TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

Dissertation

Quality Prediction and Evaluation Models for Products and Processes in Distributed Software Development

Ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Doktors der technischen Wissenschaften/der Naturwissenschaften unter der Leitung

von

a.o. Univ. Prof. Dr. Stefan Biffl

o. Univ. Prof. Dr. A Min Tjoa

Institut für Softwaretechnik und interaktive Systeme

Eingereicht and der Technischen Universität Wien

Fakultät für Informatik

von

Dindin Sjahril Fadjar Wahyudin

0527519

1030 Wien, Baumgasse 58/43

dindin@ ifs.tuwien.ac.at

Wien, am 5 November 2008

Deutsche Kurzfassung

Die moderne Entwicklung großer Softwaresysteme erfolgt typischerweise in einem verteilten – häufig global ausgerichteten – Umfeld. Eine verteilte Softwareherstellung (*Distributed Software Development*, *DSD*) ist sowohl in traditionellen Unternehmen und Organisationen als auch verstärkt im Open Source Umfeld anzutreffen. Speziell in DSD-Projekten benötigen Projekt- und Qualitätsleiter geeignete Methoden zur Evaluierung des aktuellen Projektfortschritts (beispielsweise die Qualität der bisher erstellten Produkte und durchgeführten Aktivitäten) und zuverlässige Modelle für die Einschätzung der zu erwartenden Produktqualität für neue Softwareversionen (*Releases*).

Durch die globale Verteilung der Projektteams im DSD-Umfeld wird die Einschätzung des Projektfortschritts zunehmend komplexer und unzuverlässiger, da eine persönliche Abstimmung in Form von Meetings zur Überprüfung von (high-level) Schätzergebnissen, wie beispielsweise der Fertigstellungsgrad einer Softwareversion, aufgrund der räumlichen Trennung meist nur schwer realisierbar sind.

Eine zentrale Frage für Projekt- und Qualitätsleiter im Rahmen der Projektsteuerung ist, ob eine vorliegende Softwareversion bereits freigegeben werden kann oder ob Teile des Produktes vor der Freigabe noch überarbeitet werden müssen. Zur Unterstützung dieser Entscheidung sind erforderlich:

(1) *Evaluierungs-Framework* zur Einschätzung der Produktqualität (welche Daten müssen erhoben werden und wie können sie zu aussagekräftigen Metriken aggregiert werden)

(2) *Gültigkeitsprüfung* der erhobenen Daten auf unterschiedlichen Granularitätsstufen.

(3) *Schätzmethoden* über die zukünftige Produktqualität einschließlich einer *Rückkopplung* über die Gültigkeit der Schätzergebnisse.

Die Fehlerfreiheit bzw. die Anzahl der im Produkt verbleibenden Fehler ist ein Hauptkriterium für die Feststellung der Qualität eines Softwareproduktes im Rahmen dieser Arbeit. Ein Fehler ist dabei definiert als eine Abweichung der Lösung von definierten Anforderungen (sowohl funktionale Anforderungen aber auch nicht-funktionale Anforderungen, wie Zuverlässigkeit, Sicherheit, Wartbarkeit oder Benutzerfreundlichkeit). Da Fehler in einem Softwareprodukt den realen Wert für den Anwender reduzieren, sind Fehleraufzeichnungen und Fehlervorhersagen für künftige Releases zentrale Kenn- und Steuergrößen für die Qualität eines Produktes bzw. für die Planung neuer Produkte und Releases.

Fehlerzählungen und Datensammlungen aus Qualitätsverbesserungsinitiativen sind zentrale Elemente zur Feststellung der Produktqualität (beispielsweise Fehlerkorrektur im Rahmen eines Entwicklungsprozesses). Aufbauend auf historischen Projekt- und Fehlerdaten wird ein Vorhersagemodell erstellt, das die Basis für Fehlerschätzungen zukünftiger Releases bildet.

In der industriellen Praxis werden Frameworks für die Evaluierung der Produkt- und Prozessqualität für traditionelle Softwarekennzahlen eingesetzt. Diese Frameworks unterstützen jedoch keine verteilte Entwicklung im DSD-Umfeld, wie sie im Rahmen der Entwicklung von verteilten Open Source Software Produkten (OSS) erforderlich ist. Die Ergebnisse einer systematischen Literaturanalyse zeigten folgende Einschränkungen bei Vorhersagemodellen in einem verteilten Entwicklungsumfeld:

(1) Unsystematische Planung von Qualitätsvorhersagen.

(2) Modelle, die ausschließlich auf Produktmetriken aufgebaut werden, sind zwar an sich ausreichend, weisen aber eine zu geringe Zuverlässigkeit in der Fehlerschätzung bzw. Vorhersage auf. Dieser Nachteil wirkt sich speziell bei Projekten mit kurzen Entwicklungszeiten, wie es in zahlreichen OSS Projekten üblich ist, aus.

(3) Unzureichende Qualität der erhobenen Daten aus unterschiedlichen und heterogenen Datenquellen.

Zentrale Forschungsfragen dieser Arbeit sind a) eines systematische Untersuchung großer OSS Projekte im Hinblick auf wichtige Projektergebnisse und Prozessattribute, die zur Einschätzung der Produktqualität und des Projektfortschrittes verwendet werden können und b) die Untersuchung der Angemessenheit und Zuverlässigkeit fortgeschrittener Modelle für objektive Vorhersage der Produktqualität im DSD Umfeld.

Zentrale Forschungsbeiträge dieser Arbeit:

1. *Prozess-Qualitäts-Metriken*, so genannte "*Project health indicators*", erfassen Entwicklungsaktivitäten im Rahmen der Produktverbesserung und ermöglichen die Evaluierung dieser Indikatoren in DSD-Projekten.
2. *Qualitätsindikatoren* für Softwareprodukte, die einen Wahrscheinlichkeitsbereich zur Verfügung stellen statt eines fixen Wertes ohne Berücksichtigung von Datenschwankungen.
3. *Forschungs-Roadmap* für die Software Fehlervorhersage basierend auf einem systematischen Literaturreview.
4. *Strukturiertes Framework für Qualitätsvorhersage* für Verteilte Softwareentwicklungen.
5. Verbesserte *Modelle zur Qualitätsvorhersage* basierend auf Produkt- und Prozessmetriken, die aus heterogenen Projekt-Repositories gewonnen werden können (z.B. Issue Tracker, Source Code Management).
6. *Empirische Evaluierung* im Rahmen von realen Softwareprodukten in einem realen Umfeld: Fallstudien unterschiedlicher OSS Projekte.

Der vorgestellte Forschungsansatz basiert auf den folgenden Annahmen: In einem ausreichend stabilen Prozessumfeld können durch eine Reihe von Beobachtungen der Produktqualität im jeweiligen Kontext a) signifikante Kontextparameter identifiziert und b) die Produktqualität zu einem bestimmten Zeitpunkt basierend auf den gemessenen Kontextparametern vorhergesagt werden. Dieser Ansatz wird am Beispiel von DSD (OSS) Projektumfeld vorgestellt. Eine weitere mögliche Anwendung, die nicht im Fokus dieser Arbeit steht, umfasst die Evaluierung und Vorhersage von Qualitätsattributen in kommerziellen (nicht OSS) Projekten.

Table of Contents

ABSTRACT

Modern large-scale software development is typically organized in distributed, often globally dispersed, environments. Distributed Software Development (DSD) projects occur both in traditional organizations and increasingly sophisticated open source development initiatives. Leading roles in DSD, such as the project manager and the quality manager, need to evaluate the actual project progress (e.g., quality of products produced and activities conducted) and trustworthy models for the prediction of future product quality such as release candidates. However, in a DSD context the human reporting of progress becomes increasingly complex and the reliability can become questionable, particularly if face-to-face meetings are not possible that allow to personally checking the validity of high-level estimates such as the readiness of a software version for release.

For steering the project, e.g., by deciding to release a current software version or to wait and re-work parts of the software, project managers need 1. a quality evaluation framework that defines what data to collect and convert into meaningful numbers on a higher level; 2. an approach to check data for validity on all levels; 3. an approach to predict the quality of future products with feedback on the likely accuracy of the prediction result.

In this work we focus on absence of defects as the major quality criterion of a software product, where defects are deviations from requirements that need to be repaired. The focus on defect counting and defect prediction are particularly important as defects decease value for users, and other quality criteria (e.g. reliability, security, maintainability, usability) can be formulated as requirements and thus defects can also cover these criteria. Hence, the terms of quality evaluation in our context is focus on counting defects and collect data that is related to quality improvement (e.g., development processes that related to defect removal activities). While quality prediction consists of collecting historical data from project data sources to construct prediction models that can be used to estimate number of defects or defective work products prior to release.

Unfortunately, while there are quality evaluation frameworks in traditional software metrics, to our knowledge there is no appropriate framework available that can be calibrated to modern DSD environments such as Open Source Software (OSS) environments. Moreover, our systematical literature review found that prediction models in distributed development settings have to cope with the following limitations such as a) unsystematic quality prediction planning, b) models based on product metrics alone shown sufficient accuracy but poor reliability in particular for projects with short development cycle such as in many OSS projects c) insufficient quality of collected data originated from heterogeneous project data sources.

Key research questions of this thesis are a) to investigate for large OSS projects the most important development artifact and process attributes that can indicate software product quality for project progress evaluation and b) to investigate the accuracy and reliability of advanced models for objective quality prediction in the context of DSD projects.

Main research contributions of this work are:

1. Process quality metrics, so-called project "health indicators", which capture correlated development activities in product quality improvement, and propose ways to evaluate such "health indicators" in DSD projects.
2. Quality indicators for software products that provide a range for the likely value of the indicators rather than a fixed value without indication of data volatility.
3. Research roadmap for software defect prediction based on systematical literature review
4. Structured framework for quality prediction in distributed software development settings
5. Improved quality prediction methods based on product and process metrics that can be collected from heterogeneous project repositories (e.g. issue tracker, source code management tool).
6. Empirical evaluation in a range of real-world distributed software engineering environments: case studies from different contexts OSS projects.

The general approach is based on the following assumption: in a sufficiently stable process context a sequence of observations on product quality and context parameters allow a) identifying significant context parameters and b) prediction of product quality at a point in time based on the measured context parameters. We use this approach in the context of DSD (OSS) projects; however, there are many other potential application areas such as quality evaluation and prediction for operational software systems.

ACKNOWLEDGEMENTS

I would like to thank Professor Stefan Biffl and Prof. A Min Tjoa for guiding me through my PhD and for spending lots of time for thorough discussions, and providing very constructive feedback for my research topics.

Furthermore, I want to thank my colleagues Alexander Schatten, Dietmar Winkler, Mathias Heindl, Thomas Moser, Rudolf Ramler, Richard Mordinyi, Khabib Mustofa, Munir Merdan, Jiri Kubalik, Kamil Matousek, Michael Schadler and Benedikt Eckhard for interesting discussions, research collaborations and experience exchanges.

Last but not least, I want to thank my wife Neni Novitasari and my parents for supporting me during my studies.

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

Today the engineering of software intensive systems (SISs) especially in medium to large software companies has been shifted from traditional collocated development style towards distributed software development (DSD) as it promises cost reductions, extended access to expert pools and market proximity [102]. DSD can be defined as Software development in geographically distributed settings [23] which can be occur both in traditional closed-source organizations and increasingly sophisticated open source software (OSS) development initiatives.

The success of a DSD project depends among other things on the quality of the resulting product. Thus, project manager and quality manager in DSD need a) to properly evaluate current work products quality level and activities conducted and b) to predict the quality of future software product (i.e., software release candidates) prior to release and to identify needs for improvement.

Prikladnicki et al. [102] as well as Sengupta et al [110] advocate that although DSD significantly impacts of how current software products are designed, developed, tested and deployed, nevertheless the stories of failures in distributed projects can be alarming due to poor planning, poor quality, and cost overruns. These issues are derived from the complexity of development process in DSD [49] such as:

1. Size and structural of the project. The more project participant involved in a project and the more distributed their location then the more complex is the project. In globally distributed software project (i.e., Global Software Development and Open Source Software Development) the time zones and cultural differences limit the availability of team member to work together at the same time, and motivate a concern of the transparency of the development processes [120]. Nevertheless, the increased of project complexity alleviates the challenge for project managers to plan and control the project, organize the collaborative works, as well as monitoring project participants' work performance and quality.
2. Heterogeneous changing processes and products. Each project participant is working on different processes to produce deliverables according to his assigned role. Each distributed process uses and produces heterogeneous artifacts (e.g. requirements, code set, test case, design elements, methods) that keep evolving throughout project lifecycle [126]. In metrics based quality prediction method the processes and the artifacts are

source of data that can be used to estimate of the product prior to release. The more functionally should be delivered means higher number of artifact and more involved processes which consequently increase the complexity in project monitoring and quality assurance.

3. Semantic of data collection. Ideally although project participants are distributed around the globe they work with centralized project repositories, however in most cases project participants have to store or communicate their work deliverables using different shared tools such as mailing list, issue tracker, source code management (SCM), forum, etc [124]. In some cases, these separated development teams usually owned or replicate the project repositories locally. As the results the data collection effort has been increased due to fragmented data that come from heterogeneous repositories, moreover the quality often disputed due to incomplete and missing data during collection.

Thus, to efficiently evaluate and predict the quality of development processes and products in DSD requires development of methods and technologies to address these issues.

1.1 Research Issues and Research Challenges

This section outlines the key research issue and respective research challenges that will be addressed in this thesis.

1.1.1 Research Issues

Quality can be defined as *conformance to requirements* [20] or *fitness to use* [52] of a product. From Software engineering domain, Kan [53] defines that software quality can be measured as a low level of defects in the product, since a low level of defects can be translated as high level of conformance to requirements and fitness to use. A defect can be defined as *a lack of something necessary for completeness, adequacy, or perfection* (*Merriam Webster Dictionary*). Florac [35] defines software defect as any flaw or imperfection in a ***software* work product** or ***software* process**. Project and Quality manager need to evaluate the current quality of development process and product in order to check whether the predefined requirements can be achieved and as basis for quality prediction after the product being released.

Hence, from Empirical Software Engineering perspective [31], in this work we focus on two quality aspects in Distributed Software Development:

1. **Evaluation of distributed development processes quality (EQ)**: we assume that the

quality of product is the result of correlated distributed development processes that can be defined, measure and evaluated throughout project life cycle. In this area we put our attention to correlated development processes that related to defect detection, defect validation and defect removal activities.

The research issues in this area are:

EQ1: *to investigate the distributed development processes those have impact to product quality improvement*

EQ2: *to propose ways to measure correlated development processes as project "health indicators" which reflect the quality of current development processes and may provide prognosis of project survivability*

EQ3: *empirical evaluation of proposed health indicators using data from large open source projects*

2. **Software quality prediction in distributed software development settings (QP)**: good quality software products are those with absence or low number of defects detected. Many companies perform defect prediction. Software quality prediction in particular defect prediction is important since it addresses crucial aspects of how software quality can be improved prior to release with data from product and process collected during development. In this thesis we focus on defect prediction using data collected directly from DSD project repositories to construct objective quality prediction models.

   The research issues in this area are:

   QP1: *to improve the accuracy and reliability of advanced models for objective quality prediction in the context of DSD projects*

   QP2: *to efficiently and effectively collect data from project repositories to construct objective prediction models*

   QP3: *empirical evaluation of proposed objective quality prediction models using data from large open source projects*

However, we notice that when challenged by the complexity of DSD projects, current approaches of software quality evaluation and prediction have following shortcomings:

1. Traditional evaluation and controlling of development processes are typically derived from human based reports which focuses on tracking formal achievements, manually analyzing the collaboration between team members during certain events (e.g., meeting, discussion, etc), and personal reports from each project team member [26, 100]. Nevertheless human based approach in complex distributed environment as in a typical

DSD project will likely to be biased, error-prone and expensive, moreover Keil [55] reports that often team members do not report the actual condition when the project is in critical situation.

2. Many studies in software quality prediction provide wide range selection of quality assessment methods for particular project contexts. However they lack planning step that may lead to unsystematic software quality prediction which fails to meet the business objectives assigned to the project, inefficient data collection and models construction, moreover the applicability of constructed quality model are often disputed [29].
3. To provide comprehensive quality evaluation in distributed development settings is still an expensive and error-prone activity despite of the approaches reported in practice due to several reasons such as the traditional human based reporting [55, 125] are often biased and expensive, limited data integration that come from scattered data sources across project sites, and collected data is often insufficient to construct good quality prediction models.
4. Current software quality prediction approaches such as software defect prediction focus mainly on how software quality status can be estimated using only static product metrics (e.g. code complexity, size and volume) [29]. Although many researchers reported product metrics can construct prediction models with certain level of accuracy, however such approach is often (a) fail to capture important distributed development process data (e.g. maturity level of code peer review prior to release) and (b) lack of capability in providing early warning of certain quality status or risks due to the late data availability (i.e. too close to system deployment date).

1.2 Research Contributions

In order to address proposed key research question, this thesis contributes the following deliverables:

1. Novel process metrics so called DSD project "health indicators" to evaluate the quality of development process. The concept of "health indicators" is to provide better insight of underlying development processes that have correlation to the aliveness of the development communities (e.g., developer communication pattern in the mailing list as responds to a new submitted code set) product quality improvement (e.g., defect detection effectiveness, defect resolution time), and certain risky project situation (e.g., a core developer abandon the project which may brain drain the developer community,

high number of severe defects found in a release). Thus, we can consider that "health indicators" as one quality aspect that should be evaluated continuously by a project manager in distributed development settings.

2. Quality indicators for software products based on defect data. Project manager needs to evaluate and predict the number of defect or likelihood of defectiveness of a work product prior to release or deployment. In this study we construct different quality indicators based on evaluation of defect data in release level of software product (i.e., defect growth between releases and likelihood of a release to have high defectiveness level). These quality indicators provide a range for the likely value of the indicators rather than a fixed value without indication of data volatility. In this thesis we use these quality indicators as dependent variables (estimator) to construct advanced defect prediction models. We also investigate the impact of particular process metrics which capture some aspects of quality assurance information (such as maturity level of peer review prior to release) to the accuracy of product quality predictions. We notice that the collected number of these selected metrics may not tell the whole story, e.g., a release manager may need to identify the reason why certain metrics have significant impact to increase the likelihood of a release for being highly defective. Thus for we conduct two types of analysis of empirical results: a) coefficients correlation analysis based on constructed prediction model and b) discussion with expert from to validate the findings and to obtain general and specific feedback.
3. Research roadmap in software quality prediction based on systematical literature review. Derived from systematical literature review results in software quality management and software quality prediction studies, we investigate the gap between current researches with their applicability in practices, and indentify open issues for future research works.
4. Structured framework for objective quality prediction and evaluation in distributed software development settings. This work proposes concepts of a structured software quality evaluation and estimation framework (SQF) that support a project manager in a) defining which quality aspects should be evaluated or estimated from key stakeholders perspective, (b) prediction model calibration through selection of parameters best correlated to model accuracy (c) evaluation of the prediction results to obtain internal and external validation of the constructed model.
5. Improving the accuracy of quality prediction methods based on product and process metrics. We investigate the impact of product and process metrics collected during software product evolution. Later we perform statistical data analysis to identify most

promising factors for software product and process improvement. We applied advanced parameters selection procedures and software quality prediction techniques for different project contexts to investigate which methods provide better prediction accuracy. More importantly we also conduct cross-projects modeling to have a robust quality prediction model with reasonable accuracy.

6. Empirical evaluation of quality evaluation and prediction scenarios in a range of real world distributed software engineering environment. For evaluation of the project health indicators and the SQF concepts we use empirical data from large Open Source Software (OSS) projects in Apache communities.

From a practitioner point of view, these research contributions significantly improve the effectiveness and efficiency of available software quality evaluation and prediction approaches in distributed software intensive system engineering. The following subsections outline show cases that explain how my research contributions contribute to the improvement of product and process in distributed software development context.

1.2.1 Evaluation of Distributed Development Processes Quality

Our study begins with the question how to improve the chances, that a distributed project can reach success and stay „healthy" as one aspect of development process quality. Distributed software project's survival is a result of many underlying (correlated) processes and cannot be easily determined and often very complex due to project characteristics (e.g. distributed project participants, complex project structure, and heterogeneous project repositories). Hence, the dynamic of the development process is much more difficult to understand compared to a typical collocated development project. Thus, project and quality managers, need pertinent data from the dynamics of distributed project consecutively to know the"health" status of the work.

Empirical studies in distributed software engineering [1, 47, 54, 55] agree that the quality of development process ("healthiness" of a project) should be measured by means of investigating the impact of different processes conducted by distributed project participants to the quality of software to be produced. We address this need by proposing a concept and evaluation of "health indicators" in distributed software projects.

Crowston et. al [21] as well as Collins and Fitzpatrick [18] define that a "healthy" distributed project such as in a Open Source Software Project should shows active developer communities, "a lot" of usages and feedbacks from users and rapid releases of good quality software. However, one open issue in quality assessment of development process is to identify whether

current development processes in particular those related to quality assurance activities are effective or efficient enough to produce good quality software or there are needs for improvement [124].

Thus, project "health" status monitoring should act as a) objective indicators of current quality of processes across project participants such as developer contributions level, maturity level of defect reporting and removal activities, b) investigate the correlation between processes such as defect removal activities with developer conversation in the mailing list, and c) provide prediction of certain situation or quality level status based on statistical analysis.

As the proof of concepts, we apply proposed health indicators to several large Open Source Software Projects from Apache Software Foundation (i.e., HTTPD, Tomcat, Slide, Xindice, MyFaces Core, MyFaces Tobago, MyFaces Trinidad, MyFaces Tomahawk) and later discuss the data analysis results with OSS expert to have external validation of proposed health indicators as reported in [121, 122, 124, 125].

The following 5 papers give a concise overview on this line of work; for an overview of results see Chapter 3.

Paper 1: **Wahyudin, D**., Mustofa, K., Schatten, A., Biffl, S., Tjoa, A., (2006);"Introducing "Health" Perspective in Open Source Web-Engineering Software Projects, Based on Project Data Analysis", Proceedings of the 8th International Conference on Information Integration, Web-Applications and Services (IIWAS); Austrian Computer Society (ÖCG) publishing, Yogyakarta Indonesia, 2006.

This paper proposes an evaluation process and concept for "health" indicators i.e., developer contributions into the mailing list, and correlated risk of a core committer abandon the project which will brain drain the rest of the developer community. The concept of "health" indicators can help getting an overview on a large number of OSS projects. For initial empirical evaluation of the concept, we apply the indicators to well-known OSS projects and discuss the results with OSS experts to investigate the external validity of the indicators.

Paper 2: Liem, L., **Wahyudin, D.**, Schatten, A., (2006) "Data Integration: an Experience of Information System Migration", (2006) Proceedings of the 8th International Conference on Information Integration, Web-Applications and Services (IIWAS); Austrian Computer Society (ÖCG) publishing, Yogyakarta Indonesia, 2006.

In this paper we present an experience of migrating and re-development of a legacy centralized

information system to a new distributed system. This work reports the importance of different stakeholders' participation and commitment during development process in order to obtain success in a distributed software development project.

Paper 3: **Wahyudin, D.,** Tjoa, A., (2007);"Event-Based Monitoring of Open Source Software Projects", EBITS workshop, Proceeding of the 2nd IEEE International Conference on Availability Reliability and Security (ARES), Vienna, Austria, 2007.

In this paper we propose a concept and an initial measurement approach for event-based monitoring of OSS projects to better understand the actual benefit of tool-supported gathering, correlating and analyzing processes event data from the OSS community as a supplement for traditional software project monitoring data collection. We report on an empirical feasibility study investigating "health" and risk indicators of five OSS projects listed in the Apache Incubator.

Paper 4: **Wahyudin, D.,** Mustofa, K., Schatten, A., Biffl, S., Tjoa, A., (2007); "Monitoring the "Health" Status of Open Source Web Engineering Projects", International Journal of Web Information Systems (IJWIS); Issue 3, Vol.1/2, Emerald, 2007.

This paper is an extension of Paper 1. Here once again we propose a concept and evaluation of"health indicators" (original indicators as reported in Paper 1 and additional indicators based on defect removal activities) in open source projects. The basic argument for the strategy of our approach is derived from the analysis of literature and published studies. We propose a concept of driving health indicators as derived measurements using an effect diagram of development processes in OSS project. We apply the indicators to well-known OSS projects for empirical evaluation of the concept. We perform project data analysis on the data retrieved from several successful OSS projects (Apache Tomcat and Apache HTTPD) and the challenged ones (Apache Slide and Apache Xindices). Similar to paper 1 in this study, we also discuss the results with OSS experts to investigate the external validity of the indicators

Paper 5: **Wahyudin, D**., Schatten, A., Winkler, D., Biffl, S. (2007): Aspects of Software Quality Assurance in Open Source Software Projects: Two Case Studies from Apache Project. 33rd EUROMICRO Conference on Software Engineering and Advanced Applications, SPPI Track, 2007.

This paper provides an exploration to improve our understanding of software quality practices in

different types of OSS projects. We propose a framework of Quality Assurance (QA) in an OSS project, elicit OSS stakeholder value propositions for QA, and derive additional "health" indicators. For an initial empirical evaluation we apply these indicators to 5 releases of 2 large Apache projects (Tomcat and MyFaces) to analyze the extent to which QA aspects are commonly performed during development process.

1.2.2 Software Quality Prediction in Distributed Development Settings

As suggested by [53] the absence of defects within a software product is one of important quality aspects in software engineering. Hence software quality prediction in particular defect prediction (e.g. estimates of number of defects within a release or likelihood of an software artifact to be defective) has drawn the attention of many researchers in empirical software engineering and software maintenance due to its importance in providing quality estimates and to identify the needs for improvement from project management perspective.

In this research, we propose a software quality prediction framework (SQF) for systematically conducting software defect prediction as an aid for project manager in DSD context. The framework has been aligned with practitioners' requirements and our findings from a systematical literature review on software defect prediction. We provide a guide to the body of existing studies on defect prediction by mapping the results of the systematic literature review to the framework. Based on proposed research hypotheses and results of systematical literature review we describe a research roadmap of software quality prediction.

As empirical evaluation of the SQF, we construct two quality indicators that should be predicted with product and process data collected during development. The first quality indicator is defect growth between releases while the second is the likelihood of high risk class of a release candidate. We use data from 4 Apache Projects (MyFaces Core, MyFaces Tobago, MyFaces Trinidad and Struts 2.0). We perform cross project modeling for different context of software defect prediction using advanced parameter selection techniques (i.e. correlation analysis; backward, forward and stepwise linear regression procedures) and prediction techniques (regression techniques: such as multiple linear regression, and logistic regression and classification techniques: such as Naïve Bayes, J48, Random Forest). Further, the results are discussed with OSS experts and defect prediction practitioner to have better understanding the impact of certain parameters to defect prediction accuracy, and to identify the most likely scenarios of certain correlations of different parameters [126].

The following 3 papers give a brief overview on software defect prediction in distributed software development settings; please refer to the Section 4 for an overview of the results.

Paper 6: **Wahyudin, D**., Ramler, R., and Biffl, S. (2008), A Framework for Defect Prediction in Specific Software Project Contexts, in the 3rd IFIP Central and East European Conference on Software Engineering Techniques (CEE-SET), Brno, Czech Republic, 2008.

In this paper we present the software quality prediction framework (SQF) derived from the practitioners' requirements and supported a systematical literature review results on software defect prediction. Later we use the results to derive the research roadmap in software defect prediction as guidance for future researches.

Paper 7: **Wahyudin, D.,** Winkler, D., Schatten, A., Tjoa, A. M., and Biffl, S.(2008); "Defect Prediction Using Combined Product and Process Metrics a Case Study from the Open Source Apache Myfaces Project Family"; in the 34th IEEE EUROMICRO Conference on Software Engineering and Advanced Applications, SPPI Track, Parma, Italy, 2008.

In this paper we apply SQF to investigate software defect prediction with data from a family of widely used OSS projects based both on product and process metrics as well as on combinations of these metrics. In this work our quality indicator (defect estimator) is defect growth between releases which enable project or release manager to identify whether a release candidate has potential growth of defect that may far exceed current developer capability in removing defects.

Paper 8: **Wahyudin, D**., Biffl, S, Schatten, A., and Tjoa, A. M. (2009); "Predicting the Defectiveness Risk Class of a Software Release Using Product and Process Metrics An Empirical Study Based on Data from Four Large Open Source Projects"; Submitted to the 31st IEEE/ACM International Conference on Software Engineering (ICSE), Vancouver, Canada, 2009.

In this paper, we propose a framework to characterize and predict the defectiveness risk class of a software release relative to the average defectiveness level of a reference set of releases. We collected and analyzed product and process data from 4 large OSS projects to empirically evaluate framework validity both within a project and across projects. We conduct two types of prediction model training, first is by fitting all available metrics into the model (unsupervised training) and second by conducting parameter selection and fitting only significantly correlated metrics into the prediction model (supervised training). The best prediction model is later used to predict the likelihood risk class of releases with cross project data.

1.3 Overview

The remainder of this thesis is structured as follows: Chapter 0 summarizes related work on software defect prediction and quality improvement as fundaments of this work. Chapter 3 outlines the first show case, evaluation of distributed development process quality and discussion of empirical results. Chapter 4 outlines the second show case, quality prediction in distributed software development with following discussion of empirical results. Chapter 5 summarizes the overall results with the proposed general research issues, and suggests future work.

2 FUNDAMENTS OF THIS WORK

This chapter describes a) introduction to empirical software engineering, b) quality evaluation of distributed software development product and processes, and c) software defect prediction as fundaments of this work

2.1 Introduction to Empirical Software Engineering

An improvement seeking organization or communities wants to assess the impact of process changes before introducing them to improve the way of working and the quality of product. Empirical studies are important in order to get objective and quantifiable information on the impact of changes [128]. Such organization needs to experiment and record obtained experiences from development process and eventually may depict the need to change the current way of producing a software product.

When new method or technology is substantially different from the current practice, the evaluation should be taken off-line in order to reduce risks [128]. Later, the empirical evaluation, may take the form of a controlled experiment (for detailed evaluation in the small) [5] or of a case study (to study the scale effects) [61]. In both cases, an approach of improvement life cycle such as the Goal/Questions/Metrics paradigm (GQM), as described subsequently, provides a useful empirical framework.

In this thesis we exploited and extended GQM to construct the Software Quality Prediction Framework (SQF) which will be fully described in section 4.3 .Based on metrics classification reported by literatures in software metrics (see section 2.1.2), we investigated the impact of metrics selection to quality prediction performance (e.g. accuracy, and precision) using empirical data from Open Source Software projects.

2.1.1 Goal/Questions/Metrics paradigm

The Goal Questions Metrics (GQM) [4] approach is based upon the assumption that for an organization to measure in a purposeful way it must (1) specify the goals for itself and its projects, (2) trace those goals to the data that are intended to define those goals operationally, and (3) provide a framework for interpreting the data with respect to the stated goals. The result of the application of the GQM approach is a specification of a measurement model targeting a particular set of issues and a set of rules for the interpretation of the measurement data. The

resulting measurement model has three levels:

1. Conceptual level (Goal). A goal is defined for an object, for a variety of reasons, with respect to various aspects of quality, from a range of points of view, for a particular project context. Objects of measurement are the products, development processes, and project resources.

2. Operational level (Question). A set of questions is used to characterize the way the assessment/achievement of a specific goal is going to be performed based on some characterization model. Questions try to characterize the objects of measurement (product, process, resource) with respect to a selected quality issue and to determine its quality from the selected viewpoint.

3. Quantitative level (Metric). A set of data is associated with every question in order to answer it in a quantitative way (either objectively or subjectively).

Practical guidelines of how to use GQM for measurement-based process improvement are given in [15].

2.1.2 Software Metrics

Empirical studies are used to investigate the effects of some inputs to the object under study such as accuracy of quality prediction models in an OSS project. Wohlin [128] suggests that *to control the study and to see the effects, we should measure the inputs in order to describe what causes the effect on the output, and to measure the output. Without measurements, it is not possible to have the desired control and therefore an empirical study cannot be conducted*.

Norman Fenton in his book Software Metrics [30], defines measurement and measure as:

Measurement is a mapping from the empirical world to the formal, relational world. Consequently, a measure is the number or symbol assigned to an entity by this mapping in order to characterize an attribute. The study of software metrics first published in 1976, by T. Gilb [42] since then software metrics have been heavily used in field of empirical software engineering, software project management and software quality prediction, as published in some notable text books [26, 30, 43, 48, 51].

The objects that are of interest in empirical software engineering can be divided into three different classes [31]:

a. **Product.** The products are the artifacts, deliverables or documents that result from a process activity.
b. **Process.** The process describes which activities that are needed to produce the software.
c. **Resources.** Resources are the objects, such as personnel, hardware, or software, needed

for a process activity.

In each of the classes we also make a distinction between internal and external attributes. An internal attribute is an attribute that can be measured purely in terms of the object. The external attributes can only be measured with respect to how the object relates to other objects. The mapping from an attribute to a measurement value can be made in many different ways, and each different mapping of an attribute is a *scale*. The most common scale types that can be used to measure an object attributes are the following [30, 128]:

a. **Nominal scale.** The nominal scale is the least powerful of the scale types. It only maps the attribute of the entity into a name or symbol. This mapping can be seen as a classification of entities according to the attribute. Examples of a nominal scale are: classification, labeling and defect typing.
b. **Ordinal scale.** The ordinal scale ranks the entities after an ordering criterion, and is therefore more powerful than the nominal scale. Examples of ordering criteria are; "greater than", "better than", and "more complex". Examples of an ordinal scale are: grades and software complexity.
c. **Interval scale.** The interval scale is used where the difference between two measures are meaningful, but not the value itself. This scale type orders the values in the same way as the ordinal scale but there is a notion of "relative distance" between two entities. The scale is therefore more powerful than the ordinal scale type. Examples of an interval scale are: temperature measured in Celsius or Fahrenheit.
d. **Ratio scale.** If there exists a meaningful zero value and the ratio between two measures is meaningful, a ratio scale can be used. Examples of a ratio scale are: length, temperature measured in Kelvin and duration of a in development phase.

Measures can also be classified in two other ways: (1) if the measure is direct or indirect, or (2) if the measure is objective or subjective [30, 128].

a. **Direct measure.** A direct measurement of an attribute is directly measurable and does not involve measurements on other attributes. Examples of direct measures are: Lines of code, and the number of defects found in test.
b. **Indirect measure.** An indirect measurement involves the measurement of other attributes. The indirect measure is derived from the other measures. Examples of indirect measures: Defect density (Number of defects divided by the number of lines of code), and programmers productivity (lines of code divided by the programmer's effort).
c. **Objective measure.** An objective measure is a measure where there is no judgement in the measurement value and is therefore only dependent on the object that is being

measured. An objective measure can be measured several times and the same value can be obtained within the measurement error. Examples of objectives measures are: Lines of Code (LOC), and delivery date.

d. **Subjective measure.** A subjective measure is the opposite of the objective measure. In the measure lays a judgment made by the person, who is making the measurement. The measure depends on both the object and the viewpoint from which they are taken. A subjective measure can be different if the object is measured again. A subjective measure is mostly of nominal or ordinal scale type. Examples of subjective measures are: Personnel skill, and usability.

For the empirical evaluation of the proposed concepts, in this thesis we focus on objective measure (objective modeling) which enabled both direct (basic metrics) and indirect measures (i.e. OSS Health Indicators) to construct quality prediction models.

2.1.3 Causal Modeling to Investigate Correlated Factors and Metrics

To provide effective quality prediction and evaluation, especially in a complex system like OSS project and DSD, first we should identify what would be the quality aspect to be assessed. Current practices in software defect prediction used metrics obtained during development process but often uncared for the causality of factors correlated to quality aspect that should be predicted [29]. From a practitioner perspective such studies typically report the performance of the prediction model (e.g. accuracy, error, precisions) without further discussions of the cause of the results often with vague impact factors analysis.

Max Born (1949) [13] acknowledged three assumptions that dominated physics domain until the twentieth century regarding the cause and effect [113]:

- "Causality postulates that there are laws by which the occurrence of an entity B of a certain class depends on the occurrence of an entity A of another class, where the word entity means any physical object, phenomenon, situation, or event. A is called the cause, B the effect."
- "Antecedence postulates that the cause must be prior to, or at least simultaneous with, the effect."
- "Contiguity postulates that cause and effect must be in spatial contact or connected by a chain of intermediate things in contact."

Causal model is a model or portrayal of the theorized causal relationships between concepts or variables. While, causal relationship is the relationship of cause and effect. The cause is the act

or event that produces the effect (*Source: Environmental Protection Agency of United States*)[1]. Example of popular models are Closed Loop Causal Diagram [86], Bayesian Network [31, 50] and Goldratt's Reality Tree Analysis [28].

In Figure 1, outlines the causal model for estimating software quality. In a naïve model (the left hand side figure), to predict software quality is simply by fit in the measure of software size into the prediction model. Yet, practitioners in software quality prediction are typically want to find better explanation of certain quality level status or risky condition, similar to a doctor who need to diagnose different symptoms and status of different organs to identify the patient's disease and suggest proper treatment.

On the right hand side of Figure 1, a more comprehensive causal closed loop model is represented. In causal model diagram, we can analyze how interrelated variables/factors affect one another. The diagram consists of a set of nodes representing the variables connected together through relationship. These relationships represented by arrows which labeled either as positive or negative. Positive label of an arrow, mean that by increasing the value of the origin node may likely increase the value of the destination node in the relationship, while negative labeling is vice versa.

Figure 1 illustrates what would be the impact factors and their effect to the quality of product. For example by increasing the effort in particular for conducting quality assurance activities will likely to increase the product quality. Other positive factors are assigning competent developers since the beginning of the project, and well planned work schedule will also impact the quality of the product.

[1] EPA glossary can be found at http://www.epa.gov/evaluate/glossary/c-esd.htm (last accessed 20 July 2008)

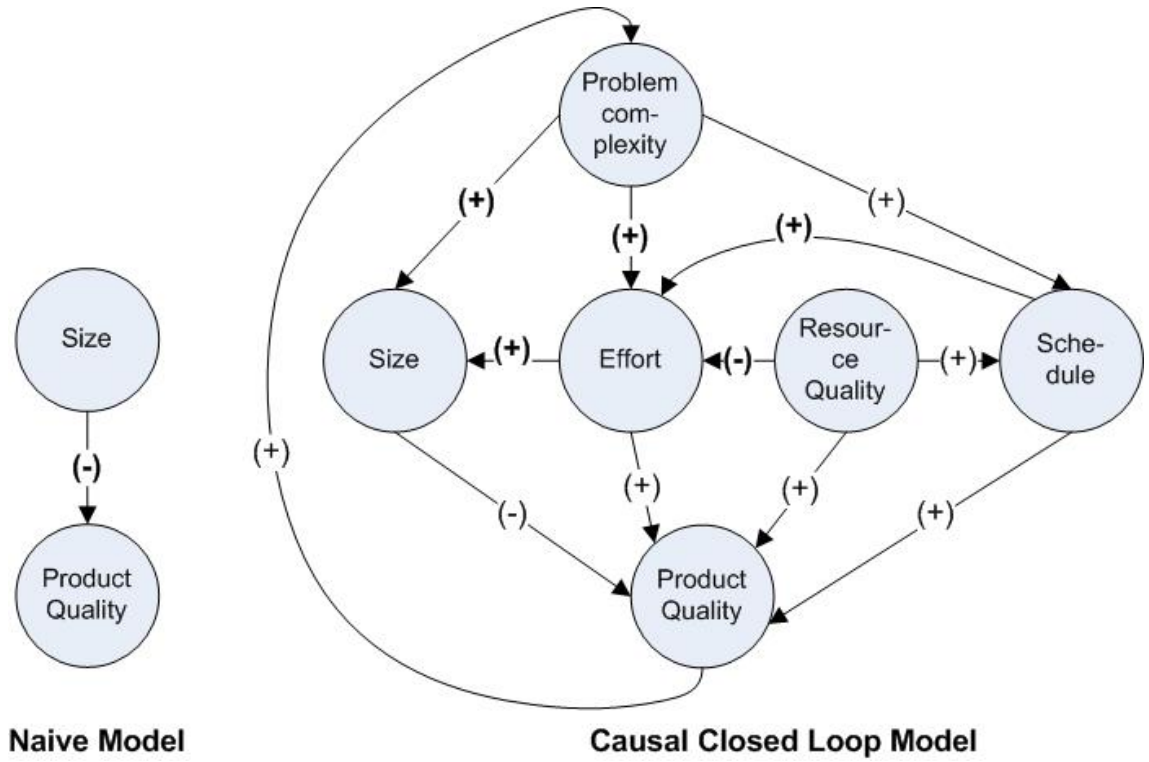


Figure 1 Simple Example of Causal Model for Software Quality Estimations (Extended from [31])

In this thesis, we use causal modeling, prior to conducting quality prediction. Derived from study literatures and expert interviews, we construct causal models that outline the context of prediction with respective cause and effect of interrelated factors. Later we evaluate the constructed model through discussion with experts of the study contexts (i.e. OSS experts and Software Defect Prediction practitioners).

2.1.4 Conducting Empirical Study in Software Engineering

The proceeding of well conducted research can be illustrated as a V model (see Figure 2). First in initiation phases "hurt" in particular domain of research may motivate some problem statements. Based on the identified problems, researchers construct model that represent the real world of the problem, and later design and develop solution ideas to address such problems. The design then implemented as method and tool for general or specific domain applications.

The right hand side of the research v-model consists of empirical validation of the solution. The first empirical validation is conducted using controlled experiment in order to assess the quality of implemented tool and system test to check whether the implementation meet all aspects of the proposed design. Errors are reported to implementation process for improvement and correction,

while the results of validated tools or methods are input for the empirical validation in industrial context.

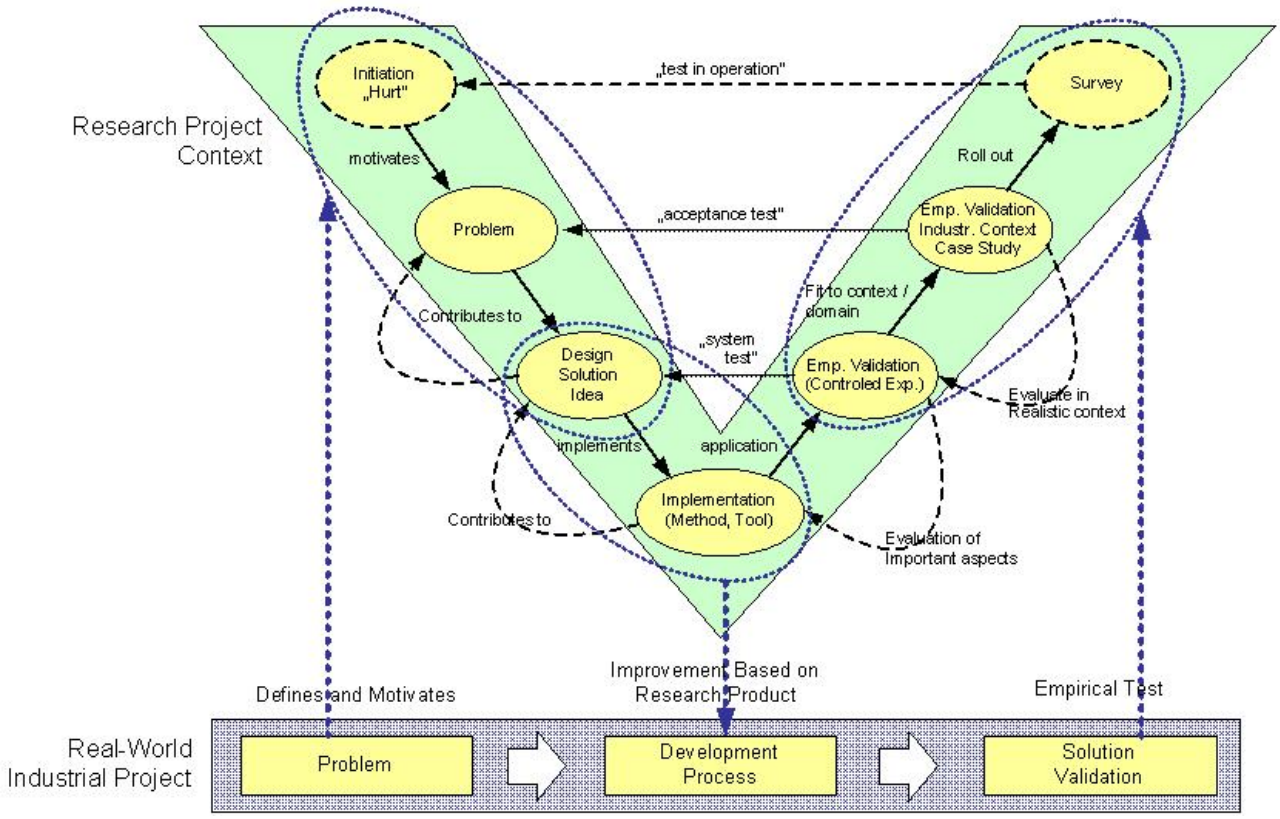


Figure 2 The V Research Model, refined from [9]

In for the industrial (*real world*) validation, use case is appropriate method to compare the new solutions, method and tools compare to other/proven approaches. The case study should provide some empirical results of benefits and challenges of the new tools/methods, and propose whether the new tools/method should be accepted for particular context of applications or domains.

Based on the results of the case study, the new methods or tools then rolled out to the real world for deployment. Then a survey can be conducted to test the new solution in operational, and to identify some new "hurts" that may initiate new researches.

A structured empirical study is a must to have better control of the study process and to have acknowledged results through real world validation. In this section we describe the proposed research framework for conducting empirical study in an OSS project context (see Figure 3). The framework is based on Wohlin's [128] guide for conducting experiment in software engineering which further simplifies the V research model with additional quality assurances

(QA) steps.

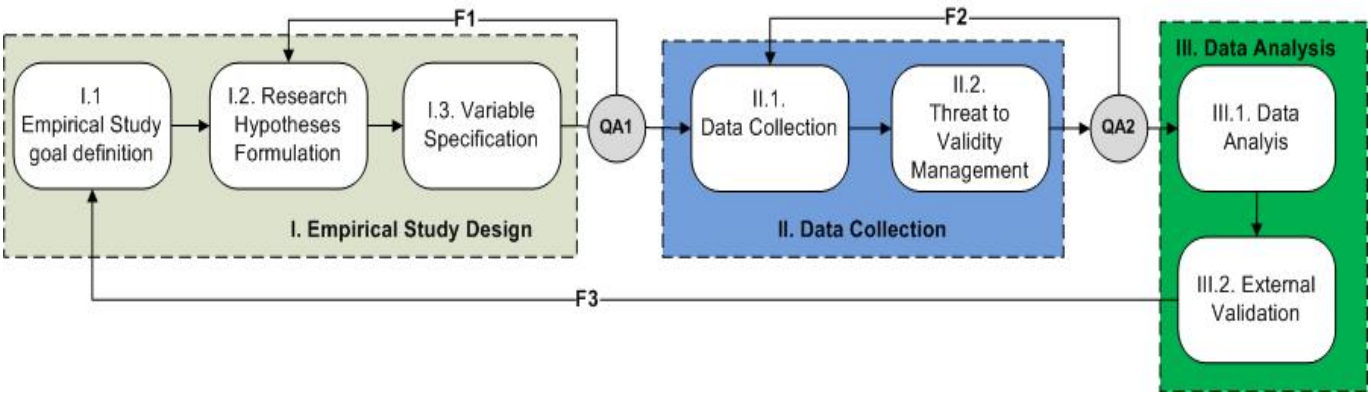


Figure 3 Framework for empirical study in context of OSS and DSD projects

Figure 3 shows the three-step approach: (a) **empirical study design** and measurement model development (b) **data collection** from project repositories (c) **data analysis**. The sequence of phases is partially derived from general frameworks for conducting empirical experiments in software engineering [128] and empirical study design [4].

I. Empirical Study Design

As precondition prior to study goal definition, first we have to describe the study objects that will be used for the experiment. Second, we need to elicit target stakeholder expectations in order to identify their requirements and performance measures of each requirement achievement. We can use The Goal Question Metric (GQM) Approach [4], a common methodology to identify empirical measures and the setup an empirical study.

Step I.1 – Empirical Study Goal Definition. Following the GQM approach, the goals represent selected stakeholders needs for certain quality aspects such as the expectation of current defect removal performance. To identify the stakeholder value expectation one can used value elicitation techniques suggested by Biffl et al. [10]. Basili et al. [4] suggested to define a goal for an object with respect to various quality models from stakeholders' points of view regarding the particular project environment. For instance: *provide an accurate estimation of the software component quality prior to the release date from the release manager's point of view in an OSS project*.

Step I.2 – Research Hypothesis Formulation. Based on empirical study goal, we derive questions that define them as completely as possible to characterize the way of assessment of

specific goal is going to be performed. For example, *how fast can the developer community provide appropriate resolution for each reported defect*? Later, we can formalize each question into research hypotheses that can be quantitatively addressed by data collection and analysis.

Step I.3 – Variable Specification. To better focus our study, we model the context of research which allows a) the creation of evaluation and predictive formula b) capture particular patterns within the project based on our study interest such as developer code contribution patterns, developer responsiveness, defect service delays [124]. The selected modeling method should provide a basic formula to determine a dependent variable using a set of independent variables [126]. After A3, one should perform validity check of the variable specification with the goal and hypotheses, as well as quick feasibility study whether a research goal and hypotheses can be addressed by the data collected from study objects (See QA1). The results of QA1 may derive a need for study design refinement as feedback to step A.1. (See feedback line F1).

II. Data Collection

The next step is to perform data collection from the project repositories, refine the collected data and address relevant threats of validity. The following steps were constructed based on our experiences in data mining of a range of OSS project repositories [121, 122, 124-126].

Step II.1 – Data Collection. The empirical design proposed in the preparation phase should be evaluated by real data collected from selected study objects (projects and project repositories). The data collection process consists of:

1. Specifying the observation objects, observation time and level of observation detail i.e. "Our study objects are Project X and Project Y hosted in the Apache Software foundation, later we conducted two months of observations from 1st February 2007 to 31 March 2007 and collected a set of metrics from the observed project issue tracker".
2. Data extraction from the project repositories is based on the observation specification and depends on the metrics to be collected. There is a variety of OSS data mining tools such as Eclipse-Metrics Plug-in[2], Jdepend[3] for static code metrics collection, and StatSVN[4], Markmail[5], Jira[6] and Bugzilla[7] to collect development metrics.

[2] Metrics project can be found at http://metrics.sourceforge.net/

[3] Jdepend project can be found at http://andrei.gmxhome.de/jdepend4eclipse/

[4] StatSVN project can be found at http://www.statsvn.org/

[5] Markmail repositories can be found at http://markmail.org/

[6] Jira for ASF can be found at https://issues.apache.org/jira/

3. Typically collected raw data are not immediately ready for analysis; we still need to exclude invalid and duplicate data, or to replace missing data [71, 124]. Filtered data needs to be grouped, refined and prepared into particular format as input for analysis tools such as Weka[8] or SPSS[9].

Step II.2 –Threats to Validity Management. The objective of an empirical study is to reach a conclusion that the selected measures reflected what we wanted them to reflect in current context of the study [118]. A Threats to Validity reveals questions and issues on the correctness of study claims [19]. A threat should be acknowledged and addressed appropriately during data collection process in order to provide counter measures to elevate the validity of empirical study results.

Wohlin [128] suggests that threats to validity can be classified into four major classes:

1. *Internal validity: is concerned with the validity within the given environment and the reliability of the results*. E.g., Ambiguity about direction of causal influence, Selection, Statistical regression, etc
2. *External validity: is a question of how general the findings are. Many times, we would like to state that the results from an experiment are valid outside the actual context in which the experiment was run.* E.g., interaction of different treatment, interaction of selection and treatment
3. *Construct validity is a matter of judging if the treatment reflects the cause construct and the outcome provides a true picture of the effect construct.* E.g., Confounding constructs and levels of constructs, hypothesis guessing, etc.
4. *Conclusion validity is concerned with the relationship between the treatment and the outcome of the experiment.* E.g., Low statistical power, Reliability of measures, Violated assumption of statistical tests, etc.

During validity evaluation, the process itself acts as the second data quality assurance by validating and checking collected data based on reality of the objects of study (see QA2). The results should outline whether collected data is sufficient enough (in quality and quantity) for data analysis or should be improved (see Feedback F2).

[7] Bugzilla project can be found at http://www.bugzilla.org/

[8] Weka "data mining with open source mining tool", can be found at: http://www.cs.waikato.ac.nz/ml/weka/, last accessed 20 July 2008

[9] SPSS software packages can be found at: http://www.spss.com/de/, last accessed 20 July 2008

III. Data Analysis

This is the last part in our framework and an important stage to validate the constructed models and measures with collected data.

Step III.1 – Data Analysis. Collected and refined data from the previous phase then fit in to the measures and model and analyze by means of statistical or machine learning methods. Some statistical tests also should be performed to validate the results, i.e. to check the distribution and the significance of data, to check model significance and accuracy.

Step III.2 – External Validation of Data Analysis Results. The last step is to provide external validation using different observation data or discussion with expert in the context of study to better analyze particular patterns in observed projects which might have impact on the analysis results.

The results of data analysis later should be used to evaluate proposed research hypotheses, and feedback for goal adjustment (see feedback F3).

2.2 Quality Evaluation of Distributed Software Development Processes and Product

Distributed software development (DSD) is a part of globalization where software project teams have become geographically distributed [102]. Large software companies employ DSD style in search for competitive advantage in terms of cost reduction, higher quality and flexibility in software development, productivity increases, and to lessen potential risks [110]. However, the notion of distributed software development has tradeoffs such as a highly complex, distributed processes, comprising a large number of highly inter-dependent parallel activities [101].

2.2.1 Quality as Software Product Conformance to Requirements

Crosby [20] defines quality as "conformance to requirements", means development processes and delivered product should be measured continuously to determine conformance to predefined requirements. On the other hand, Juran et al [52] defines quality as "fitness to use" which takes customers' requirements and their value expectation into account. Both definition are correlated to each other, and reflect the need to measure the quality of software products and development processes to produce such product. Kan [53] defines that *the narrowest sense of software quality is commonly recognized as lack of defects in the product.* He also further mentioned that the degree of defect freedom is the most basic meaning of conformance to requirements and

fitness to use, because if the software contains high number of defects it will reduce the expected functionalities.

Project and Quality manager needs to evaluate the actual status of the project and the quality of performed development processes to produce required products. Evaluation of distributed development process methods are required to supply timely, accurate and comprehensive project information as the basis for analysis and decision making.

There are several quality evaluation frameworks in traditional closed-source software projects, e.g., Quantitative Quality Evaluation model [12], Goal Question Metrics model [4], Squid model [11], and Gutman's Means-End Chain model [129]. However, to our knowledge there is no validated framework available for modern large-scale OSS [123] DSD project environments due to the characteristics of OSS development processes: globally distributed and voluntary participants, less formal project management (in particular, planning), often scarce documentation, and frequent product releases [63, 122].

The second issue with current quality evaluation frameworks, is that most of them put a focus on evaluating the quality criteria of software artefacts (work products), e.g., source code quality and architecture, rather than development processes that are likely to have significant impact on these quality criteria [125]. A critique from Norman Fenton of current quality evaluation frameworks is that most of these studies try to evaluate or predict certain quality characteristics (e.g., maintainability, reliability, security) derived from certain quality attributes (like product size, complexity, dependencies on management level) but most of them fail to report quantitatively what are the correlation between these quality attributes with actual defects reported after product release [32].

2.2.2 Human Based v.s. Tool Based Evaluation Approaches

In general, there are two evaluation approaches: *tool-based* and *human-based* monitoring. The tool-based approach seems most suitable for monitoring frequently a large number of process events data, or when human resources for monitoring are hard to obtain. On the other hand, the human-based approach seems favorable for a weekly/monthly process such as personal reporting as the summary of activities and progress status, and go to more detail if necessary by directly interviewing the team member.

Traditional software project management [100, 106] focuses on tracking formal achievements such as the progress and financial obligation and analyzing the merit of project participants based on routine personal reports and deliverables. As the consequence, most of the traditional

project management is human-based monitoring, which often misses the process and information during its project execution. This can be very risky; if a problem occurs, as Keil et.al [55] found, participants tend not to report the actual condition of the project. Hence, additional data for comprehensive balanced reporting are needed before and during a crisis for raising issues well in advance to identify and to mitigate project risks.

Advanced applications of distributed development such as Open Source Software Project and Global Software Development have led to new challenges regarding the scalability and expressiveness of project monitoring methods [108], especially when the project has to face (1) a large amount of process data to be monitored, (2) shortage of human resources for monitoring, and (3) most importantly, the loosely coupled project community as the result of a global project work. Consequently, monitoring such a system using only a human based approach is likely to be costly, time consuming, and error prone.

This complexity, of course, is the motivation behind the desire a project or quality manager to use tool supports to simplify the management and performance of the process. Tool-driven system "health" monitoring has been successfully adopted by industries, reaching from hybrid system to business activity process monitoring [117], which in principle can be adopted in software engineering domain. Other study [107] suggested an OSS tools for project monitoring, particularly for dislocated or distributed development such as in OSS projects.

During the development, [101] advocates the tool supported project evaluation should balance the observation from (a) time relevant process data (b) product-relevant artifacts data and (c) coordination inter-dependent activities data. This combination will provide more accurate and less biased project information for critical decision making process [125].

Hence, this work focuses on exploiting available extensible tools to capture product and process metrics as basis for software quality prediction and evaluation in distributed development environment such as OSS projects. In the following subsections, we describe briefly the characteristics and development processes in a typical OSS project.

2.2.3 The Needs for OSS Product Quality Evaluation

Open Source Software (OSS) projects can be considered as extreme form of distributed software development as the globally distributed participants are mostly volunteers with distinguished project management style. Eric Raymond in his famous essay "The Cathedral and The Bazaar" [105] suggested that open source development style has several advantages compared to closed source project such as:

- Rapid development and massive peer review
- Flexibility in using and modifying the source code for user interest
- Low-cost development and technology transfer
- Developer inheritance and the use of a reference implementation to help develop a standard.
- Open code base and Open development process [95]

We used open source projects to provide empirical evaluation of proposed concepts in this thesis due to the following reasons: a) rich availability of data from shared project repositories b) openness in development processes which enable reasoning for certain quality achievement as well as risk condition c) the significance of OSS product adoptions in many industry domains.

Just to give an example Netcraft Web Server Survey [93] discloses that more than 60% of the web sites on the Internet are using Apache HTTP Server (see Figure 4).

Other OSS products such as: Apache Tomcat[10] has powered large industries such as General Motor, and Wal-Mart, from automated manufacturing domain, the application of OSS solutions such as JADE platform has been successfully provides support for controlling a large distributed manufacturing system [78]. It is worth noting, that currently a number of important OSS projects are supported by companies and some participants are not volunteers (e.g., JBoss, Apache JackRabbit, MyFaces Tobago, and OpenOffice).

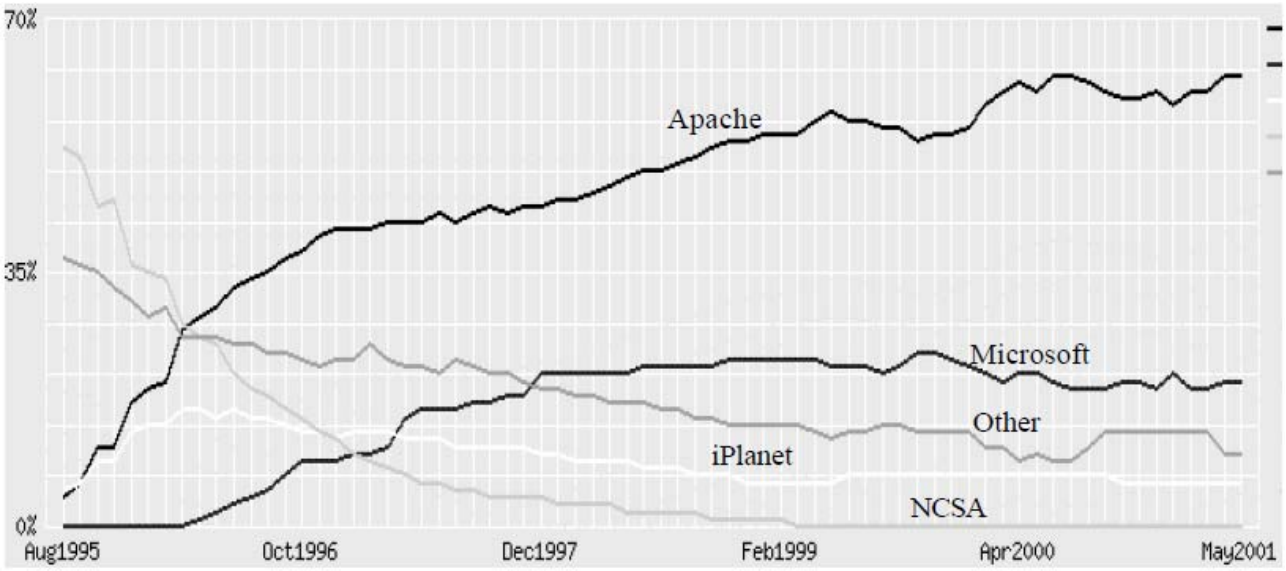


Figure 4 Growth of Apache HTTP Server Market Share (Source: Netcraft Survey [93])

Furthermore, large critical avionic industries such as Airbus sponsored TOPCASED project to

[10] List of Powered By Apache Tomcat can be found at: http://wiki.apache.org/tomcat/PoweredBy last accessed 23-July-2008

provide Eclipse based development tools for critical aerospace security systems[11], while NASA recently called for "open source specification" in Boeing's contract to build a next-generation avionics system to guide Ares rockets which is responsible for launching the manned Orion spacecraft into Earth orbit, and then driving it to the moon [92].

2.2.4 Comparison of OSS Projects to Closed Source Distributed Software Projects

In Table 1, which is partially based on Keil and Carmel [54] observations, we highlight some key differences between closed source development (custom and package) and open source software development. Given these differences, one would expect to find differences in the links that are used across the three environments.

Although OSS development style promise some appealing benefits, however [16] suggests that typical OSS development seizes several problems such as the lack of requirement elicitation, no ad-hoc development process, and poor practices of project management. Moreover, many OSS project communities are still in their initial phase, in an immature state or have reached the end of their life cycle which means their survival seems heavily uncertain [65].

Table 1 Comparison of OSS Project Development with Closed Source Development

Development Dimension	Closed Source Development [54]		Open Source Software Projects
	Custom Development	Package/COTS	
Goal	Software developed for internal use (i.e., usually not for sale)	Software developed for external use (i.e., for sale)	First software developed for internal use of the core developers later they publish the product to larger community for feed backs and attract new developers [83]
Typical point at which most customers are identified	Before development begins	After development ends and the product goes to market	Originally the customers are the developers who initiate the project [37, 69]
Number of customer organization	Usually one	Many	Depend on the project community, may consists single user to large number of users in the community[65, 124]
Physical distance between customer and developer	Usually small (e.g. customers are in same building as developers)	Usually large (e.g. customers are thousands of miles from developers)	Typically widely spread across the globe, and most of them never met each other face to face [16, 119]

[11] TOPCASED project can be found at http://www.topcased.org/, last accessed 17 June 2008

Common types of projects	New system project; „Maintenance“ enhancements	New products; new versions (major and minor)	New sub projects, new versions/releases (major and minor) [37]
Term of software consumer	User; end user	Customer	User [16] which some of them also act as developer within the project
Common measures of success	Satisfaction; acceptance	Sales; market share; good product reviews	High number of usages, active developer contributions, rapid releases [122]

2.2.5 Open Source Software Development Structure

Abedour [1] as well as Crowston et al [22] model the distributed development process in OSS as an onion shape model (see Figure 5), in this model the core engine of the project is a small group of **distributed core developers** (sometimes called as committers) that provides more than 80% of overall development contributions [83]. Some of the core developers also hold critical roles such as project management and release process. Core developers have read and write access to current body of code, as one of their responsibilities is to assure that each new code contribution are suitable to predefined specifications and do not posses any threat to the stability of the body of code.

On the second layer is a larger group of **peripheral developers** who mostly responsible in defect removal activities and patches development, this developer group only has read access to the current working code repository, thus for each patches developed they typically post the contribution to shared developer tools such as mailing list or issue tracker and wait to be reviewed by other developers before a committer can include the patches into current body of code.

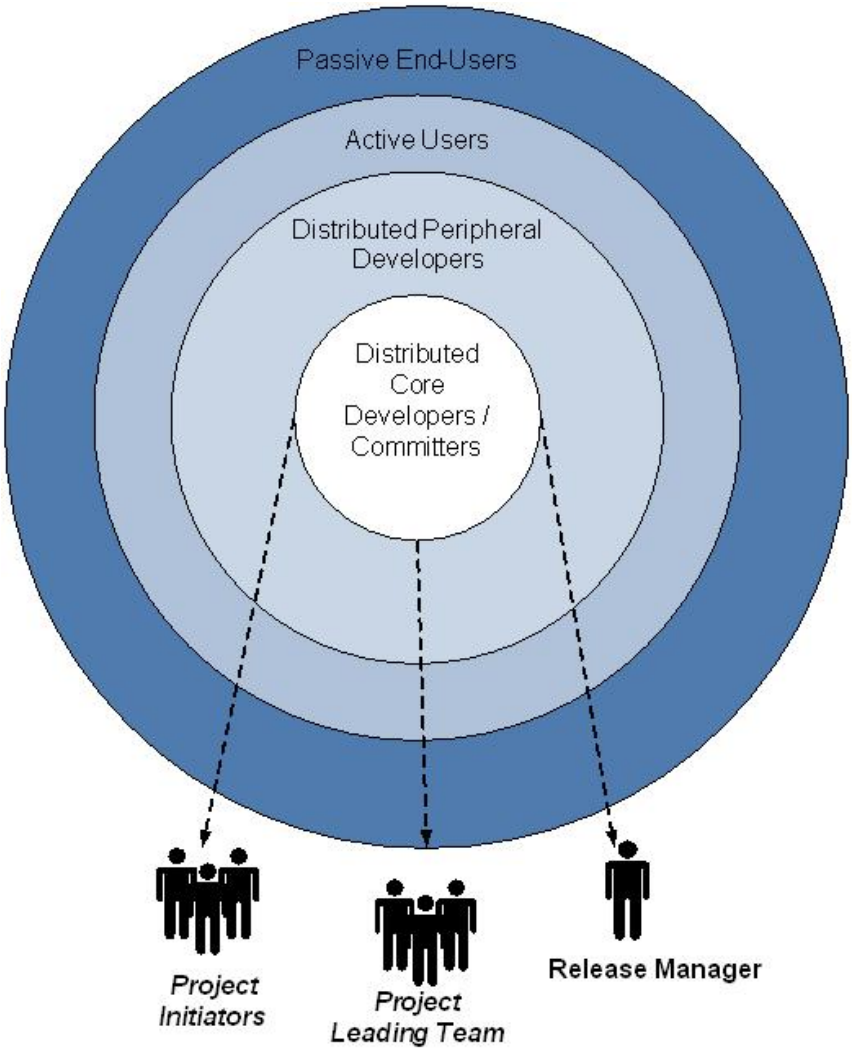


Figure 5 The Open Source Software Structure Model. Refined from [1, 22]

The user community can be divided based on their feedbacks of OSS product releases to the developer community, an active user commonly contribute in defect reporting and new feature requests, while passive users are simply end-user of a releases.

Each project participants collaborate using a set of shared development (e.g. source code management and issue tracker) repositories and communication infrastructures (e.g. mailing list, forum and wiki).

To extract useful information from such data repositories we need web data mining, starting with data preparation which may involve data cleaning, data transformations, selecting subsets of records and in case of data sets with large numbers of variables - performing some preliminary feature selection operations to bring the number of variables to some manageable range.

The development processes in an Open Source Software (OSS) project can be modeled as multi-agent event-based system: in this model the project participants are agents, and their interactions

and state changes are events (see Figure 6). During the development processes, project participants interact and move from one state to another triggered by events. They may act as producer who publish event through messaging middleware (i.e. mailing list, issue tracker, source code management), which then deliver events to other agents who act as consumers/subscriber based upon their previously specified interest.

Since most of the participants are: (a) unfamiliar with each other, (b) distributed around the globe with different time zone and work schedules, and (c) use various technologies and development-communication interfaces, thus make the OSS project as a system with loose coupling in time, space and synchronization. As result most of the messages and deliverables during development processes are made with publish/subscribe-like interaction schemas as illustrated in Figure 6.

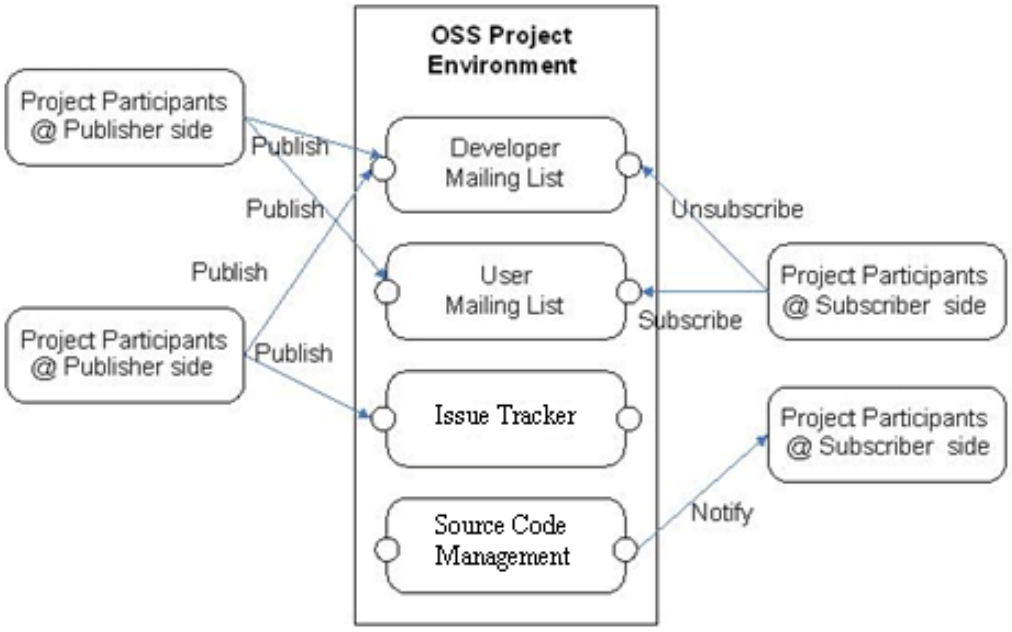


Figure 6 Publish-Subscribe mechanism as communication pattern in OSS projects

For example in a defect tracking process scenario, a user/developer who reports an issue can be considered as producer who send message about a defect existence into defect tracker (`defect_reporting_event`), then after performing some internal management operation, the defect tracker broadcasts the new defect information, e.g., through a mailing list (`new_defect_notification_event`). Later some subscribed user/developer may respond by making the diagnosis of the defect and send the result into the defect tracker (`defect_diagnoses_event`).

The event-based model and tool support allow to draw on process and artifact data from the global OSS project community that can help outsiders to better understand success and risk factors in the current state of a project and its community. This kind of data analysis can be

especially helpful if human-based reports are suspected to be unsystematic, incomplete, or inconsistent.

Several studies have used process event data of the existing open source projects to better understand the aspects of successful distributed development. These studies observed the OSS projects by manually or by tool supported mining project repositories such as mailing lists, defect database [82], Source code management tool (SCM) such as SVN/CVS [41], and changes log [17]. The results clearly portrayed the development process pattern and the importance of community involvement in OSS projects. However further works are required to better understand the OSS project, to distinguish different status of projects and to estimate the project survivability.

2.2.6 Continuous Product and Process Improvement in an OSS Projects

A good OSS project offers continuous improvement of software product releases. Just to give some examples large OSS projects such as Gnome, Mozilla, Python, Subversion and Eclipse encourage quality improvement as part of OSS community awareness.

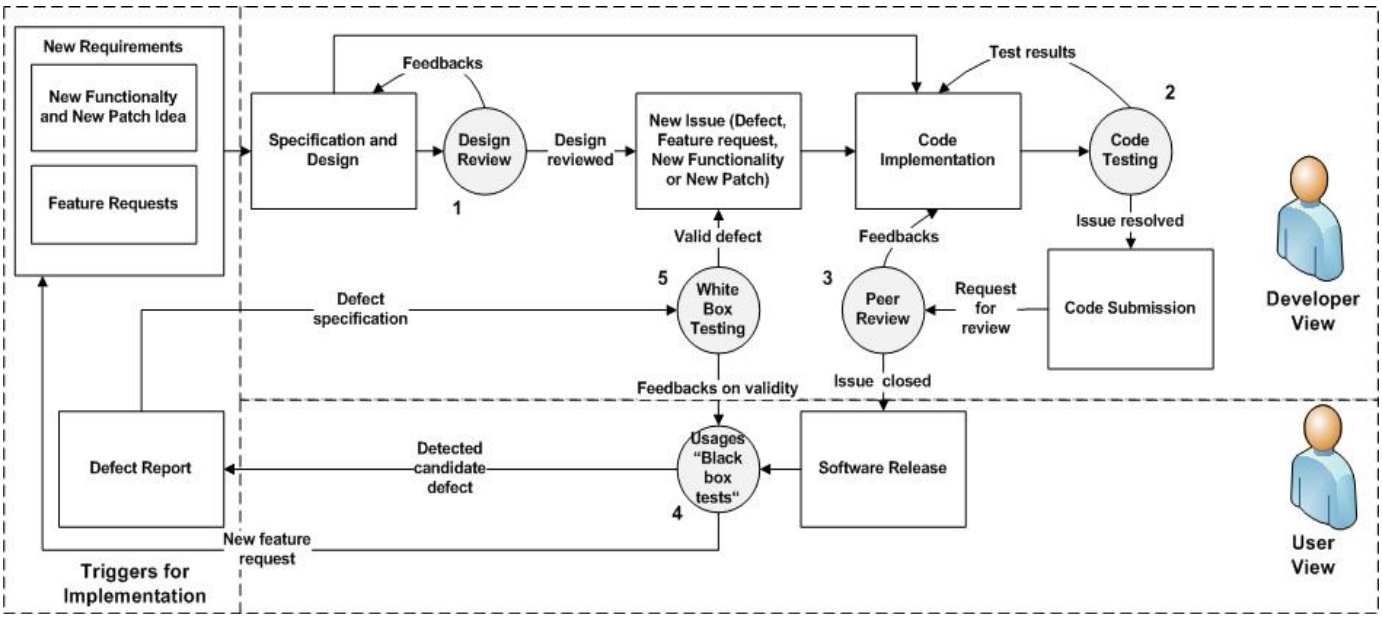


Figure 7 Continuous software product improvement within an OSS project.

Their goals are to improve the quality of the releases by involving a larger part of the project community based on principles [124] such as involvement of a developer to review the validity of a defect candidate reported by a user before submitting the report into the issue tracker (see:

Buddy System at Subversion project[12]).

Our prior study [121, 122] in four large Apache OSS projects concluded that these communities should coordinate and work together as a *symbiosis mutualism* to produce high-quality software. In [122] we found that large successful projects such as Apache Tomcat and HTTPD have faster developer response times to user community feedbacks (i.e. defect report or feature request), and higher numbers of peer-reviews of each code set or patches submitted into the project code versioning system.

The model depicted in Figure 7 is an extension of similar model proposed in Section 3.6 with some additional QA aspects (e.g. design review) and their involvement to continuously improve the software product in each release cycle. Nevertheless both model should be taken together to have a comprehensive view of development process in a "healthy" and "quality aware" OSS project communities.

Figure 7 illustrates a complete life cycle of an OSS project with five typical QA practices represented as circles as partially depicted in studies such as [36, 37, 124]; some of these practices are fully or partially observable, and thus we can measure the development activities with these practices to derive relevant Process Metrics. Afterward we investigated the usefulness of these Process Metrics to quality evaluation and defect prediction in our case studies.

Design Review

Issues reported to the tracker tool trigger most of the development activities within OSS projects. An issue can be a new requirement (feature request/new functionality, or enhancement/patch) or software defect reported by a user. Throughout the project lifetime, there are several quality assurance (QA) practices as part of product release continuous improvement.

When a developer has an idea for new functionality or a patch, he may construct specification and design and then ask other developers within the community to review his specification and design before listing them as new issues (see circle 1 in Figure 7).

The Python project[13] community encourages developers to engage in a specific design process, called Python Enhancement Proposal (PEP), which is similar to a request for comments and design technical review meeting in commercial software projects [36]. This design review process uses common information spaces of the project such as emails, forum, and project

[12] http://subversion.tigris.org/project_issues.html

[13] http://www.python.org/, last accessed on 14th February 2008.

documentation and involves different stakeholders across all project communities.

During design review, we can observe the developer activities in negotiation, collaboration, and refinement of proposed design. If the design proposal gets accepted, then the developer lists appropriate action items in the issue tracker. However, it is also common that a developer directly jumps into implementation (with his own ideas), then submits the code set, and later opens a discussion in developer communication channels and asks for technical review of his code.

Code Testing

It is worth noting that a developer in an OSS project always conducts code testing before submitting the code set into the CVS (see circle 2 in Figure 7). If the tests fail, then the developer either continues to work until the issue is resolved or returns the issue into the tracker as "open" with related documentations for knowledge preservation (i.e., refined bug recipes, development issues encountered).

Although we cannot measure the testing process directly, we can measure developer contributions from developer communication spaces (mailing list, CVS, and issue tracker) prior to a release. Hence we can obtain the following metrics: *changes to code metrics (e.g. delta, added, deleted, modified to line of codes by developers)* [89]*, number of committers/core developers and number of* peripheral *developers* [83]*, code and changes contribution of core and peripheral developers* [126].

Code Peer Review

In a quality-aware OSS project, an issue labeled "resolved" will attract other developers to review the code set. A committer then should decide based on review results whether a code set should be added into current body of code or get returned to the issue tracker (circle 3 in Figure 7).

These practices especially peer review can be observed through the project communication space, issue tracker and project CVS. Prior to a release date, a release manager needs to identify which patches and functionalities should be added to the next release package. Later he performs integration testing to assure the software quality before publishing the release package.

Code peer review effectiveness can be measured as number of defects stated as "closed" prior to

a release [124, 126]; based on the Bugzilla[14] documentation "closed" means the issue has been resolved and has passed a peer review. For example: *number of closed defects, number or resolved defects, number of resolved defects/number of reported defects, number of closed defects/number of reported defects.*

Product Release Usage and Defect Validation

The user community obtains the new release and uses it in different work contexts, and provides feedbacks to the developer community such as defects found and feature requests. This defect detection practice is similar to black box testing to find defects in a software product release (see circle 4 in Figure 7). The defect detection activities provide a list of defect candidates of a software release and considered as the primary activities performed by developers and users after a release in OSS project [39, 82]. Prior work [124] provides several examples of metrics that can be used as predictors such as: *number of defects reports prior to release, number of open defects prior to release, number of invalid defects prior to release, and defect detection effectiveness prior to release.*

Most of the defects are detected through software usage and then validated by a developer by reproducing the defect based on defect recipe report from the user (see circle 5 in Figure 7). If the defect is valid, a developer takes ownership of the confirmed defect and performs a suitable development process for resolution.

In the study we applied all of these Process Metrics as predictors and investigated their correlation to defect estimates in the case study context.

2.3 Software Quality Prediction

Software quality prediction as in our case is defect prediction has drawn the attention of many researchers in empirical software engineering and software maintenance due to its importance in providing quality estimates and to identify the needs for improvement from project management perspective.

[14]Bugzilla documentation can be found at :http://www.bugzilla.org/docs/. Last accessed 10th December 2007.

2.3.1 Objective Software Defect Prediction Methods

Schneidewind [109] suggested two approaches for objective defect prediction: (a) time-based approaches and (b) metric-based approaches. A time-based approach estimates the number of (remaining) defects from the number of defects found in a time interval after product release and fit the data to form a software reliability growth model (RGM) [76]. A metrics-based approach uses metrics obtained from historical project data before product release (called as predictors) to fit a prediction model.

The advantage of time-based approaches is a more accurate prediction compared to metrics based approaches, since estimations are derived from actual defect data; however, the availability of data for estimation purposes are mostly based on testing results. Thus the prediction is often too late to support in-time decision making [126] regarding an upcoming release. Metric-based approach promises better support for a release manager by providing defect forecast prior to release, often with less accuracy as the tradeoff [71].

To address different patterns of defect prediction, numerous statistical methods and software metric applications exist in the software maintenance and software quality research communities. Currently, there are several metric-based prediction models that are commonly used by researchers in software defect prediction:

- Regression methods that best fit to predict the *numeric value of defect prediction target (estimator)*, e.g., the number of defects in a release. For example, Khosgoftaar et al. [57] suggested as prediction models component clustering fitted to linear regression or non-linear regression [56].
- Classification methods to predict the *nominal value of defect estimators* such as module or file defectiveness. Classification techniques such as Bayesian Network (e.g., Naive Bayes, Bayes Net), regression (e.g., Logistic Regression), and tree classification techniques (e.g., J48, Random Forest) have been widely used in software defect prediction contexts [64] and risk prediction in project level [130]. For example, Yasunari et al. [130] suggested a project to be risky if the project showed confused behavior. For each observed projects they measured the confusion levels based on empirical questionnaires. Later using a certain threshold, they classified projects into two classes “confused or risky” and “not confused” and constructed a logistic regression model based on empirical data collected from these projects.

2.3.2 Metrics Categories for Software Defect Prediction

In metrics-based defect prediction, collected metrics should be selected first before fit into the model as independent variables or predictors. Norman Fenton [30] classified collected metrics to construct prediction models into three basic categories such as:

1. *Product metrics* measure attributes of intermediate and final software products, e.g., size (LOC) and complexity metrics (McCabe Complexity metrics). Product metrics are the most commonly used predictors and supported by [6, 27, 84] as important predictors in many cases of defect prediction in closed source development.
2. *Process metrics* measure attributes of development processes, e.g., project events (new defect reported into issue tracker), state changes (defect status changed from unresolved to resolved), and activities such as, number of file being changed, and LOC churned per developer within a release. Mockus et al. [82], Weyuker et al. [127], and Nachiappan et al. [89] suggested these metrics as important predictors. Some study also called this group of metrics as development metrics [127], Process Metrics [126] or changes metrics [87].
3. *Resource metrics*, which further can be classified as [70]:
    a. Project participant metrics, measure attributes of involved project participants such as number of core developers, number of peripheral developers, number of active user etc [126, 127].
    b. *Deployment and usage metrics* measure attributes of the deployment context and usage patterns of software releases, e.g., time since first release and time to next release [73, 75].
    c. *Configuration metrics* measure attributes of software and hardware configuration that interact with the software product/release during operation e.g. operating system supported by the release and type of software application [84].

Some of these metrics as well as some novel metrics proposed by us in this thesis will be used for quality prediction modeling, and to better understanding which set of metrics can provide better prediction results.

2.3.3 Software Quality Prediction in OSS Projects

The nature of open source software (OSS) development [16], such as highly distributed development by volunteer contributors; cultural and time zone differences of contributors; informal project management and modest consideration of quality assurance (QA) and

documentation during development, makes product QA a major concern to potential users of new releases. Empirical studies [1, 38] suggest that some OSS projects have created software products with quality levels similar to closed source commercial development. Ben Collins and Brian Fitzpatrick[15], committers and co-founders of the OSS Subversion project, suggested constant product improvements and releases as indicators for a "healthy" OSS project [18].

Product improvements in OSS project are directed by a strong feedback from the user community (e.g., bug reports and feature requests)[16] and active developers' contributions [37] see Figure 7.

In OSS projects, where formal QA practices such as inspection are less practicable, one feasible approach for assessing the quality of a software product is to predict the defect between releases. In a closed source software development, the prediction of defects between releases can provide benefits such as to guide testing of the next release [7], to improve maintenance resource allocation and adjust deployment [84], to guide development process improvement [27], and to enable the selection among different product releases [70].

However, Fenton [29] reported most prediction models to be based on product metrics (e.g., size and complexity metrics) obtained after product release, which seems rather late for guiding development [109] and release process [84]. Another type of metrics, which is not as popular as product metrics, is process metrics. Process metrics are measures for development activities (e.g. developer source code contributions, developer email contributions) which can be monitored and obtained through all project life cycle [30].

The quality evaluation of open source software (OSS) products, e.g., defect estimation and prediction approaches of individual releases, gains importance with increasing OSS adoption in industry applications. Most empirical studies on the accuracy of defect prediction and software maintenance focus on product metrics as predictors that are available only when the product is finished. Only few prediction models consider information on the development process (Process Metrics) that seems relevant to quality improvement of the software product.

Metrics based prediction models which enable product metrics as defect predictor are the most common prediction model in closed source software project [73]. In a short-release-cycle environment such as in many successful OSS projects, product metrics signified low value variability and weak correlation to predicted defect pattern. Li et al. [71] Moser et al. [87] and

[15] Google Speaker Series: *Successful Open Source Projects* can be found at http://www.youtube.com/watch?v=ZtYJoatnHb8. Last accessed 1st March 2008

[16] E. Raymond. The cathedral and the bazaar. http://www.catb.org/esr/writings/cathedralbazaar/cathedral-bazaar/, 2003. Last accessed 1st March 2008

Wahyudin et al. [126] confirmed that prediction models that are based only on product metrics have worse performance compared to models which enable process metrics signify by lower number of '+' as shown in Table 2

Table 2 Comparison of Metrics Selection Impact to Prediction Results in OSS Projects

Estimators	Observation Entity	Project Context	Impact of Metrics to Prediction Accuracy			Reference
			Product Metrics	Process Metrics	Combined Metrics	
Defect occurrence over time	Product Releases	OpenBSD	+	++	++	Li et al. [71]
File defect-proneness	Java Files	Eclipse	+	++	N/A	Moser et al. [87]
Defect growth between releases	Product Releases	Apache MyFaces	+	++	+++	Wahyudin et al. [126]

The findings of these studies confirm other reports that data captured from developer activities contain more discriminatory and meaningful information about the defect distribution and defect removal capacity in software project than the static product metrics [87, 104, 127]. Hence, in this study we evaluated different types of product and process metrics and investigate the potential contribution of these metrics combination to improve the accuracy of the prediction results.

2.4 Chapter Summary

In this chapter we describe the methods for conducting empirical study in software engineering. The chapter also mentions several types of metrics which later we use for evaluating and predicting software product in our study contexts. Further we illustrate the current state of the art in software quality evaluation.

We also describe the needs for quality evaluation and prediction for OSS products which currently have become more widely adopted in many industry domains.

At the end of this chapter we describe the state of the art in software quality prediction in particular objective defect prediction, and typical types of metrics utilized by researches in software defect prediction.

We later enable the methods and techniques described in this chapter to answer predefined research issues with empirical data in our case studies.

3 PROCESS QUALITY EVALUATION OF DISTRIBUTED SOFTWARE DEVELOPMENT

To ensure a project's survival, a decision maker needs to continuously evaluate health status and recognize early symptoms of illness or risky situation (e.g. particular core committer leave the project can brain drain the rest of developer community). Such indication could be obtained by correlating measures that are available during the development.

This chapter presents models and research issues for DSD process quality evaluation. It starts with concerns for OSS project survivability as the show case of the study. Further it builds on causal modeling process to elaborate on process and influence factors for project "health" status. Project "health" status is defined as quality measures of current development processes.

The evaluation of project "health" indicators attempts to have well planned quality evaluation which address, to enclose prompt status of current development process status and early warning based on prediction models for certain risky conditions that typically occur in OSS projects.

For evaluation of the concept, we perform two of empirical studies. In the first study (see 3.5), two project "health" indicators (developer contribution patterns and defect service delay) were modeled and evaluated using cross-project data as attempt to obtain robust models that can hold more than one project context. The second study (see Section 3.6) focuses on identification of more health indicators that correlated with quality assurance aspects which are commonly performed by a healthy OSS project community. Finding health indicators from current quality assurance practices will provide insight whether current practices are good enough or depict needs for improvement. Additionally, dealing with handful yet focused health indicators consequently increase the efficient and effectiveness of the evaluation effort and data collection.

3.1 Related Work

This section outlines a) the concern for OSS project survivability which derives the motivation for conducting evaluation of development process quality in OSS project contexts and b) the current approach conducted by project management and project sponsor to evaluate the quality of development process of OSS projects.

3.1.1 Concerns for OSS Project Survivability

Open source software (OSS) has caught our attention by the success and quality of its projects on the market, despite the fact, that its development does not follow traditional software development principles such as mostly voluntarily project participants, informal project management, open code and open development process [16].

In certain software product classes OSS offers comparable or even better quality than “closed source” commercial software products, making OSS a considerable alternative in many domains reaching from operating systems over web-frameworks and databases to critical mission applications [24].

OSS project management needs to assess the developments processes quality and to recognize early some risky conditions that may endanger the survivability of the project. While a prospective OSS end-user needs to evaluate the quality of product release and identify whether a new release is worth for deployment.

In OSS, the project survival is a result of many underlying (connected) processes and cannot be easily determined. Just to give an example: developers are typically not paid for their work, but contribute voluntarily on their own motivation basis. Hence, the dynamic of the development process is much more difficult to estimate compared to that of a typical commercial project. This issue is problematic for certain stakeholders in OSS community to fulfill their goals such as prospective customers of OSS products in order to decide which products will provide long-term warranty and enhancements; the hosting project (e.g. Apache, Sourceforge, Eclipse and Codehaus) to provide or to continue support for some projects under their umbrella, and for the project leading teams who steer the project’s direction based on project status in timely fashion.

The scale of the problem is escalating when a large number of projects should be monitored in parallel. The resulting OSS project monitoring faces ever-increasing demands to provide pertinent data from the dynamics of the projects, to help stakeholders cope with complex masses of data/information, to provide competitive discriminators based on the stakeholder values; and to provide the ”health” status of the project.

3.1.2 Evaluation of Development Processes Quality: Measuring the Maturity Level of Development Processes

Sourceforge [111] evaluate the maturity level of a project based on product downloads and project site hits, intensity of commits and file releases, and community communication traffics

i.e. in mailing list, forum and issue tracker [112].

Our observations on 178951 projects listed in Sourceforge at March 2007, reveals that the top 5 project categories are Internet application (15.4%), Software development (15.1%), System (12.4%), Communication (10%), and Game/Entertainment (9.3%). Sourceforge ranks these projects into several categories which are: Planning (18156 projects), Pre-Alpha (15314 projects), Alpha (17190 projects), Beta (23198 projects), Production/Stable (19531 projects), Mature (1675 projects), and Inactive (2124 projects).

This fact depicts that most of the projects are still in early stages or already at the end of their lifecycle, and only a small portion (less than 2%) of the projects in SourceForge have reached their maturity.

To elevate new project initiatives survivability level, the Apache Software Foundation (ASF) set up an incubation process called Apache Incubator as the entry path for each project initiatives called as *podling* to become part of the Foundation's efforts. As stated in ASF Incubator guideline [2], the role of incubation process is to provide guidance and support to help each *podling* engender their own collaborative community, educating new developers in the philosophy and guidelines for collaborative development as defined by the members of the Foundation, and proposing to the board the promotion of such products once their *community* has reached *maturity*.

A project considered as mature in ASF after it's graduated from the Incubator after it shows self-sustaining and self-governing communities to the Foundation board members. Such community can be achieved by having an open and diverse meritocratic community which proven to be more robust and productive compare to closed ones.

In Figure 8 an Open Source Software Project Lifecycle, we illustrate a model of an OSS project lifecycle. In this model, after a project was born, it starts its infancy states (i.e. within the Incubator for new project in ASF), depend on where this new project hosted the project initiators should attract more participants and adhere to sponsor guidelines in order to achieve its maturity status.

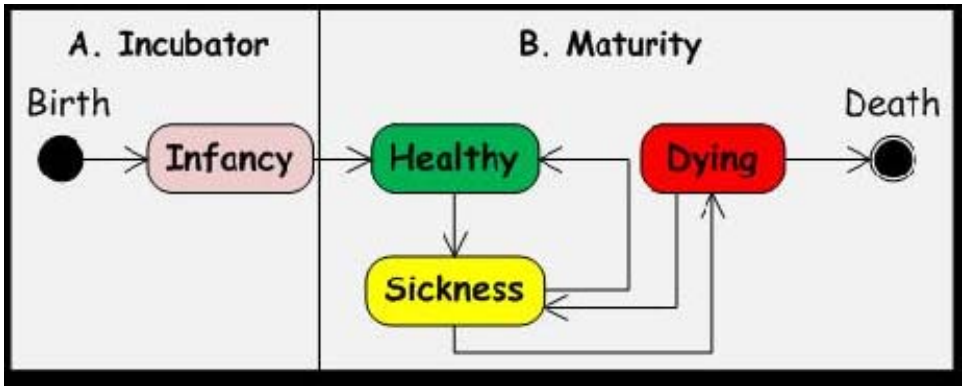


Figure 8 an Open Source Software Project Lifecycle

An OSS project that has reached its maturity basically is entering a free market, where it should compete against similar projects to attract more participants (developers and users) and to evolve its products over the time.

Both SourceForge and Apache Software Foundation have similarity regarding the concept of project healthiness which should be based on the aliveness of the community and how it should grown by attracting more active participants and produce stable releases with good quality.

In a healthy project typically has highly motivated developer community who eager to produce quality software in order to get more attention of potential participants to their project in term of development participations or at least feedbacks of product releases [18].

3.2 Causal Modeling of OSS Survivability

Crowston et al [21] suggested the success factors of an OSS project consisting of software creation/developer contribution, software use intensity and software quality.

To better illustrate the impact of these factors and typical risks during development process, in our context; we modeled the OSS project as a body consisting of three major components: the developer community, the user community and the software product.

Figure 9 outlines the success factors and risks as interrelated states and activities which indicate the project component status.

We assumed that the survivability of the project is the result of the state of well being (aliveness) of both communities indicated by facilitating rapid creation and deployment of the incremental product releases or patches. Furthermore, this release should satisfy relevant user needs. The following subsections describe more detail analysis about our causal model, based on literature review and web observation.

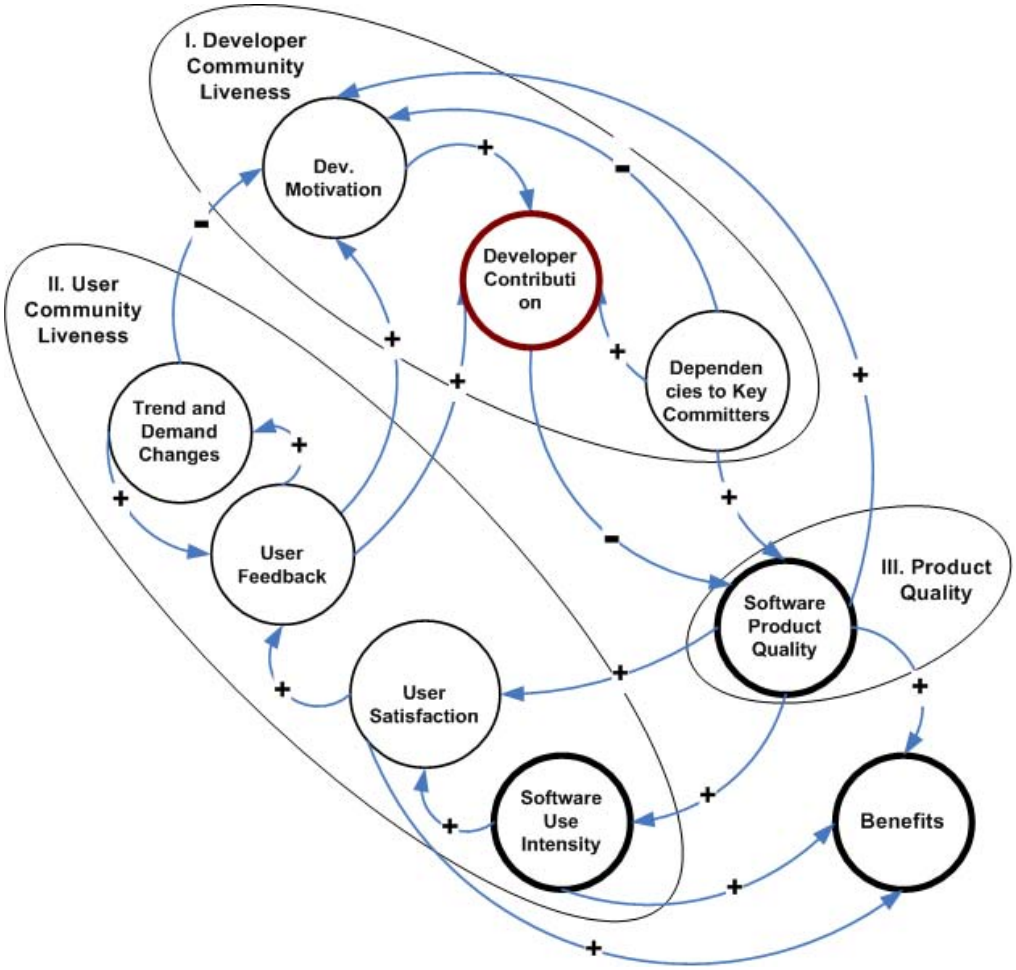


Figure 9 Causal Model of an OSS project Health Status

3.2.1 Group I. OSS Developer Community Aliveness

Successful OSS projects are not one time event. It is a process of a long life cycle which was first coined by Eric Raymond [105] as "scratching the developer itch". The developer community continues to contribute, develop, enhance, maintain and release the products developer contribution. Iteratively in a typical OSS project management style. Therefore, in order to survive a project should attract more developers and boost their motivation.

Studies from the projects listed in Sourceforge by [65, 77] signify that the developer community may consist of a single fighter up to more than 200 active developers at one time. They also disclosed that 86.2 % of the projects employ less than 6 developers during the development processes. A survey from The Boston Consulting Group [8] disclosed that the developer motivations to join an OSS project are to stimulate their intellectual, to enhance their skills or to have access to the source code and user needs. As most of the developers join the OSS project for their self satisfaction, the degradation of developer motivation are not trivial and very likely

cause project into deep problems if there is no appropriate counter measure. Lerner et.al [69] suggested that developer loyalty can be obtained by giving some incentives such as opportunity to contribute, community attention and recognition based on merit to the project.

The nature of OSS community is a social structure that provides some hierarchy of management and controlling based on self-organizing patterns [119]. According to Gacek and Arief [39], developer in OSS project consists of code/peripheral developers and core developers which also called as committers.

A project can be generally well developed and provide regular releases, which are appreciated by the user community, but it might actually be driven by very few active committers (dependencies to key committers). In the worst case, the project might depend on one particular person. This is obviously a risky situation for the project and its users as the key committers may leave which then brain drained the project, and de-motivate other developers such as in Apache Slide [121].

The dominance of the key committers may also reduce opportunities to contribute by peripheral developers, can be considered as a hostile action. This dependency has been considered as a major issue by Apache Software Foundation which should be comprehended by all new project initiatives under the Apache incubation process. Other typical risky situation that may threaten OSS project is the shift of market or the change of technology which cause project disorientation and consequently de-motivates the developers to abandon the project (Like Native XML Database).

3.2.2 Group II. OSS User Community Aliveness

The second groups in OSS project community are the users who observe, download and then use the software product for certain objectives (e.g. curiosity, work functions, and user needs). The level of software use intensity will be amplified when certain quality attributes of the software product satisfy the user value expectation.

Compared to those of commercial software products, the users in OSS project are expected to be more active to provide feedback for functionalities of product release. Some studies [1, 82, 124] reported that most of the defect reports and the feature requests came from the user community, which were then responded by the developer community by submitting patches or new features.

Eventually, these practices caused rapid changes into the code and documentation. However the user needs and expectations may change over the time due to the technology evolution, the shift of needs or some other reason. Eventually these changes may imply the trend and demand of the market of the OSS product. Hence, it is fair to say that the user community has significant impact

on the OSS project community aliveness in the whole and the quality of the software product released.

3.2.3 Group III. OSS Product Quality

The typical characteristic of OSS such as open code based and no formal project management has raised some debates about the quality of the released products. However a survey from BCG [8] suggested that open source community is mostly comprised of highly skilled IT professionals who have, on average, over 10 years of programming experience and it is not exaggerated to assume that these people are well knowledgeable to produce a good quality code which is contrary to popular belief about hackers.

Recent study in OSS quality [1] suggests several software engineering quality model that typically practiced in OSS project community such as peer review to assess whether a contribution merits from developer acceptance into the codebase.

In a large project such as Apache Server, peer review practiced not only for assessing the quality of contributed code but also applied for a new idea/solutions submitted to the developer community which need to be discussed, and reviewed before being planned for development. The bazaar style of OSS development facilitates rapid releases which make the implementation of peer review. The existence of quick response to reviewers comments and code keeps the contributor involved and interested [105].

The second typical quality practices are people management in reporting, reviewing, detecting and resolving issues and defects (more details discussed in section 5). Abernour (2007) advocates on this practice to include establishing an effective environment and culture which is as important as system design.

This means there should be a pre-defined coordination mechanism [34], conflict management(such as voting) [94], encouraging innovation and creativity [68], and affectionate attention from the community [69]. With respect to software quality assurance terms, hence for the rest of this thesis we refer to defect as bug reported about particular OSS product.

The third and most prominent quality practice is defect tracking activity. In traditional project, defect tracking is similar to inspection which is effective but also expensive quality assurance.

Mockus et al. [82] enclosed after a product release the user and larger part of the OSS community typically shift their roles in reporting, reviewing and resolving defects or issues. These practices of SE quality model influences the community *aliveness* by encouraging project participants to be involved and motivated to contribute and results in a product that extends

rapidly and reaches high quality, here we conclude that only a healthy community can produce a high quality software.

3.3 Stakeholder Value Proposition of OSS Product Quality Prediction and Evaluation

As mentioned in section 3.1, the survivability of the OSS project is the result of the state of well being (*aliveness*) [121] of developer community indicated by facilitating rapid construction, defect-fixing and deployment of the incremental product releases or patches. Furthermore, the current release should satisfy relevant user needs triggered by feedback information from the user community [122].

The stakeholders in OSS projects are represented by each individual in the community connected through project environment. Based on their role they have different expectations and (subjective) indicators of product quality [125]. Hence, to better understand their quality expectations in OSS projects we need to elicit their values, starting with eliciting their win conditions based on their roles in the project and define quality performance measures [10].

We interviewed OSS experts in January 2006 to find out the stakeholder' needs for good quality OSS product, we conduct the interview in two session first is by direct interviewing the experts with some open questions and second by asking more quality assurances (QAs) focused questions through emails such as what are the quality expectation of different project stakeholders (see Figure 10), what typical QA aspects that typically performed by developer community. In this chapter we focused on defect lifecycle as the prominent part of software product and process improvement in OSS project [105].

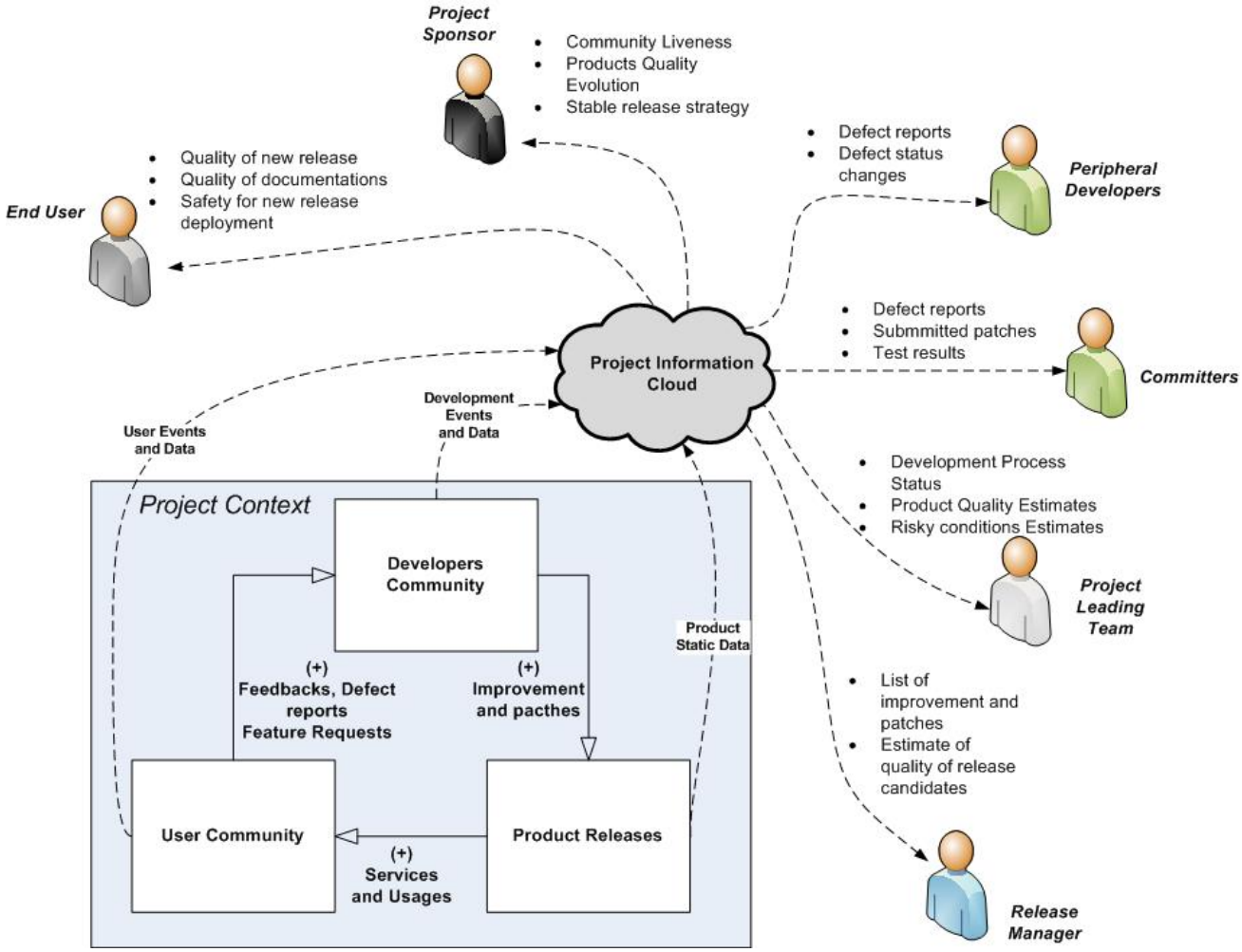


Figure 10 Expected Quality Aspects of OSS Product Releases from different Stakeholders Point of Views

As extensions to the results presented in [122, 124] and experience reports on OSS Projects [37], we elicited different stakeholders with their expected value of quality evaluation and estimation of OSS products:

1. **End User**. The user community applies OSS product releases and provides feedbacks to the developer community [16]. The results showed two typical user groups: (a) common users focus on high quality releases (i.e., more features, fewer defects, faster defect resolution time, better usability and documentations) as their primary objects of interest or win condition; (b) more professional users, such as an IT manager, need more information on stability and safety of the new release deployment to current running system and list of potential problems with relevant troubleshooting.
2. **Project Sponsor.** Project hosting sites such as Apache Software Foundation (ASF),

Sourceforge, and commercial companies such as IBM and SUN that provide support for OSS projects can be considered as project sponsors with different types of sponsorship and involvement within the hosted projects. ASF for example provides an incubation process called Apache Incubator[2] including guidance and support for each new OSS product initiative, stimulating the collaborative community, educating new developers, and proposing whether a product has reached maturity. To appraise whether a project initiative requires additional support, project sponsors need to have sufficient information of the *aliveness* of the project community, the quality of the provided product, and the community set-up including a strategy for stable releases (i.e., voting mechanism for release and feature candidates).

3. **Peripheral Developers.** Peripheral developers are the largest group of developer in the developer community, as they are mostly active in defect removal activities and patches development [83]. The meritocracy system in OSS project such as in all Apache projects put peripheral developers in the lower level of the project structure, as they do not have high level privileges such as writing access to current code base. Therefore every defect resolution and patch submitted into the issue tracker should be reviewed by other developers or a committer prior to code base modification and deployment. The larger group of developers working on the head and incorporating the current version in their productive applications, and when they do, they really find out about problems in the OSS product as well. Hence, two prominent quality sources of information of current release are: (a) defect reports including traceability of reported defects and (b) defect status reports using an issue tracker, as it is very typical that different developers work in defect report validation, defect resolution development, and peer-reviewing of submitted resolution. In quality aware communities such as MyFaces, Python, Gnome and Mozilla, a reported and implemented issue and corrected defect passes several QA activities which are similar to design review, code testing, and code peer review before deployment [122, 126]. The win conditions of developer are to have proper access to current development repositories and collaboration tools to support their works, merit based incentive from the community, less invalid defect report , adequate information of reported defect, and flexible time to resolve defect.
4. **Committers**. Committers or core developers have responsibility for assuring the quality of the software product before product release and deployment and have full access to the code base stored in the project's source code management system (SCM), i.e., CVS or SVN [1]. Typical tasks of committers are to review any defect resolutions and to

decide whether a patch should be added to the release log. A committer may expect to have adequate information of (a) valid reported defects including their severity and impact on the overall product behavior, (b) new patches to review whether every defect were solved properly according to defect specifications, and (c) test results of new patches. Therefore, they can assure that a new contribution will not possess any threat to current code base and ensure compliance of the new piece of software and the code quality standards of the developer community.

5. The **Project Leading Team** comprises of elected committers with roles in daily project management. According to the Apache Software Foundation [3], the main roles of the project leading team are to ensure that all legal issues are addressed, that the procedure is followed, the alleviation of any bottlenecks and conflicts, the overall technical success of the project and that each and every release is the product of the community as a whole. They also responsible to give strategic decision, to further the long term development and health of the community as a whole, and to ensure that balanced and wide scale peer review and collaboration do happen. Hence, they need to monitor the development process and to ensure appropriate quality assurance activities (patch discussions, unit and integration testing, and peer-review) are well performed [124]. They also need to estimate the quality level of the current developed code and identify potential project risks, e.g., portion of reported defect remaining unresolved after a certain period of development and high number of failed code tests near to release deadline.
6. **Release Manager** is an assigned committer to guide and manage release process, for example selecting stable patches, new features that should be added to the upcoming release package, and selection the best potential release candidate [37]. Typically, the release manager is a member of the project leading team. A release manager has to perform the selection based on defined quality criteria i.e. only peer reviewed and tested improvements (patches and new features) can be added to the next release. Additionally, as the tradeoff of short release cycle, release manager needs to estimate whether there will be significant growth of defects that potentially reside within a release candidate with regards to the current developer capability in solving defects after the planned release.

These roles depict the need to monitor the health of the community in timely fashion, and to quickly respond appropriately against certain status during project execution.

This study focuses on providing health indicators as prompt quality prediction and evaluation in

OSS projects. From project leading team and release manager perspective, the detailed win conditions of OSS project "health" indicators assessment are:

1. *To retrieve comprehensive indicators which indicate the actual status of project performance*. As the project may produce significant number of data, artifacts and project information which can be further processed as indicators, however due limitation of observers in OSS project as they are not a full timer, thus to compact the indicators with respect to quality of enclosed information are necessity, the key measures of this win condition probably to have a small set of indicators based on observer's own priority selection, just to give an example from IT system monitoring, our interviews with a group of system administrators result that the group daily monitor from 8 to 10 indicators, as to have more may overwhelm and mislead the observer analysis.
2. *Availability of performance metrics that provide the necessary information for different levels of detail*. This second condition is the result of the first one, as in a project we may start to monitor higher level indicator and then go deeper for better understanding the nature of development process or track back the symptoms of illness. We suggest to divide the metrics into three layers: (1) the direct metrics, which are the lowest layer consist of metrics directly obtained from project repositories such as number of developer email contributions, time stamps when a defect is resolved, number of defect resides in the defect database etc. (2) the derived metrics, which are the aggregation of several direct metrics such as the average time to resolve defects in certain time, average number of developer email contribution in one semester, etc. (3) the indicators, which are the highest level and provide comprehensive view of certain status of the project, e.g. the service delay within a release which depicted as a function of average time to respond and average time to resolve of defects within a release. Figure 4 illustrates the examples of retrieving health indicators from a hierarchy of metrics.
3. *Early availability-time relevant project information status*. The third win condition is to have quality information in a short time, as the project indicators represent the current status which is not merely historical data. Time relevant information will give more accurate indication of the project, and if the project is considered unhealthy, then the appropriate treatment can be applied to change the project into healthy one before it is to late or getting worse. To get valid information status, thus project manager should set a retrieval time constrain for each indicator he wants to monitor i.e. within

hours, days, or months.

The stakeholder values are varying based on the stakeholder roles, domain of the project application and project scales. The value elicitations will define the selection of to-be-monitored health indicators to assess the project health status.

For better illustrating the QA aspects and interaction among project community during development process, we proposed a framework of QA aspect in OSS project, which described in following section.

3.4 Modeling the OSS Project "Health" Indicators

In the previous subsections many different parameters had been taken into consideration to get an impression of the status of a project which is actually not an easy task. This is particularly problematic if a large number of projects need to be monitored. Based on the described success factor there are some indicators that experts routinely use to assess an open source project, such as the following data.

1. Proportions. We calculate the proportions of activity in the community of e.g., volume of mailing list postings, defects status changes per time slot, updates in the SCM, and use these metrics to compare projects to try to learn what "healthy" relationships are. Based on data from healthy projects we can identify whether there are correlations between different types of developer contributions, and how the likely impact of one contribution metrics to another.
2. Service delays. By measuring the time between a defect reported to the issue tracker with time of a developer responds the report we can calculate the response time, we can also calculate the defect closure time as time between reporting with time a defect state as resolved with positive resolution (*fixed* or *resolved*).
3. Communication and Use intensity. If a project has a healthy community there is indication of strong relationship between some measures such as number of downloads compared to mailing list postings and active developer interaction in (different) mailing lists.
4. In a distributed project, a set of certain Software Engineering (SE) methods and tool standards have been established to support open source projects. The tools are used where activities of developers can be interpreted as events (time-stamped date points).

Typical important tools are: source code repositories, documentation systems (source, user), defect tracking, mailing lists, forum software (web, newsgroup), and Wiki content management.

These tools provide informative but scattered pools of data during the project life-cycle. Obviously, developers use these tools and data for coordination, communication, and configuration management.

We propose an approach by unifying the data coming from different systems into a coherent format for analysis. We expect not only data for historical analysis but also for daily or weekly monitoring and analysis. If it is achieved, it is possible to monitor the status of project health regularly and receive early warning signals, if bad smells occur.

For analysis it is necessary to collect, filter, and correlate these data elements. While a human expert has to do the analysis by looking into the different systems, only little tool support is readily available for automatic and continuous ”real time” analysis of project status.

In today practices, health indicators and healthy community of OSS projects have become more important issue. Just to give an example, the Apache Incubator defined that a new project initiative may graduate from the incubation process by fulfilling some requirements: the project must have a healthy community indicated by an active collaborative works within a community and it consists of diverse core developers.

For the diversity of core developers' measurement, we can quantify the number of independent core developers based on their background profile. The diversity is important because: (a) it guarantees a sustainable development, as the project will be less dependent on a single developer, (b) it brings variety of competencies to enrich the quality, however our interview with OSS expert suggest that this indicator is best to be obtained manually by retrieving each committers personal data and analyzing the project profile.

Active collaborative works are indicated by several health indicators such as the coordination activities, conflict resolutions (number of voting), intensity of usage, defect service delays, and the proportion of the developer contribution to the project repositories. In this paper we focused on the last two health indicators:

1. *Developer Contribution Patterns*, the first health indicator is a function of proportion metrics which capture the ratio between email conversation with defect status changes and ratio between SCM commits with defect status changes. Within a healthy project most of developer activities are correlated with each other, for example is that a code submission into an SCM may trigger an email conversation within the developer mailing list which warrant a concern from other developers to peer-review submitted code. Another example is that defect status changes may also trigger an email conversation regarding the changes that for particular interested developer. Hence, this indicator is very important to obtain an outlook of software creation performance and the current

developer motivation state.

2. *The Service Delay*, in a commercial project, is the time interval to respond a service request from a customer and time to fulfill the service, while in OSS project; we define the service delay as a function of time to respond and time to resolve an issue/defect. The defect/issue service delay is important as most of the activities in OSS project derived from defect or issue report, eventually a project which has slow response and resolution time will face problems such as user dissatisfaction and bottleneck in the development process.

Both indicators derived from aggregation of metrics, which at the lowest level the metrics are obtained by mining the project repositories as illustrated in Figure 11.

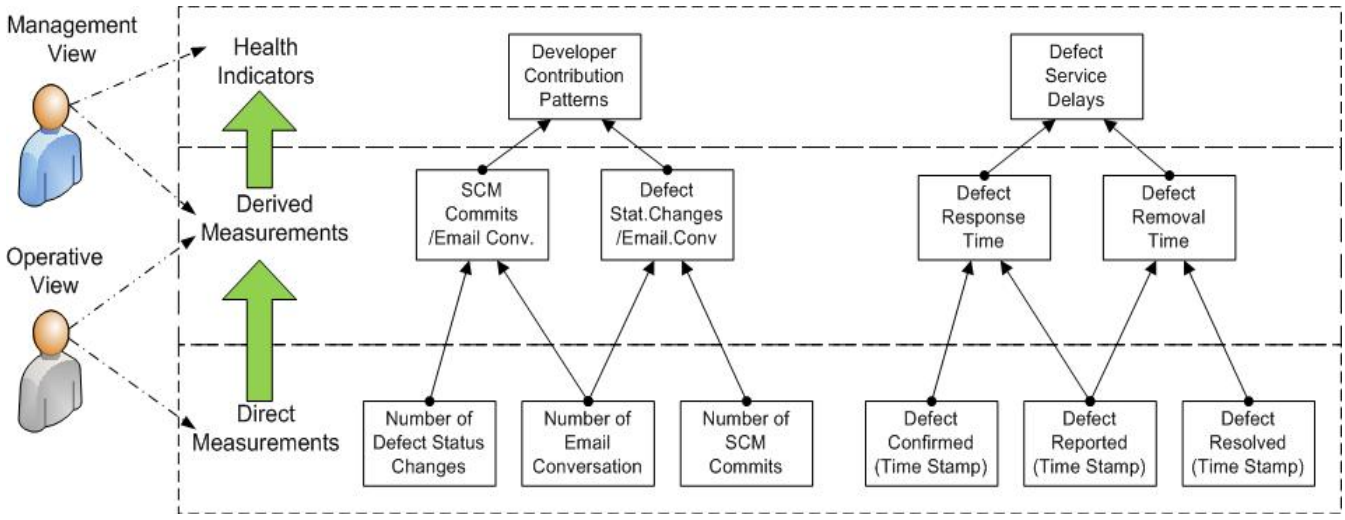


Figure 11 Deriving OSS project health indicators from basic process metrics.

The measurement selection for monitoring project status depends on the stakeholder values. The Operative level measure direct metrics from project repositories which are then transformed into higher level metrics. Project manager analysis the health indicators as the results of correlation of these aggregated metrics.

Managing and monitoring health in open source software (OSS) projects is a complex challenge, due to the typical characteristics of OSS development model. Important indicators such as activity of developers and performance of Defects' management are easier to measure as they have become part of the development nature itself. However, for the comprehensive determination of health measurement one has to consider other indicators which needs be formulated and further explored.

In the next section, we applied the proposed health indicators to well-known OSS Apache projects for empirical evaluation of the concept.

3.5 Empirical Evaluation of OSS Project "Health" Indicators: *Developer Contribution Patterns and Defect Service Delay*

This section describes the empirical evaluation process of the health indicator concepts and the initial empirical result from some projects under the Apache umbrella.

Following the systematic empirical study framework in OSS projects, it starts with design of empirical study, reports the data collection process and data analysis. Empirical data were retrieved from several successful OSS projects and the challenged ones. We compared the results to provide better understanding of project state of well being.

Later the results were discussed with OSS experts to investigate the external validity of the indicators.

3.5.1 Design of Empirical Study

The following subsections present the goal definition of the study, selected study objects, variables specification and research hypotheses formulation.

1. Goals of Empirical Study

Using goal structure in GQM model [4], we defined the goal of our research study as:

The purpose of our study is to have better knowledge of indication of project "healthiness" derived from metrics obtained from developer activities, from project manager point of view.

2. Study Objects

We apply the proposed health indicators to four cases of large-matured Apache web engineering projects indicated by more than 20 contributors (core and peripheral developers) per project which three times outsized the average number of developer in most of OSS and has already at least one major release (1.x, 2.x, etc). The set consists of two well-known Apache projects (Tomcat v.5 and HTTP Server/ HTTPD v.2), and two challenged projects (Xindice and Slide). We focused our evaluation process on the two health indicators: *the proportion of developers' participation* and *the defect service delay*. As described in Section III these indicators are very worth noted by a project manager to assure that a project is still actively running and both indicators are simple to be evaluated.

Apache Tomcat[17] is a servlet container that is used in the official Reference Implementation for the Java Servlet and Java Server Pages technologies whose code base and specifications are donated by Sun under the Java Community Process in 1999. The first Apache release was version 3.0. Since then, multiple volunteers from Sun and numerous other organizations have contributed to the product. Currently Tomcat has several major releases, employs 17 active committers and more than 50 emeritus committers. In 2005, Tomcat became its own top-level Apache project and powered numerous industries and organizations such as Wall Mart and General Motors. A survey by *TheServerSide.com* [18] pointed Tomcat as one among the market leaders in its application domain.

Apache HTTP Server[19] is an effort to develop and maintain an open-source HTTP server for modern operating systems including UNIX and Windows NT. The development started in 1994 when Brian Behlendorf and a number of users for internet servers which are developed by the National Center for Supercomputer Applications (NCSA) encountered the increasing frustration in getting NCSA to respond to their suggestion. They decided to collaborate and integrate patches to the NCSA server software. In August 1995, the group released Apache 0.8. Since then the product was called as Apache HTTPD. Later it is well-known as Apache HTTP server and has been the most popular web server on the Internet since April 1996. The HTTP server project employs 56 core contributors and hundreds of peripheral contributors. The project latest release is the version 2.2.4. Apache Tomcat and HTTP Server are considered successful large projects. Hence by examining the dynamics of both project communities, we improve our knowledge about the indication of a healthy community.

Apache Xindice[20] is a database tool designed from the ground-up for storing XML data or what is more commonly referred to as a native XML database. Xindice is the continuation of the project that used to be called the **dbXML Core**. The dbXML's source code was donated to the Apache Software Foundation in December 2001. During the development, the developer community consists of 8 active committers, 7 inactive and emeritus committers, and 19 contributors. Xindice has its first stable release of version 1.0 around March 2002 and continues with several milestone releases, such as version 1.1b4 at 8 April 2004.

Apache Slide[21] is part of the Apache Jakarta project. Slide is a content repository which can

[17] http://tomcat.apache.org/ (accessed at 20/02/2007)

[18] http://www.theserverside.com/ (accessed at 10/01/2007)

[19] http://httpd.apache.org/ (accessed at 20/02/2007)

[20] http://xml.apache.org/xindice/ (accessed at 20/02/2007)

[21] http://jakarta.apache.org/slide/ (accessed at 20/02/2007)

serve as a basis for a content management system / framework and other purposes. The original Slide codebase (Slide 0.7) was donated by Intalio Inc during May 2000. Slide has reached its maturity after release 1.x and 2.x. The project employs 14 active committers, 14 inactive/emeritus committers, 20 contributors and 3 project sponsors. However after its 2.1 release (at 12/26/2004), the project seemed to be disposed, although we still recorded some activities in the developer mailing list. Our interview with OSS expert indicates that Apache Xindice and Slide are in difficulties. This case can be taken as a fine comparison to successful ones and reveal the symptom of illness of a project.

3. Variables Specifications

In this study we evaluate two health indicators with empirical data collected from project repositories which are:

First Health Indicator: Developer Contribution Patterns

A commit into SCM can be different forms of contribution such as changes in code, new defect patches, or release documentations. In an OSS project, the developer mailing list is the main collaborative communication tool, where everyone who wants to participate in the project development can observe or join in. In Apache projects, the email archives commonly consist of three major contents: *the notification of developer commits to the SCM, notification of defect status report (the change of defect state) as a developer may work on something for the defect, and development-related short messages/email conversation* i.e. problem reports, solution recommendation, polling for opinions, and technical discussion.

Hence for the first health indicator, we define dependent variables are: a) monthly number of commits into the SCM, b) monthly number of defect status changes in the issue tracker and c) the intensity of email conversation within a month which can be triggered either by a new code/patches submission or a change of defect status.

The dependent variable is ***developer contribution patterns*** as the ratio between number of email conversation, SCM Commits and defect status changes.

To observe the relationships among the measures, we employ bivariate correlation analysis and analyze the likely impact of one variable to another. Later we perform vector analysis by constructing a multiple linear regression model to investigate whether the intensity of email conversation within the developer mailing list can be estimated using number of commits and defect status changes as predictor variables which can be formulated as:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \varepsilon \qquad \textit{Eq. 1}$$

Where Y is number of email conversation as vector of response, the predictor variables or parameters are: X_1 is number of SCM commits and X_2 is number of Defect status changes. While β_0 is intercept of the prediction model *and* β_1, β_2 are estimated parameters of the predictor variables X_1 and X_2, and ε is a vector of independent normal random variables with expectation for the constructed model.

As measure the prediction model performance, we applied average absolute error (AAE) and average relative error (ARE) which are suggested by [56].

Lets denote dependent variable (Y); and Y' as estimator of Y. Then AAE is the magnitude of the difference between the exact value and the approximation, which can be formulated as:

$$AAE = \left(\frac{1}{n}\right)\sum_{i=1}^{n}|Y - Y'| \qquad \textit{Eq. 2}$$

While ARE is the absolute error divided by the magnitude of the exact value:

$$ARE = \left(\frac{1}{n}\right)\sum_{i=1}^{n}\left|\frac{Y - Y'}{Y}\right| \qquad \textit{Eq. 3}$$

Since our observation involved different projects, hence to compare the performance of each models used ARE as the primary accuracy measure, while AAE can be used to better understand the accuracy level within the individual context of each project.

Second Health Indicator: Defect Service Delay

Although in an OSS project, the community voluntarily plays significant role in the defect tracking, and resolve the financial barrier as in traditional project, nevertheless a healthy project should employ proven defect management practices offer fast defect response and to reduce the service delay in removing defects.

To measure the defect response time and defect removal time, we used independent variables: time when a defect report filled into the issue tracker (*Treport*), time when a defect status confirmed as new defect (*Tresponse*) and time when a defect stated as resolved (*Tremoved*). Therefore we can calculate the responsiveness of the developer community for each reported defect *i* for *n* number of reported defects as:

$$Responsiveness = \frac{\sum_{i=1}^{n} Tresponse_i - Treport_i}{n} \quad Eq.\ 4$$

While the service delay performed by developer community in removing each defect *i,* for *n* number of reported defects can be calculated as:

$$Service\ Delay = \frac{\sum_{i=1}^{n} Tremoved_i - Treport_i}{n} \quad Eq.\ 5$$

4. Research Questions and Hypotheses Formulation

RI.4.5.1. the Developer Contributions Pattern. We addressed two questions for measuring the healthiness of developer contribution pattern: 1) *Are there any significant correlations among developers' contribution components (email contributions, defect status changes, and SCM commits)*?. We believe that in healthy OSS project that put strong consideration regarding the quality of contribution, for each submitted patches, code set or defect state changes should be reviewed by other developer or at least discussed in the mailing list.

Let's denote that r is a correlation function to check whether there is a significant correlation between variables (x_1, x_2) ϵ Pr and dependent variables y, where x_1 is number of commits to SCM by developers, x_2 is number of defect status changes, pr is a set of development metrics, and y is number of submitted emails to the developer mailing list, then the respective null hypothesis can be formulated as:

H04.5.1.a: $\{\forall\ x \in pr | r_i(x_i, y) = False\}$

Our second question is that 2) can we predict the potential number of email contribution from the developer as function to code submission and defect resolution activities?. Here we want to in-depth checking, whether significant correlated variables can be used as predictor variables for developer email contribution pattern. For this purpose, a linear regression method (see equation 4.1) is employed and tests the significance of constructed models using standard F-test.

The hypothesis is that in healthy project such as Tomcat and HTTPD there are linear correlations between variables. Thus to estimate the intensity of developer email conversations as one measure of project healthiness one can construct a prediction model with higher level of accuracy compare to challenged projects.

Therefore we proposed following Null hypothesis:

Let's denote ϵ is estimates from a trained prediction model using development metrics either

from a healthy project (*Ph*) or challenged project (*Pc*) then:

H04.5.1.b: ARE(e(Ph)) > ARE(e(Pc))

RI4.5.2. ***Defect Response Time and Defect Service Delay*****.** The defect status reports are also important to illustrate the project service throughput. However we suggest that monitoring the dynamics of the defect status report should be further correlated with other variables in the defect tracking activities such as the *number of defect per reporter*, *defect response per reviewer*, *defect assignment per contributor*, *service delay* (response and closure time), *defect validation time* etc.

Here we address 3 questions for assessing defect management in an OSS project: (2.) Is there any appropriate defect reporting and monitoring in place? (3.) Is there any appropriate rating of defects?, and (4.) What is the distribution of response time and closure time of defect reports?.

We assume that a healthy project with dynamic developer community can promise a better level of defect service delay, both in responding to a new defect report or resolving a valid defect. Therefore we can propose two null hypotheses:

H04.5.2.a: *Responsiveness* (Tomcat, HTTPD) < *Responsiveness* (Slide, Xindice)

H04.5.2.b: *Service Delay* (Tomcat, HTTPD) < *Service Delay* (Slide, Xindice)

3.5.2 Data Collection

This section describes the data collection process, data refinement as preparation prior to analysis, and threat validity of collected data.

1. Data Collection and Data Refinement

To answer the above mentioned research questions, we need to collect empirical data about some existing OSS projects. For the purpose of measuring the developer contributions, I developed a tool for mining the web based developers mailing list (hosted by Mailing list Archive, *MARC*[22]) of each selected project.

The tool uses the approach of a wrapper which retrieves project data web pages and then parses them to extract the information required. It retrieves the emails data (sender, subject, thread, time stamp) based on some given time interval as the input parameter.

The whole outcome is represented as *XML* and then using *XSLT* is further transformed to finally result in performance metrics, such as number of emails, number of commits to project *SCM* and

[22] Mailing list ARChives, MARC can be found at: http://marc.info/ (last accessed 20 July 2008)

number of defect status changes.

For each project we selected and examined 38 months of projects' life time with at least one major stable release. To support viewing the dynamics of the projects, these emails are then grouped into monthly archives and the proportion are calculated. We illustrate the ratio of developers' contribution over the time and compare the result between the successful projects and the challenged ones. Later, we chose Apache *HTTPD* as the role model of a healthy project and employs linear regression to figure out the correlation between email conversation, code contribution and defect report status.

Apache Project has centralized it defects tracking within repositories managed by *GNATS* and later moved to *Bugzilla* and *Jira*. The *Bugzilla* offers more features like transaction logs (history) and search facility either simple or advanced search on descriptive information of defects. This makes the dynamics of defects are relatively easy to trace.

It is not surprising that Bugzilla became very popular and widely used by 550 projects or companies[23]. This indicates that each Apache project being evaluated implements appropriate defect tracking tools.

In order to evaluate the project service delays based on defect's statistics for one stable release of each project, we retrieved each project's defect reports, measure quantitative data items, and depict the result within one software major release.

Unfortunately, some data needs pre-processing due to inappropriate or illegal values. The pre-processing steps involve: removing records indicated as "*INVALID*" and "*DUPLICATED*" in the resolution field and excluding records containing invalid date (either in the open-date or change-date).

We retrieved and examined change logs from the defect database (Issue tracker) of Apache Tomcat 5, HTTP Server 2.0, Xindice and Slide, using BugZilla 3.0rc1 query commands. Furthermore, in measuring the distribution of response time and closure time of defect reports, we formulate the calculation using the criteria: (a) response time is the length of time interval between open date and last change date for the defects having status field set to "*NEW"*, which means the defect is confirmed and accepted by the community for further processing, and (b) closure time is time to resolve a defect, calculated by measuring the length of time interval between open date and last change for defects having status "*RESOLVED*", which means the defect is already went through the development processes.

[23] http://www.Bugzilla.org/installation-list/ (accessed at 25/02/2007)

2. Threats to Validity

Internal Validity. In this empirical study data were obtained from 40 months of development (Ended at April 2007), we found that in the latest two months of observation, there is no significant number of development data collected for Slide and Xindice (as for Xindice already reveals that the project is in a dire situation), while in Apache HTTPD and Tomcat we also found large number of "Invalid" and "Duplicated" defects, therefore to maintain the validity and quality of data we just focused on 38 months of development (Ended at February 2007).

Construct Validity. The second threat derived from our discussion with an OSS expert who advised that in a large and very active project such as Tomcat and HTTPD, there is a common practice in developer community to re-open resolved defect, however in this study we assume that to re-open a resolved defect is also part of defect removal activities as the final resolution has not been reached yet, therefore for service delay calculation we ignore the re-open status of a defect which may increase the average defect service delay of a project.

3. Data Grouping

The collected data were divided into two groups; the first group consist of 34 months of observation of each projects will be used to perform the correlation analysis and to construct a prediction model. The second group consists of 4 months observation data from each project which will be used to validate the prediction model.

3.5.3 Data Analysis Results

This section describes the empirical result which is composed of data collection from the four Apache projects and data analysis.

A. The Developer Contribution Level

As described in our empirical study design section, all projects can be considered as large project (based on definition from [66]) as they employed more than 10 active developers at least for the first quarter of the development.

Table 3 shows the distribution of collected development metrics used to analyze the patterns of developer contributions in healthy and challenged projects.

Table 3 Distribution of Collected Development Metrics

	Collected Metrics from 38 Months of Observation								
	SCM commits *(X_1)*			Defect status changes *(X_2)*			Email Conversations (Y)		
	Total	Mean	STD	Total	Mean	STD	Total	Mean	STD
HTTPD	14199	373.63	99.42	8757	230.42	106.12	157814	415.22	135.90
Tomcat	9793	257.71	110.08	17816	468.84	179.70	16574	436.16	156.23
Slide	3617	95.18	93.78	1383	36.39	36.66	4479	117.87	107.68
Xindice	201	5.29	8.67	187	4.92	5.85	818	21.53	36.18

Figure 12 exhibits the absolute number of developer contribution within 38 months of development of the reviewed projects.

The developers' contribution patterns are distinguished into three line categories: the code contribution into the project SCM (number of SVN/CVS commit), the Defect status changes (number of Bug status changes in the Bugzilla), and the developer email conversation (number of email submitted into the developer mailing list).

Our prior study [121] found that in the two successful projects (HTTPD and Tomcat) a positive trend line of developers' email contributions during project life time exists (see Figure 12). On the contrary, the two challenged projects (Xindice and Slide) reveal a dire situation as revealed by diminishing developer contributions.

In this section we look further the cause of the illness symptoms and what a healthy relation should look like between components in the developer contributions.

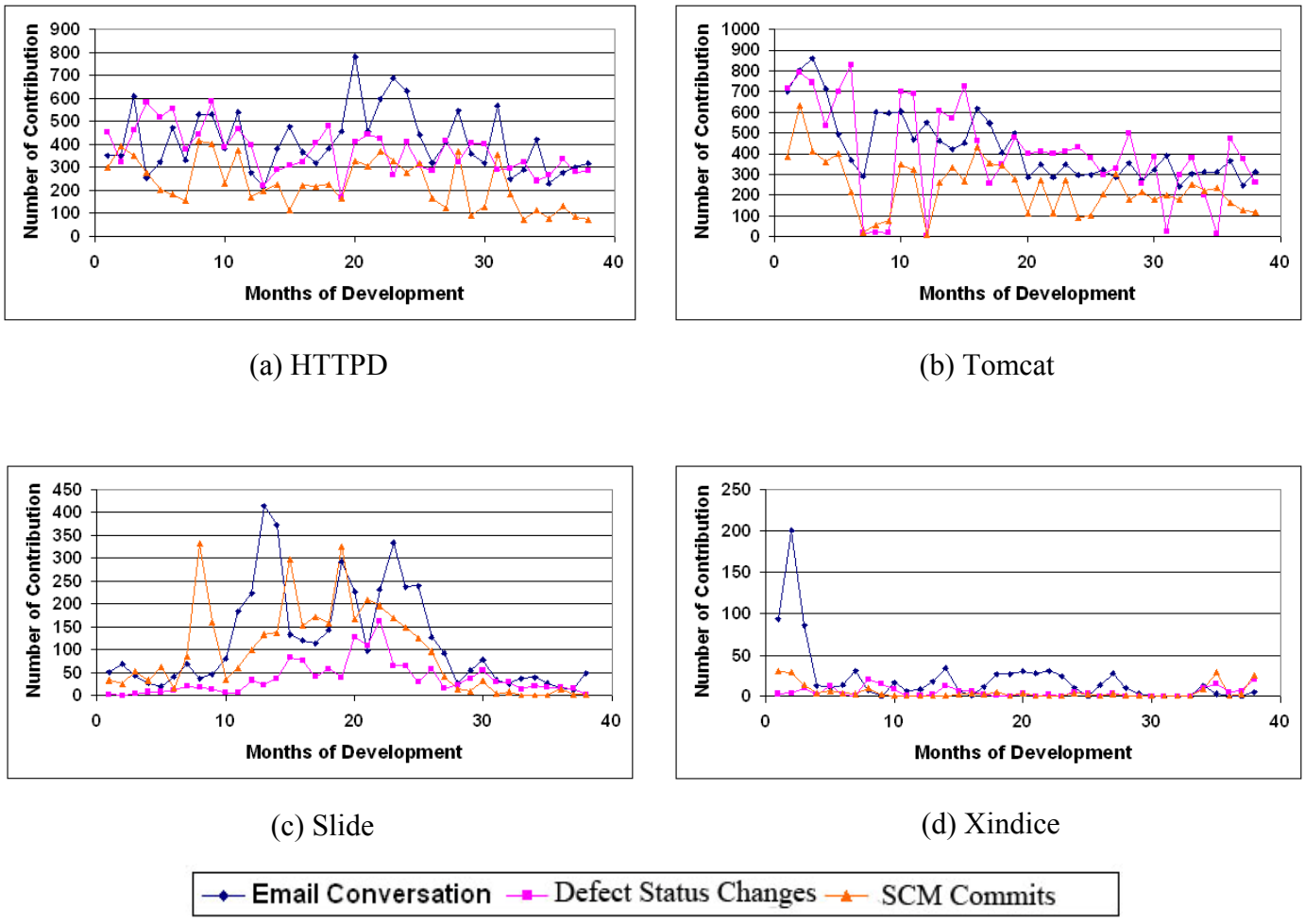


Figure 12 Absolute number of developer contributions in Four Apache Projects in 38 months of observation [122].

a.1) Developers' Contribution Patterns in the Four Reviewed Projects:

To better understand the developer contribution patterns within different types of project, we examine the retrieved data set, and calculate the ratio between code contribution (number of CVS commits), and reports of Defect status (number of Defect status change notification) with the developer email conversations.

We argue that in a healthy community the project should exhibit more *uniform* ratio among these metrics, i.e. every CVS commit ideally should be followed up by the developer discussion in the mailing list before inserted into the body of code. We found the both challenged projects exhibit more fluctuation and higher ratios as illustrated in Figure 13. It means, the developer community retrieved more notification of code contributions and Defect reports but responded less in the mailing list.

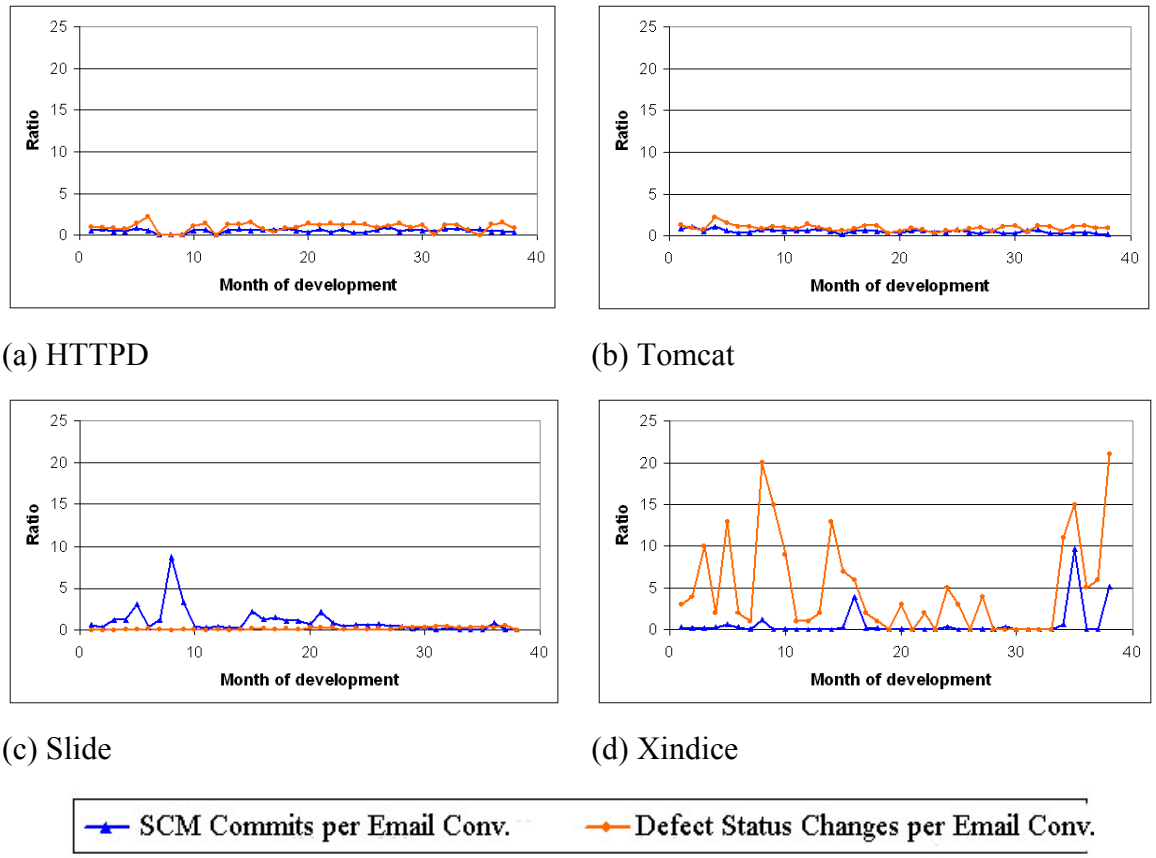


Figure 13 Developers' contribution patterns as proportion of different development metrics [122].

This situation may indicate illness symptoms. We considered some of the illnesses are the facts that the developer community pays less attention to project status' changes; the project employs small proportion of active developers which also signify developer de-motivation which further needs to be investigated by experts in OSS Community.

On the contrary, both the HTTPD and Tomcat signify more reasonable proportion in the ratio of developer contribution. This can be interpreted that most of the developer code contributions and changes of defect status may trigger some responses from the developer community.

In the healthy projects (Figure 13 (a) and (b)) the number of commits contributions per defect status changes and the ratio between numbers of email contributions per defect status changes tend to be normally distributed. On the contrary, the challenged projects (see Figure 13 (c) and (d)) show fluctuations which signify the imbalance between code contributions/developer email submissions and Defect status reports/developer email submissions.

a.2) Developers' Contribution Correlation Analysis

We analyzed the observation data from all projects and performed two types bivariate analysis to investigate the correlations of defect status changes, SCM commits to Email conversations. As shown in

Table 4 and Table 5 present that for HTTPD, Tomcat and Slide there are positive significant correlations between defect status changes, SCM commits with email conversations.

Table 4 Development Metrics Correlation Analysis in Four Apache Projects (Pearson Rank)

Pearson Rank Correlation Analysis		Email Conversations		
		Correlation	p-value	N
Defect Status Changes	HTTPD	0.679**	0.000	34
	TOMCAT	0.686**	0.000	34
	XINDICE	-0.41	0.808	34
	SLIDE	0.661**	0.000	34
SCM Commits	HTTPD	0.630**	0.000	34
	TOMCAT	0.777**	0.000	34
	XINDICE	0.576**	0.000	34
	SLIDE	0.856**	0.000	34

Table 5 Development Metrics Correlation Analysis in Four Apache Projects (Spearman Rho)

Spearman Rho Correlation Analysis		Email Conversations		
		Correlation	p-value	N
Defect Status Changes	HTTPD	0.741**	0.000	34
	TOMCAT	0.606**	0.000	34
	XINDICE	0.009	0.958	34
	SLIDE	0.505**	0.000	34
SCM Commits	HTTPD	0.582**	0.000	34
	TOMCAT	0.736**	0.000	34
	XINDICE	0.239	0.148	34
	SLIDE	0.905**	0.000	34

** Correlation is significant at the 0.01 level (2-tailed).

Statistically we can assume that an increase in defect status changes as well as SCM commits will significantly increase the number of emails sent by the developers into the developer mailing list. However in dying project such as Xindice there is no significant correlation with

email contributions, means the developer community has no customs in responding a code submission or changes of defect status collectively or simply the project has been abandoned by the majority of the developers.

As outlined in some statistical literatures [33, 88], a bivariate correlation analysis can signify the impact of one variables to another, however, we still need to assure the direction of an impact, for example, a correlation analysis results that x has a positive significant correlation with y, we can interpret this result as by increasing x will increase the value of y, however statistically we can also say that the impact may come in different direction where by increasing value y may likely increase the value of x (*another potential threat to construct validity*). Of course, one can reason based on his own knowledge to interpret a relation between variables, nevertheless we can employ more formal methods to check the variables relationship, such as by fit in the variables into a prediction model. If the constructed model tested as significant, then we can conclude that these variables have correlations and impacts to the dependent variable in the model.

a.3) Email conversation as a function of SCM commits and Defect Status Changes

As described above, a successful project community exhibits more normal distribution of developer activities metrics during the development process. The next step, we want to find out whether we can create a prediction model based on the correlations among these metrics.

For the purpose, we performed a multiple linear regression analysis with 34 months data from all projects. The results are shown in

Table 6 signify that in healthy projects such as HTTPD and Tomcat have higher linear correlation (note higher value of R^2) between the independent variables SCM Commits (X_1), Defect status changes (X_2) with dependent variable Developer Email Contributions (Y).

Table 6 Prediction Models for Four Projects using the First Group of Observation Data

		Model Coefficients			**Model Test Results**			
		Constant	X_1	X_2	R^2	F-test	P-value	N
Project	HTTPD	42.267	0.583	0.637	0.590	28.205	0.000	34
	Tomcat	62.853	0.323	0.862	0.669	35.382	0.000	34
	Slide	54.599	1.609	1.328	0.524	27.382	0.000	34
	Xindice	17.133	-2.623	3.270	0.338	5.408	0.000	34

We tested the regression model to assess its internal validity with confidence interval is 95%,

where F_{table} or $F_{0.05,34}$= 4.11, here for all constructed models we can conclude their significance as all of them has F-test value higher than the table value of $F_{0.05,34}$ and p-value < 0.001 level.

The above results conceal that in a healthy project (such as HTTPD or Tomcat) the code contribution has significant impact to amplify the number of email conversations in the developer mailing list such that the ratio between these two variables are kept in proportional during the development process which signifies one healthy status. Please refer to Appendix A1 for detail of the data analysis results.

To have external validation i.e. to check model robustness with different observation data, later the prediction models were fitted to the second group of observation data. Here we used the average absolute error (AAE) and average relative error (ARE) to evaluate the models performance.

In Table 7 outlines the result of prediction model validation for each project using the last 4 months data of observation which were not used during the model training process.

Table 7 Validation of Prediction Model using the Second Group of Observation Data

Project	Mean (AAE)	StDev (AAE)	Mean (ARE)	StdDev (ARE)	N
HTTPD	20.373	9.656	0.061	0.026	4
Tomcat	100.572	52.229	0.124	0.059	4
Slide	51.385	31.520	2.460	2.940	4
Xindice	91.533	31.531	13.239	15.299	4

We can see that ARE values for Apache HTTPD and Apache Tomcat are lower than Slide and Xindice, additionally the prediction model of Xindice suffers the lowest accuracy. Performance of prediction model of HTTPD and Tomcat indicate that the trained model can hold new observation data which is necessary in predicting the future of project "health status".

B. Defect Service Delay

Based on the measurement scenario on defect service delay as mentioned in Section 4, we present these following quantitative results. The first measurement is to see the distribution of the Defects' severity on each project.

In the Apache projects, the defects are categorized based on their severities related to security (critical) and fault (blocker), related to feature (major, minor, enhancement, normal) and related to cosmetic works (regression and trivial). Later, for further processing, the community put the development priority (P1 to P5) for each Defect, where the P1 means the top priority and needs

to be resolved as soon as possible. We retrieved the defect data from Apache Tomcat 5 (2891 Defects), HTTPD 2 (3663 Defects), Xindice (152 Defects) and Slide (420 Defects). On the data, we examine the distribution of the Defects based on severity and priority to find the proportion of defect assignment.

Table 8 Defect Distributions in Two Healthy Apache Projects

- Tomcat

		PRIORITY (%)				
		P1	P2	P3	P4	P5
SEVERITY (%)	blocker	0.7	0.8	2.1	0.0	0.0
	critical	1.4	2.0	4.1	0.1	0.1
	enhancement	0.1	5.3	5.6	0.2	0.7
	major	0.8	5.7	9.7	0.1	0.1
	minor	0.1	1.8	3.8	0.4	0.3
	normal	0.7	26.7	24.4	0.5	0.3
	regression	0.0	0.2	0.0	0.0	0.0
	trivial	0.0	0.7	0.1	0.2	0.5

(b) HTTPD

		PRIORITY (%)				
		P1	P2	P3	P4	P5
SEVERITY (%)	Blocker	0.7	0.9	4.3	0.1	0.1
	critical	1.2	1.9	6.8	0.1	0.1
	enhancement	0.1	2.7	6.5	0.1	0.5
	major	1.0	4.3	9.3	0.0	0.0
	minor	0.0	1.7	6.6	0.2	0.5
	normal	0.9	17.3	30.3	0.2	0.2
	regression	0.0	0.5	0.0	0.0	0.0
	trivial	0.1	0.5	0.1	0.1	0.2

Table 9 Defect Distributions in Two Challenged Apache Projects

(c) Slide

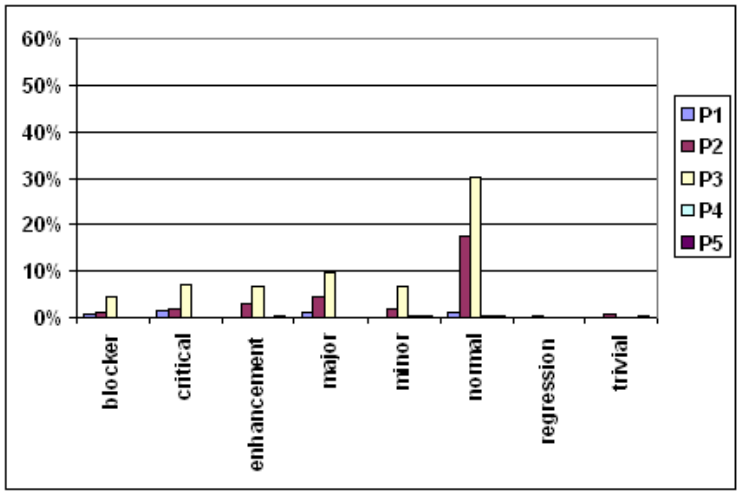
		PRIORITY (%)				
		P1	P2	P3	P4	P5
SEVERITY (%)	Blocker	0.5	0.2	1.7	0.0	0.0
	Critical	1.0	2.6	4.5	0.0	0.0
	Enhancement	0.0	1.9	7.4	0.0	0.2
	Major	1.2	5.7	5.5	0.0	0.0
	Minor	0.0	0.5	6.9	0.2	0.0
	normal	0.5	17.6	41.7	0.0	0.0
	regression	0.0	0.2	0.0	0.0	0.0
	trivial	0.5	0.2	1.7	0.0	0.0

(b) Xindice

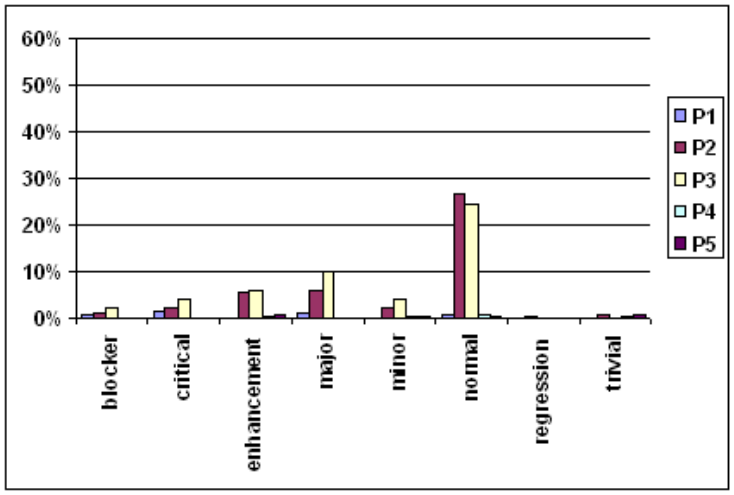
		PRIORITY (%)				
		P1	P2	P3	P4	P5
SEVERITY (%)	blocker	0.7	1.3	3.3	0.0	0.0
	critical	0.7	0.7	7.2	0.0	0.0
	enhancement	0.0	0.7	5.9	0.0	0.7
	major	0.7	1.3	14.5	0.0	0.0
	minor	0.0	0.7	4.6	0.0	0.0
	normal	0.0	7.2	48.0	0.7	0.0
	regression	0.0	0.7	0.7	0.0	0.0
	trivial	0.7	1.3	3.3	0.0	0.0

We also observed a situation that the distribution of the defects based on severity is almost ”normal” in all observed projects, as illustrated in Table 8 as well as in Table 9 (see the figures in italic bold), in the sense that most of the defect reports coming from the user community are feature requests, functionality errors or decorative ones.

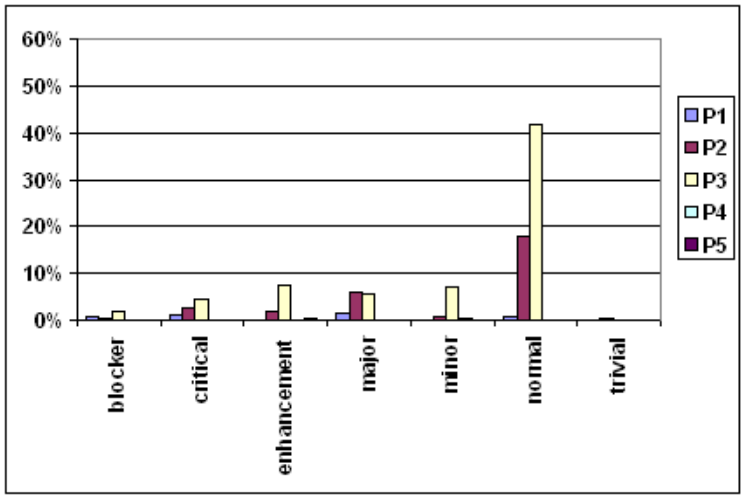
Furthermore, from the tables we can see that although the user community reports defects and considers the defects of high severity (such as “blocker”), the priority assignment by developer does not always follow the “user needs”. In other words, the developer does not always assign high severity (according to user) with high priority (see Figure 14).



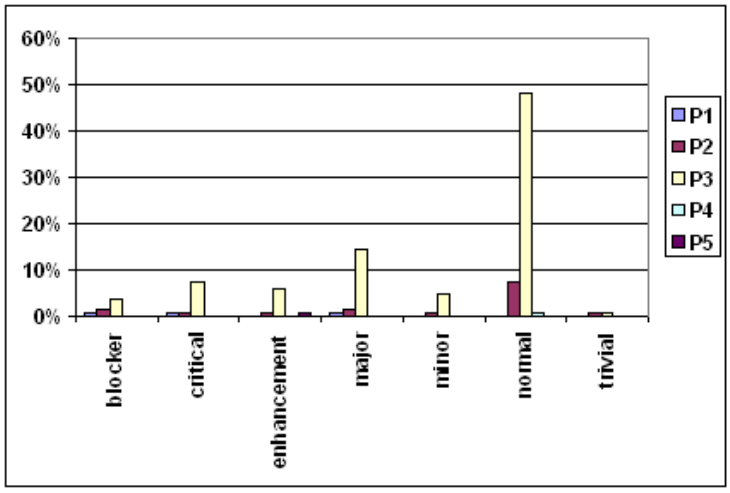
(a)HTTPD defects distribution per priority



(b)Tomcat defect distribution per priority



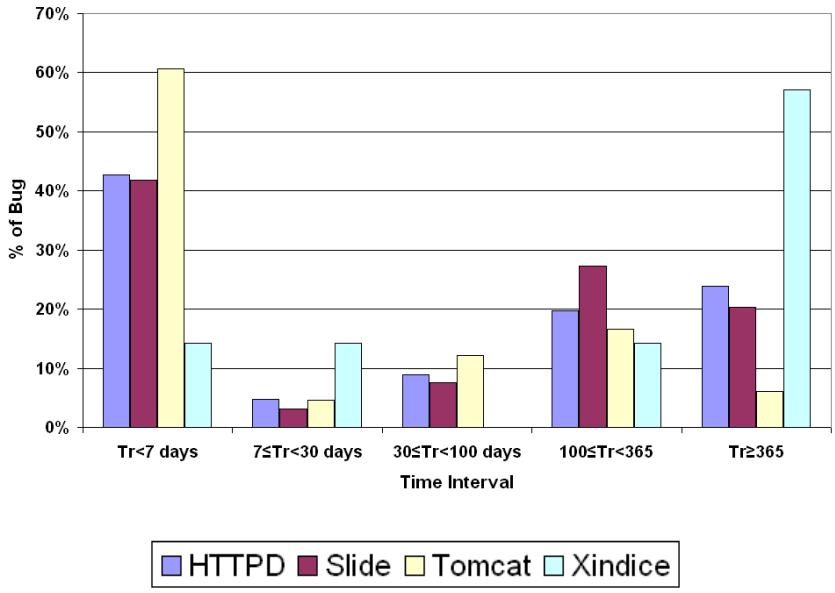
(c) Slide defects distribution per priority



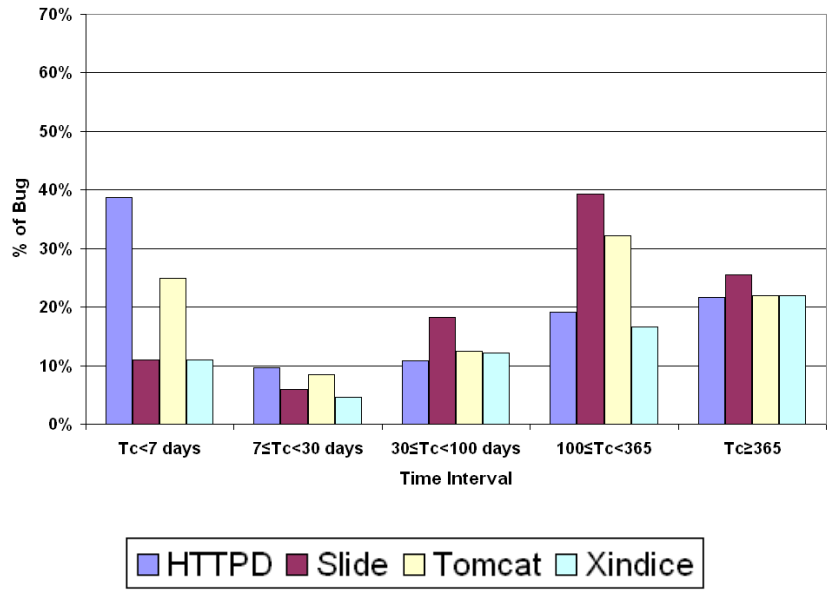
(d) Xindice defects distribution per priority

Figure 14 Defect Distributions in Four Apache Projects

To measure the performance of defect service delay in a project, we measure the defect response time (*time to respond*) and the defect closure time (*time to resolve*). We categorized such service delay into several time interval scales as shown in Figure 15.



(a) Defect response time



(b) Defect closure time

Figure 15 Defect Service Time distribution for reviewed projects. [122]

Figure 15 shows the distribution of defect response time (Tr) and defect closure time (Tc) from the four reviewed projects. From the figure it is obvious that Tomcat has the most responsive community, as 60% of the reported defects are responded less than 7 days, 79% are responded within 100 days. On the other hand, Xindice exhibits poor performance as the majority of the Defect reports (72%) were responded by the community in more than 100 days.

Our result also found intriguing fact that more than 20% of the defects were resolved more than a year in HTTPD and Tomcat. We investigated this issue by measuring the absolute number of the

resolved Defects and correlated with the assigned priority.

For HTTPD, among the 999 resolved defects, 24% are latent since they were resolved more than a year; 88% of these latent Defects are categorized as lower priority (P3, P4, P5) and 45% are considered as cosmetic work or minor feature error. For Tomcat, from 2063 resolved defects, 27% are latent defects with only 9% of top priority Defects (P1 and P2), and less than 15% are severe defects (critical and blocker). Thus, we conclude that in both successful projects, the community offers faster response to a defect with higher priority, and tends to delay the less important ones.

3.5.4 Discussion of Empirical Results

For many OSS developers, challenge is what really motivates them and it makes the project more active. Once they are drawn to a problem, they feel that they could create a better solution themselves, rather than using the existing ones. Over the time, the projects may evolve and so does the motivation of the developer.

Developer Contribution Patterns. Our study in OSS project community health revealed two extreme trend lines in the developer participation: (1) lively developers' activity as shown by Tomcat and HTTPD, (2) non responsive or dying developers' community, which we observed in particular Apache Xindice.

The developer community in the challenged projects shows less uniform proportion of developers' contribution metrics (such as code submissions, Defect status changes and email conversations). This condition is considered to signify situation in which the community is less responsive to the status changes within the project. Our empirical study results show that for Apache Tomcat, HTTPD and Slide, development metrics such as SCM commits and defect status changes have significant correlations to the number of email submitted in the developer mailing list. We can imply that statistically for each contribution by a developer whether in form of code submission, or defect patches will likely be responded by other developers in the community through discussion in the mailing list. We concluded that there is a development metrics (x) that has significant correlation with developer email contribution (y),$\{\exists\, x \in pj | r\_i\ (x\_i, y) = True\}$, hence we can reject our null hypothesis **H04.5.1a**

Xindice obviously reveals a very fluctuating proportion of developer contribution (see Figure 16) which means that there are significant imbalances between the contribution and the response from the developers' community. Furthermore based on the absolute number of the developers' contribution, Xindice indicates a"dying" project, which can be seen in Figure 16. In its last 33

months, the developers' contribution levels were low compared to its first 5 months under observation. When we discussed with some experts, according to them, the reasons behind the condition are: (a) *Xindice has been abandoned by its key developer*, which also proves that diversity of core developers is important (b) *the changes of market demand and technology trend* caused the developers' community to recognize that proposed plan and the results not to evolve expected.

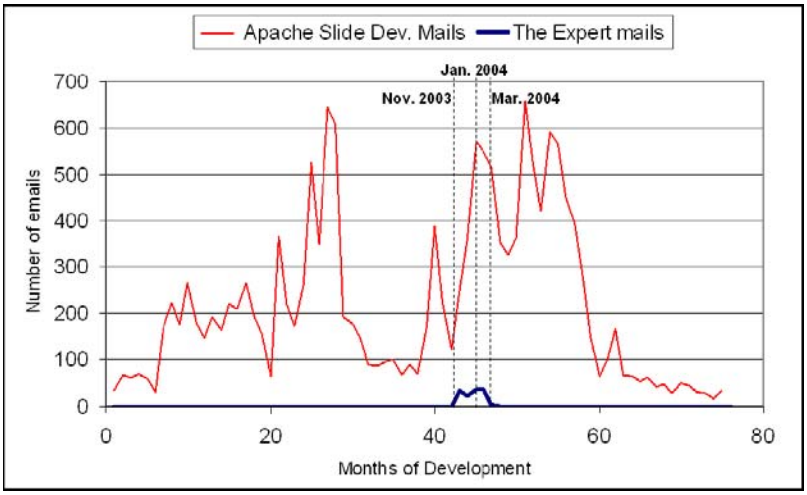


Figure 16. Impact of a Core Committer Contribution which Motivate Other Developers Contributions into the Developer Mailing List in a challenged project Apache Slide [121]

Slide has a different story as shown in Figure 16. The project exhibits proportional developer contribution, however based on the result of measuring the absolute number, on the beginning Slide was very promising, but since October 2004 the project was hit by catastrophic illness indicated by the decrement of its developer contributions.

The experts participating in the Slide project mentioned that Slide was once in a dormant state. The project woke up after a talented expert got involved in November 2003, and significantly contributed for the peak performance of Slide.

However after several months, he decided to leave the project, leading to the collapse of Slide. Our prior study on this case [121] validated this statement by measuring this experts' mail conversation and suggest the fluctuation in developers' mail follows the dynamics in "The Expert" mails.

Around March 2004, as the expert decided to leave the project, Slide developers' mailing list still showed notable activity for several months, before their number finally collapsed. Our observation in February 2007 disclosed although there are still some developer activities, the

level is relatively low compared to the zenith period when ”the Expert” was still actively involved in the project.

We performed prediction process based on collected development metrics. Prediction model is necessary to have better understanding of impact factors for developer contribution pattern (email contributions as a liner function of code submission into the SCM and defect removal activities reported in issue tracker).

As shown in

Table 6, all trained prediction models are significant (with p-value < 0.001) however when we extrapolated the prediction with data from the latest 4 months of observation of each projects, we found that models constructed for the healthy projects Apache Tomcat and Apache HTTPD outperformed the accuracy of models for Apache Xindice and Apache Slide, since we can state that ARE(ɵ(Ph)) < ARE(ɵ(Pc)), where {Tomcat, HTTPD} ∈ Ph and {Xindice, Slide}∈ Pc, so then we can reject the proposed null hypothesis **H04.5.1.b.**

Defect Response Time and Defect Service Delay. Our study also reveals that a healthy community will be more responsive and more eager to resolve issues or defects introduced to them. Tomcat and HTTPD outperformed Slide and Xindice, as the largest portion of defects reported in were responded less than a week (Tomcat=70%; HTTPD=42%; Slide=40.5% and Xindice 12.4%) which state that a healthy project signify by faster reponse time compare to challenged ones, *Responsiveness* (Tomcat, HTTPD) > *Responsiveness* (Slide, Xindice).

Hence, we can reject our null hypothesis **H04.5.2.a.** The successful projects also provide faster service as their defect closure times are shorter compared to those of Xindice and Slide.

The results show that about 46% of the defects in Tomcat and 59 % of the defects in HTTPD are resolved within 100 days, while most of the defects reported for the challenged projects are resolved within or more than a year or *Service Delay* (Tomcat, HTTPD) > *Service Delay* (Slide, Xindice), therefore **H04.5.2.b** can be rejected.

However the empirical result of defect severity distributions argued that the more severe defect does not always mean to be of higher priority, since most of the top priority defects are the normal ones as illustrated in Figure 15.

It is likely that the response time is affected also by priority set by developers due to some consideration. For example, a blocker might be given lower priority when it occurs very rarely and it is planned not to be resolved in the current release. This practice exhibits value tradeoffs between the users who report the defect, assign the defect severity and expecting quick response, with the community who assess, assign, develop and resolve the defect.

3.6 Finding "Health" Indicators from Aspects of Quality Assurance in OSS Projects

In this section we seek to make deeper exploration of aspects of quality assurances (QAs) in an open source software project. We identify several QA activities that commonly performed by a healthy OSS project community such as defect reporting, defect validation, defect removal, code testing, integration testing, code peer-review, etc.

This research is based on an argumentation that such exploration is important for the OSS project stakeholders in order to achieve better quality OSS products, and as basis for future work in process and product improvement of similar development style.

Identification and measuring quality assurance aspects in OSS project is necessary to assess current quality of OSS development process and product, which can be viewed as indication of project "health status".

3.6.1 Quality Assurance Aspects in OSS Project

Here, we propose a framework of quality assurance (QA) aspects in OSS project as an extension of stakeholder roles and activities as shown in Figure 17.

This subsection described the detail of our proposed framework of QA aspect in OSS project based on typical processes performed by different roles of project participant during defect life cycle. In this work we define defect as an error, flaw, mistake, failure, or fault in software that prevents it from behaving as intended.

A defect reported by user (or developer) at user community; however its existence should be proven and validated as defect before further processed by the developer community as illustrated in Figure 17. This framework is derived from continuous product and process improvement in OSS projects as described previously in Section 2.2.6.

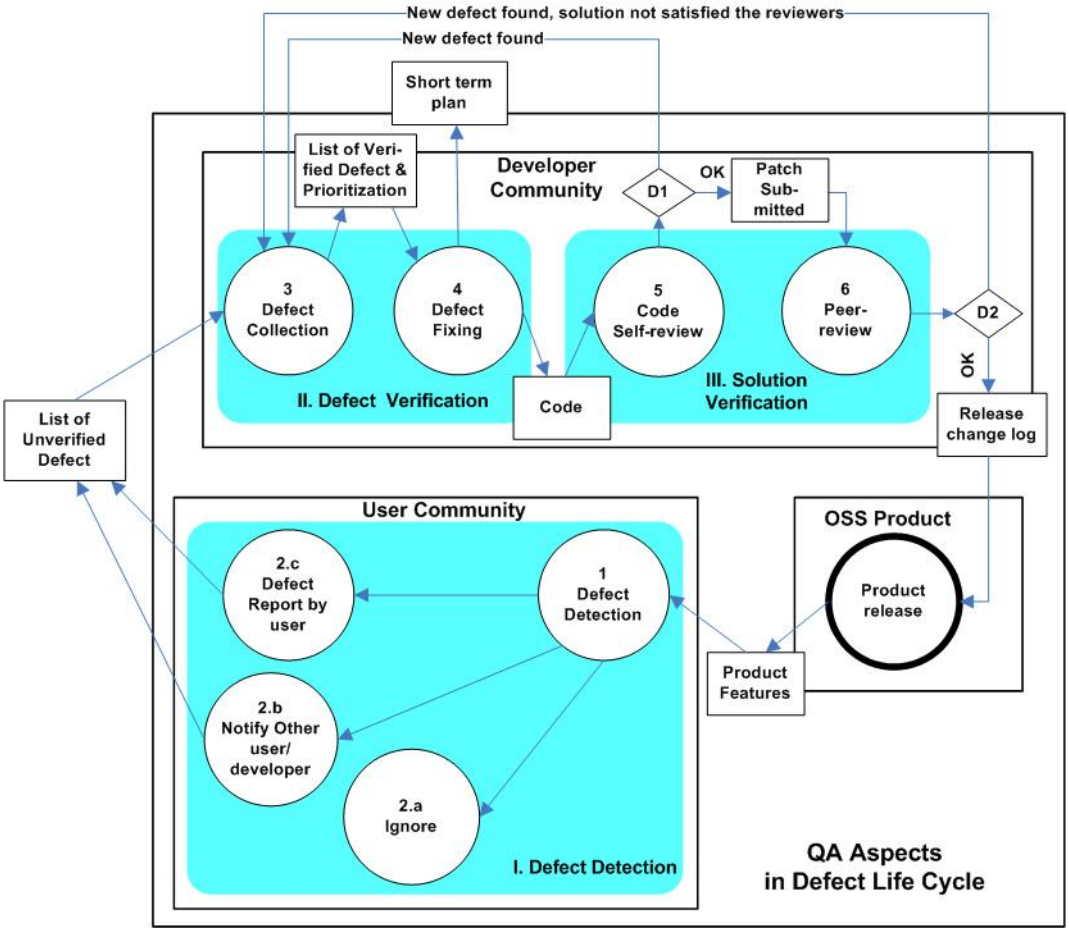


Figure 17 Framework for quality assurance processes as part of defect removal activities in an OSS project [124].

Process Group I: Defect Detection

The detection and reporting process, provide information of a defect existence, and sometime accompanies with early assessment of a defect, accompanying with early defect analysis. The results of process group I is an unverified defect list which will be further examined by the developer community. In detail process group I consist of following activities.

***Process 1*:** Defect detection, in open source is more like *black box* testing, as the user (common user or developer) have been using particular features of software release and spotted a defect, error or failure.

***Process 2(a,b,c)*:** reporting a defect into the project issue tracker heavily relies on the motivation of the user, as he may just (2a) ignore the defect and continue using the software, or (2c) he fills the issue into the tracker, accompanied with defect summary, defect description and other information needed by the tracker.

However as most of the common users are partially unknown to the project hence most common

practices in defect reporting are to (2b) notify other users or developers about the finding and ask for their opinion in the mailing list or forum. A more experience developer who noticed the issue then perform some early analysis, locates the defect and roughly estimates its effect(severity) if the defect is valid then he fills a report in, otherwise he notifies the community to ignore as it is a false defect.

Process Group II: Defect Verification

The defect verification consists of defect collection and defect fixing/correction. Both processes are very important and similar to the same named processes in commercial software inspection. The objective of group II is to validate the existence of a defect as defect of particular software release, and later perform necessary actions to correct the defect.

***Process 3*:** Defect Collection. The defect collection begins after a defect report listed into the tracker. The process is similar to *white box* testing, here defects are first stated as "open", since it is unverified and there is no action has been taken yet. One or more developer may read the report, add some comments and ask for more information from the reporter.

Later in order to validate defect existence, he needs to reproduce the defect, and then analyze defect location and its effect (severity) to the software product. Once a defect successfully reproduced and analyzed, the developer may confirm the defect existence (defect stated as „new"), the defect already reported by others ("duplicate") or false defect ("invalid") which should be ignored. The expected result from sub process 3 is to have a list of verified defect with enriched description and specification.

***Process 4*:** Defect Fixing is a set of activities to correct a defect. A developer who has interest in a verified defect may take "ownership" of the defect, create a short term plan which announce that he is working on a particular defect and set of code files and expecting other developers to avoid them or attempt to synchronize their changes.

If succeed, the process may deliver a set of code which should be first self reviewed later to be submitted to the mailing list, or tracker further review by other developers or committers.

Process Group III: Defect Solution Verification

Process 5: **Patches self-review.** During defect solution development an assigned developer produces a set of code locally and submits the results. It is worth noting that in OSS project it is very unlikely for a patch to be submitted without first being self-reviewed in order to evaluate its technical content and quality.

Eventually after self review the developer may decide (see decision: D1) to submit the results as

patch into the developer community (e.g through tracker and mailing list) or it is possible that the development of a solution will impose some new defects in the code, which should be reported as a feedback input for process 3.

Process 6: **Patches Peer-review.** Almost every code contribution, patch or commit is cross-checked by attentive people (e.g. other developers or committers) in the developer community. In a large project such as Apache HTTP Server, peer review performed not simply to assess the quality of contributed code but also applied for a new idea/solutions submitted to the developer community which need to be discussed before put into plan for development.

Later reviewed patch may be added to the body of code or written as change into release change log by the committers, however if the patches could not satisfied the majority reviewers for several reasons such as the patch does not meet the defect specification or some new defects were found, in these cases most likely the defect will be returned to the issue tracker and stated as "re-open" or "new".

3.6.2 Proposed Health Indicators Derived from QA Activities in OSS Projects

Following the health indicator construction described in section 3.4, we need to identify the expected quality aspects and investigate how to measure those using metrics that can be obtained easily from different QA activities in OSS projects.

In this study I proposed three new health indicators which are defect detection frequency, defect collection effectiveness and proportion of verified solution. I also incorporated defect closure time which already evaluated in previous section to have more comprehensive indication of QA activities in OSS projects.

1. **Defect Detection Frequency**: A defect detection frequency signifies average number of defect report filled into the issue tracker by group of reporter in certain time (monthly).
2. **Defect Collection Effectiveness**: Effectiveness is probability of valid defects against overall reported defects into the tracker during case study period.
3. Defect Closure Time: defect closure time is similar to service delay in commercial project. We expect hybrid projects should perform slower closure time of defect solution compare to pure project due their rigid documentation, guidelines and defect resolution policies.
4. **Proportion of Verified Solution**: verification after a defect has a positive resolution (e.g. patches) is important to make sure the quality of the solution meet the specification and not endangered current body of code of the software.

3.6.3 Design of Empirical Study

The design of study comprises of goal definition, brief description of study objects, variables specifications and hypotheses construction.

1. Goals

From project manager point of view, QA is very important in order to produce high quality software which satisfy user needs, and make sure current QA performance meet stakeholder win condition as depicted in section 3.3. One focus of our research is shaping performance indicators to observe QA aspect status in a pure (Tomcat) and hybrid (MyFaces) OSS project.

Therefore, from OSS project and quality manager point of view, the part of analysis has two goals: (a) to add further evidence to the validity of QA performance differences between diverse types of project, (b) to empirically evaluates the QA performance indicators that are directly applicable in every OSS project without specific expert know-how.

In general we expect QA performance in pure voluntarily project will be overall less effective than hybrid project, as in hybrid project most likely have better guidelines for QA, more rigid specification and documentation, as compulsory from their sponsor.

2. Study Objects

These projects are considered as large projects as they employ more than 20 committers and more than 50 developers and already have more than one major release when the study was conducted. Both projects were different in senses of their sponsorship, as Tomcat[24] is pure OSS project supported by volunteers, while MyFaces[25] is partly sponsored by commercial organization. The case study objects are two major releases of Apache Tomcat and two projects in MyFaces community.

The first project Apache Tomcat is a network server (system) application with a very large and diverse community background. Tomcat is pure volunteer work with 4 major releases; in this work we investigate Tomcat 5 and Tomcat 6 as the older releases (version 3 and 4) have already been abandoned by the community.

The second project is Apache MyFaces, an application considered as web framework (Internet application), the project employs more homogenous participants compared to Tomcat. Apache MyFaces consists of 4 subprojects; 3 of them (Tobago, Trinidad, and Tomahawk) are extended components that offer more functionality and flexibility than using standard Core components.

[24] Apache Tomcat Project can be found at: http://tomcat.apache.org/ (last accessed 20 July 2008)

[25] Apache MyFaces Projects can be found at: http://myfaces.apache.org/ (last accessed 20 July 2008)

Project Trinidad is a donation from Oracle to ASF, while Tobago is a hybrid project as some developers are paid and closely supported by commercial organizations.

3. Variables Specifications

The types of variables defined for the experiment are independent and dependent variables. The independent variable is the type of project (either pure voluntarily or hybrid).

The dependent variables capture the QA performance in different project type. Following standard practice in empirical studies we focus on time variables and performance measures.

(a) Defect closure time, which we defined as time spent on defect stated as "open" until the same defect stated as "resolved" in the tracker. The formula of defect closure time is presented in section 3. This indicator represent how well the developer community in responding and provide solution for each reported defect of particular software product.

(b) Defect detection frequency, defined as how many defect reported (*Dr*) by how many reporter (*Rr*) over the time into the tracker, this indicator shows how active the user community in reporting defect, which can reflect the usage of particular release

$$Defect\_Detection\_Frequency = \frac{Dr}{Rr} \qquad Eq.\ 6$$

(c) Defect collection effectiveness, we defined as: ratio between number of valid defect (*VD*) after reviewed by some developers per number of defect reported (*Dr*) by users in certain time, this indicator tells us the how many false alarm that reported by the user community which in some level can be annoying from the developer point of view.

$$Defect\_Collection\_Effectiveness = \frac{VD}{Dr} \qquad Eq.\ 7$$

(d) Proportion of verified solution, defined as ratio between defects resolved with resolution closed (*CD*) per number of defect resolved with resolution fixed (*RD*), this indicator signify the willingness of the community to resolve defect properly (e.g. by peer review every solution) before a release.

$$Proportion\_of\_verified\_solution = \frac{CD}{RD} \qquad Eq.\ 8$$

In a while to better understand the QA emphasize on defect severity, we also classified collected defects into three class of severity based on Bugzilla documentation which are Class 1 is the highest priority which related to security (critical) and fault (blocker); Class 2 those which

related to feature (major, minor, enhancement, normal) and; Class 3 are those related to cosmetics work (regression and trivial). Defect severity is typically set by the developer who reviews the defect into the tracker, hence we use defect severity to draw the red line between developer value expectations with evaluated QA performance indicators.

4. Research Questions and Research Hypotheses Formulation

In the case study we will evaluate the following research hypotheses:

RI4.6.1 Defect Detection Frequency: We expect in much larger and heterogeneous community such as Tomcat, has higher number of defect detection activities than MyFaces. Hence we replicated a negative hypothesis as:

H04.6.1: *Defect detection frequency (Tomcat) ≤ Defect detection frequency (MyFaces)*

RI4.6.2 Defect Collection Effectiveness: We expect higher defect collection effectiveness in MyFaces, as a hybrid project should have more documentation to prevent invalid defect report and the reporter may have deeper knowledge of the project compare to pure OSS project. Hence our replicated hypothesis is

H04.6.2: *Defect collection effectiveness (Tomcat) ≥ Defect collection effectiveness (MyFaces)*

RI4.6.3 Defect Closure Time: We expect hybrid projects should perform slower closure time of defect solution compare to pure project due their rigid documentation, guidelines and defect resolution policies. We proposed a negative hypothesis

H04.6.3: *Defect closure time (Tomcat) ≤ Defect closure time (MyFaces)*

RI4.6.4 Proportion of Verified Solution: We expect in less formal project environment such as Tomcat, a defect resolution will likely to be resolved faster but less frequent to be peer-reviewed (defect stated as ”closed”) compare to MyFaces. Thus we proposed following negative hypothesis:

H04.6.4: *Proportion of verified solution (Tomcat) ≥ Proportion of verified solution (MyFaces).*

3.6.4 Data Collection

This section describes the proceeding of data collection, the data refinement actions, and potential Threats to Validity.

1. Data Collection and Data Refinement

The case study was performed in March to April 2007. First we designed to be conducted empirical study which consists of case study goals definition, study variables definition, and derived research questions and following hypotheses to be evaluated by collected data.

In this work we examined both projects during their last 5 months of development (1/10/2006 to 1/02/2007). We retrieved SVN logs from each project's defect database.

We classified defects into three classes of severity derived from Bugzilla bug classification. We calculated the proposed performance indicator based on retrieved data. We use descriptive statistic analysis to compare the QA performance of both projects and discuss the results to answer our second research question.

For data refinement, similar to previous study reported in Section 3.5.2 , in this study we also removed several missing, invalid and duplicate defect data. Since the number of such missing data is quite very low (less than 2 %) compared to the number of valid data collected, hence we believe that missing data removal is reasonable in this context.

2. Threats to Validity

Construct validity. The first threat is that in Apache communities it is a common practice in developer community to re-open resolved defect in order to increase the quality of the patches. In this study we hold our assumption that to re-open a resolved defect is also part of defect removal activities. Therefore for defect closure time calculation we ignore the re-open status of a defect which of course may significantly increase the closure time.

External validity. The quality assurance aspects presented in this study is derived from QA practices in Apache project communities, thus we expect the results can be applied for projects under Apache Software Foundation or at least those which have similar characteristic to Apache projects.

3.6.5 Data Analysis Results

In this section we present empirical result and evaluate the research questions. The comparison of results from different reviewed projects is also included in following discussion.

Defect Detection Frequency

Table 10 displays average and standard deviation of monthly effort in 6 reviewed major releases. Tomcat 5 has the highest average number of defect report and reporter, which signify the project has larger reporter community. The ratio of each project exhibits that most of the time there are more than one defect report filled by a single reporter. The table shows that based on mean of reported defects and number of reporter, Tomcat 5 outsized all other project releases, which means the project has more active and heterogeneous reporter community.

Table 10 Defect Detection Frequency

		Major Releases					
Defect Severity		Tobago	Trinidad	Tomahawk	Core	Tomcat5	Tomcat6
Mean	reported defects	15.20	20.80	23.80	13.00	31.80	8.20
	active reporter	7.40	13.00	19.60	11.60	27.40	6.20
Stdev	reported defects	4.82	5.12	5.26	5.43	7.12	4.09
	active reporter	1.82	2.45	4.98	5.13	6.95	2.28

Defect Collection Effectiveness

All projects in this study show low level of report invalidity, as the majority of reported defects had been validated and listed as positive defect instead of false ones. As we expected in five months of observation all MyFaces releases has less invalid defect (closer to "1") which illustrated more effective defect collection compare to Tomcat 5 and Tomcat 6.

Table 11 Defect Collection Effectiveness

	Major Releases					
Defect Severity	Tobago	Trinidad	Tomahawk	Core	Tomcat5	Tomcat6
Class 1	1	0.75	0.9	0.63	0.2	0.33
Class 2	0.94	0.94	0.95	0.92	0.63	0.87
Class 3	0.67	1	1	1	0.82	1

Defect Closure Time

Table 12 illustrates the average of defect closure time (in days) for each major release, where "0" means a defect is resolved within the same day after the report filled in the tracker and "N/A" means there is no resolved defect in certain severity class.

In Table 12, in average all MyFaces subprojects need more time to solve class 1 defects compare to Tomcat releases. However the standard deviations in Tomcat releases show more diverge time

to closure a defect than in MyFaces, hence it will be more complicated for project manager in Tomcat to decide when a defect is delayed.

Table 12 Defect Closure Time per Class of Severity in Days

Defect Severity		Major Releases					
		Tobago	Trinidad	Tomahawk	Core	Tomcat5	Tomcat6
Mean	Class 1	62	34	11.17	43	4	0
	Class 2	45	13	9	39	22.37	4.5
	Class 3	33	2	29	N/A	13	N/A
Stdev	Class 1	24.79	40.8	16.44	57.38	7.98	0
	Class 2	27.39	26.82	76	47.48	11	9
	Class 3	39.47	0	32.99	N/A	28.72	N/A

Furthermore to better understand the defect closure time, instead of categorized defect into severity classes, we distributed defects into several class of closure time as illustrated in Figure 18

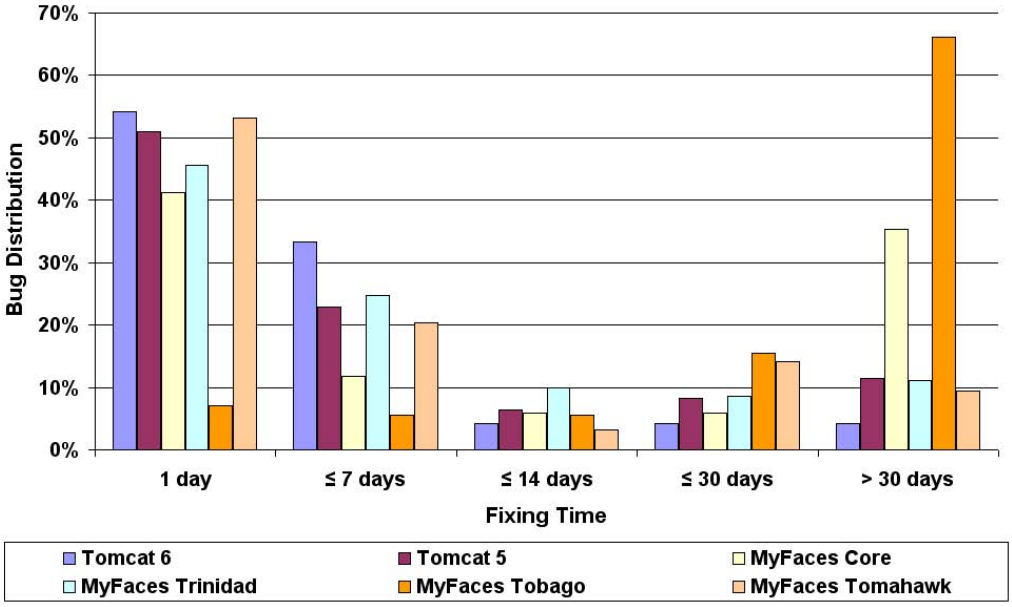


Figure 18 Defect closure time distributions in reviewed projects

In this figure MyFaces Tomahawk, and both Tomcat releases show higher performance in resolving defect, as the more than 50% of validated defects were fixed within one day. In contrary 66% defects listed in Tobago were fixed in more than 30 days which signify slower service time.

Proportion of Verified Solution

Figure 19 shows that in all projects some of reported defect have been fixed with particular resolution. Compare to both Tomcat releases, all four My Faces subprojects signify higher QA activities from assigned developer as more than 50% of resolved defect have been self reviewed

and tested (defect resolved as “fixed”).

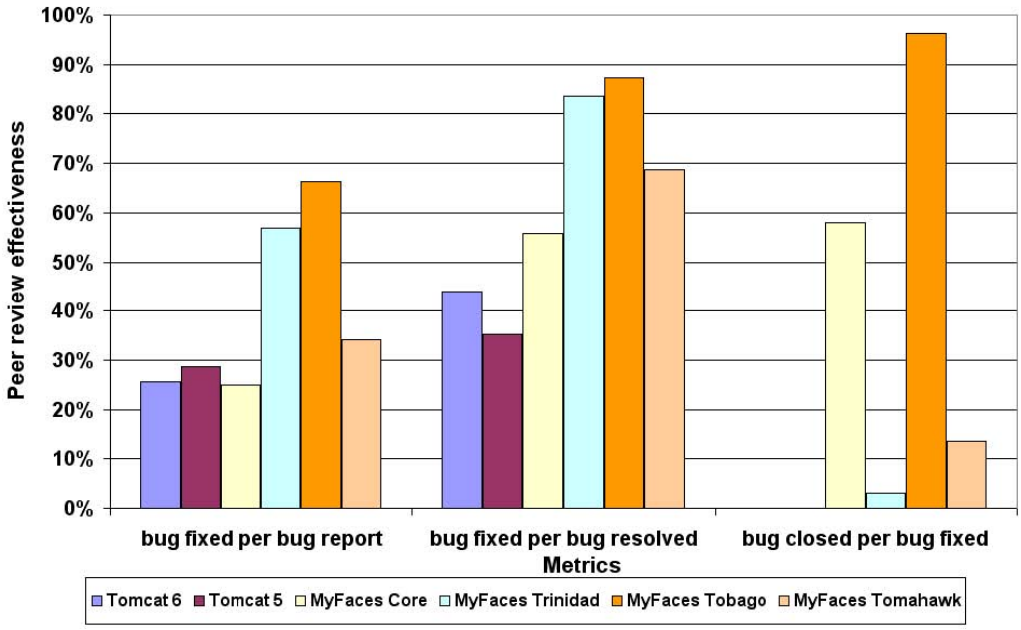


Figure 19 Proportion of verified defect resolution

Furthermore higher effectiveness of peer review illustrated by My Faces Core, and Tobago, as in both project, most of ”fixed” defect (≥ 50%) were also stated as ”Closed” means the defect has passed several QA processes by attentive developers or committers in the developer community.

3.6.6 Discussion of Empirical Results

In this section we summarize the empirical result from our case study concerning QA aspects in two Apache projects. Analyzing our empirical results, we derive following implications for performance measurement of QA processes in similar OSS projects.

Defect detection frequency. The result shows that in homogenous hybrid project community such as MyFaces obtain less number of defect reports over the time submitted by only particular people in the community compare to large and heterogeneous project such as Tomcat. In summary the data signify disagreement with our hypothesis ***H04.3***. Since,

Defect detection frequency (Tomcat) ≥ Defect detection frequency (MyFaces)

It is worth noting that in Tomcat, we also found that more project participants involved in Tomcat 5 defect collection activities rather than in Tomcat 6. The reason is in particular critical application such as web-server, instead of using the latest release (Tomcat 6), more users are still

using the previous version (Tomcat 5) for several considerations such as security, set-up overhead, etc.

Defect collection effectiveness, the results exhibit higher probability of invalid defect reports in Tomcat releases compare to MyFaces subprojects especially in defect class 1 and class 2. For these classes of defects we can reject hypothesis ***H04.4****., Since,*

$$\textit{Defect collection effectiveness (Tomcat)} \leq \textit{Defect collection effectiveness (MyFaces)}$$

Here we can expect in more formal/structured hybrid project such as MyFaces, the community is has more knowledge about the software releases, thus most of the time the defect reports are valid and should be taken into consideration by the developer community.

Defect closure time. In this work we define defect closure time as time to resolve a reported defect. The results show that majority of projects resolved defect in less than 30 days (See Figure 2) and signify a responsive developer community. In case of MyFaces Tobago exhibits more fixing time were needed, which rejects our hypothesis ***H04.5***. Since,

$$\textit{Defect closure time (Tomcat)} \geq \textit{Defect closure time (MyFaces)}$$

One possible reason is that most of the defect were also required to be properly peer-reviewed by other developers (Figure 18) which eventually took more time before a defect stated as ”closed” or ”verified”. Most of these delayed defects are in middle to lower severity classes (class 2 and class 3), which have less significant impact and tends to be delayed by the developers.

Proportion of Verified Solution. Code review at the end of defect life cycle consist of self-reviewed and team review. Due to limitation of investigation period, although in most of the projects we found practices of code self review (defect stated as “fixed”), however in both Tomcat releases we barely found any evidences of code team review, as all of fixed defects are only stated as ”resolved” instead of ”closed” or ”verified”. The result also enclosed in MyFaces Tobago, the ratio of closed defect per fixed defect is very high (96%), means most of the fixed defects had been peer-reviewed. Therefore we have to reject our replicated hypothesis ***H04.6.*** Since,

Proportion of verified solution (Tomcat) ≤ Proportion of verified solution (MyFaces).

Hence we assume the hybrid community is more responsive to each patch/code submission and highly aware about its quality. We asked expert in Apache foundation, it is probably due to complexity, maturity of its releases, and releases policy that the Tomcat's developers need to spend more time to review and verify a code or patch contribution which could not be captured within our case study time limitation. As the tradeoff of slower solution verification, Tomcat offers less re-open defects (stable solutions) compare to smaller OSS projects, however we still need to investigate this claim by providing empirical evidence, which we considered as future work.

3.7 Chapter Summary

In this chapter we deliver following research contributions:

Causal Model of OSS Projects Survivability. We proposed a model of OSS project life cycle and how the survivability of OSS project is depend on the aliveness of developer and user community and the quality of product releases.

"Health Indicators" for OSS Projects. Later we proposed a concept of "health indicators" of OSS project. "Health indicators" are ***quality evaluation*** measures of distributed development process in OSS projects, thus our proposed "health indicators" are the first quality indicator in our study context. From project management perspective to evaluate project "health indicators" in time is not only to obtain overview of project status but more importantly as early warnings of certain risks that have to be addressed to ensure the survivability of the project.

Some of these to-be-formulated indicators are "hidden" behind the development process. Hence, effectively, it is the core stakeholder who should make the decision about which indicators should be employed, based on the projects initial needs.

Empirical Evaluation of Proposed Health Indicators. On the second steps we performed empirical studies to evaluate the proposed health indicator concepts in Apache Projects. Our brief interview with an OSS expert considered that two of them as healthy projects (HTTPD and Tomcat) and two challenged projects (Slide and Xindice). The challenged projects were in dire situation, as Xindice for example shows a dying period since most of its developers left the project, while Slide shows early symptom of sickness due to a core committer suddenly abandoned the project which consequently collapses the rest of development activities.

In this chapter we outlined how such health indicators can provide more insight of development activities that correlated to each others, we also succeed in providing prediction models that can

actually predict the likely measures of developer contribution patterns for the next 4 months of development.

The second health indicator represents the capability of developer community in defect management, as in our empirical investigations reveals that one indication of a healthy project is responsiveness of developer community for newly reported defects and faster defect fixing time, which of course also address the win condition of the end user of the OSS products.

Deriving Health Indicators from Quality Assurance Activities and Defect Status Changes Data. In the second study, we focused on certain aspects of quality assurances in OSS project, in particular for projects under Apache umbrella. By correlating metrics obtained from QA activities during development processes and defect status changes data, we defined additional health indicators and evaluated empirically with data from a pure and a hybrid OSS projects. We compared the results and discuss them with an OSS expert.

Quality assurance (QA) methods such as software testing and peer review are very important to reduce the adverse effects of defects in software engineering. In this study we explore current practices of QA and possibilities for their extension in open source software (OSS) projects.

This study presented a framework for QA aspects in OSS project based on our observation from typical OSS projects. Beyond the framework we performed case study on 2 large Apache projects *Tomcat* and *MyFaces*.

Our main results were (1) base on expert interviews and literature review we found different value expectations from the members of development community for performing QA activities in an OSS project. Based on their win condition we can derived some performance measurement of QA processes which need to be monitored by the project leading teams, to address typical questions such as *"Are we doing good enough in assuring our product quality?", "How much effort should we spend to increase the quality of our next release?", and "How can we predict the quality of our software product?".* (2) Different types of project may display a variety of QA activities which depend on the nature of the developer community (e.g. size of the development team, type of project sponsorship, project complexity, and release policies). For example Tomcat which is a pure OSS project signifies defect detection activities with faster defect closure time.

However the proportion of verified solution in Tomcat is lower than MyFaces, which we can assume in hybrid project, people tend to close and verified the defect solution properly. Generally spoken the results from OSS project should be compared to proprietary or closed source project of equivalent size, which unfortunately may seem difficult to do, and we consider as future work.

4 SOFTWARE QUALITY PREDICTION IN DISTRIBUTED DEVELOPMENT SETTINGS

This chapter outlines the second quality aspect in my research. It presents models and research issues for software quality prediction in distributed development.

Start with process modeling to describe software product and process improvement in OSS projects and ways to measure the level of quality assurance (QA) activities in form of process metrics. Some of these process metrics are "health indicators" presented earlier in previous chapter.

This research focuses on defect prediction as one quality aspect of distributed software development. It builds on Software Quality Prediction Framework (SQF) proposed in Section 4.2. For empirical evaluation,

In, this chapter we offer two scenarios in software quality prediction with two different quality indicators. First scenario attempts to predict the defect growth between releases based on objective estimates on the number of defects in a release and the currently reasonable defect removal capacity of the developers. The second scenario propose a framework to characterize and predict the defectiveness risk class of a software release relative to the average defectiveness level of a reference set of releases.

For both scenarios, we investigate the potential contributions of process metrics in combination with traditional product metrics to improve the performance of quality prediction models. We collected and analyzed product and process data from 4 large OSS projects in two Apache project communities with similar characteristics to conduct an empirical evaluation of the framework across projects and across communities.

4.1 Systematic Review of the Body of Literature on Defect Prediction

The rationale for identifying defective components of a software system prior to applying analytical quality assurance (QA) measures like inspection or testing has been summarized by Nagappan et al.: "During software production, software quality assurance consumes a considerable effort. To raise the effectiveness and efficiency of this effort, it is wise to direct it to those which need it most. We therefore need to identify those pieces of software which are the

most likely to fail and therefore require most of our attention." [91] A wide range of studies provide evidence about successful prediction of defects and various scenarios on how to exploit defect prediction have been proposed, for example, focusing testing and QA activities, making informed release decisions, mitigating risks, allocating resources in maintenance planning, and supporting process improvement efforts.

These studies also provide valuable advice and share lessons learned important for those who want to adopt defect prediction in practice. Currently there are many approaches to perform defect prediction [71] and respective validation methods [56, 97]. However, Koru et al. [64] advise that in practice, the most appropriate prediction method has to be selected for the current project context and the type of defect pattern to be predicted. Thereby, a good defect prediction model has to be constructed using a set of predictor variables that represents the actual measures of the software product and process [87, 88, 126]. Furthermore, several measures to evaluate the quality of a prediction are recommended, e.g. [80], and calibrating the prediction model to align false alarm rates with prediction goals and business scenarios is recommended [79].

Despite the many findings and the comprehensive information provided by the existing studies, there still is a wide gap between published research results and their adoption in real-world projects. Studies sharing insights about the application of defect prediction in practice are rare. Li et al. [72] discuss experiences and results from initiating defect prediction at ABB Inc. for product test prioritization and maintenance resource planning. Ostrand et al. [98] describe automating algorithms for the identification of fault-prone files to support the application of defect prediction in a wide range of projects. These studies show that in many cases, research results on defect prediction cannot directly be translated to practice. Adaptation and interpretation in the context of a particular project or organization is required. Furthermore, many studies focus on specific research questions. While these studies provide a valuable contribution to defect prediction research, this contribution remains an isolated piece of a bigger picture without following the entire track of research.

4.1.1 Systematical Literature Review Procedure

Numerous empirical studies on software defect prediction have been published in journals and conference proceedings. In order to provide a systematic guide to the existing body of literature, relevant studies have been searched and selected following the approach for a systematic literature review proposed by Kitchenham et al. [58].

A systematic literature review is defined as "a form of secondary study that uses a well-defined

methodology to identify, analyze and interpret all available evidence related to a specific research question in a way that is unbiased and (to a degree) repeatable" [58].

Staples and Niazi [114] summarize the characteristics of a systematic literature review: (a) a systematic review protocol defined in advance of conducting the review, (b) a documented search strategy, (c) explicit inclusion and exclusion criteria to select relevant studies from the search results, (d) quality assessment mechanisms to evaluate each study, (e) review and cross-checking processes to control researcher bias.

A key element of a systematic literature review is the review protocol, which documents all other elements constituting the systematic literature review. They include the research questions, the search process, the inclusions and exclusion criteria, and the quality assessment mechanisms.

Research Questions. The research questions summarize the questions frequently addressed in empirical studies. These questions contribute essential findings from research to the application of defect prediction in practice and are mapped to the phases of the framework. According to the framework, we emphasize three research questions to guide the systematical literature review process:

RI3.1. How do successful studies in defect prediction design the prediction process prior to model construction?

RI3.2. How do successful studies in defect prediction construct the prediction model from collected data?

RI3.3. How can external validation of the prediction model be provided for future predictions?

Search Process. The search process describes the process to identify the list of candidate studies. Following search process advocated by Barbara Kitchenham et al. [62], the search process was organized into two separate phases. The initial search phase identified candidate primary studies based on searches of electronic digital libraries from IEEE, ACM, Elsevier, Springer, and Wiley. Search strings have been composed from search terms such as defect, error, fault, bug, prediction, and estimation. The secondary search phase is to review the references in each of the primary studies identified in the first phase looking for more candidate primary sources which repeated until no further relevant papers can be found.

Inclusion and Exclusion Criteria. The criteria for including a primary study comprised any study that compared software defect predictions which enables metric-based approaches based on analysis of project data. We excluded studies where data collected from a small number of

observations (less than 5 observations). We also excluded studies where models constructed only based on historical data of defects with no other metrics as predictor variables. The third exclusion criterion is that we only consider studies that performed internal validation and external validation of constructed prediction model. Formal inclusion criteria are that papers have to be peer reviewed and document empirical research. Regarding the contents, inclusion requires that the study addresses at least one of the defined research questions.

Quality Assessment Mechanism. This systematic literature review has been based on a documented and reviewed protocol established in advance of the review. Furthermore, in this study two researchers were involved in conducting the systematic literature review and cross validation of the results. For example, one researcher queried a digital library and extracted candidate studies while the second researcher verified the search terms, search results, and the list of identified candidate studies. Thereby we minimized researcher bias and assured the validity of the findings of the review. As suggested by [60] that a systematical literature review doesn't necessary to summarize hundreds of findings but rather to focus on small number of most related studies thus for the next step we discuss the most relevant studies that may answer our research questions properly. By following this approach we identified 12 studies on defect prediction providing findings from a total of more than 200 studies found in abovementioned digital libraries.

4.1.2 Extraction of Findings and Discussion

This section maps the findings from the systematic literature review to the phases and tasks of the framework for defect prediction. The findings summarize the contributions extracted from the studies with respect to the research questions 1 to 3 used to drive our systematic literature review.

Table 13 lists information about how current research defines the goals of defect prediction studies, questions and hypotheses, as well as how variables are specified to describe each question.

Note that Explicit mean the study describes the following terms (goal, hypotheses, etc) clearly as a separate part from surrounding texts and adhere to our term definitions in the framework. Implicit mean we need to extract the information from the text to identify a term definition. As N/A reveals that there is no information contains the definition of an expected term in the study.

Table 13 Study Related Factors- Preparation Phase

Study	Preparation Steps		
	A.1	A.2	A.3
	Goal definition	Research questions	Variables Specification
Moser et al [87]	Goal is implicitly described	Questions proposed with respective Null Hypotheses	Implicit variables specifications to predict module defect proneness
Li et al [71]	Goal is implicitly described	Explicit research question with no hypotheses	Explicit variables specification to predict defect intensity of a release
Zimmermann et al [131]	Goal is implicitly described	Implicit research question with no hypotheses	Implicit variables specifications to predict module defect proneness
Koru et al [64]	Implicit goal description	Implicit research question with no hypotheses	Implicit variables specifications to predict module defect proneness
Nagappan et al [89]	Implicit goal description	Explicit research hypotheses	Explicit variables specification
Li et al [73]	Goal is implicitly described	Explicit research question with no hypotheses	Explicit variables specification to predict defect intensity of a release
Weyuker et al [127]	Explicit goal description in later section	Implicit Research questions with hypotheses	Implicit variable specification to predict file defect proneness
Menzies et al [80]	Implict goal description	Implicit research question, hypotheses described later in the	Explicit variables specification for module defect proneness

		paper	
Graves et al [44]	Goal is implicitly described	Implicit research questions with no hypotheses	Explicit variables specification for module defect proneness
Sunghun et al [115]	Implicit goal description	Explicit research hyptoheses	Explicit variables specification of file defect proneness
Pai et al [99]	Implicit goal description	Explicit research question with no hypotheses	Explicit variables specification for number of defect per class and class defect proneness
Olague et al [96]	Explicit goal statement	Explicit research hypotheses to describe proposed goal	Implicit variable specification to predict class defect proneness

Most of the studies do not explicitly describe the goal of the study and there is no single study which identifies the target stakeholders of the results with their values expectations. 7 out of 12 studies explicitly stated the research questions and/or respective hypotheses, which provide guidance for the remaining empirical study process. Most of the study specified the variables as part of the prediction model construction prior to data collection. Thus, we assert that the preparation phase which consists of goal definition, research questions and hypotheses formulation, and variable specifications is a common practice in conducting defect prediction with different levels of detail and presentation.

Table 14 Study Related Factors- Model Construction

Study	Model Construction Steps			
	B.1	B.2	B.3	
	Parameters Selection	Prediction Methods	Internal Validation	Model Performance Measures
Moser et al	product and process	Naïve Bayess,	10 Fold cross	Number of False

[87]	metrics with no parameter selection	Logistic regression and J48 with	validation and Performance measure:	positive and Recall
Li et al [71]	Product and process metrics with no parameter selection	16 modeling methods	N/A	Average relative error
Zimmermann et al [131]	Product metrics with selection by Spearman bivariate correlation analysis	Naïve Bayess, Logistic regression and J48	10 Fold cross validation.	Performance measures: Accuracy, recall and precision
Koru et al [64]	Product (Design) metrics with no parameter selection	J48	10 Fold cross validation	Performance measures Recall, Precision and F-Measure
Nagappan et al [89]	Process (code churn) metrics with selection by Spearman correlation	Multiple regression, Step-wise regression and Principal Component Analysis (PCA)	Coefficient of determination analysis, F-test	Discriminate analysis
Li et al [73]	Product and process metrics with no prior selection	16 modeling methods	N/A	Average relative error
Weyuker et al [127]	Process (developer) metrics with no parameter selection	Negative binomial regression	N/A	Correctly identified files
Menzies et al [80]	Product (static code) metrics with no parameter selection	Naïve Bayes with log transform, J48, OneR	10 Fold cross validation	Accuracy, Number of false positive, Receiver operator curves
Graves et al	Product (changes code) metrics with no	General linear	N/A	Error measure

[44]	parameter selection	models		
Sunghun et al [115]	Process (change) metrics with no parameter selection	FixCache prediction method	Cross validation for all data set	Accuracy
Pai et al [99]	Product metrics with variable selection by correlation analysis and backward linear regression	Multiple linear regression, Logistic regression, Bayesian network model	10 Fold cross validation	False positive rate, precision , specificity, sensitivity
Olague et al [96]	Product (Object Oriented) Metrics with parameter selection by Spearman bivariate correlation analysis	Univariate and Multivariate binary logistic regression	Hold out method	Percentage of correctly classified classes

Table 14 outlines steps taken to construct the prediction model of these studies used variable selection prior to model construction. Methods such as Spearman bivariate correlation analysis and linear regression with selected methods (backward, stepwise, remove) are considered as common methods for parameters selection prior to fit them into the prediction model.

The selection of prediction methods is based on what kind of defect pattern to be predicted, i.e., classification techniques such as logistic regression can be used to predict file defect-proneness but will obtain poor performance to predict file defect rates. Similar to prediction method selection, one should also choose appropriate internal validation methods and model performance measures. We conclude that preparation and model construction phases have been identified as commonly performed by researchers in defect prediction.

Table 15 Study Related Factors- Model Usages

Study	Model Usages Steps	
	C.1	C.2
	External validation	Robustness Analysis
Moser et al [87]	Cross validation with different releases with low performance results	N/A
Li et al [71]	Constructed model were used to predict a certain period of defect growth per release	Proposed framework were used for commercial context [73]
Zimmermann et al [131]	Cross validation of trained prediction model in different releases and levels of observation	N/A
Koru et al [64]	Cross validation of trained prediction model with different class of data	Depicts the need for model calibration or refinement
Nagappan et al [89]	State briefly with no data	N/A
Li et al [73]	Cross validation with different releases	N/A
Weyuker [127]	N/A	N/A
Menzies [80]	N/A	N/A
Graves [44]	N/A	N/A
Sunghun[115]	N/A	N/A
Pai et al [99]	N/A	N/A
Olague et al [96]	N/A	N/A

For the third phase Model Usages (see Table 15), we found only two studies providing appropriate results of the two involved steps. This finding confirms the critique from Norman and Fenton [29] that most of the existing studies on defect prediction do not provide empirical prove whether the model can be generalized for different observations.

There are several reasons why many studies did not report the external validation and robustness analysis of constructed prediction model such as the availability of new observation data [99] and external validation results which signify poor performance of the model [87] for which many of the authors do not wish to report. However from practitioners' perspective such conditions

should be addressed properly by data collection process refinement and model calibrations until the model can be proven for its usefulness for prediction in particular context. Later we use the results of systematical literature review to derive the research roadmap in software quality prediction and evaluation (see Section 4.2) and to construct a systematic framework for conducting software quality prediction in distributed software development settings (see Section 4.3).

4.2 Research Roadmap of Software Quality Prediction and Evaluation in Distributed Software Development

A number of empirical studies provide evidence of successful prediction of defects using data from real-world projects conducted in an industrial or open-source context. However, practitioners are confronted with additional requirements when they try to replicate the success of these studies within the context of their specific projects and organizations. Derived from systematical literature review result reported in Section 4.1, we found several issues relevant for applying quality prediction in practice, which are currently not adequately addressed by the existing body of literature. Related future works are encouraged in order to make software quality prediction a commonly accepted and valuable aid in practice.

4.2.1 Challenge 1: Needs for well planned quality prediction

Quality prediction and evaluation remain a risky endeavor for practitioners as long as upfront investments for data collection and model construction are high and a return on these investments has to be expected late or never [64]. Thus to conduct quality prediction should adhere to these following requirements:

1. **Aligning defect prediction with project and business goals.** Empirical studies tend to focus on prevalent research questions. Practitioners, however, have to align defect prediction with the goals of their specific project. Concentrating testing on defect-prone components or planning the effort for maintenance activities are examples for such goals. Defining the goals first is therefore an important requirement as an appropriate budget has to be allocated for defect prediction and, moreover, the investment has to be justified according to estimated savings and benefits.
2. **Creating a project-specific prediction model.** Prediction models are constructed from a project's historical data. A prediction model, thus, models the context of a particular

project. As a consequence, predictors obtained from one project are usually not applicable to other projects. Nagappan et al. [90], for example, showed that predictors are accurate only when obtained from the same or similar projects and that there is no single set of metrics that is applicable to all projects. These findings were supported by Koru and Liu [64] when analyzing the PROMISE repository containing data about projects conducted at different sites. “Normally, defect prediction models will change from one development environment to another according to specific defect patterns.” [64]

3. **Evaluating the feasibility in the project or organizational context.** Despite the success reported by many studies, the prediction of defects in a particular project may not be possible. Typical reasons are the poor quality of the available data [59] or the effort required to extract and collect the necessary data [103]. Most published studies report solely successful cases of defect prediction. Only few studies point toward limitations, for example, Li et al. [72] comment on the poor accuracy in predicting field defects for one of the studied products. Most projects and organizations cannot afford this investment under such adverse conditions. Thus, means are required a) to identify the most important quality indicators, b) to conduct an early and quick estimation of the feasibility of predicting quality with acceptable performance in the context of a specific project or organization [97]. In short, the feasibility of predicting defects has to be estimated early to confirm that the defined goals will be met.
4. **Striving for fast results.** Even when the feasibility is positively evaluated, defect prediction is required to produce results fast. Defect prediction is relatively new in the software development arena, and practitioners face a high level of uncertainty concerning the return on the investment in defect prediction. Thus, when results cannot be obtained within one or a few iterations the chance defect prediction will be applied in a real-world project is low. The general concerns of practitioners have also been described by Ostrand et al. [98]: “In our experience, practitioners won't even consider using a new technology without evidence that it has worked on a substantial number of real systems of varying types. It is very unlikely that practitioners will be convinced that a new tool is worth learning and evaluating merely on the basis of its demonstration on toy systems or on systems much smaller than the ones they normally develop and maintain.” If the prediction study seems feasible then a sound plan should be assembled in order to have better controlled prediction model construction and validation, as well as to adhere to cost benefit constraints.

In Section 4.3, we describe framework that can be used to have a well planned quality prediction and evaluation which adhere to abovementioned requirements. The frameworks have been evaluated by means of systematical literature review, later we apply this framework with empirical data from different contexts of distributed software development, and reported in Section 3 and Section 4.

4.2.2 Challenge 2: Effective and efficient data collection

DSD comprised of complex distributed processes and heterogeneous project repositories. Metrics for as input parameters for prediction model are obtained through collection and correlation from these data sources. Ostrand et al. [98] found that "it is very time consuming to do the required data extraction and analysis needed to build the models, and few projects have the luxury of extra personnel to do these tasks or the extra time in their schedules that will be needed. In addition, statistical expertise was needed to actually build the models, and that is rare to find on most development projects". As a consequence, it should be possible to organize data extraction and model creation separately so it can be outsourced or – if tool support permits – automated.

Current practices are commonly using a specific data mining tool for each data sources, however different data sources and tools often means incompatible format and need for data integration. Manual reformatting and integration of data before fitting into the prediction model are time consuming and error prone tasks. Hence, most of defect prediction studies reported only a very limited number of data sources, often with only observing a single project repository.

More researches for data collection in DSD are necessary to increase the quantity and quality of data with reasonable efforts. One approach if to exploit available data mining tools that enable interfacing with a number of project repositories and store the results as a set of metrics in uniform format [125]. Another approach is to investigate the semantic relationship between data across projects which enable integration of development tools and project repositories (e.g. by using semantic web technology, ontology for integrated data modeling and data collection) [46, 67].

4.2.3 Challenge 3: Predicting under uncertainty

Fenton and Neil [29] remind that "Project managers make decisions about software quality using best guesses; it seems to us that will always be the case and the best that researchers can do is 1)

recognize this fact and 2) improve the ‘guessing’ process. We, therefore, need to model the subjectivity and uncertainty that is pervasive in software development.” Uncertainty exists besides limitations resulting from incomplete, insufficient data. It arises often about how the data has to be interpreted, which reflects the peculiarities of a project such as individual project regulations, discontinuities in workflows and processes or specific use of tools. Practitioners therefore rely on expert judgment and have to make assumptions. These assumptions should be made explicit and – as a positive side-effect – the prediction model should provide information to verify these assumptions.

4.2.4 Challenge 4: Dealing with incomplete and missing data

Existing studies on software quality prediction neglect the fact that information is often missing or incomplete in real world settings. Practitioners therefore require methods to deal with missing or incomplete information.

Treating incomplete and missing data are common practices in statistical domain. Typically if the data set is large with only small number of random value are missing the problem is not severe. On the other hand, a smaller data set which has significant missing values that are non randomly distributed will need serious attentions [116].

Li et al. [73] reported: “We find that by acknowledging incomplete information and collecting data that capture similar ideas as the missing information, we are able to produce more accurate and valid models and motivate better data collection." Hence this is imperative that future researches in software quality prediction should take closely to missing and invalid data issues using well established statistic methods.

4.2.5 Challenge 5: Providing accurate and prompt prediction results

Our systematic literature review reported that most of software defect prediction studies still put their focus on exploiting product metrics to construct good prediction model. However, as mentioned in related work, recent studies such [70, 87, 126] reported that in DSD project with short release cycle such as OSS project, prediction models which enabled only product metrics were outperformed by those that enable process metrics or combination of both classes of metrics.

In this study, we focus on metric based prediction models, as they offered early availability of data compared to time based approach (e.g. reliability growth model) with tradeoff in accuracy

of the results [73, 74]. Yet, another raised issue is that current studies are typically tried to fit all collected metrics into the prediction model without prior knowledge whether these metrics have significant correlation to the estimator (e.g. number of defects, or module defectiveness). From statistical best practices it is not wise to construct a model with a lot of weakly correlated metrics as they will reduce the performance of the constructed model [33]. For this reason, future work in quality prediction should take into account of this issue such as by comparing the results of prediction with different types of metrics as well as prediction with and without parameter selection, one should distinguish the differences and take the best option.

4.2.6 Challenge 6: Reusing and validating the existing model for upcoming releases.

To optimize the return on the investment in model creation, the model has to be reused for upcoming releases with minimal additional effort. However, over time, the project's context and the defect patterns can change. As a consequence, prediction results for a new release derived from a model created and verified with historical data have to be validated. Practitioners need a measure of reliability when they make decisions based on prediction results. Furthermore, Koru and Liu [64] point out that "as new measurement and defect data become available, you can include them in the data sets and rebuild the prediction model." As adjusting or rebuilding the model requires additional effort, the validation results should serve as an indicator when adjusting or rebuilding becomes necessary.

4.3 The Software Quality Prediction Framework (SQF)

In this section we describe a framework for software defect prediction which consists of three phases – (A) preparation, (B) model creation and (C) model usage – as well as seven steps (see Figure ***20***). This framework is in line with the requirements outlined in the previous section and has been derived from our experience and existing body literature on software defect prediction. SQF is an extension from framework for conducting empirical study proposed in Section 2.1.3, but with more focus for conducting software quality prediction.

4.3.1 Phase A – Preparation

As first phase in conducting a defect prediction, one should start by preparing the necessary preconditions prior to model construction. The intention of the preparation phase is to create a clear focus of what results should be provided by the prediction, to appropriately design the

prediction approach, and to have quick analysis whether such design will accomplish the expected results within project and organizational context.

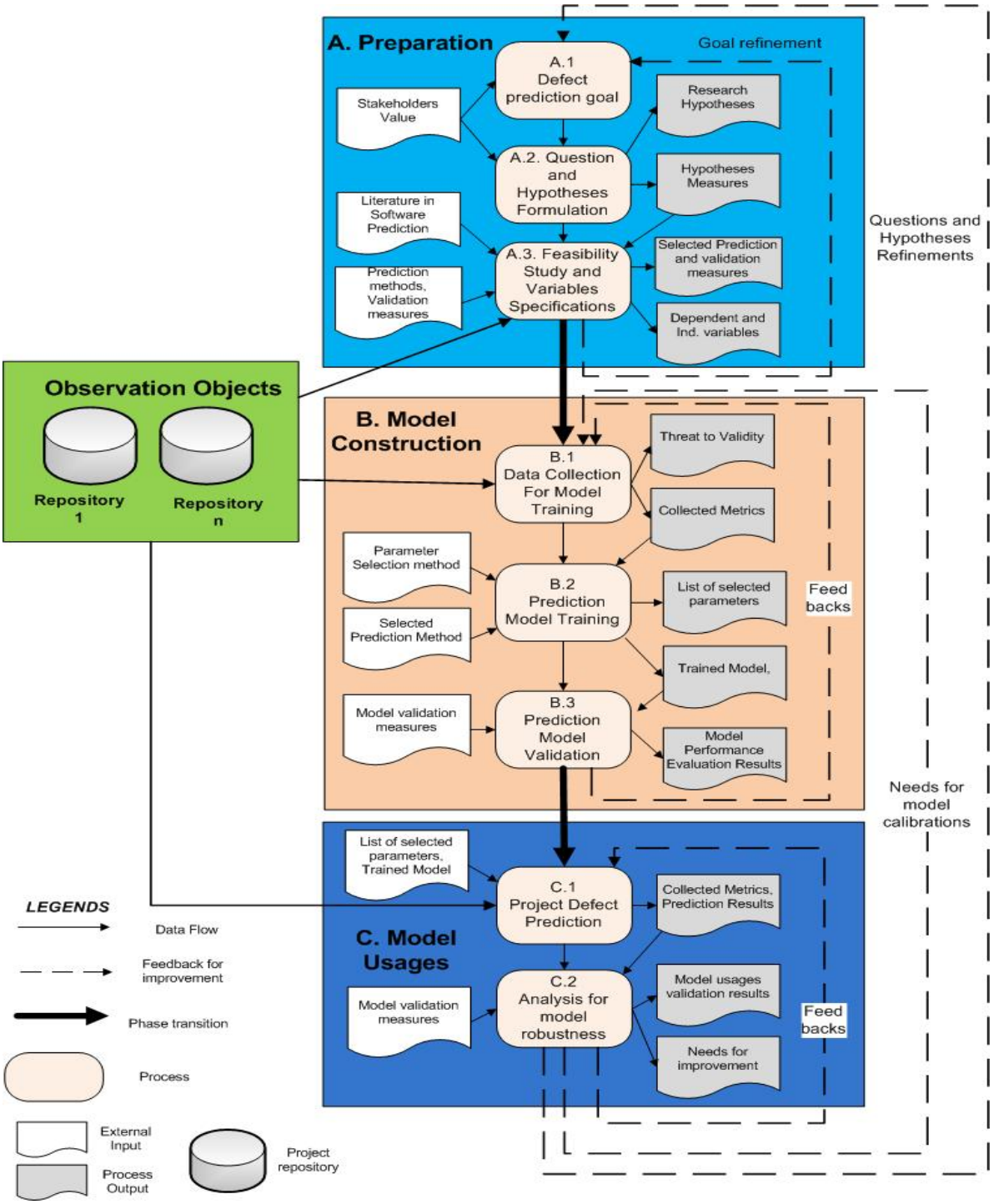


Figure 20 Software Quality Prediction Framework (SQF)

A.1. Define defect prediction goal, which represents the objective of defect prediction with respect to a particular stakeholder perspective and the current project context.

A.2. Specify questions and hypotheses. Questions are derived from the defect prediction goals.

They are used to identify relevant models of the objects of study and, then, to more precisely define the expected achievement of a specific goal. The questions can be reframed as hypotheses about the observed situation or defect pattern. We recommend specifying hypotheses that are easily measurable to enable the falsification or acceptance of the hypotheses for a sound assessment of the prediction results

A.3. Quick feasibility study and variables specification. A quick feasibility study is essential to assess whether the initial goals of the prediction can be achieved using the available data from the observation objects. A negative assessment indicates the initial goals are not feasible and shows the need for adjusting the goals and questions. After conducting a feasibility study, the set of metrics that should be collected and estimated in the prediction model is collected. These metrics act as independent variables and dependent variables in the prediction model to be constructed in the next phase.

4.3.2 Phase B – Model Construction

Constructing the prediction model is the core phase in defect prediction, here, based on the variables and the defect prediction method specified in the previous phase, data collection, model training, and model evaluation are performed.

B.1. Data collection for model training. As part of a close investigation of the available data sources, the period of observation and relevant project repositories and databases are specified. Based on the previously selected variables the data is collected from the observation objects. Invalid and missing data is thereby filter or refined. For making a sound prediction, potential threats to validity are recorded.

Collected Data Quality Assurances. One of research challenges in software defect prediction as shown in our research roadmap is the validity and sufficient quality of data prior to prediction model construction. There are some typical cases which reduce the quality of data such as missing and invalid value of data, invalid data extraction (e.g. wrong data query, wrong data source, etc), error during data integration that come from different sources, etc. In our study prior to model construction, all input parameters for the model are derived from integrated data that come from difference sources (see Figure 21). In this study we propose two quality assurance (QA) gates that should be conducted hence to assure the sufficient quality of data prior to prediction modeling.

In Figure 21, the raw data come from project repositories such as Source Code Management (i.e., SVN and CVS), Issue Tracker (i.e., Bugzilla and Jira), Mailing List (Developer Mailing List and

User Mailing List) and other repositories (e.g., Project Website, Project Wiki, Forum, etc).

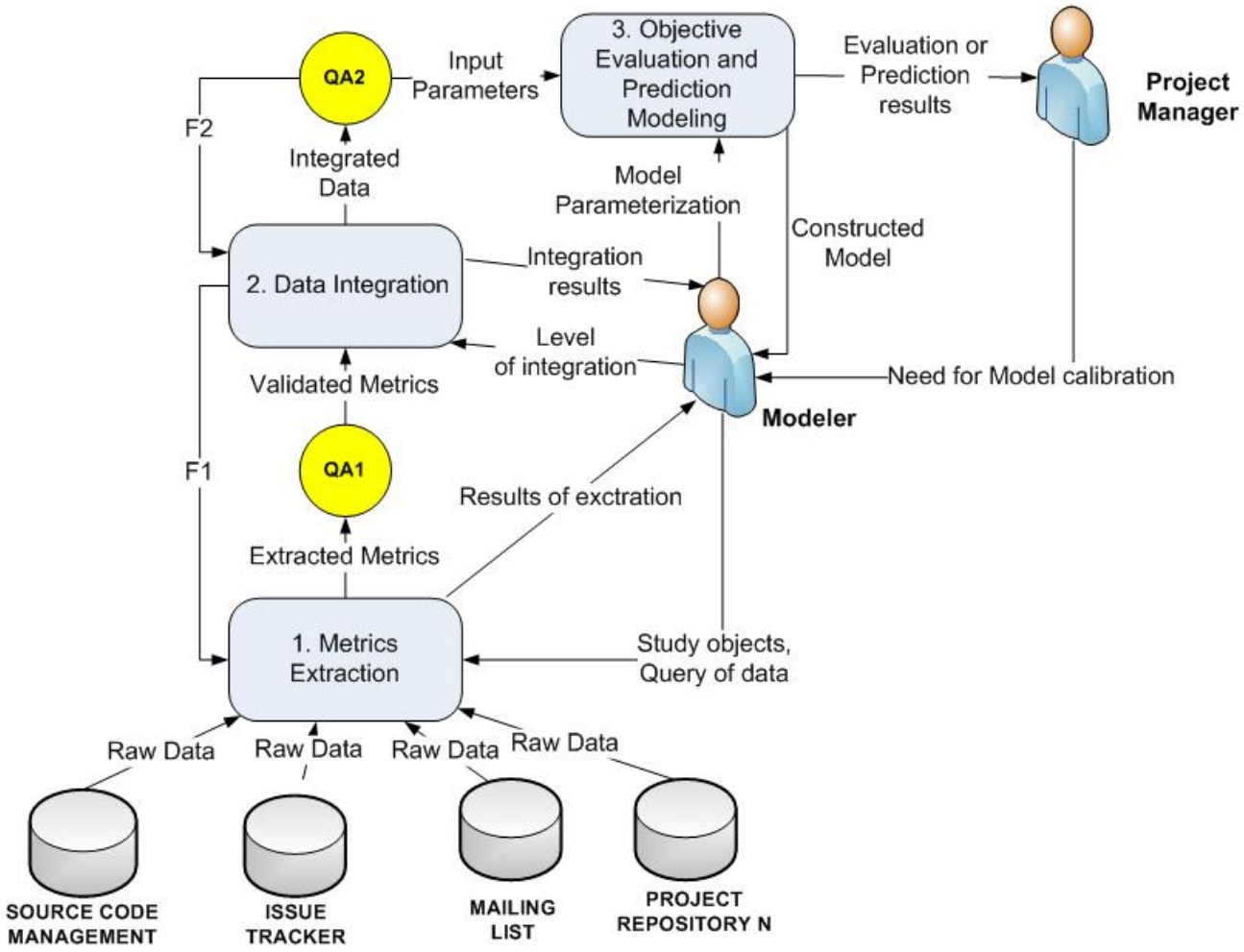


Figure 21 Data Collection and Refinement Procedure

These project repositories are sources for raw data that should be further extracted (see **Step 1**) by certain queries command or data mining tools (see Chapter 3 and Chapter 4 for overview of data mining tools we utilize in this thesis) to obtain basic metrics such as product metrics, basic process metrics and defect data.

The first QA (see circle ***QA1****)* are *a) to check the validity of query performed*, and *b) the origin correctness of data sources*. For example to retrieve the process metrics of a historical release from typical Apache project, one should perform the data extraction from the *TRUNK* directory of the SCM with specific preset date that represents the life span of the release (release date to release date of the next release version). While the product metrics should be collected from the historical releases directory of the SCM (sometimes called as *BRANCHES* directory), in this directory the community will keep the originality of code source, thus we can assure that the collected product metrics are come from source code that has not been through additional

changes since its release date. The results of QA1 are validated metrics or needs for data refinements (recollection of particular data) see feedback loop F1.

To construct the prediction model (model training), we need certain number of data points depends on the prediction techniques we use, for example for most of regression procedure we will need n+1 data points where n is number of parameters within the model. A data point represents one level of observation (e.g., in files level, module level, or release level) in study context. Hence to derive a data point we need to integrate data (see **Step 2**) built from metrics that previously we collected in step 1.

The second QA (see circle ***QA2****),* are *a) to check if there is missing or invalid value in data point, b) to check whether all data points represent the uniform level of observation required in our prediction modeling c) to check if number of data points are sufficient to construct required prediction model d) as additional QA, one can also perform parameters correlation analysis or selection procedure to indentify which parameters can only be noise and reduce the accuracy of to be constructed prediction model* (see Chapter 4 for overview of these procedures). Note, that there are some common treatments in statistic [33, 85] for missing data that can be used in software quality prediction such as:

- Obliterating missing or incomplete cases, if a case has missing values it may be removed from observation. Deleting data with missing value is typically the default option with statistical software.
- Estimate missing values and use these values during consequent data analysis. Estimates may be obtained from prior knowledge, mean values, regression techniques and spatially autoregressive models.
- Treating missing values as data which may indicate some form of behavior.
- Compare the quality prediction results with and without missing data, if they are noticeably different then try to discern the reason for the difference.

Feedback loop F2 contains information of the need for data refinement such as treatment for missing value. Later the validated input parameters are fit in to the prediction model (model training). A project manager can decide whether the performance of the prediction model is sufficient in term of accuracy and reliability or he may trigger the need for model calibration to improve the model performance.

For the following empirical evaluation of concepts in this thesis, we use this schema of data collection to ensure that the constructed models are valid with adequate quality for prediction.

B.2. Prediction model training. Parameter selection is used to identify the parameters with a significant impact on the dependent variables. These parameters are used in training the model, usually applying standard statistical or machine learning tools. Depends on the type of quality indicator we need to select best available technique for prediction model training.

As described above, in this study we propose two quality indicators: a) defect growth between releases which is predicted using multi linear regression methods such as *Stepwise* and *Forward* regression procedures and b) class risk of a release which is predicted using classification techniques such as *Logistic Regression, Naive Bayess, J48 and Random Forest* (please refer to Appendix for detailed descriptions of these selected techniques).

B.3. Prediction model validation. The trained model needs to be validated for its performance, i.e., accuracy, recall and precision. Unsatisfying results should trigger a feedback loop back to the step data collection, as it will not make sense to proceed with a low-performance model that, e.g., has a high number of false positives or errors.

4.3.3 Phase C – Model Usages

A major concern from a practitioner's point of view is that many studies reported a trained defect prediction model which show a good performance by means of cross validation with historical data [29]. Only limited studies reported the robustness of the model with different observations. This, however, is a necessity in practical usages for predicting the quality for a certain time period in the future.

C.1. Project defect prediction. In this step the model trained in the previous phase is actually used, i.e. the model is parameterized with observations form new releases to predict defects in these releases.

C.2. Analysis for prediction model robustness. Based on the results of step C.1, the robustness of the model is analyzed. Thereby, the reliability of the current prediction results are estimated to determine how to apply the prediction results in the project, e.g., to safely rely on them or to be careful. If the analysis indicates low reliability, a feedback loop back to re-creating or calibrating the model should be triggered as well as suggestions for refinement of the prediction hypotheses should be provided.

4.4 Using Combined Product and Process Metrics to Predict Defect Growth between Releases in OSS Projects

The quality evaluation of open source software (OSS) products, e.g., defect estimation and prediction approaches of individual releases, gains importance with increasing OSS adoption in industry applications. Most empirical studies on the accuracy of defect prediction and software maintenance focus on product metrics as predictors that are available only when the product is finished.

Only few prediction models consider information on the development process (Process Metrics) that seems relevant to quality improvement of the software product. In this paper, we investigate defect prediction with data from a family of widely used OSS projects based both on product and Process Metrics as well as on combinations of these metrics.

In this study we proposed our two-step predictor selection procedure. First, we use correlation analysis as suggested in [89] to identify predictors with strong correlation to potential defect growth between releases.

In this study, we call the potential defect growth between releases "delta defects". Estimates of delta defects are important indicators to evaluate the quality improvement of the current development process (e.g., potential contribution of defects of the next release) compared to prior releases.

In the second step, we use stepwise regression and backward elimination for selecting a subset of independent variables (predictors) from the strong correlated list to form a linear prediction model [56].

For evaluation, we cross validate the prediction model by comparing the average relative error (ARE) [56] of each prediction model to select which variant provides better estimates of delta defects.

4.4.1 Empirical Study Design

In this section, we describe our case study objects, define independent and dependent variables, and formulate research hypotheses for evaluation.

1. Goals of Empirical Study

For a release manager and project leading team in an OSS project, defect prediction between releases is important as decision support for release candidates such as: a) is the a release

candidate good enough for deployment or whether there is another QA cycle necessary before delivery; b) input for planning the next release cycle based on the prediction results.

The goal of this study is *to investigate different metrics contributions to the accuracy of defect prediction in OSS projects and provide robust prediction model to support release process from project and quality manager point of view*.

2. Study Objects

For an empirical evaluation we collected data from 11 releases of 2 Apache MyFaces project family (Tobago and Core), and analyzed the potential contribution of combination of product and Process Metrics for defect prediction model in OSS project context.

The objects of our case study are releases in the family of the OSS Apache MyFaces project[26]. We selected MyFaces Core and MyFaces Tobago for the study because Core is the main project of MyFaces and a pure OSS project (all voluntarily developers) while Tobago is a hybrid project where some developers are paid and well supported by commercial organizations.

Later we applied defect prediction models to six releases of Core (C.1.1, C.1.2, C.1.3, C.1.4, C.1.5, and C.1.6) and six releases of Tobago (T.1.1, T.1.2, T.1.3, T.1.4, T.1.5, T.1.6). Our selection criteria are: all releases should be announced after both projects have left the incubation process from the Apache Software Foundation[27]. Later we can regard our selected study objects as mature releases and have been promoted for larger user and developer community; therefore, we can observe more activities within the project community compare to the activities during the incubation process.

3. Variables Specifications

The measurement model defined for the empirical study consists of independent and dependent variables. Following standard practice in empirical studies we define the independent variables as: a) selection of input parameters (product, project or combination of both) and b) context parameters consisting of deployment metrics, configuration metrics, project origin, project sponsorship (pure or hybrid) and period of case study. The dependent variable in our case study is growth of defect between releases called as delta defect (*DG*). *DG* signifies the number of defects reported after release (*d*) in comparison to accumulative defect reported prior to release (*do*) and *d* (see Eq. 9).

[26] Apache MyFaces Project website can be found at http://myfaces.apache.org/. Last accessed at 10th January 2008.

[27] http://incubator.apache.org/ Last accessed at 10th January 2008.

$$DG = \frac{d}{d + do} \qquad \textit{Eq. 9}$$

Using *DG* as dependent variable we can directly assess current quality of release in term of defect reported in comparison to prior release, for example if DG > 50% means current release contributes more defects than in prior to release and signify the need for higher resource allocation for defect removal.

To select which predictors have strong correlation with independent variables, we employ the *Pearson* bivariate correlation model [91, 131], and we use multiple linear regressions to exclude insignificant predictors [130] and to develop prediction models with different combination of predictors (product metrics only, Process Metrics only, and combination of both types of metrics).

To evaluate the accuracy of the linear regression prediction models we fit the model to historical data of releases, we use the average relative error (ARE) to evaluate forecast accuracy. We apply the ARE definition as suggested by [56] to DG instead of absolute number of defects reported (*d*); and *DG'* as estimator of *DG* (see Eq. 10)

$$ARE = \left(\frac{1}{n}\right)\sum_{i=1}^{n}\left|\frac{DG - DG'}{DG}\right| \qquad \textit{Eq. 10}$$

4. Research Questions and Research Hypotheses Formulation

In the case study we will evaluate following hypotheses in order to address the research issues:

RI.5.3.1 Contribution of Process Metrics: Goal of this research issue is to investigate whether an increase of QA effort is correlated with a decrease of defects in the next release. Therefore we propose the null hypothesis as:

H0.5.3.1: There is no Process Metrics (pr) that has statistically significant impact to dependent variable DG compare to product metrics (pd).

If r is a function to check whether there is a strong correlation between variables $x \in$ pr and dependent variable DG, *then* the respective null hypothesis can be formulated as

$$\textbf{H0.5.3.1:} \quad \{\forall\, x \in pr | r_i(x_i, DG) = False\}$$

RI.5.3.2 Accuracy of Defect Prediction using Combined Project and Product Metrics: A combination of process and product metrics should be able to predict the defect growth in the next release with lower ARE value compared to prediction based on the traditional product metrics alone. Then we proposed following null hypothesis as:

***H0*.5.3.2:** A prediction model that used combination of process and product metrics has higher ARE value compare to prediction model that used only product metrics. *If the estimate of defect prediction model (e) is a function of product (pd) and/or process metrics (pr), then* the respective null hypothesis can be formulated as:

$$H0.5.3.2: \quad ARE(e(pr,pd)) > ARE(e(pd))$$

4.4.2 Data Collection

This section describes the data collection proceeding and threat to validity of collected data.

1. Data Collection and Refinement

In this work, we examined both projects during 6 months of recent development (1/10/2007 to 01/03/2008). The observed projects employ SVN as their SCM tool, and Jira for issue tracker. Hence, to measure the code development activities before release, we retrieved 24 months historical code collections using StatSVN v.0.4.1, SVN log and diff commands from the *trunk* directory of each project.

For each release observation, we created a snapshot with a specific time range (i.e., the first time a release being announced until the announcement of succeeding release). For example a release R1 was first announced in 2007-02-12 which later followed by R2 in 2007-04-19, then the snapshot for observing the development activity prior to R2 is (in SVN log creation command):

Table 16. SVN Log Command.

svn log -v --xml -r {2007-04-19}:{2007-02-12}>log.xml

After collecting the snapshot log, we applied the StatSVN tool[28] to collect code development metrics from each project SVN repository based on the given snapshot log.

Table 17. Queries for Defect and Issue Data Collection.

Addressed defects and issues prior to a release: SELECT <issue|defect> FIX FOR <release>
Number of defects after a release: SELECT defect AFFECT VERSION <release>
Resolved and peer-reviewed data: SELECT defect WITH STATUS <resolved|closed>

[28] StatSVN tool can be found at: http://www.statsvn.org/

For defect and issue removal data evaluation, we used Jira v. 3.12.3 query commands[29] to collect (see Table 17) and evaluate the defect and issue data. We exclude the *INVALID* and *DUPLICATE* defects; therefore we only include valid defect data for model construction.

Using specific *Jira query commands* and *SVN snapshot logs*, we assure the validity of collected data and assure that for each release observation all collected metrics are derived from developer activities prior to observed release date.

We applied the Eclipse Metrics v. 1.3.6 tool plug-in[30] to measure product metrics of the study objects. We used a check style plug-in[31] to analyze style violations in the source code[32]. We analyzed the collected data using SPSS v.14 for performing Pearson correlation analysis and linear regressions procedures (Stepwise and Backward). Table 18 describes collected product metrics as suggested by [30, 40, 71] with two additional code quality metrics.

Table 19 outlines 23 process and resource metrics as suggested by [30, 71, 124, 127] and newly proposed metrics (italic font). For deployment and usage metrics we used following metrics: type of release (major release, minor release and service pack), months since the 1st release, months since the previous release, month to the next release, months from release date to the end of case study.

[29] Jira Query Commands for ASF can be found at https://issues.apache.org/jira/

[30] Metrics plug-in for Eclipse: http://metrics.sourceforge.net/. Last accessed at 15th December 2007.

[31] Check style plug-in for Eclipse: at http://eclipse-cs.sourceforge.net/. Last accessed at 10th December 2007.

Table 18. Collected OSS Product Metrics [71, 126].

Source of Variation	***Metrics collected***	***Abbr.***
Volume or size	Total Lines of Code	LOC
	Method Lines of Code	MLC
	Number of packages	NOP
	Number of classes	NCL
	Number of children	NOC
	Number of attributes	ATT
	Number of methods	NMH
	Number of interfaces	NOI
	Average of the class specialization index	ASI
	Number of overridden methods	NOM
	Number of static methods	NSM
	Number of static attributes	NSA
	Average number of parameters	ANP
Control complexity	Average McCabe Cyclomatic Complexity	MCC
	Weighted Methods per Class	WMC
	NPath Complexity	**NPC**
Modularity	Average Lack of Cohesion of Methods	ACM
	Average Afferent Coupling (for each class: number of classes that uses the class)	AAC
	Average Efferent Coupling (for each class: number of classes used by the class)	AEC
	Average Instability (AEC/(AEC+AAC))	AIS
	Abstractness	ABS
	Average Normalized Distance from Main Sequence	AND
	Average Depth Inheritance Tree	DIT
	Average Nested Block Depth	NBP
	Average Specialization Index	ASI
Code quality	*Number of check style violation*	CSV
	Ratio of check style violations per number of check style methods	RCV

Table 19. Collected OSS Process Metrics.

Source of Variation	***Metrics collected***	***Abbr.***
Defect and issue removal prior to release [122, 124, 126]	*Number of targeted defects (reported defect that should be solved prior to the next release)*	TD
	Number of resolved defects	**RD**
	Number of peer-reviewed defects	CD
	Number of open defects	OD
	Number of targeted issues	TI
	Number of resolved issues	RI
	Number of peer-reviewed issues	CI
	Number of open issues	OI
	Defect resolution level= RD/TD	**RDTD**
	Defect peer review level = CD/TD	**CDTD**
	Issue resolution level = RI/TI	RITI
	Issue peer review level = CI/TI	CITI
	Number of invalid defect reports;	NIDR
	Number of invalid issue reports;	NISR
	Number of defect reporter,	NDR
	Number of issue reporter,	NIR
	Avg Number of defects reported by a reporter	NDRR
Code development prior to release [87, 126, 127]	Number of Commits	CM
	Total Changes	TC
	Total Lines of Code Deleted	TFD
	Total Lines of Code Added	TFA
	Churned LOC (Sum of added and changed LOC)	LOM
	Total Files Changed	TFC
	Average Changed File Size	**AVS**
	Changes by peripheral developers/total changes	**CBD**
	Average Revision per Changed File	ARF
	Total Commits per Core developer	CMACD
	Churned LOC per Core developer	LOMAD
	Total changes made per core developer	TCACD
Context of a release [71, 126]	Number of Active core developer prior to release	ACD
	Months from previous release	MPR

	Months to the next release	MNR
	Type of project (hybrid or pure)	TPR
	Type of release (major or minor)	TRS

2. Threat to Validity

As in any empirical study there are threats to the validity of data collection and analysis that need to be acknowledged and addressed appropriately.

Internal Validity. To reopen a resolved defect is common practice in OSS projects [124] thus there is high possibility that some of new defects reported are old defect from prior releases which most of them could not be observed. Our observation using reliability growth models (see Figure 22), reveals that a large proportion of accumulated defects originated from the incubator process hence prior to the early mature releases the developers were heavily preoccupied to resolve these defects. As the results in the first mature releases of both projects reveal very large number of defects reported which significantly increase the data *skewness* especially in MyFaces Core.

To address such issues in this paper after collecting valid defect data (by excluding invalid and duplicate defects) using Jira query we classified defect as a) "defect prior to release": a defect from prior release that has been targeted to be resolved for the next release, and b) "defect reported after release": a release defect that has been reported into the issue tracker after release. Later we normalized the number of defects data reported after release with accumulative number of defects prior to release. Later we called this normalized data as defect growth between releases or delta defects.

External Validity. In this work, we focus in one OSS community only; therefore, we consider the results would be valid for the projects in MyFaces and similar community in Apache family. However, we still need to validate the robustness of proposed estimation model with different OSS project communities.

4.4.3 Data Analysis Results

In this section, we outline the reliability growth model for MyFaces Core and Tobago derived from the whole life span of both projects. Later we perform the predictor selection procedures and estimate the defect growth between releases using variants of metrics (product metrics, Process Metrics, and combination).

Our two-step predictor selection process starts with predictor correlation analysis to find out a set

of the strongest correlated predictor to *DG* and then we use stepwise and backward linear regression to exclude some insignificant predictors.

Predictor Correlation Analysis

Table 20 shows the Pearson rank correlation among predictors with the dependent variable *DG*. In a first step we analyze predictors with Core data, and then we compare the results to Tobago data.

Table 20 Top 10 Predictors Correlation Analysis[33]

Predictors	Abbreviation	Project	Correlation	Sig.
Resolved Defects/Targeted Defects	RDTD	Core	0.927*	0.024
		Tobago	0.967*	0.020
Closed Defects/Targeted Defects	CDTD	Core	-0.879*	0.005
		Tobago	-0.969*	0.001
Closed Issues prior to release/Targeted issue	CITI	Core	0.901*	0.037
		Tobago	0.695	0.125
Changes by peripheral developers/total changes	CBD	Core	-0.768	0.042
		Tobago	-0.465	0.132
NPath Complexity	NPC	Core	-0.734	0.158
		Tobago	-0.272	0,602
Resolved defects prior to release	RD	Core	0.681	0.205
		Tobago	0.955*	0.030
Avg. McCabe Cyclomatic Complexity	MCC	Core	0.613	0.272
		Tobago	0.212	0.686
Abstractness	ABS	Core	0.582	0.303
		Tobago	0.243	0.064
Depth Inheritance Tree	DIT	Core	0.580	0.305
		Tobago	0.616	0.193
Method LOC	MLC	Core	0.460	0.012
		Tobago	0.345	0.155

[33] *) correlation is significant with p-value < 0.05 level (2-tailed)

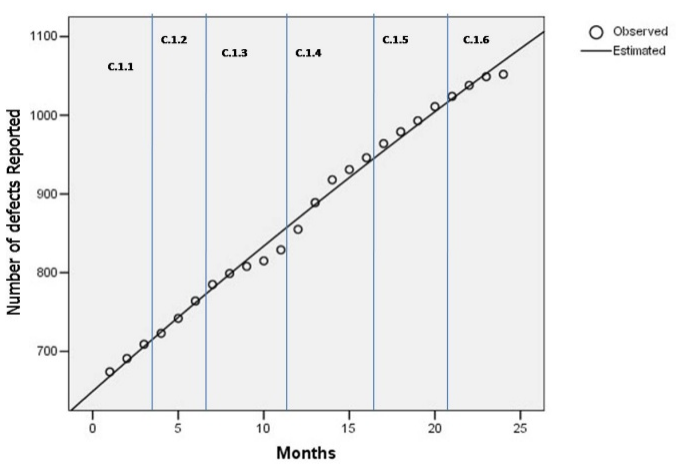
From Table 20, Process Metrics such as *RDTD, CDTD* and *CITI* considered have significant correlation to *DG. CDTD* has negative correlation with *DG* that means every peer-reviewed defect resolution may reduce the possibility of defect reported in the next release. While *RDTD* and *CITI* has positive correlation to *DG* which means that resolved defects and number of issue patched prior to a release may increase number of defects.

Reliability Growth Models

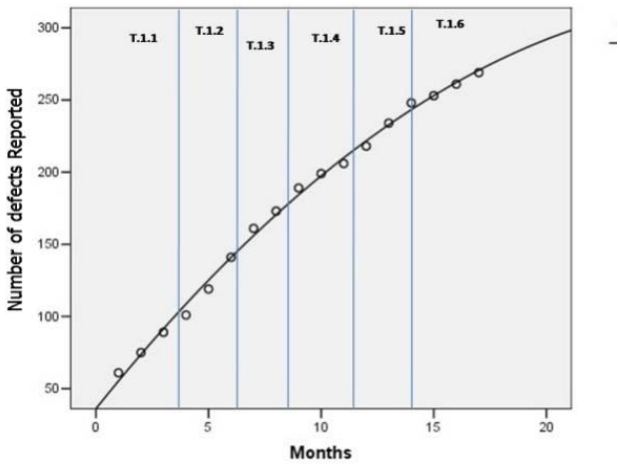
We collected defect occurrences data and use quadratic curve estimation to construct reliability growth models (RGMs) of Core and Tobago as can be seen in Figure 22.

The RGMs are useful to outline defect growth through all project life cycle, later using data from Table 20; we can perform analysis based on correlation of strong predictors with defect growth between releases. We discuss the results with an OSS expert to identify potential scenarios of the outlined RGMs

Using correlation Table 20, there are at least two scenarios that potentially accelerate the defect growth in Core as estimated in Figure 22 a as a stepwise linear defect growth which are a) new defects found in new features and patches b) a curious developer takes resolved defect prior to release and reports as new defect in current release.



RGM for MyFaces Core



RGM for MyFaces Tobago

Figure 22 Reliability Growth Models (RGM) for Myfaces Tobago and Core

Tobago has a gradual hyperbolic curve, which means potential deceleration of defect growth after 5 releases. The RGM shape of Tobago could be derived by higher number of defects closed prior to release. Using correlation data from Table 20, we can assume that that in Tobago,

the developer community spends more effort for peer reviewing defect resolutions compare to Core (see Figure 23), in which after five releases have been paid off by slower defects growth.

Figure 23 depicts the monthly performances of peer review of defect resolution (represented as defect closed per defect reported prior to release) for My Faces Core and Tobago. In average Tobago has highest level of peer review activities (Mean: 0.82) compare to Core (Mean: 0.67).

The variability of peer review practices in Core releases is higher (data are not normally distributed especially C114 with one outlier) than Tobago. The results depict in a pure OSS community such as Core although peer review of defect resolution are common practices and significantly growth over the time, however the intensity were fluctuated depend on the developers' motivation.

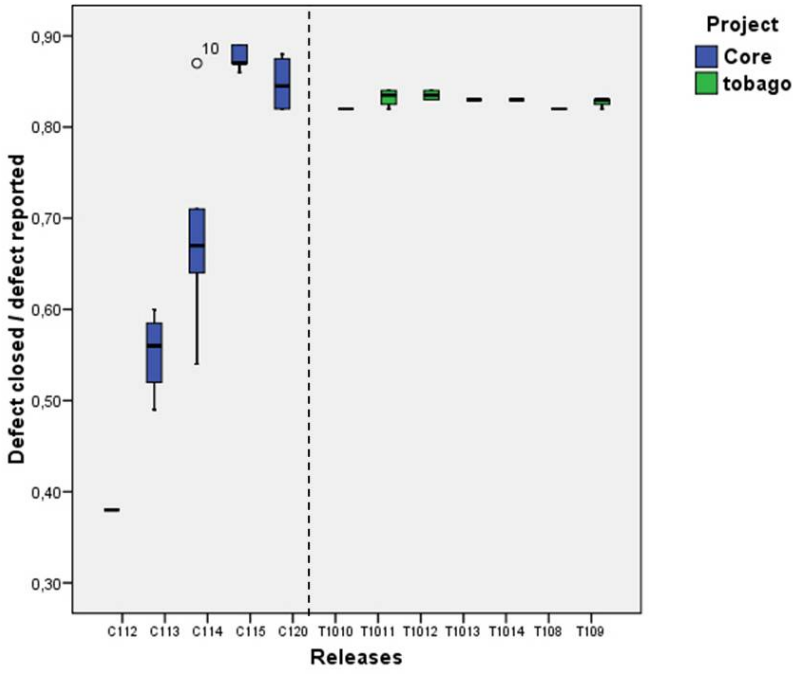


Figure 23 Ratio of monthly defect closed prior to release in MyFaces Core and Tobago

Parameters Selection and Prediction Models Construction

Stepwise regression and backward elimination procedure support selecting a subset of independent variables (predictors) from the top-ten list to form a linear model. We grouped the predictors into three groups: product, project and combination metrics, and employ the procedures for each group. Each estimation procedure was used with the three groups to fit a linear modeling expressing program defect growth in a release (*DG*).

In this case, each procedure led to selection of these following metrics: product metrics (*ABS, DIT*), Process Metrics (*CD, RDTD, CDTD, CBD*) and combination metrics (*MCC, DIT, CBD,*

ABS, RDTD, CDTD). We use these variants of metric sets from historical release and fit the data into the regression model.

The predictive quality for each estimation procedure was determined by determining the ARE values from all project releases to perform cross validation of the model. Table 21 shows the prediction results using linear regression and conclude that using Stepwise linear regression with combination metrics is superior to other prediction models.

Table 21. Comparison of Prediction Models

Prediction Model	Project	Mean (ARE)	StdDev
linear regression with product metrics	Core	0.93	1.18
	Tobago	0.12	0.08
linear regression with process Metrics	Core	0.24	0.26
	Tobago	0.06	0.05
linear regression with combination metrics	Core	0.02	0.01
	Tobago	0.04	0.01

4.4.4 Discussion of Empirical Results

Analyzing the empirical results, we derive the following implications for defect prediction in comparable OSS projects.

Contributions of Process Metrics. The results show for both MyFaces Core and Tobago that process Metrics, which are related to issue and defect resolution prior to release, have strong correlation to defect growth between releases (*DG*). Data analysis for both projects agreed for *RDTD* and *CDTD* to have strong and significant correlation with dependent variables *DG*.

For example, the increase of peer reviewed defect resolution prior to a release significantly reduces the likely number of defects in a release; while a higher number of resolved defects prior to a release are correlated to stronger defect growth. In an OSS project, this can be a result of practices such as reopening resolved defects or adding defect prior to release as a new defect in current release.

In Core *CITI* is strongly positively correlated to *DG* (p-value <0.05), this means statistically the increase of closed issues in form of patches or new features may significantly carry new defects into the next release. In summary, the correlation rank data signifies that$\{\exists\, x \in pj | r_i(x_i, Py) =$

$True$}, thus we reject hypothesis ***H0*.5.3.1**.

Accuracy of Defect Prediction using Combined Process and Product Metrics. The results in table 3 exhibit that the prediction model using combination of process and product metrics (consists of *MCC, DIT, CBD, ABS, RDTD* and *CDTD*) offers lower ARE value than using either type of metrics. Since *ARE(e(pj,pd)) < (ARE(e(pd))* thus we can reject ***H0*.5.3.2**.

In case of Apache MyFaces Core and Tobago we found strong linear correlation between selected independent variables and the dependent variable *DG*, consequently the two steps predictors' selection procedure seems straight forward to provide good prediction with only a small number of selected predictors.

4.5 Empirical Approach to Characterizing and Predicting Risk Classes of OSS Project Releases

IT managers need support in selecting OSS products based on their overall quality as well as on the quality of individual releases. From an OSS project management point of view, a typical risky situation in a software project can occur if the defectiveness level (DL) of a (planned) release is higher than the average DL of a set of reference releases. We define the DL as the weighted sum of the number of defects reported in three severity classes (critical defects, major defects and minor defects).

In this study we propose a model to characterize the defectiveness risk (DR) class of a software release based on the (predicted) DL of the release compared to the average DL of a reference set of releases, which were used for prediction model training.

We define two DR classes: 1. release candidates with higher DR (i.e., a release with a DL above the average DL of a set of reference releases) which may warrant a closer look for product improvement before actual release; and 2. Release candidates with lower DR.

We identify and evaluate as predictors several types of product metrics and development process metrics that can be efficiently collected from project repositories (e.g., issue tracker and SCM) and investigate the potential contribution of these metrics to increase the accuracy of DR class prediction models.

We conducted the empirical evaluation of the performance of prediction models (a) with several sets of metrics, (b) with parameter selection prior to model training, and (c) with data from 4 large projects listed in 2 communities of the Apache Software Foundation. For external validation we discussed the results with an OSS expert.

4.6 Research Approach

Following the Software Quality Prediction Framework in section 4.3, in this study we apply a three-step approach as framework for defect prediction of releases in the context of OSS projects.

Step 1: Design of the Empirical Study. Select the appropriate study objects and specify prediction model variables (see Section 2). Based on specified variables we collect and validate data from project data sources to ensure sufficient quality to construct a prediction model. Then we create two groups of data for *model training and model evaluation*. Common practice in software estimation is to use only one set of data to both train the model and provide internal validation, e.g., with cross validation [64, 87] or random percentage split [89, 91].

However, in practice the goal is to predict the quality of a future software release. Thus, the trained model should be externally validated with a different data set. In this study we divided the data into two groups before and after a point in time: The first group for model training consists of 70% data from older releases, while the second group for model evaluation consists of 30% of newer releases. Hence we can assure that the model is not validated with historic project data that has already been used for training. We used a 70:30 ratio following common practice in validation techniques in software estimation [91], in practice one can shift the ratios depending on the volume of data collected and the desired level of training for the prediction model.

Step 2: Defect Prediction Models Training. We fit the collected data to prediction models using statistical or machine learning techniques and measure the performance of each prediction model. The prediction models were trained using 4 classification techniques: Logistic Regression (*LG*) [87], Naive Bayes (*NB*) [64], and two tree classification techniques: *J48* [64, 87] and Random Forest (*RF*) [45]. During the training session we internally validated the prediction models with 10-fold cross validation [64, 87] by partitioning the data set into 10 equal segments. This method uses each portion once as the test set to evaluate the model built using the remaining nine segments.

One consideration from statistics is often neglected in software estimation: avoid using all collected metrics as input parameters for different prediction techniques, as the model may contain a number of weak parameters (parameters with no significant correlation to the dependent variable) which may actually reduce prediction performance [33]. In this paper we compare the performance of "unsupervised" prediction models that are trained without parameter selection with "supervised" prediction models that use a selected set of metrics. We use Stepwise

and Forward Multi-Linear Regression as parameter selection procedures following Yasunari et al. [130].

Step 3: Trained Prediction Models Evaluation. The last step in our framework is to evaluate the robustness of a trained model with data that is different from the data used for model training ("training set"). Accordingly, we evaluate the trained model with the second data set in our study ("evaluation set"). In addition to model validation, we can discuss the results with OSS experts for external validation of the prediction model results.

4.6.1 Design of Empirical Study

1. Goal

The purpose of our study is to investigate the accuracy of advanced software defect prediction models in the context of OSS projects using product and process metrics.

2. Study Objects

The study objects consist of four large Apache projects from two OSS project communities (MyFaces and Struts). Each project has been graduated from the Apache Software Foundation (ASF) Incubator process and is considered as mature project. Based on classification of the developer size in mature OSS projects by [65], all projects in MyFaces and Struts communities are categorized as large projects as they currently employ ±10 active committers (core developers) per release. These projects have been producing more than five releases within the last two years, which indicates healthy projects [1, 126]. According to these measures we assume that all study objects have sufficiently similar characteristics for cross-project validation.

Apache MyFaces[34], is a community that focuses on web-framework development; the project employs more homogeneous participants compared to Struts. Apache MyFaces consists of 4 subprojects; 3 of them (Tobago, Trinidad, and Tomahawk) are extended components that offer more functionality and flexibility than using standard Core components. Project Trinidad is a donation from Oracle to ASF, while Tobago is a hybrid project as some developers are paid and closely supported by commercial organizations. In previous section, we used MyFaces Core and MyFaces Tobago data for predicting defect growth between releases. In this study we extend the object studies into all projects within MyFaces community.

Apache Struts[35] is a Java web applications framework; in April 2008 the project offered two

[34] Apache MyFaces projects website: http://myfaces.apache.org/

[35] Apache Struts project website: http://struts.apache.org/

major versions: Struts 1 and Struts 2. Struts 1 has been recognized as the most popular web application framework for Java with proven solutions to common problems. Struts 2 was originally known as WebWork 2. After working independently for several years, the WebWork and Struts communities joined forces to create Struts 2. For this study we selected Struts 2 releases as a study object.

Our observation focuses on process and product metrics and defect data on release level. All metrics were collected from a total of 34 releases. From these 34 releases, we use 24 (70%) older releases for prediction model training and the remaining 10 (30%) releases for evaluation of the trained model.

3. Variable Specifications

The measurement model defined for the empirical study consists of independent and dependent variables (see Tables 3 and 4). We define the independent variables as: (a) selection of input parameters from a group of collected metrics (product and process metrics) and (b) context parameters consisting of project sponsorship type (pure or hybrid), release type, number of active core developers and period of release.

The variable specification in this study followed similar practices in defect and risk prediction in project level [130] which define the dependent variables (estimators) as likelihood of falling into a class representing certain level of project quality (i.e., defectiveness level).

Dependent Variables

As in our context of study, from a project management point of view, we define the defectiveness level (DL) of a release with relation to the number and severity of defects of the release. We characterize the risk of a release in one of two defect risk (DR) classes: the class of "lower-risk releases" (or *LRR*) consists of releases with a DL that is not higher than the average DL in a set of reference releases; the class "higher-risk releases" (or *HRR*) consists of releases with higher DL than the reference release set average.

Independent Variables

We focus on investigating the impact of different classes of metrics on DR class prediction. A prediction model can be generalized as $P(Y|x_1,\ldots,x_n)$ and predicts the dependent variable Y as the likelihood of a release for being in the "higher risk" HRR, using a set of independent variables $(x_1,\ldots,x_n)$, i.e., product metrics, process metrics or combined metrics. To construct the dependent variable, first, we need to calculate the DL based on the number of defects in a set of 3 defect severity classes. From defect categories in the *Jira* documentation[36] we classify the

[36] Jira documentation can be found at http://www.atlassian.com/software/jira/docs/v3.12.3/

reported defects based on their severity into 3 classes: Critical, Major, and Minor. Defect severity is typically set by the developers in an OSS project, who review and validate the reported defects in the issue tracker.

As the basic metrics to measure DL of a software release (see eqn. 1), we calculate the number of defects in each severity class: D_1 is the number of reported severe defects that related to security (critical) and fault (blocker) which may endanger system stability; D_2 is the number of major or normal defects; and D_3 is the number of defects related to minor and trivial (cosmetics) work. Later we assign weight factors (α_1, α_2, α_3) for each severity class to calculate the overall defectiveness level (DL) of a release:

$$DL = \propto_1 D_1 + \propto_2 D_2 + \propto_3 D_3 \qquad \textit{Eq. 11}$$

To determine the DR class we cluster the release data (r_i) into 2 classes based on the average DL in a set of reference releases (see eqn. 2), similar to the experiment procedure in [130] for estimating risky projects.

$$DR(r_1)\begin{cases} HRR, & DL_i > Mean\,(DL_1, \dots, DL_n) \cap (1 \le i \le n) \\ ELSE & LRR \end{cases} \qquad \textit{Eq. 12}$$

Metrics to Evaluate Prediction Models Performance

For measuring the performance of a defect prediction model, we use the standard measures *Precision, Recall, and F-Measure* commonly used in the machine learning and data mining communities [64].

a. **Precision (PC)** is defined as the ratio of the number of releases predicted correctly as *HRR* release or true positives (*TP*) and the total number of releases predicted as HRR in the study (*TP*+*FP*), where *FP* is the number of false positives.

$$Precision = \frac{TP}{TP + FP} \qquad \textit{Eq. 13}$$

b. Recall (RC) is defined as the ratio of the number of releases correctly predicted as HRR (TP) to the actual number of releases classified as HRR in the data set (TP+TN), where TN is the number of true negatives.

$$Recall = \frac{TP}{TP + TN} \qquad \textit{Eq. 14}$$

To perform well, a model must achieve both high precision and high recall. The higher the precision is, the less effort is wasted on testing and inspecting low-risk software releases; and the higher recall is, the fewer defects go undetected in high-risk releases [64].

c. F-Measure (FM) is defined as the weighted harmonic mean of precision and recall, which considers precision and recall equally important.

$$F - Measure = 2 * \frac{Recall * Precision}{Recall + Precision} \qquad \textit{Eq. 15}$$

Often, there a trade-off can be observed between precision and recall. For example, if a model predicts very few, e.g., only one release as HRR and the prediction turns out to be correct, the model's precision will be 1 but recall will be lower; in contrast if a prediction model predicts all releases in HRR; its recall will be 1 however the precision will be lower. Hence, in this study, we also apply the F-measure to evaluate the performance of prediction models, and rate model prediction with F-Measure > 0.6 as sufficiently accurate as suggested by Koru et al. [64].

4. Research Issues and Research Hypotheses Formulation

From contributions of and limitations in related work we derive the following research issues (RIs).

RI5.4.1*: Prediction model accuracy to identify higher risk releases increases when using both product and process metrics.* In an OSS project context with high release frequency, we expect defect prediction performance to be more accurate based on a combination of process and product metrics than based on only either type of metrics.

The corresponding ***null hypothesis H05.4.1*** is: There is no significant difference between the F-Measure (see section 3.3) of higher DR class predictions from models that use a combination of process and product metrics and from models that use only process or product measures.

If $\{y_1,\ldots,y_n\}$ is a set of either only *n* product metrics (*PD*) or only *n* process metrics (*PR*), and $\{x_1,\ldots,x_m\}$ is a set of *m* combined metrics (*C*), and *P* is likelihood of *Y* defectiveness risk class of a release, *then* the respective null hypothesis can be formulated as:

H05.4.1:

$$\left\{\forall\,(x \in C\ \cap y\ \in PD\ \cap z\ \in PR\,)\middle|\begin{matrix} FMeasure\big(P(Y|x_1,\dots,x_n)\big) \approx \\ FMeasure\ \big(P(Y|y_1,\dots,y_m)\big) \approx FMeasure\ \big(P(Y|z_1,\dots,z_m)\big)\end{matrix}\right\}$$

RI5.4.2*: Supervised defect prediction of higher risk releases is more accurate than unsupervised defect prediction.* Derived from best practice in statistics for estimation [33], we expect supervised defect prediction to improve defect prediction performance and robustness by calibrating the models by selecting metrics that have significant correlation in the OSS context to the prediction target, in our case the defectiveness risk (DR) class of a release. Supervised defect prediction also can reduce the effort for data collection and analysis as a project manager can focus only on a smaller set of metrics.

The corresponding ***null hypothesis H05.4.2*** is: There is no significant difference between the F-Measures of supervised and unsupervised defect prediction models for higher DR class releases.

check whether a selected set of *n* combined metrics $\{x_1,\dots,x_n\}$ can provide better accuracy (i.e. predicting defectiveness risk class 1) compared to fit all available *m* combined metrics $\{x_1,\dots,x_m\}$ into the model. Denotes that $\{x_1,\dots,x_n\} \subseteq \{x_1,\dots,x_m\}$, thus we can formulate the null hypothesis H05.4.2 as:

$$\textbf{\textit{H05.4.2}}:\left\{\forall\,(x \in C\ \cap n \le m)\middle|\begin{matrix} FMeasure\big(P(Y|x_1,\dots,x_n)\big) \approx \\ FMeasure\ \big(P(Y|x_1,\dots,x_m)\big)\end{matrix}\right\}$$

RI5.4.3*: Cross-project defect prediction using results from RI 1 and RI 2.* In OSS development a project manager often steers several projects at a time and wants to compare the quality of products and processes under his supervision with similar projects [122]. Consequently, we investigate whether the findings of RI 1 and RI 2 hold a) only within the training release data set; b) with new release data from the same project; or even c) for data from similar projects that share characteristics such as size. We define sufficient prediction model accuracy with a threshold of 0.6 for the F-Measure for predicting higher risk releases and observe data sets for training, evaluation within a project, and evaluation across projects.

Hence the corresponding ***null hypothesis H0.5.4.3*** is: there is at least one project where the best selected prediction model has insufficient performance, i.e., an F-Measure < 0.6 for higher risk releases. Lets denote that S as the set of observed projects $\{p_1,\dots,p_m\}$, and for each project *i where* $1 \le i \le m$ is described with following combined metrics $\{p_i x_1,\dots p_i x_n\} \in C$, hence we can formulate following null hypothesis:

$$\textbf{\textit{H05.4.3:}}\{\exists\ (p_i \in S \cap\ x \in C\ )\big|FMeasure\big(P(Y|p_i x_1, \ldots, p_i x_n)\big) < 0.60\}$$

4.6.2 Data Collection

1. Data Collection Procedure and Data Refinement

The observed projects employ SVN as their SCM tool, and Jira as issue tracker. Hence, to measure the code development activities before release, we retrieved 24 months of historical code using the StatSVN v.0.4.1 tool. For each release observation, we created a snapshot with a specific time range (i.e., the first time a release being announced until the announcement of succeeding release) similar to experiment in Section 4.4.2. Similar to the first study in defect prediction, after collecting the snapshot log, we also apply the StatSVN tool to collect code development metrics from each project SVN repository (Trunk directory) based on the given snapshot log, and we use Jira query commands to retrieve defect and issue data.

Following the procedure to improve the quality of collected data as described in Section 4.3 we refine the collected data by deleting invalid or duplicate issues and defects.

We apply the Eclipse Metrics v. 1.3.6 tool plug-in to collect the product metrics of the study objects (see Table 18). We use check style plug-in to collect style violations in the source code which reflects the current code quality. We perform validation of integrated data to identify there is incorrectness of data collection level that come from such heterogeneous sources.

Later, we analyze the collected data using Weka Explorer 3.4 [37] for prediction model construction and evaluation.

In total we collected 27 product metrics and 28 process metrics (including the project context metrics) that can be obtained from each project issue tracker and source code management tool (i.e., SVN). Collected metrics in this study are similar to those we used in the first case study as described in the Previous Section.

2. Creating Training and Evaluation Sets

Training sets consist of older releases for train the prediction models, evaluation sets consist of newer releases (each project donates 2 or 3 of their newest releases) that are not involved in model training but for performance evaluation.

Table 22 outlines that we trained the prediction model by using four older releases of Core (C1 to C4), six releases of Tobago (TB1 to TB6), five releases of Trinidad (TR1 to TR5), and nine

[37] Weka project can be found at: http://www.cs.waikato.ac.nz/~ml/weka/index.html, Last accessed at 10th August 2008

releases of Struts (S1 to S9).

Later we use the trained model to predict risky releases (*HRR*) in the later releases of the projects including 2 Core releases (C5, C6), 3 Tobago releases (TB7 to TB9), 2 Trinidad releases (TR6, TR7), and 3 Struts releases (S10 to S12). We use the notation such as CR1, TB1, etc for simplification reasons instead of the usual versioning names in OSS projects. For each release data set we further construct three data sets: a) with only product metrics, b) process metrics data set, and c) combined metrics data set. Consequently, in total we have 6 data sets prior to unsupervised model training and evaluation.

Table 22. Release Data Grouping.

Projects	**Releases for model training data set**	**Releases for model evaluation data set**
Core	*C1,…,C4*	*C5, C6*
Struts 2.0	*S1,…,S9*	*S10,…, S12*
Tobago	*TB1,…,TB6*	*TB7,…,TB9*
Trinidad	*TR1,…,TR5*	*TR6,…,TR7*

3. Threats to Validity

As every empirical study we identified and addressed threats to internal and external validity of the study results.

Threats to internal validity*. DL weight factors.* We use data from our prior work [121, 124] to assign the weight factors to construct the DL model, which we assume fit to represent the severity level of different classes of defects. We addressed this threat by performing sensitivity analysis of *DL* and *DR* and investigate the further impact to prediction results.

We investigated two scenarios by increasing the range between weight factors by 10% and 30%, however, the results remained stable. A reason may be that the majority of defects came from D2 (major defects) which far exceeded the number of D1 (severe defects) and D3 (minor defects) in our study context (see also Figure 1).

Threat to external validity. In this study we focused on four large Apache projects with similar size and characteristics. The selection of these homogeneous OSS projects may raise concerns whether results on the prediction models and process are also valid for other project contexts.

While we assume our approach to hold for projects similar to our study objects (i.e. under Apache umbrella, short release cycle, with active and large developer community), further work is necessary to investigate projects with strongly differing characteristics.

4.6.3 Data Analysis Results

In this section we report descriptive statistics on the empirical study objects, performance of prediction models for unsupervised (RI 5.4.1) and supervised prediction (RI 5.4.2), and finally, results from cross-project evaluation (RI 5.4.3).

1. Descriptive Statistics

Project context characterization. The observed projects in this paper are similar in the size of releases (KLOC), the involved number of active core developers/committers per release (ACD), and number of packages (NOP) as can be seen in Table 23.

Table 23. Release Sizes and Complexity.

Project	**KLOC**		**Active Core Developers**		**#Packages**	
	Mean	*StdDev*	*Mean*	*StdDev*	*Mean*	*StdDev*
Core	27.8	7.9	8.7	3.7	55.7	11.5
Struts 2.0	24.7	2.3	6.7	3.2	28.9	5.0
Tobago	25.5	1.1	6.2	2.9	37.1	4.0
Trinidad	26.3	2.5	6.6	2.1	30.7	0.6

In average Core produces larger-size releases compared to other projects, nevertheless the size differences of Core with other projects is less than 15%. As reported by [1, 83] although the number of core developers (ACD) per project release is typically ± 20% of overall developer community, nevertheless 80% of contributions within a release originate from this group.

All projects in our study have more than 5 committers involved per release; accordingly they were assisted by a larger number of peripheral developers which can be identified in each project website or release log. For the number of packages (NOP or modules), Struts 2.0 in average has the lowest number of packages per release compared to other projects in particular to Core releases. Nevertheless the total KLOC per release of each project are more and less similar, thus

a project with lower NOP typically has bigger packages.

Actual defect, DL, and DR class distributions. For training of the model we included 973 valid defects and 587 valid issues (enhancements or new feature requests) from 24 older releases. For the prediction of"higher risk" releases, we used 357 valid defects and 303 valid issues from the later 10 releases. In all observed projects, the majority of defects come from Major Defect class (D2) (> 50%) as shown in Figure 24.

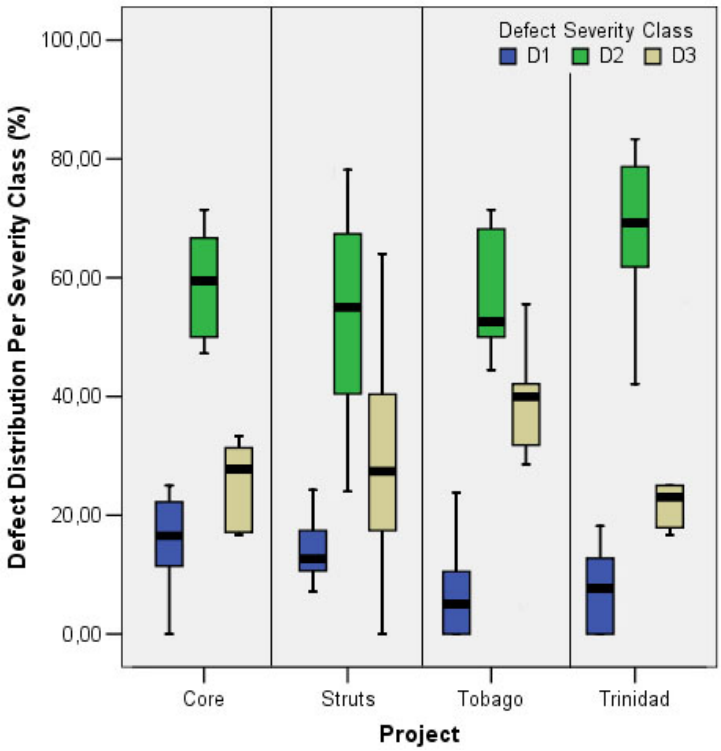


Figure 24. Defect Distribution from 34 releases in 3 severity classes.

The share of critical defects (D1) is on average lower than 20% for all projects, where Struts has the highest proportion of critical defects (*Mean:19%, StdDev: 18%*) while Tobago has the lowest one (*Mean: 6% , StdDev: 8%*). This can be one indication that in a hybrid project community such as Tobago, the community has higher awareness for release quality which results in lower numbers of critical defects compared to pure OSS projects such as Struts and Core. To identify the defectiveness risk (DR) class of each release in our study, first, we calculate the defectiveness level (DL) of each release in the first group of data. In this study we assign as weight factors: $\alpha_1 = 15$; $\alpha_2 = 10$; and $\alpha_3, = 5$ (see eqn. 1).

These weight values were based on our prior work [121, 124] on "healthy" Apache projects (such as Apache MyFaces, Apache HTTPD and Apache Tomcat projects) taking into account the average service time to resolve a more severe defect (to state Fixed or Closed) compared to a less severe defect. We take the average DL of the training set as threshold between DR classes (see eqn. 2).

As result, 8 releases out of 24 were classified as "higher risk" HRR (33.3%). Using the same threshold we characterized the remaining releases in the evaluation set: here 3 out of 10 (30%) are classified as "higher risk" HRR.

Table 24 outlines the average value of DLs per project (normalized by mean value as 100%) and number releases actually classified as LRR and HRR.

Table 24. Normalized Actual DLs and DR classes for 34 releases (Mean DL=100 %).

Projects	***Actual Release DR Classification***		***Actual LRR Release DLs (%)***		***Actual HRR Release DLs (%)***	
	LRR	***HRR***	***Mean***	***StdDev***	***Mean***	***StdDev***
Core	3	3	52	22	232	86
Struts 2.0	9	3	69	27	195	42
Tobago	6	3	70	27	245	0
Trinidad	5	2	59	31	165	0

Figure 25 illustrates the distribution of releases' defectiveness risk score in each project (normalized by mean value as 100%).

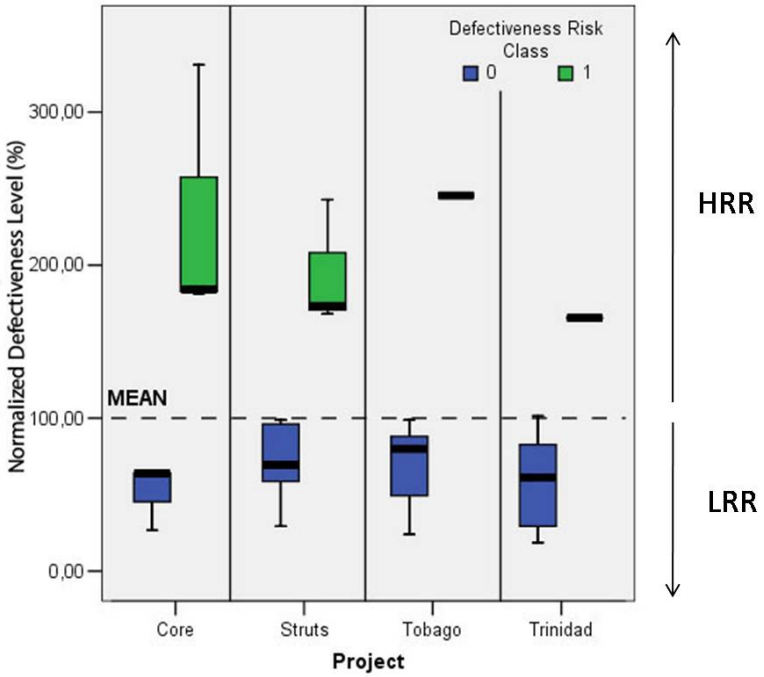


Figure 25. Actual DLs and DR classes for 34 releases

2. Unsupervised Risk Class Prediction

We perform unsupervised model training and to compare prediction model performances in our study context. Table 25 outlines the results of unsupervised training and evaluation for predicting "higher risk" HRR releases; this table compares the contribution of several *types of metrics sets*: product metrics (PD), process metrics (PR) and combined metrics (C) for each *prediction model*, i.e., Logistic Regression (LG), Naive Bayes (NB), Tree classification J48, and Random Forest (RF).

For each combination of metrics set and prediction model we calculate the evaluation measures: Recall (RC), Precision (PC), and F-Measure (FM). Highlighted cells (bold font) mark sufficient prediction accuracy (FM >= 0.6).

During training PR metrics outperformed the traditional PD metrics for most models. For three models (NB, J48 and RF) C metrics improved the prediction accuracy compared to only using either PD or PR metrics.

Table 25. Unsupervised Prediction Performance for "Higher Risk" HRR Releases.

Prediction Models		Unsupervised Models Training Data Set			Unsupervised Models Evaluation Data Set		
		PD	**PR**	**C**	**PD**	**PR**	**C**
LG	RC	0.00	**0.67**	0.33	0.67	0.67	0.67
	PC	0.00	**0.80**	0.67	0.33	0.25	0.40
	FM	0.00	**0.73**	0.44	0.44	0.36	0.50
NB	RC	0.50	**0.50**	**0.67**	0.67	1.00	0.67
	PC	0.60	**0.75**	**0.67**	0.33	0.38	0.50
	FM	0.55	**0.60**	**0.67**	0.44	0.55	0.57
J48	RC	0.33	**0.67**	**1.00**	0.33	0.33	0.33
	PC	1.00	**0.67**	**0.83**	0.50	0.50	0.50
	FM	0.50	**0.67**	**0.91**	0.40	0.40	0.40
RF	RC	0.33	0.33	**0.50**	0.00	0.33	0.33
	PC	0.50	0.67	**1.00**	0.00	0.50	0.50
	FM	0.40	0.44	**0.67**	0.00	0.40	0.40

For LG surprisingly PR metrics have better accuracy than C metrics; (Note that in this case PD show the worst possible performance in this case). Overall, J48 with PR and C metrics sets provided the best accuracy with unsupervised training.

On the second step we evaluate all trained models with the evaluation data set. However, all unsupervised models did not provide sufficient accuracy to predict DR HRR. In this case NB with combined metrics (C) offers the best performance (FM= 0.571).

3. Supervised Risk Class Prediction

Next, we investigated whether supervised modeling training and evaluation can improve prediction performance. We use two linear regression procedures (*Stepwise* and *Forward*) for parameter selection from each group of metrics (PR metrics, PD metrics, and C metrics).

From the results of each regression procedure, we selected only a significant regression model (Ftest *p-value<0.05*) with the highest *R-Square*. In this case, the procedures led to the selection of the following metrics: product metrics (*NOP, AND*), process metrics (*CDTD, RDTD*) and combined metrics (*CDTD, AND, RDTD, NOM, AVS, ASI*).

Note, that all of these metrics were not strongly correlated to each other (i.e., Pearson rank correlation > 0.8 and p-value <0.05), hence, we can say that these selected parameters are statistically independent and it is valid to include these parameters into the trained models [33].

We use these variants of metric sets from historical releases and fit the training set data to calibrate the prediction models. Later we evaluate the calibrated models using the evaluation release data set as shown in Table 26.

Table 26. Supervised Prediction Performance Results.

Prediction Models		**Supervised Models Training Data Set**			**Supervised Models Evaluation Data Set**		
		PD	**PR**	**C**	**PD**	**PR**	**C**
LG	RC	0.36	**0.63**	**0.83**	0.33	0.33	**0.67**
	PC	0.50	**0.83**	**0.71**	1.00	0.50	**0.67**
	FM	0.43	**0.71**	**0.77**	0.50	0.40	**0.67**
NB	RC	0.38	0.38	**1.00**	0.33	0.33	**1.00**
	PC	0.75	0.60	**0.86**	0.50	1.00	**0.75**
	FM	0.50	0.46	**0.92**	0.40	0.50	**0.86**
J48	RC	0.82	**0.75**	**0.83**	0.33	0.33	**0.67**
	PC	0.25	**0.86**	**1.00**	0.50	0.50	**0.67**
	FM	0.17	**0.80**	**0.91**	0.40	0.40	**0.67**
RF	RC	0.25	**0.88**	**1.00**	**0.67**	**0.67**	**0.67**
	PC	0.25	**0.70**	**0.86**	**0.67**	**1.00**	**1.00**
	FM	0.25	**0.78**	**0.92**	**0.67**	**0.80**	**0.80**

Table 26 shows that with C metrics and parameter selection *all prediction models and all data sets* show sufficient prediction accuracy in our study context, which seems to be a very promising result.

For predicting a HRR release using combined metrics (C) is better for all but one prediction models compared to only using PR or PD.

Using combined metrics, the prediction models RF and NB provide the best performance during model training (FM= 0.923 and RC =1) means that all HRR releases were correctly classified.

J48 comes in second with lower *RC* but offers the highest *PC =1,* i.e., *there* are no false positives. In evaluation phase, once again NB with combined metrics offers the best performance which reflects the ability for better data extrapolation compared to other models.

The results also depicts that Tree classification techniques (i.e., J48 and Random Forest) are only better during the training phase (data interpolation), while the results are often weaker for software defect prediction.

4. Cross-Project Evaluation of Prediction Model

To investigate whether the results in Sections 5.2 and 5.3 hold for cross-project defect prediction, we select the best model (supervised Naive Bayes with C metrics) to predict higher risk releases across projects in our study.

For this purpose we restructure the data used in RI2 into four releases group based on projects (Core, Struts, Tobago and Trinidad) and use this data set to once again validate the trained model.

Table 27. Cross-Project Evaluation with Naive Bayes for Predicting "Higher Risk" DR HRR releases.

Projects	***Model Performance for Predicting HRR releases***		
	RC	***PC***	***FM***
Core	1.00	1.00	**1.00**
Struts 2.0	1.00	1.00	**1.00**
Tobago	1.00	1.00	**1.00**
Trinidad	1.00	0.50	**0.67**

Once again, we use FM > 0.6 as the threshold for sufficient prediction model accuracy, here in Table 27 shows that using NB with selected combined metrics can be used to sufficiently accurately predict the DR classes in all projects.

The selected model can predict *fully correctly* risky releases in three projects: Core, Struts, and Tobago, i.e., without false positives or false negatives. The model shows somewhat lower performance for predicting risky releases in project Trinidad (which warrants more detailed investigation on the causes of this variance) but still with sufficient accuracy (FM > 0.6).

4.6.4 Discussion

In this section we discuss the results of the empirical study regarding research issues (RIs) and corresponding null hypotheses with related work and on OSS expert.

RI 5.4.1. *Prediction model accuracy for higher risk releases increases when using both product and process metrics.* To answer the first research question, we build distributions from the FM of prediction models (average FM) from the training and evaluation results that used the PD (PR) metrics and C metrics as reported in Table 25 and Table 26; then we compare these FM distributions for significances differences using Mann-Whitney test (with confidence level 95%) to evaluate whether there is a significant difference between models performance with different metrics sets. The results are a) overall, combined metrics improved the models performance during training session compared to product metrics from (Mean FM=0.38, StdDev 0.19) to (Mean FM= 0.70, StdDev= 0.18), with the Mann-Whitney p-value = 0.01 showing a significant performance difference; b) in the evaluation phase, combined metrics once again significantly improved the FM distribution of the model compared to process metrics from (Mean FM=0.58, StdDev 0.16) to (Mean FM= 0.70, StdDev= 0.18) with p-value=0.049. Based on these results, we conclude that using combined (C) metrics significantly improved the prediction model, thus we reject the *null hypothesis H01*.

The results also depict that although majority of models can be improved using C metrics there are several exceptions such as: a) in case unsupervised training of LG using combined metrics has lower performance to process metrics. One possible reason is that product metrics with LG in Table 25 indicates very poor performance (RC=PC=FM=0) which may impact the performance of the model when using combined metrics. b) In some model evaluation cases, i.e., the results of unsupervised evaluation of JF and RF, and supervised evaluation of RF, we found that process metrics offer the same accuracy as combined metrics. Further, the results outline that on average the performance of prediction models that use PR metrics outperformed those using PD, accordingly we confirm the finding from [87, 126] and suggest that in short-release-cycle environments such as OSS projects, product metrics are correlated to poor defect prediction performance.

RI 5.4.2. *Supervised defect prediction of higher risk releases is more accurate than unsupervised defect prediction***.** As we know that from RI1 that combined metrics provides better accuracy for prediction models, hence, we investigate whether parameter selection (Supervised training) of combined metrics can further improve the performance of unsupervised models. Using data from Table 25 and Table 26, we build distributions of the F-Measure of all prediction

models that used combined metrics and compared the unsupervised and supervised results with each other by data set (training and evaluation).

Supervised training with combined metrics improves the performance of prediction models from (Mean FM =0.67, StdDev= 0.19) to (Mean FM=0.88, StdDev=0.08) with p-value=0.046, i.e., significant average improvement. The models performances using the evaluation data set are also significantly improved from (Mean FM=0.47, StdDev=0.08) to (Mean FM=0.75, StdDev=0.09) with p-value 0.038. Note that even the supervised evaluation with combined metrics show better performance than corresponding unsupervised results with the training set in our study context. Overall, the results indicate that model calibration with parameter selection significantly increased the average of F-Measure (in each case by at least 20%) for predicting HRR releases. Thus we reject the *null hypothesis H02*.

Revisiting prior work from Moser et al [87] and critique from Norman Fenton [29] regarding the poor performance of defect prediction models for data extrapolation, hence in this paper we address this issue by proposing following improvement for advanced prediction models in an OSS project context: a) improvement through combination of product and process metrics and b) model calibration by conducting parameter selection prior to training. Parameter selection denotes that a release manager or project leading team in OSS projects can focus their work on collecting and analyzing a rather small set of representative metrics. The selection of significantly correlated metrics and data collection will require less effort compared to collect all metrics from the project repositories. In this study we conducted parameters selections across projects, in future work we consider to compare sensitivity of selecting parameters in individual projects.

The parameters selection procedures for all projects using combined metrics resulted in six parameters (see bold lines in Table 18 and Table 19) that originated from defect resolution activities (*CDTD, RDTD*), one parameter came from code development activities (*AVS*), two parameters originated from modularity of the release (*AND, ASI*), and only one parameter represents the size of release (*NOM*).

We also notice that the collected number of these selected metrics may not tell the whole story, e.g., a release manager may need to identify the reason why certain metrics have significant impact to increase the likelihood of a release for being highly defective. Thus for this study we conduct a discussion with an OSS expert from Apache to validate the findings in order to obtain feedbacks of the results. In general the expert suggests the magnitude to put focus on the improvement on particular development processes to achieve better quality products in OSS context. Later we discuss the results of impact factors analysis to obtain specific feedbacks

regarding the likely reasons for these metrics selection. We use a trained Logistic Regression model for impact factor analysis. In this discussion session, we found that the *coefficient correlation (odd ratio)* of *CDTD* is 0.778, the likely reason is that an increase of maturity level of peer-reviewed defects prior to release will decrease the likelihood of a release to be in the higher risk HRR or have less defectiveness level. *RDTD* has *coefficient correlation* of more than 1, means that an increase of defect resolution level that is not properly peer reviewed can be observed with increased likelihood of a higher risk release. The Expert also mentioned that resolved defects which were not appropriately closed through a peer review process were likely to be re-opened by developers after feedback from the community (i.e., as new defect reports). AVS has *coefficient correlation* of 0.842 which portrays that file revisions are most likely done to increase the quality of the code or to resolve reported defects after a release which confirm the finding of [82].

RI 5.4.3. *Cross-project defect prediction using results from RI 5.4.1 and RI 5.4.2.* To address RI1 and RI2 we mixed the data from all projects and divided into training and evaluation data set. The purpose is to identify which prediction model with a set of metrics can offer the best accuracy. In our case the best model is to use Naive Bayes technique with selected combined metrics. We applied this model to predict all releases in each project.

The results are surprisingly fully accurate for three projects (Core, Tobago and Struts) as shown in Table 27. We also notice that applying the selected model to project Trinidad raised concern of slightly worse model precision for this particular project. In Trinidad, although all HRR releases are predicted fully correctly, nevertheless the model suffers 50% of false positives rate, hence depicts the need for further model sensitivity analysis prior to usage in project monitoring. In overall we can conclude that the best prediction model as results of RI 1 and RI 2 can be generalized for cross project defect prediction. Hence, the corresponding *null hypothesis H03* is rejected.

4.7 Chapter Summary

Whilst a large number of studies address defect prediction, our initial literature survey found there is no specific research roadmap for software defect prediction, additionally only little support is provided about the application of defect prediction for practitioners. The second issue in software defect prediction research community is how to improve the accuracy and the reliability of prediction model in different context of projects. Hence, in this chapter we delivered following contributions:

Software Quality Prediction Research Roadmap. We conducted a systematical literature review for software defect prediction using data from several major digital libraries such as ACM Portal, IEEE Explore, and Springer. Based on the results of the review we draw a research roadmap by identifying open issues in defect prediction and provide guidelines for future research with regards to practitioners' requirements. Some of these open issues were adopted as key research issues in this thesis and later we proposed methods and conducted empirical studies to address these selected research issues.

Software Quality Prediction Framework. In this chapter we proposed a framework for conducting software defect prediction as an aid for the practitioner establishing defect prediction in the context of a particular project or organization and as a guide to the body of existing studies on defect prediction.

More importantly, the framework has been aligned with practitioners' requirements and supported by our findings from a systematical literature review on software defect prediction. The systematic literature review also served as an initial empirical evaluation of the proposed framework by showing the co-existence of the key elements of the framework in existing research on software defect prediction. The mapping of findings from empirical studies to the phases and steps of the framework show that the existing literatures can be easily classified using the framework and verifies that each of the steps is attainable.

Improving the Accuracy and Reliability of Software Defect Prediction Models. We conducted two empirical studies to evaluate our proposed Software Quality Framework (SQF) with goals a) to investigate the important factors (e.g. product and process metrics) that have strong correlation to quality improvement of a work product (e.g. to reduce the level of defectiveness of a release candidate) and b) to improve the accuracy and reliability of trained defect prediction models.

In the first case study we perform defect prediction to predict defect growth between releases with different combination of metrics to investigate which combination can provide better prediction results. As for the second case we perform unsupervised and supervised model training to investigate whether a handful set of metrics with strong correlation to quality indicator can improve the accuracy of the model. Later we performed cross project prediction, which is necessary to check the robustness of the model and its general applicability to a set of

different projects.

Current studies on the accuracy of defect prediction mostly focus on product metrics and only a few prediction models consider information on the development process. In this paper we reported on an empirical study of software defect prediction using combined product and Process Metrics from Apache MyFaces project family, following the project life over a period of two years. Process Metrics can be obtained from several QA practices in OSS project that can be observed and measured. Most of these QA practices were performed to improve the quality of the next release and to overcome each defect reported.

Our case studies reveal that in a quality-aware OSS project such as the MyFaces and Strusts community, a selected group of Process Metrics has strong correlation to defect growth between releases compared to the traditional product metrics. Furthermore, the combination of selected process and product metrics may provide more accurate prediction model, hence provide better guide the release process or indicate areas for process improvement in context of OSS project.

Major results of the empirical study were:

1. In our study context, a combination of product and process metrics provided a reasonably accurate estimation approach for both quality indicators: a) defect growth between releases and b) identifying higher risk releases. More importantly the combined metrics significantly increased the performance of the majority of the prediction models.
2. In the second case study, we found that by calibrating prediction models with parameter selection improved prediction accuracy by at least 20% for all models. The data analysis results suggested that some models which have good performance during model training but poorly performed during model evaluation can be calibrated by means of parameter selection to obtain a higher level of robustness. We also found that all supervised models with combined metrics offered sufficient accuracy (F-Measure>=0.6) during both model training and evaluation.
3. Empirical results of the second case study imply that the best prediction model (in our case, Naïve Bayes) trained from cross-project data can be used as general model to predict higher risk releases from these projects with full accuracy for most projects and with sufficient accuracy for all projects. Such prediction model generalization is particularly useful for a project manager who needs to supervise several projects.

5 CONCLUSION AND FUTURE WORK

From project manager point of view, the goal of quality evaluation is to objectively assess current quality of work product and process and to identify typical risk conditions that may occur and take necessary countermeasure to address such risks. Meanwhile, the goal of quality prediction is to predict the likely quality of candidate product for a certain time point based on current and historical data, further the prediction results act as basis for necessary product and process improvement and strategic decision process such as release decision.

In this work we focused on quality evaluation and prediction in large Open Source Software projects as these type of projects offer "openness" in both work product data and distributed processes [95]. Moreover many of OSS products have been widely recognized to have better or at least comparable quality to commercial products hence motivated many industry domains to adopt OSS products as an alternative solution.

5.1.1 Summary of Research Issues and Results for Evaluation of Distributed Development Processes Quality

One challenge for quality evaluation in a distributed development environment such as in large OSS projects context, is that the human reporting of progress becomes increasingly complex and the reliability can become doubtful [55], particularly if direct communications between project participants are not possible [25] that prohibit personal checking of the validity of high-level estimates such as the readiness of a software version for release or depict needs for further improvement. Thus for steering distributed projects, project managers need objective and trustworthy models for quality evaluation with objective data directly come from project repositories. The second challenge as suggested by Norman Fenton [29, 31] and Tim Menzies [80] is that most of current approaches are focused on assessing the software quality based on product metrics (i.e., static code metrics). They further mentioned that although static code metrics have some merits such as assessing particular quality criteria such as maintainability, security and reliability of software product, however static code metrics contain very limited information of related development processes to produce such product.

In OSS projects, many experts agree that in order to survive OSS initiatives should focus on improving the development processes with assumption that the improvement of product quality should follow. It is normal for a new OSS initiative to have a large number of defects reported for their early releases; this doesn't indicate that the project is in a bad shape, but rather indicates

that the community is active and has concern for the further product improvement. Hence in this thesis despite a concern for the product data we also put more attention to OSS development processes and to investigate these processes contributions to OSS product quality for progress evaluation.

We conducted case studies with empirical data from large OSS project under the Apache umbrella as reported in Chapter 3. In Chapter 3, based on OSS expert suggestion, we compared data from two large healthy OSS project to two challenged projects to identify what would be the symptom of "illnesses" that typically occur in an OSS project and may endanger its survivability.

The first research hypothesis that we want to investigate is that product quality improvement should be a result of correlated development processes conducted by the project participants (e.g., developers and users).

EQ1. Mockus et al [82] suggested that in large successful OSS projects, after a product release typically most of the project participants involve in defect detection/reporting, defect validation and defect fixing. Hence our first research issue is to investigate the distributed development processes which have impact to product quality improvement in term of reduction of defect counts. Based on literature survey and expert interviews, we identified several development processes and quality assurance practices that typically conducted by OSS developer and user communities (please refer to Chapter 3).

EQ2. Later we proposed several "health indicators" that can be used to assess quality of current development process in OSS projects and may provide diagnosis of current project health status. A health indicator is derived measures of two or more correlated development process (see Chapter 3) that have impact to product quality improvement.

EQ3. We conducted some intensive case studies with data from large Apache projects, and divided these projects into successful ones (healthy) and challenged ones (sick or dead) based on two OSS experts interview. Our empirical results found that in healthy projects such as Apache Tomcat and Apache HTTPD, these quality assurance activities such as defect removal or defect reporting will likely trigger some responses from the developer community i.e. through email conversations. Other improvement activity such as new code submissions will likely trigger similar responses in healthy projects, but on the other hand similar pattern could not be found in challenged projects in our study. This result signifies that in a healthy project in order to improve the quality of a "to be released" software product, it should go through several interconnected development processes or quality assurance activities which typically involved more than one project participant.

We also found several typical risks for project survivability, for example in one challenged

project the development activities were mostly triggered and steered by one core developer (committer) as the result when this committer decided to leave the project, his decision has brain drained the rest of developer community which led the project into “sickness” state before it’s finally “dying”.

The assessment of development process quality is important for project monitoring or to capture current project status; nevertheless from project point of view another important issue is to predict the quality of the product based on current status data, in order to identify the needs for improvement prior to product release date.

5.1.2 Summary of Research Issues and Results for Software Quality Prediction in Distributed Software Development Settings

Software quality prediction in particular defect prediction currently is one important topic in empirical software engineering community. Researchers in this area are typically dealing with issues regarding a) the accuracy of prediction results, and efficiency and effectiveness of prediction model constructions.

From project management point of view, defect prediction is also important to provide estimate of software quality such as to identify software components that will likely to be defective or to estimate how many defects will likely to be found in particular release candidate. This information is particularly important for product improvement prior to release, or to delay a release schedule.

In this thesis we conducted a systematical literature review to provide a profound basis for conducting defect prediction in DSD / OSS contexts. Derived from the systematic review results we outlined a research roadmap which briefly described several open research issues that still need to be further investigate by researcher in software defect prediction. Later we also use the findings to derive a systematic Software Quality Prediction Framework (SQF) which we applied to some scenarios of defect prediction with empirical data from selected large OSS projects.

QP1. In a short development cycle environment such as in many OSS projects, product metrics have weak correlation to defect count between releases, as the results recent studies in software defect prediction reported poor performances of prediction models that enabled only product metrics [87, 126]. Based on our prior investigation of development processes within OSS projects, we derived some process metrics and employed them as input parameters for the prediction models along with the traditional product metrics as reported in Chapter 4. We also investigated the potential contributions of these process metrics to software quality.. To improve

the accuracy and reliability of advanced models for objective quality prediction in the context of DSD projects we propose two steps approaches, 1) by combining development process metrics with tradition code static (product) metrics and 2) calibrating constructed model with parameter selection techniques. Our empirical results show that a) combination of product and process metrics significantly increased the performance of the majority of the prediction models and in short release cycle environment as in many OSS projects b) prediction models calibration with parameter selection improved prediction accuracy by at least 20% for all prediction models. We also found that the reliability of prediction models (model capability for data extrapolation) significantly improved by applying these two approaches (see Chapter 4). Hence the results can be used by project or release manager as early guidance for product and process improvement

QP2. To efficiently and effectively collect data from project repositories to construct objective prediction models. The efficiency of data collection can be measured by how much effort one should spend to collect and to refine data prior to model construction, to address this issue, we exploited several available data mining tools which can directly extract process and product metrics from project repositories (see Section 4.4.2). Later the parameter selection as described in **QP1**, can also increase the efficiency of data collection, as a data collector now can only focus on collecting a handful set of metrics instead of collecting and refining a lot of metrics with less accurate results. Data collection effectiveness refers to sufficient data quality to construct a prediction model. To address this issue we conducted two steps of data quality assurances, first by checking the quality of metrics collected directly from project repositories by data mining tools, second by checking the quality of integrated metrics in the same level of observation (data point) such as in release level or component level (see Section 4.4.2).

QP3. Empirical evaluation of proposed objective quality prediction models using data from large open source projects. We conducted two scenarios of defect prediction: 1) predicting defect growth between releases and 2) predicting risk classes of a release candidate. We used data from large "healthy" OSS projects such as Apache MyFaces Core, Tobago, Trinidad, Tomahawk and Apache Struts 2.0. The analysis of empirical results, in addition to improvement of prediction models accuracy and reliability as mentioned in **QP1,** we also found that there are several development processes that statistically have significant impact to increase or to decrease the quality of a release candidate. Hence release manager or project manager in OSS project can use this information to adjust certain development processes in order to improve to be released software product.

5.1.3 Future Work

Evaluation of distributed development processes quality. The concept of "health status" evaluation attempts to complement the current project monitoring model in OSS projects based on health indicators (quality) of development processes. However, major challenges for future work were identified: a) how to better formulate such indicators as the basis of meaningful notifications about the status of product quality for different stakeholders, b) how much effort seems reasonable to spend on creating, maintaining and monitoring the indicators in OSS project context; c) the need for empirical evaluation of the concept using larger set of OSS projects; and d) application of proposed concept in closed source distributed software developments. In principle our approach is applicable for commercial project, however further work should investigate the commercial project structure and the culture within the team development to reveal appropriate health indicators.

The investigation of important health indicators is just the beginning. The next step is the development of assessment methods that allow observers to get semi-quantitative measures of project health itself. This will help people to learn how these processes work in-depth, allow to enhance cooperation and give monitoring and "early-warning" capabilities to the project stakeholders. Nevertheless more works should be done to define relevant dynamics indicators, empirical rules and measurement metrics for an ongoing OSS project quality and project community assessment

Software quality prediction in distributed software development settings. In this thesis we proposed a research roadmap based on systematical literature review, nevertheless the findings should be further evaluated by external experts in software quality engineering or by involving larger collection of literatures in order to externally validate and to derive more fine grained explanation of each open research issue.

For a release manager or project leading team in an OSS project, the proposed software quality prediction framework, the predictor selection approach and defect prediction model can be a starting point for evaluating a product before release, release decisions or needs for improvements. For example to boost performance level of peer review of defect resolutions before release, or to select which release candidate should be considered for further improvement and which candidates should be dismantled.

While the initial empirical results in the study context are promising, nevertheless further work is necessary to strengthen the external validity of the findings with data from a wider range of

project types and OSS communities.

Additionally, a growing number of commercial projects focus on global software development within a professional and commercial environment, which might be comparable to highly distributed OSS projects with volunteer contributors. Thus, the continuous product improvement approach within OSS projects might be a promising approach for closed source commercial projects. As closed source commercial products usually can have a similar structure of (short) releases in a quality-aware environment, similar project and process metrics might be used for quality prediction. This approach might be a second major direction for future work based on the results of this study.

Our experiences suggest that the effort for data collection, data integration and data quality analysis using data originated from heterogeneous sources (e.g., SCM, Issue tracker, mailing list) are time consuming, and error prone tasks. Although we already utilized available data mining tools still the effort are high, thus as future work a more efficient and effective ways for data collection and data quality validation that come from multi sources are needed. One approach is by using a Semantically-enable data warehouse with capability for automatic data collection and data quality validation.

REFERENCES

1. Aberdour, M.: Achieving quality in open-source software. IEEE Software **24** (2007) 58–64
2. ASF: Apache Software Foundation Incubation Process can be found at http://incubator.apache.org/. (2008)
3. ASF: Apache Software Foundation, Meritocracy in Action, can be found at:http://www.apache.org/. (2008)
4. Basili, V., Caldiera, G., Rombach, H.D.: The Goal Question Metric Approach. . **2** (1994)
5. Basili, V.R.: The Experimental Paradigm in Software Engineering". Experimental Software Engineering Issues: Critical Assessment and Future Directives. Springer-Verlag, #706, Lecture Notes in Computer Science (1993)
6. Basili, V.R., Perricone, B.T.: Software Errors and Complexity: An Empirical Investigation. Communications of the ACM **27** (1984) 42-52.
7. Bassin, K.A., Santhanam, P.: Use of software triggers to evaluate software process effectivenessand capture customer usage profiles. The Eighth International Symposium on Software Reliability Engineering - Case Studies (1997)
8. BCG: Boston Consulting Group: The "Hacker Survey". (2002)
9. Beer, A., Ramler, R.: The Role of Experience in Software Testing Practice. 34th EUROMICRO Conference on Software Engineering and Advanced Applications SPPI Track Parma Italy (2008)
10. Biffl, S., Aurum, A., Boehm, B., Erdogmus, H., Gruenbacher, P.: Value-Based Software Engineering. Springer Verlag (2005)
11. Boegh, J., Depanfilis, S., Kitchenham, B., Pasquini, A.: A method for software quality planning, control, and evaluation. Software, IEEE **16** (1999) 69-77
12. Boehm, B.W., Brown, J.R., Lipow, M.: Quantitative evaluation of software quality. Proceedings of the 2nd international conference on Software engineering. IEEE Computer Society Press, San Francisco, California, United States (1976)
13. Born, M.: Natural Philosophy of Cause and Chance. Dover Publications New York. (1949)
14. Breiman, L.: Random Forests. Machine Learning **45,** (2001)
15. Briand, L.C., Differding, C.M., Rombach, H.D.: Practical Guidelines for Measurement-Based Process Improvement. Software Process Improvement and Practice **2** (1996)
16. Capiluppi, A., Lago, P., Morisio, M.: Characteristics of open source projects. . In Proceeding of the 7th European Conf. Software Maintenance and Reengineering (CSMR 03). IEEE CS Press (2003)
17. Chen, K., Schach, S.R., Yu, L., Offutt, J., Heller, G.Z.: Open-Source Change Logs. Empirical Softw. Engg. **9** (2004) 197-210
18. Collins, B., Fitzpatrick, B.: Successful Open Source Projects. Google Speaker Series (2008)
19. Cook, T.D., Campbell, D.T.: Quasi-Experimentation: Design and Analysis Issues for Field Settings. . Houghton Mifflin, Boston (1979)
20. Crosby, P.B.: Quality is Free: The Art of Making Quality Certain. McGraw-Hill (1979)
21. Crowston, K., Annabi, H., Howison, J.: Defining Open Source Software Project Success. The 24th International Conference on Information Systems. (2003)
22. Crowston, K., Howison, J.: The social structure of Free and Open Source software development. First Monday **10** (2005)

23. Damian, D., Dustdar, S.: International Workshop on Distributed Software Development. 13th IEEE Requirements Engineering Conference (2005)
24. David, P., Waeselynck, H., Crouzet, Y.: Open Source Software in Critical Systems. Building the Information Society (2004) 667-677
25. de Souza, C., Basaveswara, S., D., R.: Supporting Global Software Development with Event Notification Servers. Int. Workshop on Global Software Develop-ment, ICSE IEEE (2002)
26. DeMarco, T.: Controlling Software Projects. Yourden Press, New York (1982)
27. Denaro, G., Pezze, M.: An Empirical Evaluation of Fault-Proneness Models. International Conf on Software Engineering (ICSE2002) IEEE, Miami, USA (2002)
28. Dettmer, H.: Goldratt's Theory of Constraints: A System Approach to Continuous Improvement. Quality Press (1997)
29. Fenton, N., Neil, M.: A Critique of Software Defect Prediction Models. IEEE Trans. Softw. Eng. **25** (1999) 15
30. Fenton, N., Pfleeger, S.L.: Software metrics (2nd ed.): a rigorous and practical approach. PWS Publishing Co. (1997)
31. Fenton, N.E., Neil, M.: Software metrics: roadmap. the Conference on The Future of Software Engineering. ACM, Limerick, Ireland (2000)
32. Fenton, N.E., Ohlsson, N.: Quantitative Analysis of Faults and Failures in a Complex Software System. IEEE Trans. on Software Engineering **26** (2000) pp. 797-814.
33. Field, A.: Discovering Statistics Using SPSS (Introducing Statistical Methods S.). Sage Publications Ltd (2005)
34. Fielding, R.T.: Shared leadership in the Apache project. Commun. ACM **42** (1999) 42-43
35. Florac, W.A.: Software Quality Measurement: A Framework for Counting Problems and Defects. CMU-SEI (1992)
36. Flore, B., Fran, oise, D., tienne, Jean-Marie, B., Warren, S.: A socio-cognitive analysis of online design discussions in an Open Source Software community. Interact. Comput. **20** (2008) 141-165
37. Fogel, K.: Producing Open Source Software How to Run a Successful Free Software Project. O'Reilly (2005)
38. Fuggetta, A.: Open source software--an evaluation. Journal of Systems and Software **66** (2003) 77-90
39. Gacek, C., Arief, B.: The Many Meanings of Open Source. IEEE Softw. **21** (2004) 34-40
40. Galorath, D.D., Evans, M.W.: Software Sizing, Estimation, and Risk Management: When Performance is Measured Performance Improves Auerbach (2006)
41. German, D.M.: Mining cvs repositories, the softchange experience the 1st International Workshop on Mining Software Repositories (MSR2004) (2004)
42. Gilb, T.: Software Metrics. Chartwell-Bratt (1976)
43. Grady, R., Caswell, D.: Software Metrics: Establishing a Company-wide Program. Prentice Hall Engiewood Cliffs, New Jersey (1987)
44. Graves, T.L., Karr, A.F., Marron, J.S., Siy, H.: Predicting fault incidence using software change history. IEEE Transactions on Software Engineering **26** (2000) 8
45. Guo, L., Ma, Y., Cukic, B., Singh, H.: Robust prediction of fault-proneness by random forests. 15th International Symposium on Software Reliability Engineering (ISSRE 2004) (2004)
46. Happel, H.-J., Maalej, W., Stojanovi, L.: Team: towards a software engineering semantic web. Proceedings of the 2008 international workshop on Cooperative and human aspects of software engineering. ACM, Leipzig, Germany (2008)
47. Heng, C.-S., Tan, B.C.Y., Wei, K.-K.: De-escalation of commitment in software projects: Who matters? What matters? Information & Management **41** (2003) 99-110
48. Humphrey, W.: Managing the Software Process. Addison-Wesley Reading,

Massachusetts, (1989)
49. IEEE: IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. Institute of Electrical and Electronics Engineers., New York, NY (1990)
50. Jensen, F.: An Introduction to Bayesian Networks. UCL Press (1996)
51. Jones, C.: Applied Software Measurement. McGraw Hill (1991)
52. Juran, J.M., Gryna, F.M.: Quality Planning and Analysis: From Product Development Through Use. McGraw-Hill (1970)
53. Kan, S.H.: Metrics and Models in Software Quality Engineering Addison-Wesley Professional (1995)
54. Keil, M., Carmel, E.: Customer-developer links in software development. Commun. ACM **38** (1995) 33-44
55. Keil, M., Smith, H.J., Pawlowski, S., Jin, L.: 'Why didn't somebody tell me?': climate, information asymmetry, and bad news about troubled projects. SIGMIS Database **35** (2004) 65-84
56. Khoshgoftaar, T., Bhattacharyya, B., Richardson, G.: Predicting Software Errors, During Development, Using Nonlinear Regression Models: A Comparative Study. IEEE Transaction On Reliability **41** (1992) 5
57. Khoshgoftaar, T., John Munson, Lanning, D.: A Comparative Study of Predictive Models for Program Changes during System Testing and Maintenance. International Conference on Software Maintenance (1993) 72-79.
58. Kitchenham, B.: Guidelines for performing Systematic Literature Reviews in Software Engineering. (2007)
59. Kitchenham, B., Kutay, C., Jeffery, R., Connaughton, C.: Lessons learnt from the analysis of large-scale corporate databases. Proceedings of the 28th international conference on Software engineering. ACM, Shanghai, China (2006)
60. Kitchenham, B., Mendes, E.: A Comparison of Cross-company and Single-company Effort Estimation Models for Web Applications. EASE 2004 (2004)
61. Kitchenham, B., Pickard, L.M.: Evaluating Software Engineering Methods and Tools, Part 9: Quantitative Case Study Methodology. Software Engineering Notes **23** (1998)
62. Kitchenham, B.A., Mendes, E., Travassos, G.H.: Cross versus Within-Company Cost Estimation Studies: A Systematic Review. Software Engineering, IEEE Transactions on **33** (2007) 316-329
63. Kogut, B.M., Metiu, A.: Open-Source Software Development and Distributed Innovation. Oxford Review of Economic Policy **17** (2001)
64. Koru, A.G., Hongfang, L.: Building Defect Prediction Models in Practice. IEEE Softw. **22** (2005) 23-29
65. Krishnamurthy, S.: Cave or community?: An empirical examination of 100 mature open source projects. First Monday (2002)
66. Krishnamurthy, S.: Cave or community?: An empirical examination of 100 mature open source projects.: First Monday (2002)
67. Lanubile, F., Mallardo, T.: Tool Support for Distributed Inspection. Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment. IEEE Computer Society (2002)
68. Lawrie, T., Gacek, C.: Issues of dependability in open source software development. SIGSOFT Softw. Eng. Notes **27** (2002) 34-37
69. Lerner, J., Triole, J.: The simple economics of open source. Journal of Industrial Economics **52** (2002) 37
70. Li, P.L., Herbsleb, J., Shaw, M.: Finding Predictors of Field Defects for Open Source Software Systems in Commonly Available Data Sources: A Case Study of OpenBSD. Proceedings of the 11th IEEE International Software Metrics Symposium. IEEE Computer Society (2005)

71. Li, P.L., Herbsleb, J., Shaw, M.: Forecasting Field Defect Rates Using a Combined Time-Based and Metrics-Based Approach: A Case Study of OpenBSD. the 16th IEEE International Symposium on Software Reliability Engineering. IEEE Computer Society (2005)
72. Li, P.L., Herbsleb, J., Shaw, M., Robinson, B.: Experiences and results from initiating field defect prediction and product test prioritization efforts at ABB Inc. Proceedings of the 28th international conference on Software engineering. ACM, Shanghai, China (2006)
73. Li, P.L., Herbsleb, J., Shaw, M., Robinson, B.: Experiences and results from initiating field defect prediction and product test prioritization efforts at ABB Inc. the 28th International Conference on Software engineering. ACM, Shanghai, China (2006)
74. Li, P.L., Nakagawa, R., Montroy, R.: Estimating the Quality of Widely Used Software Products Using Software Reliability Growth Modeling: Case Study of an IBM Federated Database Project. the First International Symposium on Empirical Software Engineering and Measurement. IEEE Computer Society (2007)
75. Li, P.L., Shaw, M., Herbsleb, J., Ray, B., Santhanam, P.: Empirical evaluation of defect projection models for widely-deployed production software systems. SIGSOFT Softw. Eng. Notes **29** (2004) 263-272
76. Lyu, M.: Handbook of Software Reliability Engineering. McGraw-Hill (1996)
77. Manenti, F., Comino, S., Parisi, M.: From planning to mature: on the determinants of open source take-off. Discusion Paper. Universita Degli Studi Di Trento (2005)
78. Marik, V., Vrba, P., Hall, K.H., Maturana, F.P.: Rockwell automation agents for manufacturing. Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems. ACM, The Netherlands (2005)
79. Menzies, T., Di Stefano, J., Ammar, K., McGill, K., Callis, P., Chapman, R., Davis, J.: When Can We Test Less? : Proceedings of the 9th International Symposium on Software Metrics. IEEE Computer Society (2003)
80. Menzies, T., Greenwald, J., Frank, A.: Data Mining Static Code Attributes to Learn Defect Predictors. Software Engineering, IEEE Transactions on **33** (2007) 2-13
81. Merdan, M., Moser, T., Wahyudin, D., Biffl, S.: Performance Evaluation of Workflow Scheduling Strategies Considering Transportation Times and Conveyor Failures. The International Conference on Industrial Engineering and Engineering Management (IEEM). IEEE, Singapore. (2008)
82. Mockus, A., Fielding, R.T., Herbsleb, J.: Two case studies of open source software development: Apache and Mozilla. ACM Trans. Softw. Eng. Methodol. **11** (2002) 309-346
83. Mockus, A., Fielding, R.T., Herbsleb, J.: A case study of open source software development: the Apache server. The 22nd International Conference on Software Engineering (ICSE) (2000)
84. Mockus, A., Weiss, D., Zhang, P.: Understanding and Predicting Effort in Software Projects. International Conference on Software Engineering (ICSE), IEEE (2003)
85. Moore, D., McCabe, G.: Introduction to the Practice of Statistics. W.H. Freeman and Company, New York (1993)
86. Moray, N., King, B., Turksen, B., Waterton, K.: A closed-loop causal model of workload based on a comparison of fuzzy and crisp measurement techniques. Hum. Factors **29** (1987) 339-348
87. Moser, R., Pedrycz, W., Succi, G.: A Comparative Analysis of the Efficiency of Change Metrics and Static Code Attributes for Defect Prediction the 30th International Conference on Software Engineering. ACM, Leipzig, Germany (2008)
88. Nachtsheim, C.J., Kutner, M.H.: Applied Linear Regression Models. McGraw-Hill Education (2004)

89. Nagappan, N., Ball, T.: Use of relative code churn measures to predict system defect density. the 27th International Conference on Software Engineering. ACM, St. Louis, MO, USA (2005)
90. Nagappan, N., Ball, T., Zeller, A.: Mining metrics to predict component failures. Proceedings of the 28th international conference on Software engineering. ACM (2006) 452-461
91. Nagappan, N., Ball, T., Zeller, A.: Mining metrics to predict component failures. the 28th International Conference on Software Engineering. ACM, Shanghai, China (2006)
92. NASA: NASA OPEN SOURCE AGREEMENT VERSION 1.3. (2007)
93. Netcraft: Web Server Survey, can be found at http://news.netcraft.com/archives/web_server_survey.html. (2008)
94. O'Reilly, T.: Lessons from open-source software development. Commun. ACM **42** (1999) 32-37
95. O'Reilly, T.: What Is Web 2.0, Design Patterns and Business Models for the Next Generation of Software, can be found at http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html. (2005)
96. Olague, H.M., Etzkorn, L.H., Gholston, S., Quattlebaum, S.: Empirical Validation of Three Software Metrics Suites to Predict Fault-Proneness of Object-Oriented Classes Developed Using Highly Iterative or Agile Software Development Processes. Software Engineering, IEEE Transactions on **33** (2007) 402-419
97. Ostrand, T.J., Weyuker, E.J.: How to measure success of fault prediction models. Fourth international workshop on Software quality assurance: in conjunction with the 6th ESEC/FSE joint meeting. ACM, Dubrovnik, Croatia (2007)
98. Ostrand, T.J., Weyuker, E.J., Bell, R.M.: Automating algorithms for the identification of fault-prone files. Proceedings of the 2007 international symposium on Software testing and analysis. ACM, London, United Kingdom (2007)
99. Pai, G.J., Dugan, J.B.: Empirical Analysis of Software Fault Content and Fault Proneness Using Bayesian Methods. Software Engineering, IEEE Transactions on **33** (2007) 675-686
100. Phillips, J.: IT Project Management: On Track from Start to Finish. McGraw-Hill, Osborne Media. (2005)
101. Pocock, J.N.: Distributed software development and VSF. IEE Colloquium on Architectures for Distributed Development Support Environments (1991) 6/1-6/5
102. Prikladnicki, R., Audy, J.L.N., Damian, D., de Oliveira, T.C.: Distributed Software Development: Practices and challenges in different business strategies of offshoring and onshoring. Global Software Engineering, 2007. ICGSE 2007. Second IEEE International Conference on (2007) 262-274
103. Ramler, R., Wolfmaier, K.: Issues and Effort in Integrating Data from Heterogeneous Software Repositories and Corporate Databases. 2nd International Symposium on Empirical Software Engineering and Measurement (ESEM'08), Kaiserslautern, Germany (forthcoming)
104. Ratzinger, J., Pinzger, M., Gall, H.: EQ-Mine: Predicting Short-Term Defects for Software Evolution. FASE'07 Braga, Portugal (2007) 14
105. Raymond, E.S.: The cathedral and the bazaar. Can be found at: http://www.catb.org/esr/writings/cathedralbazaar/. (2003)
106. Royce, W.: Software Project Management: A Unified Framework. Pearson Education (2000)
107. Schatten, A., Tjoa, A., Andjomshoa, A., Shafazand, M.: Conference invited spech: Building a web-based open source tool to enhance project management, monitoring, and collaboration in scientific projects.: the third International Conference on Information

Integration, Web-Applications and Services (2001)
108. Schewe, K., Thalheim, B.: The co-design approach to web information systems development. Journal of Web Information System **1** (2005) 9
109. Schneidewind, N.F.: Body of Knowledge for Software Quality Measurement. IEEE Computer **vol. 35** (2002) 77-83.
110. Sengupta, B., Chandra, S., Sinha, V.: A research agenda for distributed software development. Proceedings of the 28th international conference on Software engineering. ACM, Shanghai, China (2006)
111. SourceForge: SourceForge Open Source Software Projects can be found at http://sourceforge.net/. (2008)
112. SourceForge: SourceForge Statistics, DocumentD04. Can be found at http://sourceforge.net/docs/D04/en/, 12.03.2008. (2008)
113. Sowa, J.F.: Processes and Causality. (2000)
114. Staples, M., Niazi, M.: Experiences using systematic review guidelines. Journal of Systems and Software **80** (2007) 1425-1437
115. Sunghun, K., Thomas, Z., E. James Whitehead, J., Andreas, Z.: Predicting Faults from Cached History. Proceedings of the 29th International Conference on Software Engineering. IEEE Computer Society (2007)
116. Tabachnick, B., Fidell, L.: Using Multivariate Statistics. Harper & Row Publishers, New York (1989)
117. Thai, J., Pekilis, B., Lau, A., Seviora, R.: Aspect-oriented implementation of software health. The Eighth Asia-Pacific Software Engineering Conference (APSEC'01) (2001)
118. Trochim, W.: Threats to Construct Validity. Research Methods Knowledge Base (2006)
119. Valverde, S., Theraulaz, G., Gautrais, J., Fourcassie, V.A.F.V., Sole, R.V.A.S.R.V.: Self-organization patterns in wasp and open source communities. Intelligent Systems, IEEE **21** (2006) 36-40
120. Wahyudin, D., Heindl, M., Berger, R., Schatten, A., Biffl, S.: In-Time Project Status Notification for All Team Members in Global Software Development as Part of Their Work Environments. Software Cockpit (SOFTPIT) Workshop at the 1st IEEE International Conference on Global Software Engineering (ICGSE), Munich, Germany (2007)
121. Wahyudin, D., Mustofa, K., Schatten, A., Biffl, S., Tjoa, A.M.: Introducing Health Perspective In Open Source Web-Engineering Software Projects, Based On Project Data Analysis.: The 8th International Conference on Information Integration and Web-based Applications & Services (IIWAS2006). ÖCG, Yogyakarta, Indonesia (2006)
122. Wahyudin, D., Mustofa, K., Schatten, A., Biffl, S., Tjoa, A.M.: Monitoring the "health" status of open source web-engineering projects. International Journal of Web Information Systems **3** (2007) 116 - 139
123. Wahyudin, D., Ramler, R., Biffl, S.: A Framework for Defect Prediction in Specific Software Project Contexts. the 3rd IFIP Central and East European Conference on Software Engineering Techniques (CEE-SET) Brno, Czech Republic, 2008 (2008)
124. Wahyudin, D., Schatten, A., Winkler, D., Biffl, S.: Aspects of Software Quality Assurance in Open Source Software Projects: Two Case Studies from Apache Project. 33rd EUROMICRO Conference on Software Engineering and Advanced Applications, SPPI Track (2007)
125. Wahyudin, D., Tjoa, A.M.: Event-Based Monitoring of Open Source Software Projects. In: Tjoa, A.M. (ed.): The Second International Conference on Availability, Reliability and Security, 2007. ARES 2007. (2007) 1108
126. Wahyudin, D., Winkler, D., Schatten, A., Tjoa, A.M., Biffl, S.: Defect Prediction using Combined Product and Project Metrics A Case Study from the Open Source "Apache" MyFaces Project Family. 34th EUROMICRO Conference on Software Engineering and

Advanced Applications SPPI Track Parma Italy (2008)
127. Weyuker, E.J., Ostrand, T.J., Bell, R.M.: Using Developer Information as a Factor for Fault Prediction. the Third International Workshop on Predictor Models in Software Engineering. IEEE Computer Society (2007)
128. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimental Software Engineering -- An Introduction. Kluwer Academic Publishers (2000)
129. Wong, B., Jeffery, R.: A Framework for Software Quality Evaluation. Product Focused Software Process Improvement (2002) 103-118
130. Yasunari, T., Osamu, M., Tohru, K.: An Empirical Approach to Characterizing Risky Software Projects Based on Logistic Regression Analysis. Empirical Softw. Engineering **10** (2005) 495-515
131. Zimmermann, T., Premraj, R., Zeller, A.: Predicting Defects for Eclipse. International Workshop on Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007 (2007) 9-9

DINDIN WAHYUDIN CURRICULUM VITAE

Dindin Wahyudin, MBA conducted his PhD at the Institut für Softwaretechnik und interaktive Systeme (ISIS) with an ASEA UniNet scholarship since November 2005. He received his master degree in business management at the Bandung Institute of Technology, Indonesia, where he also received his bachelor degree in Informatics. Later, he works as a researcher in the area of software process and product quality evaluation and estimation in distributed software-intensive systems and software engineering since 2005 at ISIS. His main research interests are empirical software engineering and advanced data analysis such as software defect prediction from project data and software quality management. In addition, Dindin Wahyudin participated in several research projects such as Open Source Software Health Status Monitoring, in-time notification in global software development, and quality measurement of multi-agent systems for production automation.

1. Personal Information	
Parentage	Born on 26th August 1976 in Bandung, Indonesia, married Indonesia citizen
Home address	Baumgasse 58/43, A-1030, Vienna, Austria Phone:+43-650-470-1644 eMail: dindin@ifs.tuwien.ac.at
2. Education	
PhD (Doctor of technical Sciences)	Started in November 2005, Vienna University of Technology Focus of education: Software Quality Evaluation and Prediction Models, Empirical Software Engineering, Advanced data analysis, Distributed Software Development
MBA (Master of Business Administration)	2000-2002, Bandung Institute of Technology, Indonesia Focus of education: E-Government Concept and Application for Municipalities Government Offices in Indonesia

BSc (Bachelor of Science)	1994-1999, Bandung Institute of Technology, Indonesia Focus of education: Informatics, Software Engineering, Information System
High-School	1991-1994, High School, Bandung, Indonesia Focus of education: General education, Foreign Languages (English, Arabic)

3. Work experience

Vienna University of Technology, Austria	2005-Now. Main duties: 1. Scientific research in software quality evaluation and prediction in distributed development settings, empirical software engineering 2. Supervisor for master thesis and praktika in field of empirical software engineering and software quality management
Bandung Institute of Technology, Indonesia	1999-2004. Main Duties: 1. Scientific research in software engineering, and software project management 2. Tutor for software engineering, software project management, information system and enterprise information architecture
PT LAPI Divusi, Indonesia (a Medium Scale IT consulting company)	1999-2004. Main Duties: 1. IT Consultant for many GOs and NGOs in Indonesia 2. Project manager for software developments 3. Business Analysis for software developments

During my PhD study, I have authored more than 10 international publications in the areas of information system, software engineering, quality management, software process and product improvement and software management.

- **International Journal and Book Chapter**

1. "Monitoring "Health" Status of Open Source Web Engineering Projects", Dindin Wahyudin, Khabib Mustofa, Alexander Schatten, Stefan Biffl, A Min Tjoa (2007) International Journal of Web Information System IJWIS vol 3 (1/2): 116-139
2. "In-Time Role-Specific Notification as Formal Means to Balance Agile Practices in Global Software Development Settings", Dindin Wahyudin, Matthias Heindl,

Benedikt Eckhard, Alexander Schatten, Stefan Biffl (2007) at the 2nd Central and East European Conference for Software Engineering Techniques (CEE-SET), *Springer-Lecture Notes on Computer Science* (LNCS), Poznan Poland

- **International Peer-Refeered Conferences**

3. "Predicting the Defectiveness Risk Class of a Software Release Using Product and Process Metrics An Empirical Study Based on Data from Four Large Open Source Projects"; Wahyudin, D., Biffl, S, Schatten, A., and Tjoa, A. M. (2009); Submitted to the 31st IEEE/ACM International Conference on Software Engineering (ICSE), Vancouver, Canada, 2009.
4. "A Framework for Defect Prediction in Specific Software Project Contexts", Dindin Wahyudin, Rudolf Ramler, and Stefan Biffl. (2008), in the 3rd IFIP Central and East European Conference on Software Engineering Techniques (CEE-SET) Brno, Czech Republic, 2008.
5. "Performance Evaluation of Workflow Scheduling Strategies Considering Transportation Times and Conveyor Failures", Munir Merdan, Thomas Moser, Dindin Wahyudin, Stefan Biffl, (2008). The International Conference on Industrial Engineering and Engineering Management (IEEM), Singapore. *(Nominated for Best Paper Award),* in print
6. *"*Simulation of Workflow Scheduling Strategies Using the MAST Test Management System*",* Munir Merdan, Thomas Moser, Dindin Wahyudin, Stefan Biffl, Pavel Vrba (2008). The 10th International Conference on Control, Automation, Robotics and Vision (ICARCV), Hanoi, Vietnam. *In print*
7. "Reconfiguration Process Improvement with UML and Ontology" Thomas Moser, Kamil Matousek, Klemens Kunz, Dindin Wahyudin (2008). the 24th IEEE Euromicro Software Process and Product Improvement Conference, Parma Italy. *In Print*
8. "Defect Prediction Using Combined Product and Project Metrics, A Case Study from Apache Myfaces OSS Project Family", Dindin Wahyudin, Alexander Schatten, Dietmar Winkler, Stefan Biffl (2008), the 24th IEEE Euromicro Software Process and Product Improvement Conference, Parma Italy.
9. "Ontology Based Software Quality Assurance for Component Based Configuration", Stefan Biffl, Richard Mordinyi, Thomas Moser, Dindin Wahyudin (2008) Workshop on Software Quality (WoSQ) in IEEE/ACM International Conference on Software Engineering (ICSE), Leipzig, Germany.

10. “Model-Driven Development of Intelligent Mass Customization Systems“, Kamil Matousek, Dindin Wahyudin, Stefan Biffl (2008) The 19th European Meeting in Cybernetics and System Researches (EMCSR08), Vienna Austria.
11. "In-Time Project Status Notification for All Team Members in Global Software Development as Part of Their Work Environments", Dindin Wahyudin, Matthias Heindl, Ronald Berger, Alexander Schatten, Stefan Biffl (2007), Software Cockpit (SOFTPIT) Workshop at the 1st IEEE International Conference on Global Software Engineering (ICGSE), Munich, Germany
12. "Aspects of Software Quality Assurance in Open Source Software Projects: Two Case Studies from the Apache Project", Dindin Wahyudin, Alexander Schatten, Dietmar Winkler, Stefan Biffl (2007) Proceedings of the 23rd IEEE Euromicro Software Process and Product Improvement Conference; Munich, Germany
13. "Event-Based Monitoring of Open Source Software Projects", Dindin Wahyudin, A Min Tjoa (2007) EBITS workshop, Proceeding of the 2nd IEEE International Conference on Availibility Reliability and Security (ARES), Vienna, Austria
14. "Introducing "Health" Perspective in Open Source Web-Engineering Software Projects, Based on Project Data Analysis", Dindin Wahyudin, Alexander Schatten, Khabib Mustofa, Stefan Biffl, A Min Tjoa (2006) Proceedings of the 8th International Conference on Information Integration, Web-Applications and Services; Yogyakarta Indonesia
15. "Data Integration: an Experience of Information System Migration", Inggriani Liem, Dindin Wahyudin, Alexander Schatten (2006) Proceedings of the 8th International Conference on Information Integration, Web-Applications and Services; Yogyakarta Indonesia.

APPENDIX

A1. Predicting the Number of Developer Mail Response for a Defect Status Change and a New Code Submission

In this section we describe the detailed empirical data analysis of proposed health indicators with data from four large Open Source projects in Apache Community.

A1.1 Single Project Modeling using Apache Tomcat Data

The following tables (

Table 28 to Table 30) show the summary of linear regression model to predict the number of developer mail response using Tomcat data, and test results of constructed model and predictors' significance.

Table 28. Tomcat Prediction Model Summary

R	R Square	Adjusted R Square	Std. Error of the Estimate
0.818(a)	0.669	0.650	95.20320

a Predictors: (Constant), SCM, Defect

Table 29 Tomcat ANOVA Test Results of Constructed Model

Model	Sum of Squares	df	Mean Square	F	Sig.
Regression	641295.848	2	320647.924	35.377	0.000(a)
Residual	317227.731	35	9063.649		
Total	958523.579	37			

a Predictors: (Constant), SCM, Defect

b Dependent Variable: Email

Table 30 Tomcat Coefficients and Predictors Test Results

Model	Unstandardized Coefficients		Standardized Coefficients		Sig.
	B	Std. Error	Beta	t	

(Constant)	62.853	47.209		1.331	0.192
Defect	0.323	0.124	0.327	2.613	0.013
SCM	0.862	0.188	0.572	4.577	0.000

a Dependent Variable: Email

A2.2 Single Project Modeling using Apache HTTPD Data

The following tables (Table 31 to Table 33) show the summary of linear regression model to predict the number of developer mail response using HTTPD data, and test results of constructed model and predictors' significance.

Table 31. HTTPD Prediction Model Summary

R	R Square	Adjusted R Square	Std. Error of the Estimate
0.768(a)	0.590	0.567	87.22455

a Predictors: (Constant), SCM, Defect

Table 32. HTTPD ANOVA Test Results of Constructed Model

Mode	Sum of Squares	df	Mean Square	F	Sig.
Regression	383527.466	2	191763.733	25.205	0.000(a)
Residual	266284.245	35	7608.121		
Total	649811.711	37			

a Predictors: (Constant), SCM, Defect

b Dependent Variable: Email

Table 33. HTTPD Coefficients and Predictors Test Results

Model	Unstandardized Coefficients		Standardized Coefficients	t	Sig.
	B	Std. Error	Beta		
(Constant)	42.267	56.430		0.749	0.459
Defect	0.637	0.157	0. 495	4.069	0.000

SCM	0.583	0.176	0.404	3.318	0.002

a Dependent Variable: Email

A2.3 Cross Project Modeling using Apache HTTPD and Apache Tomcat Data

The following tables (Table 34 to Table 36) show the summary of linear regression model to predict the number of developer mail response using HTTPD and Tomcat data, and test results of constructed model and predictors' significance.

Table 34. Cross HTTPD-Tomcat Prediction Model Summary

R	R Square	Adjusted R Square	Std. Error of the Estimate
0.783(a)	0.612	0.602	92.43275

a Predictors: (Constant), SCM, Defect

Table 35. Cross HTTPD-Tomcat ANOVA Test Results of Constructed Model

Model	Sum of Squares	df	Mean Square	F	Sig.
Regression	985766.478	2	492883.239	57.689	0.000(a)
Residual	623698.403	73	8543.814		
Total	1609464.882	75			

a Predictors: (Constant), SCM, Defect

b Dependent Variable: Email

Table 36. Cross HTTPD-Tomcat Coefficients and Predictors Test Results

Model	Unstandardized Coefficients		Standardized Coefficients	t	Sig.
	B	Std. Error	Beta		
(Constant)	76.195	34.890		2.184	0.032
Defect	0.413	0.094	0.389	4.388	0.000
SCM	0.726	0.130	0.493	5.563	0.000

a Dependent Variable: Email

A2.4 Single Project Modeling using Apache Xindice Data

The following tables (Table 37 to Table 39) show the summary of linear regression model to predict the number of developer mail response using Xindice data, and test results of constructed

model and predictors' significance.

Table 37. Xindice Prediction Model Summary

R	R Square	Adjusted R Square	Std. Error of the Estimate
0.684(a)	0.468	0.438	27.13077

a Predictors: (Constant). CVS. defect

Table 38. Xindice ANOVA Test Results of Constructed Model

Model	Sum of Squares	df	Mean Square	F	Sig.
Regression	22682.721	2	11341.360	15.408	0.000(a)
Residual	25762.753	35	736.079		
Total	48445.474	37			

a Predictors: (Constant). CVS. defect

b Dependent Variable: Email

Table 39. Xindice Coefficients and Predictors Test Results

Model	Unstandardized Coefficients		Standardized Coefficients	t	Sig.
	B	Std. Error	Beta		
(Constant)	17.133	5.873		2.917	0.006
defect	-2.623	0.875	-0.424	-2.998	0.005
CVS	3.270	0.590	0.783	5.541	0.000

a Dependent Variable: Email

A2.5 Single Project Modeling using Apache Slide Data

The following tables (Table 40 to Table 42) show the summary of linear regression model to predict the number of developer mail response using Slide data, and test results of constructed model and predictors' significance.

Table 40. Slide Prediction Model Summary

R	R Square	Adjusted R Square	Std. Error of the Estimate
0.875(a)	0.766	0.753	98.34407

a Predictors: (Constant). CVS. defect

Table 41. Slide ANOVA Test Results of Constructed Model

Model	Sum of Squares	df	Mean Square	F	Sig.
Regression	1109904.916	2	554952.458	57.380	0.000(a)
Residual	338504.479	35	9671.557		
Total	1448409.395	37			

a Predictors: (Constant). CVS. defect

b Dependent Variable: Email

Table 42. Slide Coefficients and Predictors Test Results

Model	Unstandardized Coefficients		Standardized Coefficients	t	Sig.
	B	Std. Error	Beta		
(Constant)	54.599	25.054		2.179	0.036
defect	1.329	0.589	0.207	2.258	0.030
CVS	1.609	0.193	0.762	8.328	0.000

a Dependent Variable: Email

A1. Statistical Methods for Software Quality Prediction

In this section we describe briefly the statistical methods used for conducting software quality prediction with empirical data from OSS projects.

A1.2.Multiple Linear Regression Techniques

Multiple linear regression[38] attempts to model the relationship between two or more explanatory variables and a response variable by fitting a linear equation to observed data. Every value of the independent variable x is associated with a value of the dependent variable y. Formally, the model for multiple linear regressions, given n observations, is [88]

$$y_i = \beta_o + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \beta_i \quad for\ i = 1,2,\ldots,n. \qquad Eq.\ 16$$

In the least-squares model, the best-fitting line for the observed data is calculated by minimizing the sum of the squares of the vertical deviations from each data point to the line (if a point lies on the fitted line exactly, then its vertical deviation is 0). Because the deviations are first squared, then summed, there are no cancellations between positive and negative values. The least-squares estimates $\beta_0, \beta_1, \ldots \beta_p$ are usually computed by statistical software.

The values fit by the equation $\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}$ are denoted $\hat{y}_i$, and the residuals e_i are equal to $y_i - \hat{y}_i$, the difference between the observed and fitted values. The sum of the residuals is equal to zero.

A1.2.Classification Techniques

Logistic Regression

Logistic regression is a model used for prediction of the probability of occurrence of an event by fitting data to a logistic curve. *Logistic Regression* has been proven to have sufficient capability for data extrapolation which are necessary for prediction with different dataset that are not used during the training session. Yasunari et al, [130] outlined the basic model of logistic regression which is based on following equation (see eq. 17).

[38]Multiple Linear Regression Manual can be found at the http://www.stat.yale.edu/Courses/1997-98/101/linmult.htm (Last Accessed 20 January 2008)

$$P(Y|x_1, x_2, .., x_n) = \frac{e^{b_0+b_1x_1+b_2x_2+\cdots+b_nx_n}}{1+e^{b_0+b_1x_1+b_2x_2+\cdots+b_nx_n}} \quad \textit{Eq. 17}$$

$x_1 \ldots x_n$ represent the independent variables, which are collected candidate predictors for defect estimation. While, $b_o,\ldots,b_n$ define regression coefficients of the independent variables. Y represents the predicted dependent variable as a binary value such as a module for being *defective* or *not*.

J48

J48 is a decision-tree learner Decision-tree learners generate a simple tree structure where nonterminal nodes represent tests on one or more attributes and terminal nodes reflect decision outcomes. As described in J48 Manual[39], J48 has the useful feature of generating tree-based models that human experts can easily interpret. We can summarize the general approach, as the following:

1. Choose an attribute that best differentiates the output attribute values.
2. Create a separate tree branch for each value of the chosen attribute.
3. Divide the instances into subgroups so as to reflect the attribute values of the chosen node.
4. For each subgroup, terminate the attribute selection process if:
    a. All members of a subgroup have the same value for the output attribute, terminate the attribute selection process for the current path and label the branch on the current path with the specified value.
    b. The subgroup contains a single node or no further distinguishing attributes can be determined. As in (a), label the branch with the output value seen by the majority of remaining instances.
5. For each subgroup created in (3) that has not been labeled as terminal, repeat the above process.

The algorithm is applied to the training data. The created decision tree is tested on a test data set, provided on is available. If test data is not available, J48 performs a cross-validation using the training data. The created decision tree is then output in the Model section of the Learner Model Output.

[39] J48 Decision Tree Manual can be found at the: http://grb.mnsu.edu/grbts/doc/manual/J48_Decision_Trees.html#sec:dtso (Last Accessed 20 January 2008)

Random Forest

A random forest is a classifier consisting of a collection of tree-structured classifiers [45]. The random forest classifies a new object from an input vector by examining the input vector on each tree in the forest. Each tree casts a unit vote at the input vector by giving a classification. The forest selects the classification having the most votes over all the trees in the forest[40]. Each tree is grown as follows:

- If the number of cases in the training set is N, sample N cases at random, with replacement from the original data. This sample will be the training set for growing the tree.
- At each node, m predictors are randomly selected out of the M input variables (m _ M) and the best split on these m predictors is used to split the node. The value of m is held constant during the forest growing. By default, m = √M (to achieve near optimal results).
- Each tree is grown to the largest extent possible. There is no pruning. When the training set for the current tree is drawn by sampling with replacement, about one-third of the cases are left out of the sample. This oob (out-of-bag) data is used as a test set to get an unbiased estimate of the classification error. Therefore, there is no need for cross-validation or a separate test set to get an unbiased estimate of the test set error. The out-of-bag estimates are unbiased [14].

Random forest is a good candidate for software quality prediction, especially for large-scale systems, because [45]:

- It is reported to be consistently accurate when compared with current classification algorithms.
- It runs efficiently on large data sets.
- It has an efficient method for estimating missing data and retains accuracy when a large portion of the data is missing.
- It gives estimates of which attributes are important in the classification.

[40] Random Forest Manual can be found at the http://www.stat.berkeley.edu/users/breiman/RandomForests (Last Accessed 20 January 2008)

Naive Bayes

From the Naive Bayes clasiffier manual[41] defines Naive Bayes as a rule generator based on Bayes's rule of conditional probability. It uses all attributes and allows them to make contributions to the decision as if they were all equally important and independent of one another, with the probability denoted by the equation:

$$\Pr[H|E] \frac{\Pr[E_1|H] \cdot \Pr[E_2|H] \ldots \Pr[E_n|H]}{\Pr[E]} \qquad Eq.\ 18$$

Where Pr[A] denotes the probability of event A, Pr[A|B] denotes the probability of event A conditional on event B , E_n is the *n* th attribute of the instance, H is the outcome in question, and *E* is the combination of all the attribute values.

In the case of categorical input for evidence class E_i, $\Pr[E_i|H]$ is simply the number of instances in the data where the evidence in category E_i divided by the total number of instances in the dataset.

Characteristic of Naive Bayes:

- Simple technique results in high accuracy, especially when combined with other methods.
- Treats variable as independent and equally important, which can cause skewed results, especially if many of the variables are interrelated, as that relation will have a greater effect on the decision, for better or for worse.
- Naive Bayes classification does not allow for categorical output attributes

[41] Naive Bayes Manual can be found at the http://grb.mnsu.edu/grbts/doc/manual/Naive_Bayes.html (Last Accessed 20 January 2008)

A3. In Time Notification Tool Support for Distributed Development Processes

In highly distributed software development environment, demands effective collaboration and communication among the team members to deliver good quality software. Project manager may need summarized reports on project performance, software quality status and quality predictions of future product, a quality assurance (QA) person may want to monitor detailed reports test performance over the time to determine the quality status of software artifacts before an approaching release; a developer should receive immediate feedback if a change in his work causes a quality problem with other (concurrently evolving) components of the software product.

In this line of work we adopt the concept of role-specific in-time notification on the status of project artifacts supported by an event-driven monitoring infrastructure. We extend this concept with providing the team members with relevant notifications in the set of tools they typically use, e.g., as part of the software development environment rather than in a separate management or collaboration tool. We argue that by providing such notification well represented as part of team member's work tools will better support collaboration among the team members during distributed development process (see Figure 26).

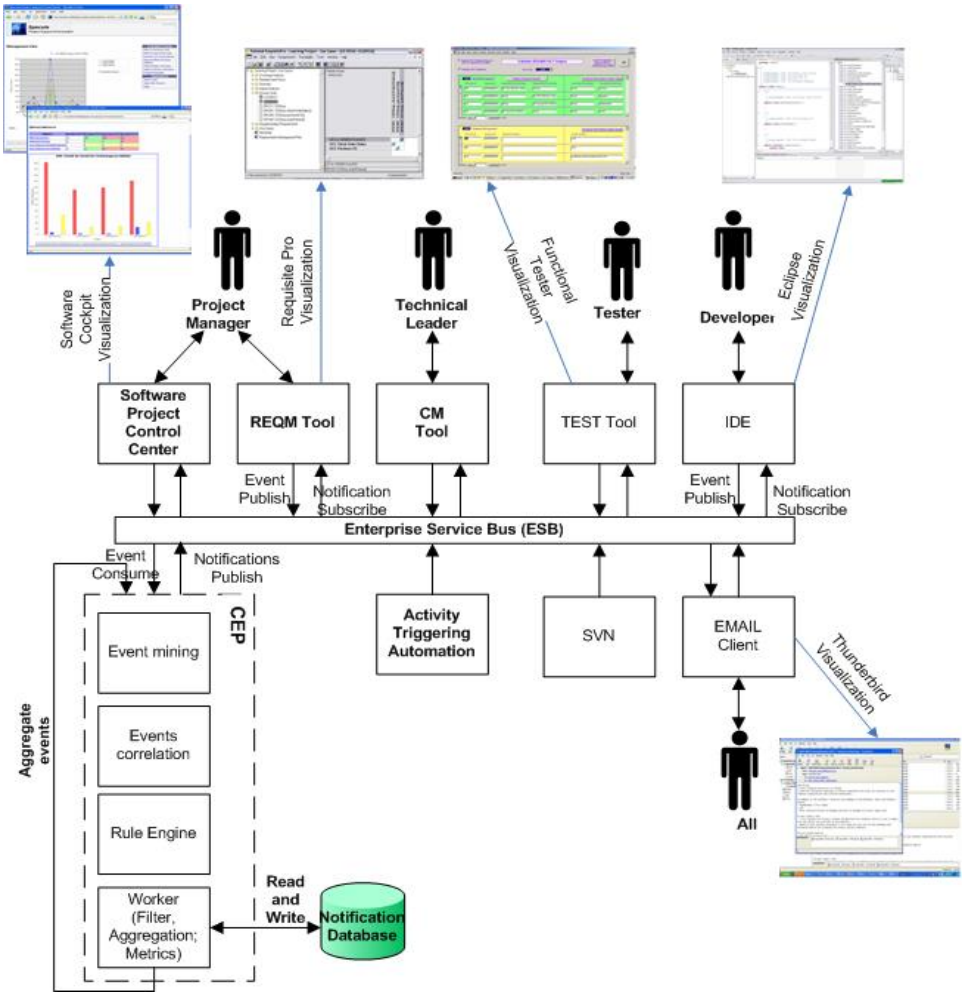


Figure 26. Integrated tool support for In-time Role-Specific Notification in Agile-GSD settings [120]

The following two papers outline our contributions in providing data collection and notification tool in globally distributed software development;

Paper 9: **Wahyudin D.,** Heindl, M., Berger, R., Biffl, S., Schatten, A. (2007); "In-Time Project Status Notification for All Team Members in Global Software Development as Part of Their work environments", International Conference on Global Software Engineering (ICGSE), Workshop on Measurement-based Cockpits for Distributed Software and Systems Engineering Projects (SOFTPIT), Munich, August 2007.

In this paper we suggest the needs and propose an initial concept to allow keeping all relevant roles informed using in-time notification on significant project events. Distributed team members can subscribe to specific notification services provided by project infrastructure. In a typical usage scenario these notification services promise to provide information for more effective and efficient change impact analysis and quality analysis in concurrently evolving software

development artifacts.

Paper 10: **Wahyudin D.,** Heindl, M., Eckhard, B., Schatten, A., and Biffl, S. (2007) ; "In-time role-specific notification as formal means to balance agile practices in global software development settings", in the 2nd IFIP Central and East European Conference on Software Engineering Techniques (CEE-SET), Springer LNCS, Poznan, Poland, 2007.

In this paper we extend the contributions of Paper 9 by introducing a framework to define in time notification for distributed development team members that allows a) measurement of notification effectiveness, efficiency, and cost; b) formalizing key communication in an agile distributed environment i.e., as in agile DSD and OSS projects; and c) providing a method and a tool to implement communication support. We illustrate, with an example scenario from an industry background, the concept and report results from an initial empirical evaluation.

A4. Potential Application of Quality Evaluation and Prediction Framework in Operating Software Systems

In this line of work we bring the knowledge and experiences in conducting quality evaluation and prediction in distributed software development into different context of application. The work is originated from development of an agile multi agent simulation tool (MAST) for production automation in cooperation with Rockwell Automation, Czech Technical University, and Automation and Control Institute, Vienna University of Technology. MAST was first developed by Rockwell Automation to simulate a new configuration of workshop floor prior to real implementation in hardware based systems. One requirement of MAST extension which has become our research focus is to have the capability to automatically measure the quality (performance) of the system when enabling new configuration.

From DSD point of view, MAST can be seen as distributed development environment where the agents (e.g., Machines, Conveyor belts, robots, storages, etc.) as can be seen in Figure 27 are the project participants while the work order consists of products to produce through cooperation of these agents. Using this analogy, we use the SQF approach to define the quality indicators of the system (e.g., system throughput, production effectiveness and efficiency, etc.), and to investigate the most promising parameters (e.g., workflow scheduling strategies, number of pallets, redundancy of machine functions, etc.) that have significant impact to the quality indicator using advanced statistical analysis.

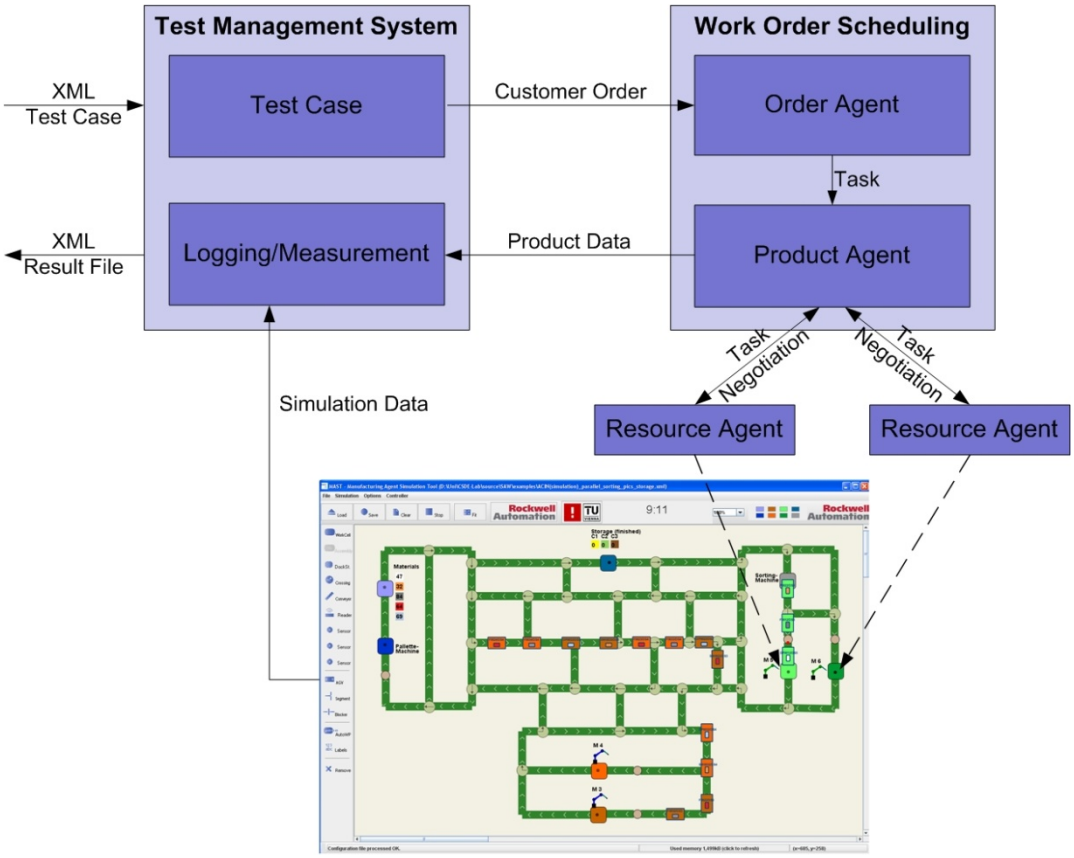


Figure 27 Overview of Extended MAST System Architecture [81]

The following two papers outline our contributions in application of quality evaluation and prediction models in MAST environment

Paper 11: Merdan, M., Moser, T., **Wahyudin, D.,** Biffl, S., (2008); "Performance Evaluation of Workflow Scheduling Strategies Considering Transportation Times and Conveyor Failures", The International Conference on Industrial Engineering and Engineering Management (IEEM), IEEE Comp Soc, Singapore.

In this paper we report on a test management system for the evaluation of a range of workflow scheduling strategies based on multi-agent negotiation, where each resource agent performs local scheduling using dispatching rules. The newly developed test management system runs test cases on the Multi Agent Simulation Tool (MAST), which provides comprehensive support for performance measurement and data analysis reporting.

Paper 12: Merdan, M., Moser, T., **Wahyudin, D.,** Biffl, S., Vrba, P., (2008); "Simulation of Workflow Scheduling Strategies Using the MAST Test Management System". In The IEEE 10th

International Conference on Control, Automation, Robotics and Vision (ICARCV), IEEE Comp Soc, Hanoi, Vietnam, 2008.

In this paper we augment the scheduling calculations to explicitly consider the transportation durations between the machines. In addition, we introduce scenarios with failures of the transport system, e.g., conveyors, which influence the variation of transport durations and evaluate the robustness of workflow scheduling strategies regarding these variations.