the enterprise architect file format (which is a format that can be used to store UML files) within Google code, assembla, and GitHub. They identified 121 models. They further assessed the model lifespan (between introduction and last update) to be in average 1 247 days[10]. However, studying a single file format is a rather limited view on UML. Furthermore, the project perspective is not considered and they rather put a focus on the used UML concepts.

2.2 Mining

Mining software repositories has mainly focused on aspects related directly to (programming) source code. However, projects may include non-source-code sources such as images, translation, documentation or user interface files, that can be usually identified by their extension [16]. By doing so, research has shed some light on the variation and specialization of workload that exist in FOSS communities [19].

The study of specific file formats that are non-source code can be found as well in the research literature: McIntosh et al. have investigated the build system for its evolution [11] and effort [12], or the analysis of infrastructure as code that has become mainstream in the last years [8].

3 Methodology

In this section, we describe our study approach. The overall process is shown in Figure 1.

First, we obtained a list of 10% of GitHub repositories from GHTorrent [6] that are not forks. This resulted in a list of file of 1 240 000 repositories, those that had a downloadable branch. From this list, potential UML files were collected using several heuristic filters based on the creation and storage nature of UML files (Step 1).

An automated process was built on the basis of image processing [7] and textual analysis to examine the existence of UML notation in the obtained

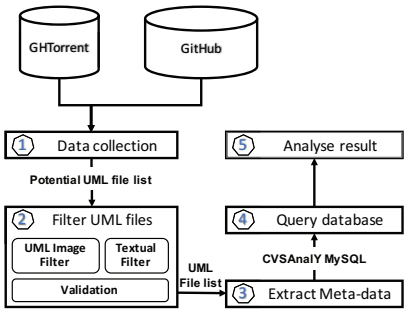


Fig. 1. Overall processes

files (Step 2). A manual validation step is taken in order to consolidate the classification result.

We have then obtained the meta-data from those repositories where a UML file has been identified by means of using the CVSAnalY tool [15] (Step 3).

In step 4, we queried the metadata (taken in Step 3) with respect to our research questions. We answer the research questions by analyzing the result (Step 5). Note that during the data analysis further files got lost for diverse reasons (see discussion section 4). Thus, we were finally able to analyze a set of 21 316 UML model files.

4 Discussion

Considering our initial expectations we were surprised to find such a big number of projects with UML. Surely, 3 295 projects are still a small number compared to the overall number of GitHub projects. Nonetheless, the identification of 21 316 UML models exceeds by far the expectations that we had based on the numbers of models found so far in open source projects in related work, e.g. 121 models by Langer et al.[10] or 19 projects with UML by Ding et al.[4].

Data consistency We want to shortly discuss the type of data that we can get with the presented mining method. The method we applied is not trivial and consist of several steps of data collection. For example, we search for UML candidates using a GHTorrent dump, but accessed the GitHub API to retrieve further information about model versions and project contributors. Due to the difference in time between the creation of the GHTorrent dump and the request to the GitHub API, we had drop outs of identified models and projects during the second step.

In addition, we performed this method for the first time, which had an exploratory component in trying out what kind of data we can (and need to) retrieve. This led to the situation that we accessed the GitHub API several times, leading to different drop-outs in models and projects for the different types of information collected.

A *lessons learned* is that, for the next analysis, we have to make a clear planning of all required data in advance, to ensure that at least the second threat to data consistency can be reduced. For this paper we addressed the problem with a reduction of the finally analyzed data set to models and projects for which we had the data points that are necessary to answer the different research questions.

Static models A finding is that many projects use UML only in a very static way. In such projects models are never updated and often all models are introduced at the same point in time. These results confirm findings from smaller studies such as Yatani et al.'s or our own Osman et al.ś, who both found that updates of models are rare[20, 13]. This can have different reasons. One optimistic interpretation would be that models are just introduced as first architectural plans that are followed and used as documentations, but never changed. Another rather pessimistic interpretation would be that modeling is just "tried-out" at some point in time and then dropped. An observation that at least supports the idea that the first interpretation plays a role is that in most projects the main activities of introducing models happen during the first 10%-20% of all commit activities.

Projects with regular model usage Another number that we consider surprisingly high is the number of projects (or models) with more than 20 model updates as well as projects with more than 1 year of active UML creation and modeling. Again, compared to the number of overall GitHub projects the here found number seem small. Nonetheless, it was unexpected to find several projects that seem to use modeling on a regular basis.

We found out that a minority of projects (less than 10%) have UML active phases that cover nearly the whole project life time. For a majority of projects the active UML phase is very short and often concentrated in the first commits. It has to be noted that the results we found are in contrast to the study of Langer et al. who found an average model lifespan of 1 247 days, while studying 121 enterprise architecture models in open source [10]. We found much lower lifespans. The difference in the findings might be caused by the fact that enterprise architect is a modeling tool that is rather used in an industrial context. Thus, the probability that the projects studied by Langer et al. have industrial support is very high.

Different populations A preliminary finding of our research is that there seem to be different populations of model usage. There is a difference in the number of model updates between projects with more than 100 model files and projects with less than 100 model files. One reason for different populations could be the actual form of model usage and creation. Models might be created manually or automatically (e.g. through reverse engineering). They might solve as plans for system design or as description for an already existing system. Model updates might be performed in order to make small corrections after an initial

creation (leading to updates within in short span of time) or in order to make a documentation up to date after a longer phase of system change.

At the current state we do not know whether these populations can actually be distinguished on their characteristic commit and update pattern. However, a further hint that they might play a role can be seen in the relatively constant distribution models by the amount of commits that were already done within a project. We can see model introductions at all project ages. The in average short time of active UML creation and modification speaks against the idea that these introductions at different points in time happen within the same projects. Thus, it seems that we have to deal with different groups of projects introducing their models at different points in time.

In future work we plan to have a closer look at the model usage in order to study whether we can associate pattern to different populations of model use.

Duplicates The large number of identified duplicates leads to questions. What are the reasons for duplicates? Missing model versioning techniques alone cannot explain the found results. Furthermore, it is not clear yet whether these duplicates represent a form of model use. E.g. if models are adopted together with code from other projects, they might be used to understand the alien code that is embedded in a new project.

Paving the way for future research Finally, one of our main contributions is that we presented a method to systematically mine for UML models in GitHub and that this leads to an enormously promising set (much larger than any existing set of projects) for future analysis. On the one hand this will help us to address in future question that arise from the findings of this paper. For example, concerning the model updates, it would be interesting to consider following questions:

– Are models updated by their original authors or by other people?
– In how many projects are UML files obsolete?

Further considering the time of model introduction, we would like to address the following question further: Has the time of introduction an influence on the "success" of an open source project, i.e., the question how many developers join a project? And of course we would like to address the question whether different populations of model-usages can be statistically distinguished.

Even more important, the hereby published list of open source projects using UML can help other researchers to progress in their studies. For example:

– What kind of UML diagrams are used most often?
– What coding languages are used most often in combination with UML?
– What files are changed together with changes in architectural models?
– Can UML help to attract and integrate inexperienced developers?

Furthermore, the data can be used as input to find case studies for other model or architecture related research, such as:

– Does a good architectural design in models help to create a good architecture in the code?
– Tools for traceability management and model merging can benefit from the real case studies.
– Research that integrates models into fault prediction can be evaluated with the help of that data.

Thus, we believe that the identified initial list of open source projects with UML will be of great help for other researchers, too.

5 Conclusions

In this paper we joined forces in repository mining and model identification in order to identify open source projects on GitHub that contain UML models.

As a result we can present a list of 3 295 open source projects which include together 21 316 UML models. This is the first time the modeling community can establish a corpus comparable to collections already exist for source code only, such as QualitasCorpus[3]. Furthermore, the relatively low amount of UML projects amongst the investigated GitHub projects (0.28%) reconfirmed that our systematic mining approach was required in order to establish the corpus.

We analyzed the data to gain first descriptive results on UML model usage in open source. One finding is that the majority of models is never updates, but that projects exist that do update their models regularly. Furthermore, we learned that models can be introduced during all possible phases in the lifespan of an open source project. Nonetheless most models are introduced during the first 10% of the duration of projects.

A few projects are active with UML during their whole lifetime. However, most projects work very shortly actively on UML, usually at the beginning. We found that 12% of the distinct models occurred several times. Duplicates are in average spread across 1.88 projects.

In the future we plan to further explore the possibilities that arise with the here presented new method to collect data about UML usage in open source projects. For example we plan to analyze the impact of model usage on project dynamics, such as the number of people joining projects. We are planning to proceed with mining GitHub in future work. Based on the now investigated 10% of GitHub we expect that GitHub includes around 34 000 projects with UML and together around 200 000 UML models. Furthermore, we will investigate possibilities to identify UML models that are embedded in other files such as manuals stored in pdf.

[3] QualitasCorpus `http://qualitascorpus.com/`

References

1. O. Badreddin, T. C. Lethbridge, and M. Elassar. Modeling practices in open source software. In *Open Source Software: Quality Verification*, pages 127–139. Springer, 2013.
2. G. Booch, J. Rumbaugh, and I. Jacobson. *Unified Modeling Language User Guide, The (2Nd Edition) (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2005.
3. E. Chung, C. Jensen, K. Yatani, V. Kuechler, and K. N. Truong. Sketching and drawing in the design of open source software. In *Visual Languages and Human-Centric Computing (VL/HCC), 2010 IEEE Symposium on*, pages 195–202. IEEE, 2010.
4. W. Ding, P. Liang, A. Tang, H. Van Vliet, and M. Shahin. How do open source communities document software architecture: An exploratory survey. In *Engineering of Complex Computer Systems (ICECCS), 2014 19th International Conference on*, pages 136–145. IEEE, 2014.
5. R. France, J. Bieman, and B. H. Cheng. Repository for model driven development (remodd). In *Models in Software Engineering*, pages 311–317. Springer, 2007.
6. G. Gousios and D. Spinellis. Ghtorrent: Github's data from a firehose. In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pages 12–21. IEEE, 2012.
7. T. Ho-Quang, M. R. V. Chaudron, I. Samúelsson, J. Hjaltason, B. Karasneh, and H. Osman. Automatic classification of uml class diagrams from images. In *Proceedings of the 2014 21st Asia-Pacific Software Engineering Conference - Volume 01*, APSEC '14, pages 399–406, Washington, DC, USA, 2014. IEEE Computer Society.
8. Y. Jiang and B. Adams. Co-evolution of infrastructure and source code - an empirical study. In *12th IEEE/ACM Working Conference on Mining Software Repositories, MSR 2015, Florence, Italy, May 16-17, 2015*, pages 45–55, 2015.
9. B. Karasneh and M. R. Chaudron. Online img2uml repository: An online repository for uml models. In *EESSMOD@ MoDELS*, pages 61–66, 2013.
10. P. Langer, T. Mayerhofer, M. Wimmer, and G. Kappel. On the usage of uml: Initial results of analyzing open uml models. In *Modellierung*, volume 19, page 21, 2014.
11. S. McIntosh, B. Adams, and A. E. Hassan. The evolution of ant build systems. In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pages 42–51. IEEE, 2010.
12. S. McIntosh, B. Adams, T. H. Nguyen, Y. Kamei, and A. E. Hassan. An empirical study of build maintenance effort. In *Proceedings of the 33rd international conference on software engineering*, pages 141–150. ACM, 2011.
13. M. H. Osman and M. R. V. Chaudron. UML usage in open source software development : A field study. In *Proceedings of the 3rd International Workshop on Experiences and Empirical Studies in Software Modeling co-located with 16th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2013), Miami, USA, October 1, 2013.*, pages 23–32, 2013.
14. G. Reggio, M. Leotta, and F. Ricca. Who knows/uses what of the uml: A personal opinion survey. In *Model-Driven Engineering Languages and Systems*, pages 149–165. Springer, 2014.

15. G. Robles, J. M. González-Barahona, D. Izquierdo-Cortazar, and I. Herraiz. Tools for the study of the usual data sources found in libre software projects. *International Journal of Open Source Software and Processes*, 1(1):24–45, 2009.
16. G. Robles, J. M. Gonzalez-Barahona, and J. J. Merelo. Beyond source code: the importance of other artifacts in software development (a case study). *Journal of Systems and Software*, 79(9):1233–1248, 2006.
17. H. Störrle, R. Hebig, and A. Knapp. An index for software engineering models. In *International Conference on Model Driven Engineering Languages and Systems (MoDELS) 2014*, pages 36–40, 2014.
18. M. Torchiano, F. Tomassetti, F. Ricca, A. Tiso, and G. Reggio. Relevance, benefits, and problems of software modelling and model driven techniques - A survey in the italian industry. *Journal of Systems and Software*, 86(8):2110–2126, 2013.
19. B. Vasilescu, A. Serebrenik, M. Goeminne, and T. Mens. On the variation and specialisation of workload - a case study of the gnome ecosystem community. *Empirical Software Engineering*, 19(4):955–1008, 2014.
20. K. Yatani, E. Chung, C. Jensen, and K. N. Truong. Understanding how and why open source contributors use diagrams in the development of ubuntu. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 995–1004. ACM, 2009.

Evolution and Influence of Sub-groups on Group Productivity and Success

Pinar Ozturk
School of Business, Stevens Institute of Technology
1 Castle Point on Hudson
Hoboken, New Jersey USA 07030
pozturk@stevens.edu

Abstract. In this study, we investigate the determinants and impact of community-level collaboration network structure on group success. We focus one specific type of success measured by the productivity of the group. We examine how the sub-groups (modules) form within the larger group, how they change and how their existence within the group affects the success of the overall group. Specific hypotheses are developed and proposed using longitudinal panel of 512 WikiProjects at Wikipedia.

Keywords. Online community, social networks, modularity, productivity, project success

1 Introduction

The relationship between the group and the individual is at the heart of the many long-standing scholarly debates in literature. One of the main debates concerns the task-performing capabilities of groups relative to individuals. While individuals create most user-generated content, an increasing amount emerges from groups of people working collectively in knowledge-producing communities. These communities represent unique forms of organized, collaborative activity. Examples of such communities include open-source software projects such as Sourceforge and Linux, where developers work together to create software products; the wiki websites such as Wikipedia, where contributors work together on articles; and virtual worlds such as World of Warcraft, where players create shared spaces and perform shared tasks [32]. These examples all involve collaboratively generated content.

Given that content can be generated either by several people working together as a group or by those same people working individually, does it matter which work arrangement or collaboration structure is employed? Are some collaboration structures more conducive to productivity than others? If so, what are the determinants and impact of such collaboration structures on organizational outcomes and success?

Understanding the collaboration structure and identifying different sub-groups have both been active areas of research in sociology, physics, biology and computer science. This line of research characterizes the organization of collaborative relationships as networks of individuals and discusses the effects of the structural properties of these networks on organizational outcomes. Many studies have showed that the network structure and organization of people and groups can in large part affect the quality and quantity of organizational outcomes and success [13, 24]. These collaboration networks have been found to organize into densely linked sub-groups that are commonly referred as network communities, clusters, groups or modules [14, 41]. The network modules represent dense and non-overlapping structural groups of actors and can affect information diffusion [12], conflict [31] and emergence of cooperation [26]. Previous research has suggested that presence of modular structure and network clusters shape the degree to which groups acquire and interpret knowledge [13].

Similarly in knowledge-producing communities, the structure of the community can be regarded as a dynamic collaboration network of individuals whose connections shapes and is reshaped through their interactions with each other and the content they work on. The resultant structure in turn can implicitly influence how they interact with each other and perform as a whole. Social network analysis techniques have been used extensively to study the collaboration network of open source communities and its impact on group performance, project outcomes and success. For example, Crowston and Howison [10] examined 120 project teams from SourceForge, and compared the centralization measures of the different projects. They concluded that not all open source software projects are associated with a particular social structure, and suggested that open source software project teams should spend more effort on creating social structures, which are considered to be favorable. Colazo [7] found that while network ties may improve OSS project quality, they may slow down the speed of project development. Grewal et al. [16] found that project network embeddedness positively influences project technical success, while the effect of project manager network embeddedness is more complex and different for older projects when compared with younger projects.

These studies adopting network approach for OSS success mostly have focused on the individual measures (e.g. centrality measures), with only few considering the implications of the collaboration network structure at the level of a particular project. While other studies of network structure investigated the implications on project success at the project level, they have used cross-sectional data and did not explore the determinants – what leads to such collaboration network structures. For example, an analysis of development community at Sourceforge showed that the networks of

developers tend to be highly clustered [42]. The authors proposed that this observation could be a consequence of homophily and the success of a project may be related to the underlying clustered structure of the OSS development community; however, they did not explore these propositions. Further analysis of Sourceforge development community showed that clustering of developer networks was indeed positively related to the success of the projects, measured by the number of CVS commits [35, 36, 37]. While the reasons why such clustering exists wasn't explored, it was argued that the positive effect of high clustering on success might be due increased levels of trust among developers and the skill composition of the developers embedded in the clusters.

Few studies have also investigated the collaboration network structure of other knowledge-producing communities such as Wikipedia. However, similar to studies on open source communities, most have focused on individual measures (e.g. centrality measures) of articles rather than implications of the collaboration network structure at the level of a particular groups or projects. An analysis of the collaboration networks associated with 300 articles on medical and health related topics showed that the position of the article, measured by degree and eigenvector centrality, within the global network of articles is positively related to the article quality [17]. Another analysis of WikiProject Medicine contributor and article network revealed that embeddedness of the article (measured through local and global centrality) is positively related to viewership of that article [32]. Collaboration networks of 3085 featured articles also showed that more centralized collaboration networks lead to faster article promotion or getting featured [29].

In this study, we investigate both the determinants of collaboration network structure and its implications on group success. Specifically, we focus on understanding how the network modules forms, how they change and how the existence of such modules within the group affects the success of the group. The key idea behind this approach is to consider groups (modules) of contributors instead of individual contributors. By doing so, this approach accounts for potential synergies between groups of contributors that become visible only when the contributors are analyzed jointly. Using longitudinal data we are able to account for project specific unobserved heterogeneity and lag relationship between project productivity, success and network measures. Since most previous studies exploring the collaboration network structure at the group level have been carried out in the context of OSS projects, we chose Wikipedia WikiProjects as our research setting. To operationalize project success, we use the number sticky (non-reverted) edits on articles within the scope of the project.

2 Theoretical Development

2.1 Modularity

Modularity is a property of a network and a specific proposed division of that network into smaller subgroups or modules. The modularity of a network is a scalar value between -1 and 1 and measures the density of links within modules as compared to links between modules [30]. These sub network modules, by definition, include individuals who have more frequent interactions connecting them to each other than to others in the network. Since individuals are free to choose who they work with, repeat interactions indicate the presence of strong bonds and engenders a sense of belonging among members of a module. This reduces uncertainty in the behavior of individuals and, hence, increases levels of trust and reciprocity in the relationships [20].

Trust and reciprocity in a relationship promotes diffusion of information and resource sharing [33] and can facilitate the productivity of members by allowing them to access the local pool of knowledge through short direct ties. Frequent interactions can encourage the accumulation of shared background knowledge [38] and using this shared background knowledge, members could more easily identify a range of opportunities for recombining complementary knowledge available within community.

Frequent interactions between the members of a module also lead to an increased sense of mutual accountability [39] and ensure the reliability of information received [8], which in turn helps with norm enforcement [5] and carry out sanctions against individuals who do not follow those norms [8]. This discourages members from acting opportunistically and try to gain advantages for themselves at expense of others [28] since such information would be quickly related to other members. By fostering trust and promoting the enforcement of social norms, social cohesion that occurs within modules offers the facilitating conditions for coordination and collaborative endeavors.

A project with high modularity means there are well-separated modules within the project and these modules differ from each other based on the frequency of interactions among their members. While frequent interactions and social cohesion within modules increases trust, shared background knowledge and rapid diffusion of information within modules, the presence of well-separated modules on the project level can help with co-existence of several ideas and preserve the fundamental diversity of knowledge conducive toward the production of further new knowledge [23]. Having high modular structure may also allow for high specialization on the group level [43], which in turn might lead to effective task distribution and higher productivity. Hence, we posit that

Hypothesis 1: *The modularity of a project will be positively related to project success.*

2.2 Determinants of Modularity

In previous studies, scholars have articulated rationales for how and why collaboration and social action emerge from both external social foci (content focused attributes) as well as internal human agency (editor focused attributes). While our first hypothesis focuses on a content attribute (i.e. popularity), our second hypothesis focuses on contributor attributes, (i.e. editor expertise).

2.2.1 Content Popularity

Each contributor individually decides to contribute and participate in knowledge-producing communities. An individual's contribution behavior depends jointly on his/her motivation and intention, which is deeply related to his/her individual interest and ability [1, 34]. Hence, potential contributors of knowledge-producing communities actually participate in the community in order to work on the content they are interested in. Consequently, it can be expected that the content that are able to interest more people would gather more actual contributors. And when interest is higher for certain content, the popularity of that content would also be high. Therefore, higher levels of interest of potential contributors in certain content would lead to higher numbers of actual contributors.

Frequent interactions of members within modules represent that they have worked on the same content. The more content they work together, the stronger the tie that connects them to each other than to others in the network. Hence, the modules can be seen as epistemic communities of people bounded by shared interest around certain content [27]. If modules are driven by content, popularity of content would increase number of people being clustered around certain topics and lead to more separated sub-groups on the project level. Hence, we posit that

Hypothesis 2: *The popularity of content (articles) within the scope of a project will be positively related to project modularity.*

2.2.2 Contributor Experience

It has been widely observed that in knowledge producing communities the minority of contributors provides a majority of the content, resembling a power law distribution of content contribution [40]. This has been seen in Wikipedia [19], Usenet posting [11] as well as open source communities [2]. This power law distribution causes knowledge producing communities to have a core periphery structure, where the majority of contributors participate at the community's periphery, while a relatively small portion of contributors take on additional responsibilities and constitute the core [4, 6].

Prior studies investigating core-periphery structure of knowledge producing

communities have defined contributors' position on the core-periphery continuum based on the quantity and types of activities they perform [21, 25]. It was found that while highly experienced editors exhibit a tendency toward concentrating their work in a few articles by deep and sustained involvement; inexperienced apprentice editors play a crucial role of contributing to many articles and acting as crucial brokers providing bridges [18].

Since modularity of a network represents how well separated sub-groups exist in a network, we expect experienced editors to drive the formation of modules around certain content and inexperience editors to act as a bridge between the modules. Hence we posit that

Hypothesis 3: *The experience level of contributors within a project will be positively related to project modularity.*

3 Research Setting and Method

In this paper, we use social network analysis to examine how the collaboration network structure characteristics affect project success. We specifically focus on a type of approach which has proven particularly effective in identifying sub-groups and which is based on the optimization of a quality function, the modularity [30]. The modularity of a network is a scalar value between -1 and 1 that measures the density of links between subgroups as compared to links between sub-groups.

3.1 Research Setting

WikiProjects are illustrative examples of these online communities, embedded in the broader ecosystem of Wikipedia. Each WikiProject consists of participants with a variety of motivations, expertise and levels of interest, tasks designed around shared goals and interests, and digital features that facilitate sustained interactions. Wikipedia hosts over 2000 WikiProjects, of which 603 are considered active as of August 2015.

The dataset used in this study is extracted from the February 2016 archive of English Wikipedia, which includes full text of all pages and their complete edit history from the beginning of Wikipedia. To gather the sample of WikiProjects, the main directory page of WikiProjects was traversed. Inactive projects and the projects that are not topical (e.g., WikiProject Deletion sorting) were excluded from the sample. In addition to projects that have less than three members and projects whose scope cannot be estimated using categories were also excluded. The resulting dataset included 512 topical WikiProjects.

The scope of each WikiProject (the pages fall under the project) was estimated by finding the Wikipedia category that matches project's title (i.e., Category: Cycling for WikiProject Cycling) and finding all articles that fall under that category.

Since Wikipedia category and subcategory structure resembles a graph structure, all the subcategories of the matched category down to the 3rd level deep were traversed and all articles in those categories were included in the scope of the WikiProject. Among the WikiProjects in our sample, the lowest scope included 13 wiki articles while the largest scope included 65,656 articles.

As an alternative measure, the claimed scope of each WikiProject (i.e. Category: WikiProject Sports articles) was also collected. However, claiming articles under such categories is a manual process and only one third of the projects were found to have created such claimed scope. Consequently, project scopes were determined using the category structure, which seemed a more reliable measure to compare breadth across projects.

3.2 Dependent Variable

Knowledge producing communities in general, and Wikipedia specifically, are notable in that they produce content systematically, in a coordinated fashion but without the luxury of incentives and direct profit motives [22] that exist in traditional firm-driven endeavors. Therefore, it is not always easy to define success for such projects.

As the success measure that would most closely relate to our theory and provide validation for project success, we chose the number of sticky (non-reverted with a month) edits on the articles within the scope of a project. Sticky edits signifies meaningful addition to the articles and represent a way in which knowledge creation is instantiated [9, 16].

3.3 Independent Variables

Modularity is measured as the modularity of the collaboration network in which nodes represent the contributors of a WikiProject and edges represent their editing interaction on the pages under the scope of the project. If two members had edited the same page, a tie exists between them. The more pages they edit together, the stronger the tie that connects them. In this study, the modularity of interaction networks is quantified by using Newman's algorithm [30].

We operationalize content popularity as the number of times a Wikipedia article is viewed in a given month. For each article within each WikiProject, we collect the number of views each day for the time period the project network is constructed.

Contributor experience is measured as the number of edits a contributor has performed until the end of the time period the project network is constructed.

3.3 Control Variables

Project Scope is measured as the number of articles within the project scope. Scope is determined by the category and subcategory structure of Wikipedia.

Project Size is measured as the number of contributors who have contributed 5 or more edits to the articles within the project scope.

4 Analysis

4.1 Performed Analysis

As an exploratory analysis, we created visualizations for modularity and productivity of each project (see Figure 2 as an example). With one-year iteration, starting from the year the project was founded, we created a collaboration network of the project in which nodes represent the members of a WikiProject and edges represent their editing interaction on the pages under the scope of the project. If two members had edited the same page, an edge exists between them. At every year iteration, we calculated the modularity of collaboration networks using Newman's algorithm [30] and collected the number of sticky edits made on the articles within the scope of that WikiProject. For example, WikiProject Lakes was founded in 2006. The first network we created for WikiProject Lakes included 5842 Wikipedia editors and their 25,286 edits made on the WikiProject Lakes articles until the end of 2006. The last network created included 82,299 editors who made 453,309 edits on the project articles until the end of 2015. In total we have created 11 networks for WikiProject Lakes (from 2006 to end of 2015).

Our exploratory visualizations suggest that the modularity of a project is positively related to project success (Figure 1).

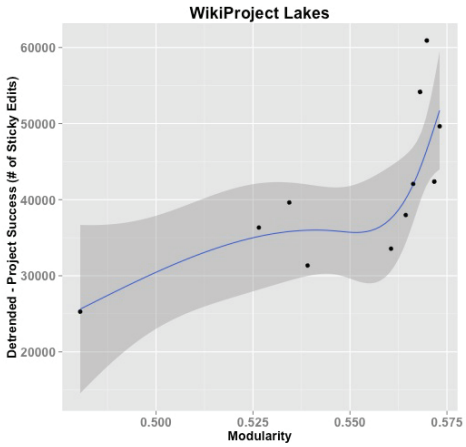


Fig. 1. Relationship of WikiProject Lakes collaboration network modularity and project success (measured as the number of sticky edits)

4.2 Planned Analysis

We plan to use pooled OLS, Random effects and Fixed effects estimation for testing our hypotheses. We will create one model for testing hypothesis 1 – the effect of network modularity on project success and another model for testing hypothesis 2 and 3 – the effects of content popularity and contributor experience on the modularity of the collaboration network.

Empirical study of the effect of collaboration network structure on project success suffers from an issue related to reverse causality – whether a project's success shapes the network rather than the reverse. In order to address the issue of reverse causality, we plan to employ a lag structure and measure our modularity construct prior to the dependent variable (project success). To select appropriate lag structure of the independent variable, we will employ alternative lags of independent variable relative to dependent variable. Hence, we measure the dependent variable, $Project\ Success_{it+j}$, as the number of sticky edits for project i in j years subsequent to network construction date t and $Modularity_{it}$. For instance, for network constructed in December 2005 and j=1, we count the number of sticky edits for a project from December 2005 to December 2006.

Recent research has shown the importance of understanding collaboration network in understanding open source communities. It has also been shown that structural pattern of relations in collaboration networks can have significant impact on how participants actually behave. However, these collaboration networks evolve over time due to changing community boundaries, fluctuating participation and dynamic interactions of the participants. In this context, we suggest that it is important to understand how different network structures emerge, how they might co-evolve with the interactions of the participants and how they impact productivity and success of the community.

4 References

1. Ajzen, I. (1991). The Theory of Planned Behavior. *Organizational Behavior and Human Decision Processes* 50(2), 179-211.
2. Arafat, O., and Riehle, D. (2009). The commit size distribution of open source software. *HICSS 2009. Proceedings*, 1-8. IEEE.
3. Berkes, F., and Folke, C. (1998). Linking social and ecological systems for resilience and sustainability. *Linking social and ecological systems: management practices and social mechanisms for building resilience*, *1*, 13-20.
4. Bryant, S. L., Forte, A., and Bruckman, A. (2005). Becoming Wikipedian: transformation of participation in a collaborative online encyclopedia. In *Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work*, 1-10. ACM.
5. Burt, R. S. (2005). *Brokerage and closure: An introduction to social capital*. New York: Oxford University Press.
6. Butler, B., Joyce, E., and Pike, J. (2008). Don't look now, but we've created a bureaucracy: the nature and roles of policies and rules in wikipedia. In *Proceedings of the SIGCHI conference on human factors in computing systems*, 1101-1110. ACM.
7. Colazo, J. A. (2010). Collaboration structure and performance in new software development: findings from the study of open source projects. *International Journal of Innovation Management*, *14*(05), 735-758.
8. Coleman, J. (1988) Social Capital in the Creation of Human Capital. *American Journal of Sociology* (94), 95-120.
9. Crowston, K., Annabi, H., and Howison, J. (2003). Defining open source software project success. *ICIS 2003 Proceedings*, 28.
10. Crowston, K., and Howison, J. (2005). The social structure of free and open source software development. *First Monday*, *10*(2).
11. Fisher, D., Smith, M., and Welser, H. (2006) You Are Who You Talk To: Detecting Roles in Usenet Newsgroups. *HICSS 2006 Proceedings*, vol. 3. IEEE.
12. Galstyan, A., and Cohen, P. (2007). Cascading dynamics in modular networks. *Physical Review E*, *75*(3), 036109.
13. Gibson, C., and Vermeulen, F. (2003). A healthy divide: Sub- groups as a stimulus for team learning behavior. *Administrative Science Quarterly,* 48: 202–239
14. Girvan, M., and Newman, M. E. (2002). Community structure in social and biological networks. *Proceedings of the national academy of sciences*, *99*(12), 7821-7826.
15. Granovetter, M. S. (1973). The strength of weak ties. *American journal of sociology*, 1360-1380.

16. Grewal, R., Lilien, G. L., and Mallapragada, G. (2006). Location, location, location: How network embeddedness affects project success in open source systems. *Management Science*, *52*(7), 1043-1056.
17. Kane, G. C. (2009). It's a Network, Not an Encyclopedia: A Social Network Perspective on Wikipedia Collaboration. In *Academy of Management Proceedings* (Vol. 2009, No. 1, pp. 1-6). Academy of Management.
18. Keegan, B., Gergle, D., and Contractor, N. (2012). Do editors or articles drive collaboration?: multilevel statistical network analysis of wikipedia coauthorship. In *Proceedings of the ACM 2012 conference on computer supported cooperative work*, 427-436. ACM.
19. Kittur, A., Chi, E., Pendleton, B. A., Suh, B., and Mytkowicz, T. (2007). Power of the few vs. wisdom of the crowd: Wikipedia and the rise of the bourgeoisie. *World wide web*, *1*(2), 19.
20. Kollock, P. (1994) The Emergence of Exchange Structures: An Experimental Study of Uncertainty, Commitment, and Trust. *American Journal of Sociology* (100:2), 313-345.
21. Kriplean, T., Beschastnikh, I., and McDonald, D. W. (2008). Articulations of wikiwork: uncovering valued work in wikipedia through barnstars. In *Proceedings of the 2008 ACM conference on Computer supported cooperative work* , 47-56. ACM.
22. Lakhani, K., and Wolf, R. G. (2003). Why hackers do what they do: Understanding motivation and effort in free/open source software projects.
23. Lambiotte, R., and Panzarasa, P. (2009). Communities, knowledge creation, and information diffusion. *Journal of Informetrics*, *3*(3), 180-190.
24. Lau, D. C., and Murnighan, J. K. (2005). Interactions within groups and subgroups: The effects of demographic faultlines. *Academy of Management Journal*, *48*(4), 645-659.
25. Liu, J., and Ram, S. (2011). Who does what: Collaboration patterns in the wikipedia and their impact on article quality. *ACM Transactions on Management Information Systems (TMIS)*, *2*(2), 11.
26. Lozano, S., Arenas, A., and Sánchez, A. (2008). Mesoscopic structure conditions the emergence of cooperation on social networks. *PLoS one*, *3*(4), e1892.
27. Margolin, D., and Monge, P. (2013). Conceptual Retention in Epistemic Communities: Examining the Role of Social Structure. *Communication and community*, 1-24.
28. McGinn, K. L., and Keros, A. T. (2002). Improvisation and the logic of exchange in socially embedded transactions. *Administrative Science Quarterly*, *47*(3), 442–473.
29. Nemoto, K., Gloor, P., and Laubacher, R. (2011). Social capital increases efficiency of collaboration among Wikipedia editors. In *Proceedings of the 22nd ACM conference on Hypertext and hypermedia,* 231-240. ACM.
30. Newman, M. E., and Girvan, M. (2004). Finding and evaluating community structure in networks. *Physical review E*, *69*(2), 026113.

31. O'Leary, M. B., and Mortensen, M. (2010). Go (con) figure: Subgroups, imbalance, and isolates in geographically dispersed teams. *Organization Science*, *21*(1), 115-131.
32. Ransbotham, S., Kane, G. C., and Lurie, N. H. (2012). Network characteristics and the value of collaborative user-generated content. *Marketing Science*,*31*(3), 387-405.
33. Rogers, E. M. (2003). *Diffusion of innovations* (5th ed.). New York: The Free 21 Press.
34. Simon, B., Loewy, M., Stürmer, S., Weber, U., Freytag, P., Habig, C., Kampmeier, C. and Spahlinger, P. (1998). Collective Identification and Social Movement Participation. Journal of Personality and Social Psychology 74(3), 646-658.
35. Singh, P. (2007). Open source software development and the small world phenomenon: An empirical investigations of macro level colaboration network properties on project success. *ICIS 2007 Proceedings*, 18.
36. Singh, P. V., Tan, Y., and Mookerjee, V. (2008). Network effects: The influence of structural social capital on open source project success. *Management Information Systems Quarterly*.
37. Singh, P. V. (2010). The small-world effect: The influence of macro-level properties of developer collaboration networks on open-source project success. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *20*(2), 6.
38. Strang, D., and Meyer, J. W. (1993). Institutional conditions for diffusion. *Theory and Society*, *22*(4), 487–511.
39. Uzzi, B. (1997). Social structure and competition in interfirm networks: The paradox of embeddedness. *Administrative Science Quarterly*, *42*(1), 35–67.
40. Voss, J. (2005). Measuring Wikipedia. In Proceedings of the International Conference of the International Society for Scientometrics and Informetrics (ISSI), 10, Stockholm
41. Yang, J., and Leskovec, J.(2015). Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, *42*(1), 181-213.
42. Xu, J., Gao, Y., Christley, S., and Madey, G. (2005). A topological analysis of the open souce software development community. *HICSS 2005* Proceedings, 198. IEEE.
43. Whitley, R., 2000. The Intellectual and Social Organization of the Sciences.New York: Oxford University Press.

Crowston, K., Hammouda, I., Lindman, J., Lundell, B., and Robles, G. (Eds.) Proceedings of the Doctoral Consortium at the 12th International Conference on Open Source Systems, Skövde University Studies in Informatics 2016:1, ISSN 1653-2325, ISBN: 978-91-978513-9-8, University of Skövde, Skövde, Sweden.

Copyright of the papers contained in this proceedings remains with the respective authors.

Skövde University Studies in Informatics 2016:1
ISSN 1653-2325
ISBN 978-91-978513-9-8

www.his.se

UNIVERSITY
OF SKÖVDE