UNIVERSITÉ DE MONTRÉAL

ON THE IMPACT OF AFFECT IN SOFTWARE ENGINEERING

PARASTOU TOURANI
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIÆ DOCTOR
(GÉNIE INFORMATIQUE)
JANVIER 2017

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée:

ON THE IMPACT OF AFFECT IN SOFTWARE ENGINEERING

présentée par: TOURANI PARASTOU
en vue de l'obtention du diplôme de: Philosophiæ Doctor
a été dûment acceptée par le jury d'examen constitué de:

M. BELTRAME Giovanni, Ph. D., président
M. ADAMS Bram, Ph. D., membre et directeur de recherche
Mme SHARIF Bonita, Ph. D., membre
M. KHOMH Foutse, Ph. D., membre
M. ACHICHE Sofiane, Ph. D., membre

DEDICATION

This thesis is lovingly dedicated to my mother,
Akhtar Ganji.

ACKNOWLEDGEMENTS

Firstly, I would like to express my sincere appreciation to my advisor Dr. Bram Adams for the continuous support of my Ph.D study and for his patience, motivation, and immense knowledge.

Besides my advisor, I would like to thank the rest of my thesis committee: Dr. Bonita Sharif, Dr. Foutse Khomh, and Dr. Giovanni Beltrame for accepting my invitation to be jury members, Dr. Sofiane Achiche to be representative of my defense.

My sincere thanks also goes to Prof. Giuliano Antoniol, Prof. Yann-Gaël Guéhéneuc, Dr. Alexander Serebrenik, and Dr. Peter Rigby that gave me insightful advice in my academic life.

Last but not least, I would like to thank every single one of my labmates who created such a good positive atmosphere in the lab: Zohreh, Yujuan, Mahdis, Rodrigo, Azadeh, Zephyrin, Armstrong, Le, Mbarka, Ons, Laleh, Venera, and Rubén. Thank you so much for be the part of my life.

RÉSUMÉ

Selon les études scientifiques, l'une des meilleures façons d'améliorer la performance des développeurs de logiciels peut être réalisée en se concentrant sur les individus. Par exemple, Agile Manifesto a évalué les individus et leurs interactions sur les processus et les outils, tout en mettant l'accent sur le fait de fournir un environnement favorable et souhaitable aux employés. En outre, les recherches menées dans les autres domaines comme la psychologie, le comportement économique et organisationnel révèlent que le bien-être émotionnel des gens agit comme une force causale de leur productivité. Les affects d'individus, c'est-à-dire leurs émotions, leurs sentiments et leur humeur influencent sur leur comportement et leurs interactions et, par conséquent, le développement de logiciels en tant qu'activité collaborative des développeurs peuvent également être influencé par ces affects. Surtout sachant que le développement logiciel est un processus complexe, cognitif et intellectuel, ces affects peuvent avoir un impact significatif sur son succès. Jusqu'à présent, très peu d'études ont été menées dans un contexte de génie logiciel pour étudier comment les affects des développeurs pourraient influencer le processus de développement logiciel et quelles conséquences ils peuvent apporter.

Cette thèse vise à étudier le lien entre les affects des développeurs et les aspects cruciaux du processus d'ingénierie logicielle, il s'agit de la qualité du travail en résultat et le temps nécessaire pour y parvenir. À cette fin, nous avons effectué plusieurs études empiriques sur des projets "open source" populaires. Dans un premier temps, nous avons essayé d'analyser la présence d'affects dans des artefacts de génie logiciel/ de l'ingénierie du logiciel, d'inspecter leurs significations dans ce contexte et de vérifier la faisabilité du support automatique d'outil pour la détection de l'affect dans le domaine du génie logiciel/ de l'ingénierie du logiciel grâce à ces études empiriques. Ensuite, deux grandes études de cas empiriques ont été réalisées pour analyser l'impact des systèmes métriques liés affectifs. Dans la première étude, nous avons examiné l'impact de divers facteurs, y compris le sentiment de l'issue et les commentaires sur la défectuosité-pronostic d'un correctif. Ensuite, les commentaires ont laissé dans le repérage de la sortie et des dépôts utilisés comme proxy pour les discussions humaines. Les modèles explicatifs ont montré une précision et un rappel significatif, en particulier en raison de l'impact des facteurs humains de la discussion humaine, tandis que les paramètres liés au sentiment y jouent également un rôle. Dans notre deuxième étude, nous nous sommes concentrés sur la durée de l'examen du code et la résolution des problèmes, qui sont deux activités principales dans le développement de logiciels. Bien que cette étude utilise les mêmes données que l'étude précédente, les facteurs analysés l'ont plus largement été sélectionnés,

par exemple, pour les facteurs liés à l'affect, les mesures, y compris le sentiment, l'émotion et la politesse des commentaires ont été calculés. Les facteurs liés à l'affect sont apparus parmi les 10 principaux éléments influents des modèles proposés. Tandis que, des modèles explicatifs forts ont été obtenus avec une haute précision et un rappel élevé, c'est-à-dire plus de 75% pour aussi bien la précision que pour le rappel.

Les découvertes de ces deux études ont révélé que des facteurs concernant l'affect ont un impact sur de principales mesures du processus de développement de logiciels, incluant la qualité et le temps pris pour mettre fin aux activités principales, il s'agit de l'examen de code et la résolution des problèmes (publication). Cela est encore plus important pour les projets "open source", car récemment, de nombreux projets "open source" populaires et même des référentiels de code comme Github ont commencé à chercher une solution possible pour traiter des conflits, sous forme de codes de conduite. De grands projets logiciels, comme des projets "open source", sont construits et entretenus en collaboration par des équipes de développeurs et de testeurs, qui sont typiquement et géographiquement dispersés. Cette dispersion crée une distance entre des membres de l'équipe, cachant des sentiments de détresse ou (un) bonheur de leur gestionnaire, ce qui les empêche d'utiliser des techniques de remédiation de ces sentiments.

La mesure automatique de l'incidence des artefacts du génie logiciel peut aider des gestionnaires à contrôler leurs environnements d'équipe en fonction de leurs besoins et à augmenter leur conscience émotionnelle. Cela pourrait finalement conduire à une meilleure prise de décision pour ces directeurs au moment désiré à stimuler l'ambiance de leur équipe et de traiter des conflits ou des obstacles avant qu'ils deviennent perplexes et insolubles. Nous avons conduit une étude empirique pour comprendre des codes de conduite et leur rôle dans des projets "open source". Nous espérons que cette évaluation de codes de conduite et nos résultats avec des mesures d'affect automatisées pourront permettre aux praticiens de non seulement remarquer des problèmes, mais également d'évaluer des contre-mesures éventuelles à prendre.

ABSTRACT

According to scientific studies, one of the best ways to improve software developers' performance is by focusing on people. For instance, Agile manifesto valued individuals and their interactions over processes and tools, while emphasizing a supportive and desirable environment for employees. Moreover, research in other fields like psychology, economic and organizational behaviour reveals that emotional well-being of people acts as a causal force for their productivity. Affects of individuals, i.e. their emotions, feelings, and mood influence human behaviour and interactions, and therefore, software development as a collaborative activity of developers can be influenced by affect too. Especially knowing that software development is a complex, cognitive, and intellectual process, affect might have a significant impact on its success. So far, very few studies have been conducted in a software engineering context to investigate how developer affect may influence the software development process and what consequences they can bring about.

This PhD dissertation aims to investigate the link between affect of developers and important aspects of software engineering process, i.e., quality of the resulting work and the time taken to achieve it. For this purpose, we conducted several empirical studies on popular open source projects. First, we tried to analyze the presence of affects in software engineering artifacts, inspect their meanings in this context and verify the feasibility of automatic tool support for affect detection in the software engineering domain through empirical studies. Next, two large empirical case studies were done to analyze the impact of affective-related metrics. In the first study, comments left in the issue tracking and review repositories were considered as proxy for human discussions. Then, we investigated the impact of various factors including sentiment of issue and review comments on defect-proneness of a patch. Explanatory models showed significant precision and recall, especially due to the impact of human discussion factors while sentiment-related metrics play a role in them too.

In our third study, we focused on the duration of code reviewing and issue resolution, which are two major activities in software development. While this study used the same data as the previous study, the analyzed factors were selected more widely, for instance, for affect-related factors, measurements including sentiment, emotion and politeness of comments were computed. Affect-related factors appeared among the top 10 influential factors of proposed models. While, strong explanatory models were obtained with high precision and recall, i.e., mostly higher than 75% for both precision and recall.

Findings of these last two studies revealed that affect-related factors impact major metrics in

software development process including quality and time taken to finish main activities, i.e., code reviewing and issue resolution. This is even more important for open source projects, since recently, many popular open source projects and even code repository like Github started to look for a possible solution to deal with conflicts, in the form of codes of conduct.

Large software projects like open source projects, are constructed and maintained collaboratively by teams of developers and testers, who are typically geographically dispersed. This dispersion creates a distance between team members, hiding feelings of distress or (un)happiness from their manager, which prevents him or her from using remediation techniques for those feelings. Automatic affect measurement from software engineering artifacts can help managers to monitor their team environments affect-wise and increase their emotional-awareness. This finally may lead to better decision making for managers at the appropriate time to boost the atmosphere of their team and to deal with conflicts or obstacles before they are getting perplexing and insoluble. We conducted an empirical study to understand codes of conduct and their role in open source projects. We hope that this evaluation of codes of conduct and our results with automated affect measurements will be able to enable practitioners to not only notice problems, but also evaluate possible countermeasures to take.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

Emotions and feelings play an important role in the life of most people including their professional work, for instance happy people are more creative and also productive at work (Robertson and Cooper, 2011; Rao, 2010; Barron and Barron, 2013). One of the firm and famous theories proposed in 1996 (Weiss and Cropanzano, 1996), i.e., Affective Events Theory(AET), is a psychological model explaining the impact of feelings and mood on job satisfaction and job performance. That can explain why some well-known high tech companies like Google, Facebook, and Twitter offer perks to their employees at workplaces during the working hours.

Particularly, software development is dominated by human factors as a collaborative, cognitive, and intellectual activity (Mistrìk et al., 2010; Wagner et al., 2010), and studies like (de Barros Sampaio et al., 2010; Wagner and Ruhe, 2008) revealed the human related factors are among influential productivity factors in software development. Moreover, cognitive activities are impacted by affects of individuals involving in it and software engineering process cannot be considered exempt from such impact (Khan et al., 2011). Hence, it seems that feelings of people involved in software development process play an important role in the process, however, there is a lack of evidence and scientific studies on the role of affect in software engineering domain.

Finding out the presence of people's affect in software development process which is a contentious activity, Murgia et al. in 2014 in their study (Murgia et al., 2014) referred to one exemplary incident that happened in July 2013: one of the top developers (Sarah Sharp) felt offended by the project leader (Linus Torvalds) during their discussions in the Linux kernel mailing list (Brodkin, 2013). This example clarified that during software construction and maintenance, there might be situations where people get emotional and freely express their feelings. Therefore, it seems software engineering research needs to incorporate affect measurements to advance human factors study and consequently improve management styles.

More in-depth research revealed software artifacts like issue reports can carry emotions, in other words developers may express their emotions during the work through their communications (Murgia et al., 2014). Moreover, processes like issue-fixing and code review activities recently equipped with tools that people can discuss using them and their discussions being recorded. These electronic communications in addition to emails, chat rooms and, video conferencing are popular means of communications in many organizations and software projects that are dispersed geographically. By using them people are not just conveying their mes-

sage, but their feelings too (Bacchelli et al., 2010; Rigby et al., 2008). People are getting so accustomed to these communication means more and more every day, therefore it is more expected to see their actual feelings through electronic communications.

In this thesis, we planned not only to inspect but also measure the impact of affects of software stakeholders including developers on software development process considering several output values. To this purpose, we selected different software repositories including mailing lists, issue comments of issue repository and investigated whether there are symptoms of members' affects in them. After finding the feasibility of automatic affect extraction from those artifacts, various metrics related to affects of people were defined using various tools discussed later. Next, we conducted several empirical studies to study the impact of aforementioned affective metrics on the quality of resulting work and on the time spent for the work. Finally, one recent solution, adopting in open source software projects to prepare more safe and positive atmosphere for participants, has been studied.

In this chapter, we will first present definitions and differences of emotion, sentiment, affect and their quantitative measurements followed by our research hypothesis and the technologies we used. Finally, we discuss how our studies address the hypothesis.

1.1 On Affect, Emotion, Mood, Feeling, Sentiment, and Politeness

Many different terms are being used when talking about affect, some of them synonyms, while others have a different meaning. We base our definitions on those given by Matsumoto (Matsumoto, 2009) defining affect, emotion, mood, and sentiment.

Affect: **1.** "The subjective feeling or evaluative component of human experience or thought". **2.** "A transient neurophysiological response to a stimulus that excites a coordinated system of bodily and mental responses including facial expressions that inform us about our relationship to the stimulus and prepare us to deal with it in some way. The basic affects are anger, fear, surprise, happiness, disgust, and contempt". These formal definitions refer to people's opinion, attitude, appraisal or feelings toward entities, events and their attributes.

Emotion: Emotion is a synonym for affect, except that emotion shows more detail of human feeling like sadness, happiness, shame, and anger. Both of them are used automatically whenever people communicate, since it helps people convey their message or understand other people's reactions.

Mood: An affective state that persists from several minutes to several weeks which directs and colors perception, thought, and behaviour.

Sentiment: "Thoughts concerning feelings or emotions that are attached to objects or people.

Thinking about a dog, for example which one feels attached to and positive about may be a sentiment." (In particular, for each affect related to an object or person, one can feel range of sentiments from very negative to very positive.)

Politeness: "Central force in communication, arguably as basic as the pressure to be truthful, informative, relevant, and clear" (Grice, 1975). In simple words, "the ability to make all the parties relaxed and comfortable with one another."[1]

1.2 The Quantitative Measurement of Affect

This section discusses current approaches to quantitatively measure affect based on findings of Leiman's work (Leiman, 2013). Most of these approaches were developed in the area of marketing, since in the competitive world of marketing where there are many reviews left by customers towards product or services, measuring the affect of customers, has found its application. For example, it has been used in decision making systems and also in predicting some trends in future.

The basis of most of current measurement approaches is two-dimensional model of emotion comprising the dimensions of arousal and valence. Arousal represents the level/amount of physical response and can range from calm to excited and valence represents the emotional direction and spread from negative to positive. For instance, anger would be tagged by high arousal but with negative valence, while happiness is marked by high arousal and very positive valence.

However, this model has several shortcomings, like inability to target a broad range of emotional detail and nuance, and the fact that it relies on some form of self-report like through the use of rating scales to be generated.

Measuring emotions, it is generally acknowledged that there are both inner and outer signs of emotion. The outer signs are related to expressive reactions and behavioral responses, while the inner signs include both subjective feelings and physiological reactions. So, based on these manifestations, there are several ways of quantitative measuring:

- Expressive Reaction: facial, vocal and postural responses ('body language')
- Physiological Response – Autonomic Responses: blood pressure, heart rate, pupillary response, and electrodermal activity (EDA)

[1] `http://en.wikipedia.org/wiki/Politeness`

- Brain Imaging: using various forms of brain imaging like functional magnetic resonance imaging (fMRI) to study a human's emotional response to questions
- Measuring Emotion Through Opinion Mining and Sentiment Analysis: Recent developments and studies suggest that text analytics focused on opinion mining and sentiment analysis of unstructured textual data help to measure emotions.

While physiological measures appear attractive, as they directly measure physical symptoms of emotions, they are difficult and expensive to use. Therefore, economically and organizationally speaking it is impossible to study larger samples of developers in this way. Furthermore, since some of the techniques require placing consumers in rather constrained, controlled environments, one might also argue whether the obtained results are really representative of one's true emotions. For these reasons, the sentiment analysis approach has become the most lightweight, practical approach to measure emotions, even though it does not directly measure someone's emotions (just the communication records about the emotions). Note that recent development and research in the text analysis domain has enabled sentiment analysis to not simply tag text for positive or negative sentiment, but also to identify the expression of a variety of emotional states, such as curiosity, confusion, embarrassment, excitement, bemusement, shock, irritation, etc. The next section discusses how modern sentiment analysis techniques work.

1.3 Sentiment Analysis

Sentiment or affect occur automatically when people communicate, since it helps people convey their message or understand other people's reactions. This implicit behaviour is not just limited to communications in real world, but even when people interact through computer aided communications, like comments or feedback that people make on community fora or chat rooms, or in more conventional electronic media like emails (Thelwall). These sentiments are universal, in that they occur as much in politics as in business contexts (Pang and Lee, 2008). Even in software engineering (Murgia et al., 2014), identification of sentiments and emotions in software artifacts has the potential to provide indications about someone's opinions towards certain project decisions or other people.

Sentiment analysis and opinion mining is the field of study that analyzes people's opinions, sentiments, evaluations, attitudes, and emotions from written language (Liu, 2012). These emotions or attitudes are aimed towards entities such as products, services, individuals, issues, events, topics and their attributes. The automatic measurement of such attitudes from recorded (typically textual) transcripts of communication is based on a number of measures

of subjectivity and opinion in text. Those measures usually measure the polarity (positive or negative) and "strength" (also "degree") of a document. Strength shows to which degree a word, phrase, sentence or document is positive or negative towards a subject topic, person, or idea (Osgood et al., 1957). For example, a positive opinion with strength 1 is much more positive than a positive opinion with strength 0.1. Some examples of approaches to extract the above measures are sentiment analysis (Pang and Lee, 2008), opinion mining (Pang and Lee, 2008), and affect mining (Batson et al., 1992).

In general, sentiment analysis can be computed in three levels:

- *Document level*: Analysis at this level scores a positive or negative sentiment for a whole document.
- *Sentence level*: Analysis at this level studies individual sentences to classify whether a sentence expresses positive or negative sentiment.
- *Aspect/Feature level*: The definitions of document and sentence level above did not consider the specific subject that people like or dislike. However, aspect level analysis has a finer granularity, since it is not only based on *sentiment* expressed in a sentence (positive or negative), but also a *target* of the sentiment.

Sentiment analysis algorithms mainly use two approaches: machine learning or a lexical approach. With machine learning, typically a training set of text documents are taken as input to train a classifier, which then can be used on other documents to score the sentiments expressed in them (Socher et al., 2013). Lexical approaches use language information in the form of a list of known sentiment-related words, their polarities and the grammatical structure of the language, then uses those to score the sentiment of the text. Word lists or dictionaries for lexical approaches can be created manually, or could be expanded by using seed words (sentiment words) (Taboada et al., 2011).

1.3.1 Lexical Sentiment Analysis

The lexical approach is based on the assumption that the contextual sentiment analysis is the sum of sentiment orientation of each word. Positive sentiment words are used to express desired states like *nice, interesting* and negative sentiment words express undesired states like *ugly, unpleasant* (Liu, 2012). Apart from words there are also phrases that express positive or negative states. These words and phrases together are called *sentiment lexicon*, and such a lexicon can be compiled using a *manual approach, dictionary-based approach,* or *corpus-based approach.*

The manual approach is labor intensive and time consuming, and hence usually combined with automatic approaches. Dictionary-based approach uses dictionaries like WordNet to find sentiment-related words and all their synonyms and antonyms. In this approach, generally, a small set of sentiment words (seeds) is first collected. Then, based on the synonym and antonym structure of a dictionary the set is extended. In corpus-based approach , based on a given seed list of known sentiment words, other sentiment words and their orientations from a domain corpus are discovered. Turney et al. (Turney, 2002) introduced a technique that performs classification based on some fixed syntactic patterns used to express opinions.

1.3.2 Machine Learning-based Sentiment Analysis

Sentiment classification interpreted as a kind of text classification usually has three outcome classifications: positive, negative and neutral. Since sentiment classification is a kind of text classification, any existing supervised learning method can be applied, such as naïve Bayes classification or support vector machines (SVM). In this approach, engineering of a set of effective features is an effective key for sentiment classification. Features include *opinion rules*[2], *sentiment shifters*[3] and *syntactic dependency, sentiment word an phrases, part of speech, sentiment shifters,* and *sentiment dependency.*

1.3.3 Hypothesis of Thesis

Research on various domains including organizational behaviour and psychology showed the impact of employees' feelings on their performance and consequently on their organizations outcome (Robertson and Cooper, 2011; Rao, 2010; Barron and Barron, 2013). Since software engineering is a collaborative, human activity, one would expect that these results hold as well for software developers, architects or any other stakeholder of the software development process. However, how much do these factors impact software development? Given the abundance of open source development data and recent advances in automated sentiment analysis, this thesis sets out to quantify the impact of affect on the software development process. In particular, our research hypothesis is:

[2]The sentiment of a compound expression is a function of the meaning or sentiment of its constituents and the syntactic rule they are combined with

[3]Expressions that change the sentiment orientation like negation words.

First, there is a link between the affect of participants involved in open source software development and (1) the quality of and (2) time taken by their work. Second, (3) affects of participants in open source projects can be improved through concrete measures.

Measuring the affect of software engineering stakeholders and understanding its impact on software quality enables managers to gain an understanding of a team's status and prospects in further stages of the software development process, and could help them to establish baselines for comparison between different teams' projects or releases. In this way, they can evaluate and predict major issues related to quality and time as well as identify roadblocks and opportunities.

To explore this hypothesis, this thesis performs case studies on various software engineering data sources. We studied open source software projects as they are free and they publicly expose large amounts of data sources and repositories that do not only include source code, but also cover the different communication, planning and documentation media, such as mailing lists and issue repositories, utilized by open source community. We especially investigated large, well-known projects like the Apache, OpenStack, and Eclipse projects that have been studied before in many research studies.

To be more specific this thesis dealt with following sections to inspect the above hypothesis, we performed a preliminary study of the feasibility of measuring affective metrics from software artifacts automatically, 1) studied the influence of affective metrics on the quality of the commits, 2) analyzed their impact on the time taken to do code reviewing and issue resolution and, finally, 3) explored one general solution taken by open source communities to provide participants with a more comfortable and friendly environment.

1.3.4 Thesis Contributions

Studying the feasibility of affect detection from software engineering artifacts

We aim at detecting affects automatically from software engineering artifacts by mining issue repository and open source mailing lists. Instead of self-assessment of emotional states done by developers, we tried to extract their affect states from the texts they left on various software artifacts like mailing list and issue comments. Such self-assessment based approaches suffer

from problems, e.g. participants might not be comfortable enough to disclose their different feelings and also those approaches can not be scaled easily. However, our approach detecting affects from software artifacts, first of all increases affect-awareness in open source projects environments dispersed geographically while equipped with different tool, like mailing lists and issue tracking systems like Jira and Bugzilla. In addition, these results can be used to analyze the impacts of affects in software engineering process.

To this end, we run two empirical studies, one on issue comments (792 developer comments) extracted from issue repository of Apache Software Foundation(AFS), and one on mailing lists of AFS (635,906 emails from Tomcat and Ant projects). We investigated the feasibility of extracting affects from those texts written by developers and software stakeholders during their discussions. In our first study, manual evaluation of AFS issue comments showed that issue reports carry emotions, in the second study we applied the SentiStrength tool(Thelwall), to score the sentiment of emails followed by manual evaluations of the emails. Again our results showed the presence of both positive and negative sentiments in email communications of developers and users. In addition, we found that the expressed sentiments evolve across the time, which can be adopted by project management to be informed about extremely positive and negative sentiment in projects. However, we also observed that, while the overall sentiment trend seems to be meaningful, the individual sentiment measures are quite noisy.

Using MSR to understand the relation between human affectiveness and the quality of product

In order to understand the impact of affective metrics on the quality of the projects, we performed a large empirical case study on 10 OpenStack and 5 Eclipse open source projects. As a measure of quality, we built a model of defect-prone commits adopting "Just In Time"(JIT) models (Kamei et al., 2013b). JIT models have several advantages in comparison with other proposed models, since their granularity, i.e. individual commits, provides more actionable and efficient context for bug-fixing. Previously, JIT models have been applied using measures related to source code, however here we integrated measures related to human aspects who are developing the product. To this purpose, we focused on issue and review discussions as the major discussions, during software development, occurred through them. We needed to link issues and reviews to commits which was done heuristically in our work. To the best of our knowledge no work has done that before, to link both reviews, issues and commits together.

Our results showed that sentiment-related metrics also play a role in more than half of the

generated explanatory models.

Using MSR to understand the link between human affectiveness and the time taken by code reviewing and issue resolution

Since time-to-market is one of the most important success measures, software organizations have been trying to accelerate their activities continuously (Wohlin and Ahlgren, 1995a). Since issue resolution and code reviewing are two major activities in recent software development, the time spent on them can impact the whole development process time. For example, another study performed by Ortu et al. considered issue fixing time as one potential measure of the productivity in a project.

Therefore, we conducted a detailed empirical study on 10 OpenStack projects, for which we were able to link issue report data with patch and review data (see chapter 6), to analyze important factors influencing their time. To this end, first we investigated the actual interaction between issue resolution and review process, found their timing proportions and scales. To our knowledge, there have been no studies done on the interaction patterns and scaling of issue resolution time and code reviewing time. Next, we studied the impact of different factors with different dimensions, including metrics related to the human affects dimension. We found that affective-related metrics are among top influential metrics after experience and churn metrics while positive affects have negative or decreasing impact on the time (in contrast to politeness), i.e., more positive affects are related to shorter time.

Understanding Code of Conduct as a solution to improve affect

While thus far we have identified the presence of and potential impact of affect on the quality and undertaken time to code reviewing and issue resolution, here we examine one possible solution open source projects are taking to improve the affect in their community, i.e., codes of conduct. A code of conduct is a set of rules or standards articulating behaviours of participants to protect the community from offensive and unacceptable behaviours retaining a safe and friendly environment.

However, still some communities are against codes of conduct, as they think applying it leads to being censored and to a suffocating environment.

We conducted an empirical study to understand the role of code of conduct in open source projects using both quantitative and qualitative study. We presented an estimation of the proportion of the open source projects adopting codes of conduct while we tried to identify the major codes of conduct in use. Further studies including structured and semi-structured

interviews with experts were performed to understand the nature of a code of conduct, i.e., its attributes and scope, limitations, and also to identify the process and thought behind it, the reason they have been emerged in open source communities.

CHAPTER 2 CRITICAL LITERATURE REVIEW

In this chapter, we will introduce two state-of-the-art sentiment analysis tools, then go through the related work, briefly in the domains other than software engineering, followed by work in the software engineering domain. We discuss how these works are different from our work.

2.1 Tools

2.1.1 SentiStrength tool

SentiStrength is a lexicon-based automatic sentiment analysis tool designed by Thelwall (Thelwall) for sentiment strength detection across the social web. It primarily uses direct indications of sentiment, i.e., words or phrases of the text for calculating sentiment score, and by default returns a pair of integers, one from -5 (signifies strong negative sentiment) to -1 (signifies no sentiment), and one from 1 (signifies no sentiment) to 5 (signifies strong positive sentiment). A given text can express both positive and negative sentiments at the same time, and a text is considered sentimentally neutral when the emotional scores for the text appear to be -1 and +1. SentiStrength has other configurations for returning sentiment score discussed in subsection 8.2.1. In the following there is an example showing how SentiStrength scores given text regarding its default configuration i.e. dual [1]:

The text: "I love you but hate the current political climate."

Sentiment Score: positive strength 3, negative strength -4

SentiStrength has several key features like spell correction algorithm, or using a booster word list to strengthen or weaken the sentiment strength of the words coming following the booster words.

2.1.2 Stanford tool

The Stanford sentiment extraction tool, based on machine-learning, developed by Richard Socher et al. (Socher et al., 2013) for positive/negative sentence classification. To capture the meaning of longer phrases instead of just single words, Socher et al. introduced the Stanford Sentiment Treebank and a powerful Recursive Neural Tensor Network. The corpus consists of 11,855 single sentences extracted from movie reviews. This is the first corpus with

[1] http://sentistrength.wlv.ac.uk

fully labeled parse tree that enable complete analysis of compositional effects of semantic in language. New deep learning model, Recursive Neural Tensor Network (RNTN) obtained the highest performance in comparison with all previous methods with 80.7% accuracy for predicting fine-gained sentiment labels. This model considers not only the words, but also builds up a representation based on a structure of a sentence. According to Socher et al. it is the only model that can accurately captures the sentiment change and negation scope also the sentiment of phrases following the contrastive conjunctions like 'but'. RNTN predicts 5 sentiment classes from "very negative" to "very positive" (very negative, negative, neutral, positive, very positive) for each sentence or at every node of a parse tree.

Jongeling et al. focused on application of sentiment analysis tools in software engineering context (Jongeling et al., 2015) and examined 4 different tools including SentiStrength, Stanford and, NLTK (Bird et al., 2009a). (For each given text, NLTK returns different values separately that show the probability of being positive, negative, and neutral of the text.) Based on their results, SentiStrength and NLTK show the highest degrees of correspondence (correspondence between the scores provided by the tool and manual labeling) and also have the higher agreements with each other. While, Jongeling et al. concluded more precise sentiment analysis for software engineering context is needed.

2.2 Related Work

Previous studies mostly used sentiment analysis in the areas of marketing and financial markets, but not the area of software engineering.

2.2.1 Sentiment Analysis for Reviews of Consumer Products and Services

Many online markets like mobile app stores or Amazon provide facilities for customers to assess their products and give their opinion. In such cases, sentiment analysis can be applied on the reviews of a customer for products and services. Twitter and Facebook are popular websites being used for sentiment analysis purposes too, like monitoring the reputation of a specific brand (Feldman, 2013). Pak et al. (Pak and Paroubek, 2010) focused on Twitter, which is the most popular microblogging platform, to do sentiment analysis. They collected a corpus of 300,000 tweets that are evenly distributed across positive emotions, negative emotions and absence of emotions. By performing statistical linguistics analysis on the corpus, they build a sentiment classifier that uses the corpus as training data. Finally, they conducted an experimental evaluations on a set of microblogging posts.

A large and rapidly growing number of businesses and merchants sell their products and

services on the Web and consequently ask their customers to review their products and give their ideas about those products. For example, according to Hu et al. (Hu and Liu, 2004) for a popular product, the number of reviews can be in hundreds or thousands. With this huge amount of reviews and comments, keeping track of the customer reviews and managing them (essential for decision making) have become one of the difficulties for relevant manufactures or other fellows require this information. Hu et al. provided feature-based summaries of customer reviews based on data mining and language processing methods. They mined product features commented by customers, opinion sentences from customer reviews and semantic orientations (positive vs. negative) of the opinion words, using WordNet. Their experimental evaluations showed their proposed techniques are very promising.

Bing Liu et. al proposed a novel framework, *Opinion Observer*, for analysing and comparing customers' opinions of competing products (Liu et al., 2005). With their proposed system, the user can see with a single and simple look the strengths and weaknesses of each product. A manufacturer can easily gather marketing intelligence and product benchmarking information.

Cataldi et al. (Cataldi et al., 2013) presented an approach for feature-level sentiment detection. Their proposed approach extracts users' opinions, from user generated reviews, about specific features of products and services. For this, they used both natural language processing and statistical techniques. First, they extracted domain features so that each review was modelled as a lexical dependency graph, then the polarity linked to each feature was estimated. To achieve this, each sentence is modelled as a set of terms in a dependency graph connected through syntactic and semantic dependency relations. Through a user study their results compared against 39 human subjects for hotel reviews and obtained high precision and recall on the features, with the computed polarity degree slightly below the average human performance. The proposed approach can be used to extract sentiment patterns from any social networks as well.

2.2.2 Sentiment Analysis in Financial Markets

Similar to consumer reviews, analysis of financial markets uses sentiment analysis on news items, articles, blogs and tweets about companies to drive automated trading systems like StockSonar (Feldman, 2013). Sehgal et al. introduced an approach for stock prediction based on sentiments of online messages, from which correlations between stock values and sentiments are learnt to enable prediction (Sehgal and Song, 2007). Das et al. (Das and Chen, 2007) designed an algorithm to train a small investor sentiment classifier from stock message boards, which can be used to assess the impact of small investor behaviour on stock

market activity.

2.2.3 E-learning and affective framework

Agnieszka proposed a framework applicable in Intelligent Tutoring Systems (ITS) (Landowska, 2013). He defined affect-aware system as a software program that identifies emotional state of a user (in their context, a learner), then uses its control mechanism and application log process information on affect which might lead to some interventions (in their context: support or disturb learning process) done by the system. Therefore their proposed framework or design model consists of 3 major parts: user affect elicitation, user affect analysis, and affect-aware reaction intervention. He also conducted a case study using one conversational ITS, Gerda. Their emotional state recognition was based on lexical analysis by using user inputs. Their keywords lexicons were ANEW extension (Bradley et al., 1999) with WordNet and ANEW extension with ConceptNet. Finally, they evaluated the framework based on the various uncertainties assumed for ITS emotion recognition and control mechanism parts of the framework. The author believed that although the proposed framework is for ITS, it can be applied in multiple disciplines.

2.2.4 Affects in Software Engineering

In this section we discussed about the software engineering studies that applied affects in their researches from different perspectives. We clarified how our work differentiates from them.

Inclusion of Affects in Software Design

Colomo-Palacios et al. conducted a study to bring the stakeholder's emotions into requirement engineering, and treat them such as one criteria like stability in this process (Colomo-Palacios et al., 2010). They discussed about the importance of human dimension even in comparison to technical dimensions in software engineering. Their goal was to build an instrument that is able to predict requirement evolution using expressed emotions of stakeholders. To this end, first they showed the importance of understanding stakeholder's viewpoint from their emotion expression, then presented a method that include these emotion in requirements. In their method, for each requirement, all related stakeholders express their emotion about that requirement through the Affect Grid (Russell et al., 1989) by determining pleasure and arousal. The proposed method implemented in two different projects. Finally, they looked into 1,175 emotional ratings finding out any patterns and they concluded that

high arousal and low pleasure levels are predictors of high versioning requirements in their studied projects.

Miller et al. stated that a user's acceptance of products not only can be understood by their cognition but also by emotion, while emotion shows deeper level of appealing (Miller et al., 2015). They explained that in traditional system design, software engineers gathered functional and non-functional requirements, while not including the desires of the end-users and stakeholders. Based on their opinion, it is a prevalent misconception to postpone addressing these issues by designing appropriate user-interface. They believed that emotional goals are drivers to elicit new requirements, either functional or non-functional. Therefore, they proposed a flexible modeling notation, called people-oriented software engineering (POSE) model, by adding emotional goals (which represent the desired feelings of stakeholders) to the notations and methods in different disciplines and specifying their relations to other parts of the system. Finally they evaluated their models, using a case study and a user study, while observing improvements in user satisfaction and found that inclusion of emotions is a positive step in software engineering.

These studies shed light on the usage of affects during software design. They highlighted the importance of stakeholders' and users' affects towards software design as one influential dimension that can improve quality of the design. In contrast to our work, they did not deal with the affects of developers during development process and their influence on the final work.

Investigating the Importance of Understanding Developers' Affects

De Choudhury et al. explored various emotional expressions of employees of 500 large software corporation. For this purpose, they inspected the posts on an internal Twitter-like microblogging tool, called OfficeTalk, to characterize emotional expression of employees at the workplace and in a fine-grained, continuing manner (De Choudhury and Counts, 2013). Empirical analysis showed that affective expression in the enterprise can be the result of various workplace factors grounded by theoretical foundations in organizational behaviour research. These factors can be exogenous and endogenous workplace factors, geography of organization or the organizational hierarchy. Toward this end, they extracted a score for positive affect(PA) and negative affect(NA), from the microblog posts of employees over time, by measuring the textual content of the microblog posts using Linguistic Inquiry and Word Count (LIWC). PA and NA are computed as the ratio of the number of positive or negative words to the total number of words in a post. They concluded that affective expression in the workplace can provide an efficient tool for assessing key factors and performance relevant

outcomes.

Rigby et al. also used LIWC, a pyschometrically-based linguistic analysis tool, to study the Apache httpd developer mailing list (Rigby and Hassan, 2007). They tried to assess the personality of four top developers, and also two top developers that have left the project. They also examined the word usage on the mailing lists near the releases to find the general attitude of developers in these times. They found some promises in understanding why developers join and leave a project.

Bazelli et al. (Bazelli et al., 2013) chose StackOverFlow.com, which is one of the most popular Question and Answer websites used by all kinds of programmers including professionals or amateurs to interact. They explored the personality traits of the authors of the site, by analysing answers and questions of the community. Like Rigby et al. (Rigby and Hassan, 2007), they applied LIWC for their analysis, then categorized the extracted personalities based on their reputations (also determined from the site). They found that top reputed authors are more extroverted and have less negative emotions. They stated that their work is a partial replication of Rigby and Hassan's work.

These studies show that monitoring the developers' affects can provide companies or managers with facilities to assess critical factors such as understanding the behaviours of developers toward the community. Measuring affect, they applied LIWC on the recorded communications of developers (including authors of StackOverFlow). However, in our study, we specifically investigated the impact of developers' affects on quality of the work and the time they spent doing their work. To this purpose, we considered affect from different dimensions i.e. sentiment, emotion, and politeness of developers. Applying appropriate tools on developers' discussions, the obtained affects and their impacts were investigated in our work.

Impact of Developers' Affects on Their Performance

Khan et al. (Khan et al., 2011) argued that people's mood can affect their activities, a programmer's mood also affect their work specially their debugging's performance. They tested the effect of mood on debugging in two experiments. In the first one, 72 programmers saw short movie clips to provoke specific moods. In the second experiment, 19 participants were asked to perform some physical exercises before running of algorithms. In their first experiment, there is no significant effect that valence (positive or negative affect like sadness and happiness) has an impact on performance. In the second experiment, they could not separate the effect of valence and arousal. They concluded that possible future research is needed to measure the impact of emotions on the debuggers' performance. However, they did not use any measurement to extract and quantify the developers' affects while assumed

that watching movie clips induce positive or negative emotions in developers. In our study, instead of relying on the movie clips to change affects of developers for a while, we tried to measure developers' affects during the whole development process applying different tools on various software collaborative textual artifacts. In addition, we focused on the affects of the entire team not individual members.

Impact of Developers' Affects on Their Problem Solving

Graziotin et al. (Graziotin et al., 2014) believe that the software development process, as a primarily intellectual process, is substantially more complex than other industrial processes. They introduced psychological measurement for affect, analytical problem solving, and creativity in empirical software engineering. Then, they conducted a study with 42 student participants and investigated the correlation between affect vs creativity and between affects vs analytical problem solving performance of software developers. There was no significant difference in the number of generated creative ideas based on the affects, however, their results showed that happier software developers are more productive in problem solving performance. They also found that there is more need for studying the human factors in software engineering while applying a multidisciplinary viewpoints. They also conducted a study how affects(including dominance, valence, arousal dimensions) can impact self-assess productivity of developers (Graziotin et al., 2015). This time, they selected 8 participants (4 students, 4 professional developers), with questionnaires for measuring the affects (using psychological measurements) of participants and their productivity. Participants work for 90 min on software development tasks while the researcher observed their behaviors too, each 10 min (this interval was the result of a pilot test), the participants complete a questionnaire, in this way their affects and productivity are measured nine times per participants. In addition, after task completion participants were interviewed about the factors impacting their performance and self-assessed productivity. Final results denoted that the affect dimensions (high happiness) are correlated with self-assessed productivity positively. They mentioned the experiment designed in their study is not applicable for continuous application in the industry. However, instead of using psychological measurement or questionnaire, we applied appropriate tools and software artifacts, to automatically measure developers' affects which is applicable in many OSS projects and can be extended for further studies too.

Automatic Detection of Developers' Emotions

Murgia et al. analyzed whether development artifacts like issue reports carry any emotional information about software development (Murgia et al., 2014). That was a first step towards

verifying the feasibility of an automatic tool for emotion mining in software development artifacts. They applied emotion mining, which tries to identify the presence of human emotions like *joy* or *fear* from text, voice and video artifacts produced by humans. (It is different from sentiment analysis, which instead evaluates a given emotion as being positive or negative.) They mined 800 issue comments from the issue repository of the Apache software foundation[2], and with four raters identified the emotions associated to each extracted comment. As a ground truth, they consider agreement on a particular comment's emotions as a "correct" classification for their research questions and to measure the degree of inter-rater agreement on identified emotions, they calculate either Cohen's κ value (Cohen, 1960) or Fleiss' κ value (Fleiss, 1971).

Their study confirms that issue reports do express emotions towards design choices, maintenance activity or colleagues and some emotions like love, joy and sadness are easier to agree on. Their findings suggest that for love, joy and sadness it makes sense and eventually might be feasible to automate emotion mining although challenges need to be studied more closely while more systems and data sources are needed to be taken into account too.

Sentiment Analysis of Software Collaborative Artifacts

Guzman et al.(Guzman and Bruegge, 2013) proposed an approach to help finding emotional awareness in software development teams. For this purpose, they used the latent Dirichlet allocation to identify the topics discussed in the collaboration artifacts like texts from mailing lists and web discussions. By applying lexical sentimental analysis, they then obtained an average emotion score for each of the topics. They evaluated their approach on student projects by interviewing the project leaders, it seems their work needs more details in the generated summaries. Their designed experiment also run on university students.

Later, Jurado et al. (Jurado and Rodriguez, 2015) emphasized on the Global Software Engineering [3] and exposed the necessity of new techniques for monitoring software development process, similar to social network analysis field. They also pointed to the role of emotions in professional work and its impacts on productivity and task quality, and job satisfaction. Therefore, they conducted an exploratory case study and analyzed the sentiment expressed by developers in issues of nine well-known and large projects. Their results also showed that the developers express sentiments in issues and tickets, which could be used in order to analyze various factors in the development process.

[2]https://issues.apache.org/jira/secure/Dashboard.jspa

[3]A variety of terms exist: Distributed Software Development, (DSD), Global Software Development (GSD), or Global Software Engineering (GSE).

These studies first discussed the importance of understanding developers' affects at work, then applied sentiment analysis to analyze the presence of sentiments in collaborative artifacts. In contrast to our study, they just focused on sentiments extracted from software artifacts, while we considered emotion and politeness too. In addition, they have not studied the impact of obtained sentiments, however, we continued our study by investigating the impact of affects regarding different factors.

Md Rakibul et al. (Md Rakibul and Minhaz, 2016) conducted a quantitative empirical study to investigate emotional variations of developers regarding type of development tasks and development time. To this purpose, they extracted commit messages of 50 open source projects, while these commit messages were associated to other information like their timestamp, revisions, type of underlying task including 1)bug fixing, 2)new feature, 3)refactoring, 4)energy-aware development task. For each commit message, they themselves computed sentiment score (they called it emotional score) using SentiStrength tool which we also adopted in our studies. The majority (65%) of the commit messages had neutral sentiments, positive sentiments were found in smallest portion (13%) of the commit comments, and negative sentiments in 22% of the commits. Their results showed that positive sentiments for energy-aware development are much higher than sentiments of other types, and oppositely more negative sentiments appeared in commit messages related to new feature implementation tasks. Then, they run another experiment to see whether sentiments are varied based on developers considering one particular activity, i.e. bug fixing. Their results revealed that some developers are happier(with more positive sentiment) than others during bug fixing. However, another part of their study showed that developers' sentiments have no significant difference based on different time and days of a week. They also found that when developers are emotionally active, either positive or negative, write longer comments in their commits. In this study, they have applied commit messages as sources to extract developers' sentiment, however we used issue reports and mailing lists of developers as two major communication media in software development (Bacchelli et al., 2010, 2012) to extract developers' affects. They considered factors like energy-aware development and also focused on individual developer's sentiment too in their study while we have not considered these concerns. We investigated the correlation between overall affect related metrics achieved from collaborative artifacts and software quality and development time.

Investigating Developers' Affects using Biometric Data

Based on the fact that software developers experience a broad range of emotions during their work, Müller et al. (Müller and Fritz, 2015) conducted a study to first investigate whether

biometric data can classify developers' emotions, then to analyze the correlations between developers' emotions and their progress on the assigned tasks. To this purpose, they ran an experiment with 17 developers working on two change tasks. The change tasks assigned to participants were representative of general change tasks that could not be solved easily, thus they stressed both positive or negative emotions of participants. While participants work on the two change tasks wearing biometric sensors like Empatica E3 wrist band or Eye Tribe eye tracker. Therefore, biometric measurements that are linked to emotions such as skin temperature, heart rate, blood volume pulse and, pupil size could be measured during their work. In addition, before each change task, participants relaxed and watched a calming video, so that their biometric features are leveled to the baseline.

To measure emotions, participants were periodically asked to assess their emotions using Russell's 2-dimensional Circumplex model (Russell, 1980) in two axes, valence and arousal. For measuring the progress, participants were asked to rate it. Their analysis on the collected data shows that a classifier trained on biometric data is able to predict positive and negative emotions significantly, 71.36% of all cases. The accuracy of predicting the achieved progress based on the emotions was also high, 67.70%. Their further analysis based on machine learning techniques also revealed that emotions and perceived progress are highly correlated. However, their approach is tedious to be applied in other studies.

Affects of Developers and Software Quality

Canfora et al. (Canfora et al., 2014) examined whether the personality factors of team members and also team climate factors are related to the quality of the developed software by the team. They designed two quasi-experiment studies with a laboratory environment, run on university students who had to develop a software system. Most of the values of the experiment variables, independent and dependent variables, were taken directly from questionnaires. For example, to measure software quality, Canfora et al. referred to the selected software quality criteria taken from SWEBOK 2004, such as testability and functionality, and to quantify the personality factors, the NEO-FEI test was used, which is composed of 60 questions covering five personality factors. The results show that software quality has a significant correlation with personality factors of team members like extroversion and some team climate factors such as participative factor as well. Finally, they proposed some guidelines for software project managers with respect to team formation.

Biometric measurements explained in section 2.2.4 are logistically expensive. Both biometric measurements and self-assessment (section 2.2.4) approaches also may disrupt people while are difficult to be applied at workplaces especially considering geographically distributed

teams like open source projects. In our study, affects extracted from textual software artifacts such as issue reports or review comments, since developers' discussions during software development process recorded in these artifacts.

In this thesis, we have investigated the correlation between affect related metrics of developers and quality of the software system they produced. Therefore, we briefly explained about software quality metrics and measurements in the following.

2.2.5 Software Product Quality

Quality is one of the essential measures of success and improvement of a software product, but because of this role it is a multidimensional subject with many levels of abstraction and different viewpoints. From a customer's perspective, quality is the value that he/she obtains from the product or service based on different variables like price, performance and reliability (Guaspari, 2004). Considering the practical definition of quality, it is defined as "conformance to requirements" (Juran and Godfrey, 1999) and "fitness for use" (Crosby, 1980). "Conformance to requirements" denotes that first requirements should be identified completely and correctly, secondly it implies that during development and the production process measurements should be taken to guarantee that those requirements have been met. With this definition, nonconformances are concerned as defects. The "fitness for use" definition regards customers' needs whether the proposed products or services are right for their uses. According to Juran et al. (Juran and Godfrey, 1999), since each customer might have different use of a product, product must cover multiple elements of fitness for use, each of these elements is a quality characteristic or a parameter for fitness for use.

Product Quality Metrics

Generally we can conclude that software quality contains two aspects, intrinsic product quality and customer satisfaction. Based on Kan, the following metrics cover both of them(Kan, 2003):

- Mean time to failure
- Defect density
- Customer problems
- Customer satisfaction

Intrinsic product quality usually measured by the number of functional defects (bugs) and by how long the system can run before a crash happens. These key properties indicate two metrics respectively: defect density (rate) and mean time to failure (MTTF). The first metric measures the time between failures while the second one measures the defects with respect to the software size like lines of code, function points.

However, customer's viewpoint can play a role in measuring software quality. Generally, a good defect rate leads to reduction in the number of defects in each upcoming release.

Customer satisfaction could be obtained based on overall quality and various dimensions like: capability, functionality, usability, reliability, installability, documentation/information, etc.

According to Kim et al. (Kim and Whitehead, 2006), the number of bugs or fixes as common factors are used to measure the quality of software. When a software system has cumulative bugs in its history seems more unstable in comparison with the time there is no bugs in its history. Kim et al. also discussed that the bug-fix time can be used to measure software quality. We believe that it is an indicator of MTTF, likely long bug fixing time is a reason of unstable situation in the software system which may need more attention. Fixing bugs is one major part of software maintenance activities and huge amount of cost is dedicated to it, perpetually for example it was estimated to cost 70 billion US dollars per year in the United States (Lerner, 1994). For recent large and long-lived software, with more complexity and evolution, the importance of bug fixing time increases significantly. They receive larger number of bug reports, short bug fixing time beside decreasing number of bug reports can help to less cost and achieving a product with more quality.

Code review as one of the most effective QA practices improves software quality by identifying defects in code changes before they are integrated into the code base and released to the customer (Fagan, 1999). To ensure quality of long-lived and large projects, code review process should be applied during software development process (Kononenko et al., 2016). However, code review process is relatively expensive in terms of time and effort (Fagan, 1999). Investigating how factors related to human affects can influence the code review time, beside factors from other domains, as one major issues has been studied in this research. Kononenko et al. (Kononenko et al., 2016) studied the relationship between reviewers' code inspection quality and several parameters ranged from technical aspects to the personal and social aspects of reviewers. Their study suggested that developer participation in discussions around bug fixing and review loads can impact code review quality. While we focused on code review time instead of its quality, and various kinds of factors specifically human affect related ones, which extended reviewers' personal factors, were studied in understanding how they can influence code review time.

Müller et al. (Müller and Fritz, 2016) used biometric sensing to identify code quality issues online i.e. while a developer is working on the code. Using ten professional developers as participants, some of the specific biometric data of developers were monitored and extracted in 14 days. Then, in companion with other metrics explained in the following, Müller et al. investigated how they can impact software quality concerns. Their metrics are:

- Biometrics: by using chest strap and wristband, they collected various measurements related to heart, skin and breathing. It was also needed that they applied several data extraction, data cleaning, and feature extraction on the biometric sensor data.
- Code metrics: A small, self-written interaction monitor plugin installed into the participants' IDE to calculate some famous code metrics that have link with program comprehension and code quality based on their previous research like McCabe's complexity (e.g. (Nagappan et al., 2006; McCabe, 1976)), and fanout (e.g. (Zimmermann et al., 2007; Henry and Kafura, 1981)).
- Change metrics: They extracted the number of files added or removed for each code element using commits to the repository.
- Interaction metrics: The number and ratio of edit and select events were collected for each code element. Since, based on previous studies (Lee et al., 2011), these interaction metrics are effective on defect prediction.

As one output metric, one to three reviewers looked for actual bugs and violations of coding styles or documentation as code quality concern factors in commits. Their results suggest that it is possible to use biometrics to predict quality concerns while they outperform traditional metrics. Their biometric classifier was able to predict half of all bugs reported in code review. They repeated a second study with five developers from different country and company within 1 week to asses the generalizability of the obtained results, and it shows that findings related to software quality prediction can be replicated.

In contrast to their research, instead of biometrics we focused on metrics derived from individual's affect that can be seen in their text communications. Based on Jurado et al.'s study (Jurado and Rodriguez, 2015), the biometric measurements are linked to positive and negative emotions which are core of our affective parameters in this study, and so they may impact software quality too.

2.2.6 Mining Software Repositories

Recently, many software development activities and stages being performed using various tools. For instance, in writing source code, mostly version control systems like Git are applied, for code review and testing, code review tools like Gerrit and issue tracking systems like Bugzilla are adopted, mailing systems or IRCs in various work environment are used for people communications too. Hassan et al. (Hassan and Xie, 2010) categorized three groups for the examples of the tools' related repositories in MSR: **Historical** (such as source control or bug repositories), **Runtime** (such as deployment logs), and **Code** (such as Google code) repositories. These tools provide us with ample valuable data source help better understanding of software projects such as development-related behaviour, defect prediction and, code change recommendation. Thanks to open source software projects, they give the researchers this possibility to access to the various software repositories specially large software systems that so far were the center of attention for many researches in this domain like (Zimmermann, 2007; Moura et al., 2015; Tian et al., 2012). Then, MSR researchers try to analyze these rich data sources to achieve practical and applicable facts and information from software systems and projects and thereby support software projects with recommendations and guidelines based on these information. In this thesis, we applied MSR techniques on historical repositories (like issue reports and mailing lists) in open source projects to extract affects from the recorded discussions and communications of developers.

However, one of the major challenges in MSR research, and in our study as well, is finding the linkages between different kinds of repositories. Because there is not necessarily any standard or enforced practices to push developers to link them precisely, for instance to link a bug issue and a commit in source control. In our study, finding the link between issue reports and reviews of patches was one of our challenges too. We tried to solve using proper data sets from open source ecosystems like OpenStack and Eclipse, as these ecosystems are among the leaders of pushing developers to clearly link code changes to issues and reviews. In addition, recently almost every known tools are integrated with GitHub, therefore, this type of problem should be mitigated. However, this is part of a critical problem that MSR researches may suffer from, called "systematic bias" and impacts both build prediction models and generalizability of hypotheses (Bird et al., 2009b). Systematic bias happens when the distribution of data is not fully random and balanced, i.e., the data is not representative of the population. For example, Bird et al. (Bird et al., 2009b) showed for bug-fix data sets used normally in bug prediction analysis in MSR, it is probable that just some developers submit bug reports like developers of a fragile component, thus the data set is not representative of the whole components.

Recently, Bird et al. (Bird et al., 2015) illustrate best practices in MSR using leading data scientists' experiences. They discuss different sorts of data sources such as code reviews, app stores and log files. Hassan et al. (Hassan and Xie, 2010) see the future of MSR as Software Intelligence (SI), like business intelligence but will support software practitioners including owners and developers in decision making process by using fact based support systems. Decisions such as when to release a software system or which parts of the system should be tested and so on can be done based on recent and pertinent information offered by SI. Therefore, this process, which is based on a well-studied science, likely prevents wasting large resources and co expensive costs. They believe that recent advances in MSR is promising for SI realization in the near future.

Based on our literature review, we discussed the current researches that have focused on the impact of human affects in software engineering by applying different techniques including biometric approaches. Our research will aim to understand the importance of human affects related metrics obtained from various repositories related to recorded communications among developers during software engineering process. To study their importance, we considered an important criteria "software quality" based on the defects rate and time to fix the defects.

Moreover, knowing the emotional state of the development team helps the manager to create an environment capable of combating the effects of "bad" emotions. Thus, training the development team on stress management, communication and assertiveness will improves the coping ability of the RE.

CHAPTER 3 RESEARCH PROCESS AND ORGANIZATION OF THE THESIS

This chapter presents the methodology of the research process and the structure of this whole dissertation. This thesis will focus on the three parts mentioned in the research hypothesis, as well as one preliminary part: 0) affect measurements in software engineering context, 1) analyzing the link between affect-related factors and quality of work, 2) analyzing the link between affect-related factors and time taken by work, 3) investigating one possible solution to deal with conflicts in open source projects.

Our work tries to show the importance of affects of developers in software development process and help practitioners to measure affect-related factors from software artifacts. Chapter 4 analyzes the presence and evolution of sentiments in developer and user mailing lists, moreover evaluates the application of one tool in software engineering process. Chapters 5 and 6 investigate the link between affect-related factors and the quality of the resulting work, and the time of code reviewing and issue resolution, respectively. Finally, chapter 7 analyzes the characteristics of one possible solution adopted in open source projects to deal negative affects rooted in diversity that may cause debates, conflicts, and battles among participants.

3.1 Investigating the Presence and Evolution of Sentiment in Mailing Lists (Tourani et al., 2014)

Before studying the link between affect and quality/time in later chapters, we first need to analyze whether affect exists in software engineering communication. In particular, we wanted to explore the use of sentiment mining tools to identify such affects. For this study, we chose mailing lists of two mature and successful open source projects, Tomcat and Ant, to extract the sentiment of developers or users during their communications as mailing lists are one of the most popular media for discussion in open source software projects. This study was in following with our previous study Murgia et al. (2014), which confirmed the existence of emotions in issue reports and also the feasibility of automated emotion mining from issue reports.

Using the state-of-the-art tool, SentiStrength, we set out to identify extremely positive or negative feelings in emails, then manually evaluated these emails to understand their topics. Although, we showed the presence and evolution of the sentiment in studied open source mailing lists, we also observed how noise in the sentiment measures appeared (because of

special way of sampling). To further explore this noise, section 8.2 evaluates and compares SentiStrength to one other cutting-edge sentiment mining tool.

3.2 Investigating the Link Between Affect-Related Factors with Quality and Time

After finding the presence of affects in software engineering artifacts, we analyzed their relation with two important measures i.e., *quality*, and *time* spent of code reviewing and issue resolution, since these measures are two of the major concerns in software maintenance and quality assurance. Software practitioners are interested in identifying and understanding factors influencing quality and time of the work so that they can produce high quality products in shorter time.

In our preliminary studies, including the one explained in chapter 4, we examined the presence of affects in software artifacts. Particularly, two major communication media in open source projects, i.e., issue repository and mailing lists were studied. In chapters 5 and 6, we study extracting affect-related factors from issue and review repositories of OpenStack and Eclipse projects. These projects adopted dedicated code reviewing tool, Gerrit, and there is a reasonable amount of links between their commits, issue reports and reviews. For linking accepted reviews and commits, we applied following techniques:

- commit identifiers mentioned in the code reviews.
- additional reviews added after rebasing (Kalliamvakou et al., 2014).
- identifiers of the reviews in the commit messages.

For linking issue report to commits, we used:

- issue identifiers mentioned in commits
- Git commit identifiers mentioned in issue reports
- issue identifiers mentioned in review comments of a corresponding commit

3.2.1 Studying the Link Between Affect-related Metrics with Just-In-Time Quality (Tourani and Adams, 2016)

In our first study, explained in chapter 5, we investigated the impact of various factors, including affect-related ones, on quality using Just-In-Time(JIT) prediction models. JIT

models have several advantages, since they consider patches or commits as granularity for their work, these models properly fit to our study. Each patch has dedicated review and issue comments for which we can measure affect. A JIT model hence allows to study potential relations between these affects and the quality of the corresponding patch.

Sentiments of issue comments or review comments (as proxies for human discussions) were computed, as well as variety of other factors defined in our study. Using the proposed factors as input variables, we built explanatory models regarding quality (in terms of defect-prone commits) as output variable for each project separately. Analysis of the top influential factors in the model helped determine whether affect has a strong relation with quality of a patch and, if so, in what way.

3.2.2 Studying the Link Between Affect-related Metrics and Time Duration of Code Reviewing and Issue Resolution (Tourani and Adams, 2017)

We also studied the relation between affect-related factors and the time taken for code reviewing and issue resolution. At first, we investigated whether these two variables, time of code reviewing and time of issue resolution, are independent or measure the same thing. Since, detailed comparison between them showed important differences (which we manually validated), we decided to build separate models for both. For these, we defined a wider range of metrics in comparison to the metrics defined in chapter 5. For example, for affect-related factors, we not only computed sentiment scores of comments (both issue and review comments), but also obtained the emotion and politeness of the comments.

Explanatory models were built for code reviewing time and issue resolution time of each project separately. Then, evaluated using 10-fold cross validation. To find out the importance of metrics in generated models, top powerful metrics were computed and discussed.

3.3 Understanding One Popular Solution for Dealing with Conflicts (Tourani et al., 2017)

Recently, many open source projects started adopting codes of conduct as possible solution to deal with the potential debate and battles that are likely to happen in environments like open source communities with such a huge diversity. We tried to understand this solution as it is one of the first explicit countermeasures for negative affect in large open source projects, aiming to provide a healthier and more comfortable atmosphere for participants.

As explained in chapter 7, we conducted our study applying principals of a systematic literature review to find out the role of codes of conduct in open source projects. Two electronic

databases were searched, i.e., Github and Google, to obtain the major codes of conduct identified. We manually studied these major codes of conduct to understand their contents, guidelines and key characteristics.

We also performed a qualitative study by interviewing leaders and creators of codes of conduct to understand the process behind codes of conduct in open source, and their impact and constraints. To this end, we prepared a list of specific questions and interviews done in semi-structured and structured form. Finally, the interviews were analyzed using *open coding* technique. This allowed us to understand the reason and process behind emerging codes of conduct in open source communities, and to identify their impact and limitations in open source projects.

CHAPTER 4 ARTICLE 1: MONITORING SENTIMENT IN OPEN SOURCE MAILING LISTS – EXPLORATORY STUDY ON THE APACHE ECOSYSTEM

Abstract

Large software projects, both open and closed source, are constructed and maintained collaboratively by numerous teams of practitioners including developers and testers, who are typically geographically dispersed. Although often schooled in management techniques to act on motivational or emotional problems in their team, not being able to work face-to-face with their team members causes team leads to lack a crucial piece of information in order to apply these techniques: awareness of problems. This paper evaluates the usage of automatic sentiment analysis to identify distress or happiness in a development team and project community. Since mailing lists are one of the most popular media for discussion in software projects, we extracted sentiment values of the user and developer mailing lists of two of the most successful and mature projects of the Apache software foundation. The results show that (1) user and developer mailing lists carry both positive and negative sentiment and have a slightly different focus, while (2) work is needed to customize automatic sentiment analysis techniques to the domain of software engineering, since they lack precision when facing technical terms.

4.1 Introduction

Research in psychology, economic and organizational behaviour shows the importance of happiness and job satisfaction at work. Various books and papers (Robertson and Cooper, 2011; Rao, 2010; Barron and Barron, 2013) emphasize that happy people are more creative, learn more and achieve greater success at their work. Work in administrative science (Amabile et al., 2005) highlights the direct linear relationship between positive affect (i.e., feelings like happiness, joy, excitement or, contentment) and creativity in organizations. Researchers also have found evidence of the economic impact of happiness and attitude of employees. For example, a recent study (Oswald et al., 2009), saw 10-12% greater productivity for happier individuals, concluding that social scientists may need to pay more attention to emotional well-being as a causal force for productivity at work. Finally, organizational behaviour research (909, 1975), showed that affective factors are closely tied to the feelings of employees about their work and company.

Since software development and maintenance are a collaborative activity as well (Mistrìk et al., 2010), it seems intuitive that the affect of software project members also plays a pivotal role in the success or failure of a software product, however software managers have a hard time keeping track of their people's feelings. Of course, the best way to know how people feel is by talking and working with them in person. Surprisingly, this becomes more and more difficult, not just in open source development (where by definition most people do not even work for the same employer), but also in traditional companies. For example, larger companies are distributed across a different campus or even country. Recently, companies like Mozilla even started promoting remote work. For example, in 2012 the release engineering team (12 people), which is the backbone responsible for bringing new features to customers through official releases, was spread across 4 different (non-contiguous) time zones to offer around-the-clock service and improve quality of life. Their manager noted "out-of-sight, out-of-mind is a real concern". It is even harder to know how the user community feels about a project, since this group is several orders of magnitude larger and spread all across the globe.

As such, in many cases electronic communication in the form of email (1-to-1 or mailing lists), chat rooms, video conferencing or phone calls have become the de facto means of communication. While one could expect people to be more reserved or careful when using those media since almost all of them record conversations (in contrast to face-to-face discussions), Bacchelli et al. (Bacchelli et al., 2012) noted that people have become so accustomed to these communication channels that most of them freely express their actual feelings in their communications. In other words, for many organizations, emails and chat messages are one of the primary means of conveying and picking up signals and indications of good or bad feelings of colleagues and employees (Rigby et al., 2008), (Bacchelli et al., 2010), (Bacchelli et al., 2012).

To help organizations and open source projects in picking up signals of good or bad affect easier and more accurately, this paper explores, as the first step, whether automatic sentiment analysis tools are able to identify periods of extremely positive or negative feelings in the developer and user community of two major Apache projects. Automatic sentiment analysis is an emerging area that blends natural language analysis and psychology to obtain cues about an individual's opinion or feelings towards a product. Our work, if successful, opens the door towards a new research area of customizing sentiment analysis and other techniques for software development and software maintenance. In particular, we address the following research questions:

RQ1) *How accurate is existing sentiment analysis on software engineering data sources like mailing lists?*

Since sentiment analysis (and the corresponding tools) primarily have been used in the context of psychology, finance or organizational behaviour, we first evaluate how well they perform on software engineering data sources, which typically contain technical terms. We find that the precision of SentiStrength for positive periods is 29.56%, and for negative periods 13.18%. These low values are due to ambiguities in technical terms as well as the difficulty of SentiStrength to distinguish extremely positive/negative documents from neutral ones. For Ant, some correlations between the number of closed bugs and appearance of sentiments in the user mailing list have been observed.

RQ2) *What types of sentiment can be observed in software engineering mailing lists?*

To understand the role of affect in software projects, we manually studied a representative sample of developer and user emails. 19.77% of the emails contain positive sentiment, compared to 11.27% for negative sentiment. We could distinguish 6 categories of positive sentiment, and 4 categories of negative sentiment.

RQ3) *Do developers and users show different sentiment?*

Finally, we studied whether users and developers of a software project show different affect in mailing list communication. The user mailing list and developer mailing list of each project show only little similarity in their sentiment trends. For emails with positive sentiment, user mailing lists contain substantially more "Curiosity", but less "Announcement" and "Socializing", while for emails with negative sentiment, user mailing lists contain more "Sadness" and less "Aggression" than developer mailing lists.

In the remainder of this paper, we first describe the background notions for sentiment analysis (section 7.2). Next, we describe the experimental setup (section 6.3). We then address the three research questions (section 6.4) and discuss our findings. After threats to validity (section 7.4) and related work (subsection 7.2.3), we finish with conclusions (section 7.5).

4.2 Background

This section provides background about sentiment, sentiment analysis, and the SentiStrength tool used in this paper.

4.2.1 Sentiment Analysis

The term "sentiment" (also "affect") refers to people's opinion, attitude, appraisal or emotions toward entities, events and their attributes (Mishra and Jha, 2012). Sentiment differs from "emotion", which is a state of feeling and shows more detail of human feeling like sadness, happiness, shame, and anger. Sentiment and emotion are used automatically whenever people communicate, since it helps people convey their message or understand other people's reactions. This implicit behaviour is not just limited to communications in real world, but even when people interact through computer aided communications (Thelwall), like comments or feedback that people make on community fora or chat rooms, or in more conventional electronic media like emails. These sentiments are universal, in that they occur as much in politics as in business contexts (Pang and Lee, 2008). Even in software engineering (Murgia et al., 2014), identification of sentiments and emotions in software artifacts can provide an indication of someone's opinions towards certain project decisions or other people.

In order to automatically measure sentiment from recorded (typically textual) transcripts of communication, semantic measures have been proposed as a measure of subjectivity and opinion in text. Those measures usually measure the polarity (positive or negative) and "strength" (also "degree") of a document. Strength shows to which degree a word, phrase, sentence or document is positive or negative towards a subject topic, person, or idea (Osgood et al., 1957). For example, a positive opinion with strength 1 is much more positive than a positive opinion with strength 0.1. Some examples of approaches to extract the above measures are sentiment analysis, opinion mining, and affect mining (Batson et al., 1992; Pang and Lee, 2008). This paper focuses on sentiment analysis.

Sentiment analysis algorithms mainly use two approaches: machine learning or a lexical approach. With machine learning a, typically text documents are taken as input and a classifier is produced as output (Socher et al., 2013). According to (Aue and Gamon, 2005) classifiers perform very well when they are applied in the domain on which they were trained, otherwise their performance decreases significantly. Lexical approaches use language information in the form of a list of known sentiment-related words, their polarities and the grammatical structure of the language, and based on them scores the sentiment of the text. Word lists or dictionaries for lexical approaches can be created manually, or could be expanded by using seed words (Taboada et al., 2011).

4.2.2 Sentistrength

SentiStength uses a lexical approach (Thelwall et al., 2010) to extract sentiment strength from informal English text, which means that it calculates sentiment of a document from the sentiment of words or phrases in the document (Taboada et al., 2011). SentiStrength also divides the given text into sections, then based on the words or phrases within each section, assigns both a positive and negative value to the section since according to psychological research a human can experience negative and positive feelings together (Thelwall). These values for positive values range from 1 to 5, and for negative from -5 to -1. To calculate the sentiment of each word or phrase, SentiStrength looks up the word or phrase in its lexicon and (if found) uses the associated sentiment strength. Modifier words like "very" and "extremely" act as boosters can alter the score, while symbols, punctuations like "!" and smilies can do this as well. SentiStrength also consider the structure of a sentence, such as negations and questions.

We decided to use a lexicon-based sentiment analysis tool rather than a machine learning-based one as such algorithms are simpler and it has been used successfully in several research projects like (Thelwall et al., 2010; Taboada et al., 2011; Kucuktunc et al., 2012). In addition, compared to many existing commercially-oriented opinion mining tools, SentiStrength considers sentiments related to expressing friendship or showing social support (Thelwall et al., 2010).

4.3 Experimental Setup

This section explains the methodology used to address the research questions of the introduction.

4.3.1 Selection of Subject Systems

Mailing lists are the core means of project communication in open source communities like Apache, where developer and user mailing lists are used during software development and maintenance to discuss technical issues, propose changes, report bugs, or ask how-to questions about configuration or any other parts of the product.

For this reason, this study investigates the mailing lists of two major projects of the Apache Software Foundation, i.e., Tomcat and Ant. Tomcat is an open source web server and servlet container first released in 1999[1], while Ant is a software tool for automating software build

[1] http://tomcat.apache.org/

processes[2]. Both of them are mature and widely deployed, successful projects.

Of both projects, we obtained the developer and user mailing list data in the textual mbox format from the official Apache archive pages[3].

4.3.2 Pre-processing of Data

Before being able to analyze the email data, we first had to filter out emails that did not contain actual human content. A major category of emails to filter out were automatic confirmation emails. Such emails are sent automatically by various servers like Bugzilla issue tracking systems or version control systems. Obviously, these emails do not contain any sentiment, since they only contain source code patches or reports originally submitted elsewhere (not on the mailing list). To filter out automated emails for each project, we manually identified the different patterns that they might have. Some of these patterns have special subjects, while others have a specific sender of email. Using regular expressions, we reduced the email data down to a total of 595,673 emails. Table 4.1 shows how this number is distributed across the four studied mailing lists. To avoid duplication of contents of emails, we also removed the quoted parts of email threads as they have considered in their original emails.

Table 4.1 Number of emails per mailing list

Ant Developer	Ant User	Tomcat Developer	Tomcat User	Total
20,292	169,329	360,733	45,319	595,673

After recovering all non-automated emails, the next step is to filter out any non-natural language text inside these emails. The unstructured and noisy nature of the emails related to the development of a software system causes many emails to contain technical information about design, implementation (e.g., source code or excerpts related to reported bugs) and defect-related information like stack traces. Bacchelli et al. (Bacchelli et al., 2012) founded that the content of development emails can be classified into five categories: natural language, source code, patch, stack trace and junk (i.e., textual information like the signatures or spam status of authors).

To find sentiment values of emails, only the natural language category of email content should be taken into attention. For this, Bacchelli et al. (Bacchelli et al., 2010) found that lightweight methods based on regular expressions were the most effective. For this reason,

[2]http://ant.apache.org/

[3]http://mail-archives.apache.org/mod_mbox/

we used a combination of regular expressions and searching for lines with special characters and keywords to filter out uninteresting email content like source code or stack traces.

4.3.3 Sentiment Score Computation

In order to automatically detect the sentiment expressed in emails, we applied the SentiStrength tool and ran it over the pre-processed emails of developers and users. SentiStrength scores each line of email with a value from -5 to +5, however we need one sentiment score for each email as a whole. To find the best way to aggregate the scores of all lines of an email into one value, we ran an experiment on a random sample of 100 emails from the Tomcat project. Given the large number of emails, we sampled enough emails to have a confidence level of 95% and confidence interval of 5%. Two of the authors manually scored the sentiment of each email. Then, we compared the manual score to the following aggregation methods across SentiStrength's line-level scores: minimum, 1st-quartile, average, median, 3rd-quartile and maximum. Note that the "maximum" method corresponds to finding the most extreme value, be it negative or positive.

Table 4.2 shows that the Mean, Median and Max Value methods are the most accurate aggregation methods. However, Mean and Median only worked well for neutral sentiment, whereas the Max method also had an accuracy of 36% for positive sentiment and 21% for negative sentiment. For this reason, we chose the Max Value for our purposes. This seems reasonable, since an email usually consists of a small number of sentences and the sentence with the maximum value of sentiment likely dominates the overall sentiment of an email.

Table 4.2 Accuracy of different aggregation methods for SentiStrength.

	Min	1st Qu.	Median	Mean	3rd Qu.	Max
Positive	0%	0%	0%	0%	100%	36%
Negative	100%	100%	0%	0%	0%	21%
Total	14%	14%	64%	64%	19%	39%

4.3.4 Analysis of the Sentiment Values

The specific analysis used for each research question is presented in the next section with the corresponding findings. When studying the evolution of sentiment, we abstract up from the sentiment of individual emails to the average Max sentiment of all emails sent in one month. A period of one month in open source development strikes a nice balance between being too short (nothing significant happening) and being too long (multiple releases happening).

4.4 Experimental Results

RQ1. How accurate is existing sentiment analysis on software engineering data sources like mailing lists?

Motivation. Detecting sentiment of software team members is essential in modern software development and maintenance, where members are mostly geographically distributed and therefore physical face-to-face meetings are impossible or scarce. In such environments, automatic monitoring of sentiment, either of individuals or of a whole community, can play an important role in managing software projects and identifying potential risks that might threaten the sustainability of a project.

Our first research question explores how precise a modern sentiment analysis tool is in detecting the sentiments of software-related mailing lists. In particular, we use a popular sentiment analysis tool to identify positive and negative peaks of sentiment across time, with the aim of enabling managers or team leads to identify good or bad trends in the sentiment of project stakeholders. Depending on the outcome of the analysis, existing techniques could (a) be used as is to monitor sentiment in a software engineering context, (b) might need customization, or (c) might need to be reconsidered for the purpose of identifying the sentiment in a software project.

Approach. To address this research question, we picked for each project and mailing list the 4 months with the most positive and most negative sentiment. To avoid being stuck with 4 consecutive top months (limiting our evaluation to a too narrow period), we choose at most one positive (or negative) peak per year (starting from the year with the highest/lowest peaks). We ended up with 4 months for the Apache Ant developer mailing list, and 4 for its user mailing list. Similarly, we have 4 months for Apache Tomcat's developer and user mailing lists. Given the large number of emails, we sampled enough emails to obtain a confidence level of 95% and confidence interval of 5%, i.e., 400 in the positive months and 400 in the negative months out of a total of 595,673 existing emails. The 25 sampled emails of the positive months consist of the 25 emails with the highest SentiStrength score, while the emails of the negative months correspond to the most negative scores. In cases where there were more than 25 candidates with the same top positive or negative value, sample emails were selected randomly out of the candidate set.

After sampling, two separate raters read the emails and manually scored them with a positive, negative or neutral value. They ignored the amplitude of the SentiStrength scores, just focusing on the sign (positive/negative/neutral), since by definition these emails correspond to the most extreme (positive/negative) emails and our goal was to validate whether this was

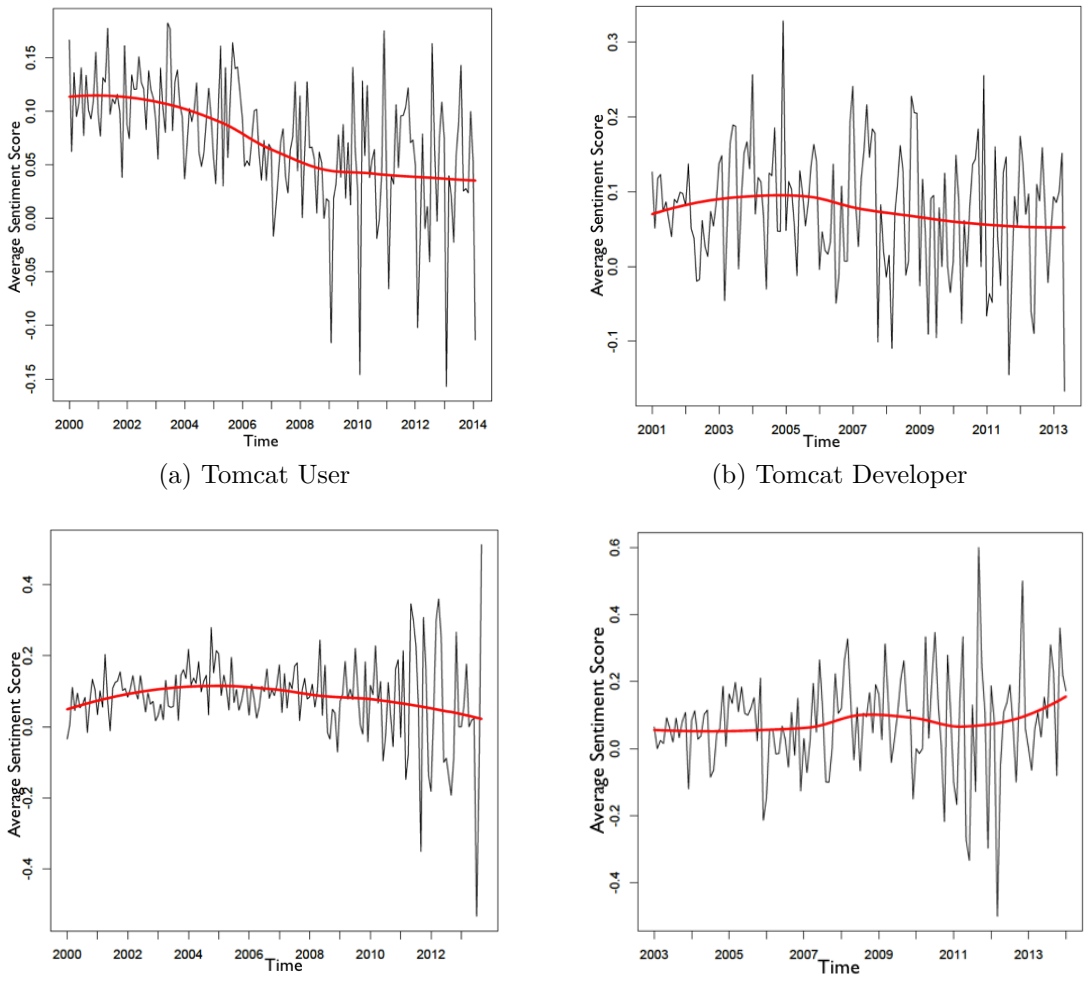
correct. We then used the manual validation to calculate a precision value for SentiStrength. Since the two raters obtained an agreement of 76.62%, our precision values are relative to the emails for which both raters agreed.

Findings. Sentiment evolves over time, with a lot of variation in the form of upward and downward trends. Figure 4.1 shows the evolution of the average SentiStrength score per month for the 4 mailing lists. We can see how average sentiment is bounded between -0.15 and 0.2 for the Tomcat mailing lists, with a peak up to 0.3 for the Tomcat developer mailing list. The Ant mailing lists go from -0.2 to 0.2, with peaks over -0.4 and 0.4 for the user mailing list, and even 0.6 for the developer mailing list.

Given the jagged nature of the plots, a lot of noise is present. For practical applications, one should either filter the noise (i.e., putting average values below a certain threshold to zero) or focus only on the most extreme peaks, since those are indicators of major problems or opportunities in the project that could be worth investigating. As an example of the first kind of filtering, we added loess local regression lines (Kabacoff, 2013), which are smoothed regression lines based on a running average. They are ideal to identify the predominant trend in a noisy curve. We can see how the Ant user and Tomcat developer mailing lists have a more or less constant trend, while the Tomcat user mailing list sees a clearly downward trend and the Ant developer mailing list an upward trend. It is important to note that the average values of the trend line remain slightly positive, even for the downward trends. This indicates that, overall, both projects have a healthy, i.e., positive, community.

The second kind of filtering, i.e., only focusing on the most extreme peaks, yields for each mailing list a small number of very large values. For Ant, the peaks get more extreme towards the right, as can be seen in Figure 4.1. This seems to be linked to a decreasing number of emails being sent to the mailing lists (the volume dropped from an average of 1758 emails in 2000-2007 to an average of 365 in 2008-2014). Tomcat sees many more extreme values over time, often in bursts. This is why our manual analysis considered maximum one positive/negative peak per year, since otherwise our analysis would only consider a very narrow period of time.

The precision of SentiStrength for positive months is 29.56%, while for negative months it is 13.18%. Table 4.4 shows the evaluation results of SentiStrength by the two raters, while Table 4.3 shows the confusion matrix of each group. We see how for positive emails, SentiStrength obtained a precision of 29.56%, while for negative emails a precision of 13.18%. For reference, the SentiStrength documentation mentions a 60.7% precision for positive texts and 64.3% for negative texts on documents on the social web, which is much higher than the numbers that we obtained for the emails.



(a) Tomcat User

(b) Tomcat Developer

(c) Ant User

(d) Ant Developer

Figure 4.1 Average monthly sentiment value across time for the four mailing lists.

One possible explanation for the relatively low precision is that most of the emails in the top positive and negative months turned out to be neutral. This is confirmed by calculating the empirical recall, i.e., recall where we use the set of all emails identified as positive (resp. negative) in our sample of 800 as oracle (basically ignoring the neutral emails). Table 4.4 shows that positive empirical recall is higher than 72.46%, which indicates that positive sentiments primarily are found in the set of 400 positive emails, not the negative emails. In other words, precision is low because the sets of 400 emails contained too many neutral emails rather than emails of the opposite sentiment (e.g., negative emails in positive sample). Only for the user mailing lists we see a relatively low empirical recall, which means that only half of the negative emails showed up in the negative data set.

Investigating the result of the SentiStrength, another possible explanation can be found in

Table 4.3 Confusion matrices of SentiStrength

Actual by raters for developer	Computed by SentiStrength: -1	1
-1	26	10
0	113	89
1	19	50

Actual by raters for user	Computed by SentiStrength: -1	1
-1	15	18
0	125	95
1	13	50

Actual by raters for both	Computed by SentiStrength: -1	1
-1	41	28
0	238	184
1	32	39

Table 4.4 Precision and empirical recall of SentiStrength.

	Positive Precision	Negative Precision	Total Precision	Positive Recall	Negative Recall	Total Recall
Developer	33.55%	16.45%	24.75%	72.46%	72.22%	72.38%
User	25.65%	9.80%	17.70%	75.00%	45.45%	63.52%
Total	29.56%	13.18%	21.24%	73.55%	59.42%	68.42%

the nature of software development and maintenance emails, where people mostly write about problems or solutions in a very technical manner. Many technical keywords are used that are (a) unknown to SentiStrength's pre-compiled list of phrases or (b) sometimes already known to have a positive or negative sentiment when used in a non-technical context. For example, "Safe", "Security", "Value", "Support" and "Dynamic" are existing English terms with a known positive sentiment, while "Kill", "Defect", "Error", "Disabled", "Failure" and "Default" are known to be negative, while neither of these are interpreted as positive/negative in software development or maintenance area: they are just technical terms used in a different (technical) meaning. Table 4.5 shows examples of sentences identified to be incorrect due to technical jargon.

Positive months have a larger variation in precision than negative months. Table 4.6 shows the distribution of precision per month. Especially for negative months, precision is concentrated in a narrow band [0.09,0.19], while for positive months we see more variation [0.18,0.41]. In other words, although the negative peaks have a low precision, this

Table 4.5 Examples of Sentences with Incorrect SentiStrength Score.

Incorrect Positive Samples	Incorrect Negative Samples
that dynamically add JARs to lookup classes (I share Connor's concerns about this)	And I can only (try to) fix errors which occur ...
trunk with support for types and tasks only., At least for starters.	can even enforce dependencies of the code compiled by the different
It seems eclipse has a security manager enabled. That means tasks that have	We thought this was an error on WinRARs side, so an user contacted the
security checks will perform them and if they are loaded by ant class	It might be good to have 1) EXIT_ON_INIT_FAILURE=3D by default in TC8.
on the objects currently declared on that role and their respective XML	duplicate. While the original problem was indeed related to using EL in
was to provide something like ProcessHelper class that can be stored as a ref	Currently Tomcat HTTP 1.1 Connector disables the use of chunked encoding if
	The problem is with those locales for which CharsetMapper.getCharset(locale)
	returns null.,There is an error in ResponseBase.setLocale() that it will set
	land, so that we could e.g. log an error if we encounter invalid data t

low precision seems evenly distributed across each month and hence the general trend of negative sentiment (as in, for example, Figure 4.1) remains correct. For positive months, this is less the case, yet those months tend to have a higher precision. Hence, although the low precision values on the one hand mean bad news, the sentiment results can still be interpreted.

Automated lexical sentiment analysis obtains a precision of around 30% and 13% for positive and negative months, respectively, especially because of neutral emails and ambiguous terms being misinterpreted.

Table 4.6 Distribution of precision per month.

	Min	1st Qu.	Median	Mean	3rd Qu.	Max
Positive	0.04167	0.17800	0.27530	0.31050	0.40530	0.68750
Negative	0.00000	0.09348	0.15790	0.14050	0.18900	0.25000

RQ2. What types of sentiment can be observed in software engineering mailing lists?

Motivation. As mentioned earlier, mailing lists play an important role in communication among team members involved in software development of open source software. Different individuals including developers and users talk about numerous issues in various stages of the project via emails: developers discuss about the problems that they encountered in design or implementation, announce some essential news about the project like a product release or some critical changes that they have made, acknowledge other people's help, while users ask questions about using the product, suggest solutions to improve the product and so on (Guzzi et al., 2013). During these discussions, both developers and users also convey their opinions or sentiment. Our second research question seeks to understand the different kinds of such sentiment in OSS mailing lists.

Approach. We investigated the same emails sampled in RQ1 to address this research question. The same two raters categorized the emails based on the emails' sentiment value. Even if an email was classified incorrectly by the algorithms, we still analyzed its true sentiment(positive/negative), such that each of the 800 emails was tagged with a category. Our categorization is orthogonal to that of Bacchelli et al. (Guzzi et al., 2013), since they focused on the content of the emails while we focus on whether or not the person writing the email feels positively or negatively. We started the categorization on one mailing list, then applied (and enhanced) it on the other mailing lists. As a third step, any discrepancies between the two raters' categories were discussed and resolved.

Findings. Table 4.8 gives an overview of the categories of the sentiments identified as well as their relative proportions, while Table 4.7 shows representative examples of each category. 34.21% of sample emails expresses sentiments in the developer mailing lists, compared to 27.87% for the user mailing lists. The percentages for individual categories should be interpreted like this: 34.71% of the 19.77% developer emails with positive sentiment correspond to "Satisfactory Opinion". In the following, we define each category and discuss the results across all analyzed emails ("Total" in Table 4.8), while RQ3 compares categories between user and developer mailing lists.

Positive Sentiment.

- *Satisfactory Opinion:* In its simplest form, satisfactory opinions are feelings of sympathy or positive impressions that people have towards a software system, new release, new feature or code change. Depending on the specific mailing list (developer or user), these subjects vary from users talking about a special feature or a product as a whole to developers dealing with new changes or parts of the code. According to Table 4.8, this group of emails is the most popular among emails with positive sentiment.

- *Friendly Interaction:* Since software development is a collaborative activity, constructive communication amongst the people involved might lead to higher productivity (Murgia et al., 2014). Well-mannered interactions with a positive undertone are a good start towards this. In response, if these emails are answered and guided with respect and a positive attitude, the interaction continues in a friendly and constructive manner. This group of email usually contains expressions of appreciation and support, such as "Hope this helps", "Thanks and really appreciate it". Since the goal of development mailing lists is to ask and answer questions of practitioners and users during software development and maintenance, being able to measure the amount of *Friendly Interaction* can help us to identify the heartbeat and soul of a community. Together with Satisfactory Opinion, *Friendly Interaction* is the most popular positive sentiment in the analysed mailing lists.

- *Explicit Signals:* Independent of the content type, authors often directly write indications about their good mood, for example in the form of emoticons (e.g., smilies), which reinforce the positive sentiment.

- *Announcement:* Many emails are written by someone to announce good news from the author's perspective, such as a new release that mitigates severe problems, or advertisement for a person during a vote for new members to join the developer or committer team. This group of emails expresses valuable interactions and milestones in a project.

- *Socializing:* Emails are used to socialize between community members by sending and discussing invitations to visit each other or attend community events or meetings. Such emails obviously contribute to a positive sentiment since they might strengthen good relationships among people.

- *Curiosity:* In the large amount of emails containing questions and answers, we found that there are other indicators for positive sentiments too. One important one is when

an email shows signals of curiosity when asking a question, or encourages and reveals signs of hope while answering a question. Such emails have positive sentiments since they provide a hint of participative and aspiring behaviour of a community. Hence, despite their relatively small market share in Table 4.8, this group of emails is highly valuable for a community.

Negative Sentiment.

- *Unsatisfied Opinion:* In contrast to *Satisfactory Opinion*, emails with negative sentiment might contain unpleasant or even offensive opinions towards various issues that people complain about. Similar to the positive counterpart, these issues can cover topics ranging from the software system as a whole to individual contributions or characteristics of a project like a code change, or even a new feature proposed by a person. *Unsatisfied Opinion* is one of the most important kinds of emails with negative sentiment.

- *Aggression:* This category covers emails with signs of poor and destructive communications, like flamewars, or people attacking or insulting each other. A second group of emails in this category consists of less extreme emails that ask their question or report a problem while complaining or while answering to an email in an angry way. As shown in Table 4.8, *Aggression* is one of the most frequently occurring negative sentiments, together with *Unsatisfied Opinion*.

- *Uncomfortable Situation:* Some emails have indicators that reveal the author of the email to be in an uncomfortable situation, such as suffering from a problem for ages, or being confused about unexplainable behaviour of the software system, or worrying about risks and fears. Some emails even reveal their authors to be under severe pressure like time constraints that might overwhelm them. This category is as common as the *Aggression* category. Independent of their specific rationale, these emails refer to negative symptoms reflecting poor quality of the software or parts of it (in the eyes of the unhappy author of the email), or simply to disagreement with management of the project.

- *Sadness:* Finally, there are also emails in which authors explicitly apologize or express feelings of sadness towards a problem. Although not aggressive, such emails also are carriers of unpleasant news or events, which is why we grouped them under negative sentiment. Fortunately, in many cases other people follow up comforting the sad author, giving rise to *Friendly Interaction*.

Neutral Sentiment. Emails with neutral sentiment refer to emails that show no sentiment, for example the author just describes a solution or a problem in a (possibly detailed) technical way, without showing specific emotions or other subjective signs. Another example of this group are the typical howto emails seen a lot in developing mailing lists in which guidelines or steps for doing a task are explained.

Based on Table 4.8, emails in which authors give their opinion towards an issue or towards the answer that they have got, can have a significant impact on the sentiment of emails. Similarly, emails in which the authors express situations such as the constraints or limitations that they have encountered, play an important role in negative sentiment emails. Finally, the quality of interactions among a community during the development of the project polarizes sentiment in emails substantially. By quality of interaction, we refer to how grateful, or supportive and helpful stakeholders are when interacting with each other or, conversely, in contrast how offensive they are. Apart from these factors, there are some minor factors that also can affect the sentiment of emails, like the amount of desire that people have towards a task, or friendship among people, as shown in the *socializing* category.

19.77% of the sampled emails were positive, 11.27% negative. We identified 6 positive sentiment categories, and 4 negative ones.

RQ3. Do developers and users show different sentiment?

Motivation. Now that we have categorized emails with different sentiment (positive, negative or neutral), we can analyze potential differences between developer and user mailing lists in terms of stakeholder sentiments. Intuitively, these two mailing lists have different purposes and different individuals subscribed to them. Typically, a developer mailing list is used for discussions about the actual development of the project, such as changes to the source code and related issues including bug fixes. On the other hand, configuration, how-to and support questions about the product are sent to the user mailing list. Despite these different purposes, one could expect sentiment in the developer list to be coupled to the user list, for example when complaints about a major bug trigger stressful discussions between developers to fixing this bug. Our third research question analyzes whether such coupling of sentiment does occur.

Approach. To answer this question, we compare the average monthly sentiment plots of Figure 4.1 between the developer and user mailing lists of both projects. We also use time

series analysis to compute the cross-correlation between both pairs of user/developer mailing list to quantitatively measure a (possibly lagged) correlation between both lists (Kabacoff, 2013). Finally, we compare the popularity of sentiment categories in Table 4.8 between both types of lists.

Findings. The user mailing list and developer mailing list of each project do not necessarily follow a similar trend. First of all, comparing the plots of users and developers shows that Tomcat and Ant follow different trends. While the Tomcat mailing lists both feature a downward trend from sentiment values around 0.1 to 0.05, the Ant mailing lists see an opposite trend, with the developer mailing list suddenly seeing a surge in average monthly sentiment towards 0.15 and higher instead of a downward trend towards 0.05. However, even for Tomcat the trends are not that highly correlated: the highest cross-correlation between developer and user mailing list occurs for a lag of 4 months, but only reaches a correlation of 0.19. Ant has a slightly higher (but still low) correlation of 0.22 for a lag of 1 month. Interestingly, in both cases the lag is positive, which suggests that the sentiment of the developer mailing list tends to follow (in time) that of the user mailing list. A potential hypothesis is that bugs and new features typically are proposed by users, then trickle down to developers.

The developer plots show substantially more fluctuation in sentiment value than the user plots, with a very large variation between the lowest and highest sentiment values. For example, the Tomcat developer list has positive peaks reaching 0.3, and the Ant developer list even reaching 0.6. The user mailing lists, even though varying as well, seem more compact, except for the last couple of months. The latter is likely due to the lower mailing list volume in that period, as discussed earlier.

User mailing lists contain substantially more "Curiosity", but less "Announcement" and "Socializing". Considering positive sentiment emails, developer and user mailing lists are quite similar since both have the highest proportions for *Satisfactory Opinion* and *Friendly Interaction*, which in total comprise around two-thirds of all positive emails. For users, these proportions are even a little higher. Furthermore, users express more their *Curiosity* about different issues, which means that users also convey more desire in comparison with developers. This seems normal, as most of the time, there are more newcomers among the users that are in the process of becoming more familiar with the system. Hence, those users show more desire to learn and obtain answers such that they become able to use the system properly. On the other hand, we can see a higher percentage of developers in the *socializing* category, which means that among developers there is a more friendly and decontracted atmosphere. Considering that developers need to collaborate more closely, this

observation indeed makes sense. The *announcement* category also takes up a bigger part in the developer mailing list. The reason for this is that among developers there are often announcements for a candidate during a vote in addition to regular announcements related to a new software release, while there are hardly any such announcements for users.

User mailing lists contain substantially more "Sadness", while developer mailing lists contain a lot more "Aggression". Indeed, comparing emails with negative sentiment, we can detect two differences between developers and users. Users adopt apologies and direct expressions for revealing their *Sadness* about two times more frequently than developers. This might be due to the fact that users are more likely to inadvertently make certain mistakes, after which they apologize or demonstrate similar expressions.

Finally, we have found that "Aggression" emails are a lot less common in the user mailing list, i.e., interactions among users rarely involves bad manners (6% vs. 19%). This means that developers more often state their negative opinions about bugs or new features. This might show that developers are very passionate about their work and the project as a whole, while (most of the) users are less negative than one would expect up front when complaining about their problems. The latter is a bit surprising, since in half of the negative user emails the user is unhappy because she is in a "Uncomfortable Situation" and definitely needs help.

Generally, we can say that developers and users show different proportions of sentiment categories during the construction and maintenance of the software project. This seems to confirm the different roles and perspectives of both groups of stakeholders towards a software system, and hence their attitude towards each other can be different. The developer mailing list comprises communication among colleagues, while the other list basically contains customer support communication. The main constant factor in both is the software product that is being discussed.

Sentiment on developer mailing lists chronologically seems to follow that on user mailing lists. User lists feature more "Curiosity" and "Sadness", but less "Aggression", "Announcement" and "Socializing".

4.5 Discussion and Threats to Validity

Although we have identified different categories of sentiment in developer and user mailing lists, and motivated our work based on the link between affect and productivity in other

domains, thus far we have not analyzed the possible link between affect and productivity in software development teams. Although this is outside the scope of the exploratory study performed in this paper, we did perform a small, initial study checking possible correlation between sentiment and productivity in the form of bug fixing activity.

For this reason, we have extracted the number of closed bugs for each month in the Tomcat and Ant bug repositories as an indicator of the effectiveness of the developers in fixing bugs. Similar to RQ3, we then calculate the cross-correlation between the monthly number of closed bug reports and lagged versions of the average Max SentiStrength emotion score. We then check large correlation values (positive or negative), as well as the corresponding lag. Only for the two Ant mailing lists, we found significant correlations of -0.43 for the user mailing list and 0.65 for the developer list. In both cases, we obtained this correlation for a lag of 5 months, in the sense that 5 months after a month with a particular average Max sentiment score, a higher number of closed bugs is observed. For the Ant developer list, we also observed a negative correlation of -0.62 for negative lag of 7 months. For the Tomcat project, no such high correlations could be observed.

As an start for an explanation, we studied the Ant release history. We found that the mean time between successive releases is around 6.5 months and the median time is around 4.2 months. The correlations for a lag of 5 months might be related to this release cycle time. Even if this would be correct (more analysis is needed for this), it is still not clear why the user and developer mailing lists show opposite correlation signs, nor why the Ant developer list shows a second negative correlation. We plan on exploring this in future work.

Regarding the threats, internal validity threats concern factors that might mistify the obtained results. We assume a causal relationship between a developer's sentiments and what he or she writes in emails, based on empirical evidence conducted in different domains (Shadish et al., 2001). In addition, the emails used in this study were collected over an extended period from developers or users not aware of being monitored, hence we are confident that the sentiments that we found are genuine. Another internal threat to validity is whether one can deduce the correct sentiment based on emails in isolation, without considering earlier emails in the thread. In most of the analyzed emails, the individual emails were indeed clear. In a minority of cases, when suspecting irony or observing specific cases of jargon, the raters looked up the earlier emails in the thread.

Threats to construct validity focus on how accurately the observations describe the phenomena of interest. Different stages in preprocessing of data such as filtering out automatic emails or extracting the natural text from emails might introduce some inaccuracies. However, after each stage enough testing has been done to assure the correctness of the data. To determine

the correct sentiment of each email, we relied on human raters. In earlier work (Murgia et al., 2014), we performed a user study with a large group of raters in the context of emotion mining. This showed that human raters agree sufficiently on "joy" and "sadness", which roughly coincide with positive and negative sentiment. For this reason, we used only two raters for this study.

Threats to external validity correspond to the generalizability of our experimental results (Shadish et al., 2001). In this paper, we study emails from two popular open source projects. We chose the two successful mature projects as a representative sample of the universe of open source software projects, with different development teams and from different domains. We have no evidence to support the assertions that these results are generalizable even to other projects that have most similarities with the studied projects. Replication studies should confirm whether other similar open source projects confirm our study results or not.

4.6 Related Work

Substantial work (Ambler, 2002; Brodkin, 2013; Fredrickson, 2001) has shown the influence of emotions on work results as well as on personnel effectivity in different workplace types. For example, De Choudhury et al. describe how assessment of employees' feelings enables an organization to detect causes of joy, sadness and frustration among the employees, based on which plans can be made to improve general emotions, workgroup dynamics, employee collaboration and hence work effectiveness (De Choudhury and Counts, 2013). Positive feelings inside a community can be an indicator of the quality and value of the interactions between people, which is why it is vital to support managers to discover the emotions of their teams (Murgia et al., 2014).

Previous studies mostly used sentiment analysis in the areas of marketing and financial markets but not in software engineering. For example, many online markets like mobile app stores or Amazon provide facilities for customers to assess their products and give their opinion. In such cases, sentiment analysis can be applied on the reviews of customers for products and services. Twitter and Facebook are also popular websites for sentiment analysis applications like monitoring the reputation of a specific brand (Feldman, 2013).

Similarly, analysis of financial markets uses sentiment analysis on news items, articles, blogs and tweets about companies to drive automated trading systems like StockSonar (Feldman, 2013). Vivek Sehgal et al. (Sehgal and Song, 2007) introduced a new approach for stock prediction based on sentiments of online messages, from which correlations between stock values and sentiments are learnt to enable prediction. Sanjiv R. Das et al. (Sehgal and Song,

2007) designed an algorithm to train small investor sentiment classifier from stock message boards, which can be used to assess the impact of small investor behaviour on stock market activity.

Despite extensive work on sentiment analysis for product reviews, marketing and financial markets, few research has studied the role of sentiment or emotion analysis in software engineering. Recently, Marta N. Gómez et al. (Canfora et al., 2014) examined whether the personality factors of team members and team climate factors are related to the quality of the developed software by the team. Analysis of student projects showed that software quality has a significant correlation with personality traits of team members like extroversion and team climate factors such as participation. Finally, they derived guidelines for software project managers with respect to team formation. Peter C. Rigby et al. (Rigby and Hassan, 2007) also used LIWC, a psychometrically-based linguistic analysis tool, to study the Apache httpd developer mailing list. In their study, they assessed the personality of four top developers, and two top developers that have left the project. They also examined the word usage on the mailing lists near releases to find the general attitude of developers in these periods. Blerina Bazelli et. al. (Bazelli et al., 2013) studied the personality traits of authors of questions on StackOverFlow.com, which is one of the most popular Question and Answers website used by all kinds of programmers. As a replication of Rigby et al.'s work, they applied LIWC (this time on SO questions), then categorized the extracted personalities based on the online reputations of the analyzed authors. They found that top reputed authors are more extrovert and issue less negative emotions. Against these studies, which are about intrinsic personality of developers, our paper looks at instantaneous sentiments to obtain the general trend of community sentiment.

Munmun et al. explored various emotional expressions of employees at 500 large software corporation by characterizing the emotional expression of the employees in a fine-grained continuous manner via posts on an internal Twitter-like microblogging tool (De Choudhury and Counts, 2013). They empirically show that affective expression in the enterprise can be the result of various workplace factors. These factors can be exogenous and endogenous workplace factors, geography of organization or the organizational hierarchy. This analysis extracted sentiment of employees over time, by analysing textual content of microblog posts using LIWC. They concluded that affective expression in the workplace can provide an efficient tool for assessing key factors and performance relevant outcomes.

Guzman et al. (Guzzi et al., 2013) used latent Dirichlet allocation to find the topics discussed in collaboration artefacts like messages from mailing lists and web discussions in university projects. They then used lexical sentimental analysis on the topics to obtain an average

emotion score for each of the topics. They evaluated their approach by interviewing the project leaders, which revealed the need for more details in the generated topic summaries.

Regarding the use of development mailing lists as source of valuable information related to software development, comprehension, and maintenance, Bacchelli et al. classified email content at the line level (Bacchelli et al., 2012). By combining parsing techniques and machine learning, they partitioned the content of development emails in five categories, i.e., natural language, source code, patch, stack trace, and junk. Later, Bacchelli et al. (Guzzi et al., 2013) also conducted research to better understand mailing list communication. They analysed OSS mailing lists both quantitatively and qualitatively, showing the wide range of topics discussed in email threads apart from source code, such as project status and social interactions. Our paper analyzes sentiment in the natural language category of email content.

4.7 Conclusion

Instead of algorithms or techniques to improve technical software development or maintenance issues, this paper focused on the human aspects involved with these activities. In particular, we studied the presence and evolution of positive and negative sentiment in the email communication of users and developers of two large open source projects.

On the one hand, we found that a state-of-the-art automatic sentiment analysis tool obtains only a modest precision due to the presence of ambiguous technical terms and the difficulty of distinguishing neutral (technical) emails from positive or negative ones. Hence, substantial work is needed to customize off-the-shelf sentiment analysis tools to the domain of software engineering. Still, the relatively uniform precision across each month already allowed to observe certain trends in the data.

On the other hand, we observed that developer and user mailing lists do contain sentiment (resp. 19.77% and 11.27% of the emails). We identified 6 categories of positive sentiment in emails, and 4 categories of negative sentiment. Furthermore, the two types of mailing lists have their own focus, with user mailing lists having more curiosity and sadness, and developer mailing lists more aggression, announcements and socializing. Furthermore, we found weak correlations that suggest sentiment in the developer mailing list to chronologically follow that of the user mailing list.

This paper only scratched the surface of sentiment analysis in mailing lists, hence a lot more work is needed on other systems and other tools. Ultimately, the goal of this field is to warn managers and other leading stakeholders of extremely positive or negative sentiment in a project, such that they can choose which of the widely known team building or other

activities are necessary to improve the stakeholders' affect, morale and productivity.

Table 4.7 Examples of categorization of email sentiments.

Category	**Example**
Satisfactory Opinion	Thank you very much Gianluca!,Great work so far.,And I know it will... We (UF, I cannot claim to represent the tomcat devs) are happy with a,simple round robin distribution for new requests.
Friendly Interaction	Thanks and really appreciate your assistance. butwith a good community effort we should be able to be done within a reasonable timeframe and enjoy a successful 2.0 version!
Explicit Signals	Fixing leaks is good :) Oh wait, there'sWindow$, so I guess there are takers ;)
Announcement	I am pleased to announce that I have a version of IvyDE ready to be released. With 8 +1 votes and no 0 or -1 votes, the vote is successful and Charles Duffy is now a committer.
Socializing	If anyone is interested in getting together for some drinks or exploring the city (I've never been to Vancouver) on Thursday, email me privately [. . .] Beer is always acceptable, though sometimes tough to ship.
Curiosity	I am keen on having two web applications be able to share sessions. I can't reproduce the scenario that causes the deletion but the "autoDeploy" attribute has piqued my curiosity.
Unsatisfied Opinion	Isn't this a bit premature, junit 4 isn't even "out" yet. And no, Tomcat is not reliable at all, it's more kinda toy for bored developers such as me.
Aggression	Anyone volunteering to buy me a second pair of glasses? Or does gump drink too much ? who are the stupid people who manages this group.
Uncomfortable Situation	I'm absolutely furious that Tomcat did not say (almost) anything in its logs. I have not been into ivy(ide) (yet) and currently heavily constrained ontime (new job)
Sadness	Oh geez... really?,We're going to have a top-post vs bottom-post flame-war?? I was very sloppy and changed the wrong one.

Table 4.8 Categorization of email sentiment.

		Total		Developer		User	
Positive Sentiment	Satisfactory Opinion	19.77%	34.71%	22.48%	33.33%	17.05%	36.54%
	Friendly Interaction		33.88%		33.33%		34.62%
	Explicit Signals		13.47%		14.49%		12.12%
	Announcement		6.61%		8.70%		3.85%
	Socializing		5.79%		7.25%		3.85%
	Curiosity		6.61%		2.90%		11.54%
Negative Sentiment	Unsatisfied Opinion	11.27%	18.84%	11.73%	19.44%	10.82%	18.18%
	Aggression		11.59%		16.66%		6.06%
	Uncomfortable Situation		52.17%		52.78%		51.51%
	Sadness		17.39 %		11.11%		24.24%
Neutral Sentiment		68.95%		65.80%		72.13%	

CHAPTER 5 ARTICLE 2: THE IMPACT OF HUMAN DISCUSSIONS ON JUST-IN-TIME QUALITY ASSURANCE

Abstract

In order to spot defect-introducing code changes during review before they are integrated into a project's version control system, a variety of defect prediction models have been designed. Most of these models focus exclusively on source code properties, like the number of added or deleted lines, or developer-related measures like experience. However, a code change is only the outcome of a much longer process, involving discussions on an issue report and review discussions on (different versions of) a patch. Similar to how body language implicitly can reveal a person's real feelings, the length, intensity or positivity of these discussions can provide important additional clues about how risky a particular patch is or how confident developers and reviewers are about the patch. In this paper, we build logistic regression models to study the impact of the characteristics of issue and review discussions on the defect-proneness of a patch. Comparison of these models to conventional source code-based models shows that issue and review metrics combined improve precision and recall of the explanatory models up to 10%. Review time and issue discussion lag are amongst the most important metrics, having a positive (i.e., increasing) relation with defect-proneness.

5.1 Introduction

The later defects are identified and fixed, the more expensive they become (Boehm and Papaccio, 1988). This is why companies try to limit the number of major defects discovered by end users to a minimum, as such defects may degrade a company's reputation and lead to irreversible financial loss. In addition to a well thought-out quality assurance strategy comprising tests, reviews and other activities (Galin, 2003), prediction models of defect-prone files and defect-introducing patches have gained importance.

Initial work on defect prediction considered files or modules as granularity for predictions (Gyimothy et al., 2005; Hassan, 2009; Munson and Khoshgoftaar, 1992), where models would predict which files have the highest probability of containing defects and hence should be tested more thoroughly. Later studies (Fukushima et al., 2014; Kamei et al., 2013a; Kim et al., 2008; Mockus and Weiss, 2000; Shihab et al., 2012) suggested Just-In-Time (JIT) defect prediction models that focus on software patches instead of on files or modules. Such models are more actionable and provide large effort savings (Kamei et al., 2013a), since they

locate specific patches as defect-prone, instead of large files or modules. In addition, predictions can be applied at the exact time when a patch is being reviewed and consequently, the responsible developer can intervene quickly.

State-of-the-art approaches for JIT prediction mainly use measures related to the actual code that is being changed, such as the size of the code change (churn) or the developer's track record in the project, but ignore the actual feelings and insights of the stakeholders involved in reviewing a patch or (earlier on) commenting on the bug or feature request that the patch is implementing. For example, when an issue report provokes many comments by different people in a relatively short time frame, this could (amongst others) indicate that the corresponding bug or feature is complicated and hence has a higher risk of introducing bugs than an issue report with only one or two comments. Similarly, a patch requiring multiple revisions before being accepted or yielding many code comments during review could indicate an increased risk, even though the reviewers in the end accepted the patch.

To understand the relation between issue/review discussions and the defect-proneness of a patch, this paper performs a large empirical case study on 10 OpenStack and 5 Eclipse open source projects. We chose these projects, as they are large projects that have adopted code reviews on a large scale and have a reasonable traceability between commits, reviews and issue reports. Using comments in the issue tracking and review repositories as proxy for human discussions, we compare explanatory models containing issue and/or review metrics to a baseline model containing only change-related metrics. Using this comparison, we address the following research questions:

RQ1) *How well can issue discussion metrics explain defect-introducing changes?*

> For five of the project, explanatory JIT models with issue discussion-related metrics show improvements of 3% to 10% in precision or recall.

RQ2) *How well can review discussion metrics explain defect-introducing changes?*

> Models with review discussion-related metrics show similar improvements as RQ1 in precision or recall for 9 projects . Some review discussion metrics figure amongst the most impactful metrics.

RQ3) *How well can issue and review discussion metrics explain defect-introducing changes?*

> For half of the projects, models augmented with both review and issue discussion metrics improve precision and recall by 3% up to 17% (in one case). Review time and issue discussion lag are amongst the most important metrics in the combined models.

In the remainder of this paper, we first describe the necessary background notions for our work (section 7.2). Next, we describe the case study setup (section 6.3), then present the results of the three research questions (section 6.4). After discussion and threats to validity (section 6.6 and section 7.4), we discuss related work (subsection 7.2.3) and we finish with conclusions (section 7.5).

5.2 Background

This section provides background information about JIT models and issue/review environments.

5.2.1 Source Code-based JIT Models

Various researchers have built JIT models (Fukushima et al., 2014; Kamei et al., 2013a; Kim et al., 2008; Mockus and Weiss, 2000; Shihab et al., 2012). Here, we focus on the more recent work of Kamei et al. (Kamei et al., 2013a). They considered 14 software change metrics, grouped into 5 dimensions, to explain whether a code change introduces a defect. These metrics, which are derived from the source code repository data of a project, would be measured for a new code change and plugged into a prediction model trained on earlier code changes (for which one knew in the meantime whether they introduced a defect) to obtain a risk probability. Based on a threshold, this probability would then suggest whether or not this new code change is expected to be defect-prone. Developers could then immediately review, test and revise the code change and resubmit.

Table 5.1 shows the 14 metrics and 5 dimensions, as well as their rationale. All of them can be calculated using only the project's version control system, such as Subversion or Git. The resulting models performed well on a large set of open and closed source systems. Hence, we use these change metrics to build a baseline model to which we compare our new models that add issue and/or review discussion metrics.

5.2.2 Issue and Review Repositories

An issue tracking system (e.g., Bugzilla, Launchpad or Jira) is a repository used by a software organization to enable users and developers to report defects and feature requests. It allows such a reported issue to be triaged and (if deemed important) assigned to team members, to discuss the issue with any interested team member and to track the history of all work on the issue. During these issue discussions, team members can ask questions, share their opinions and help other team members. Some projects also use an issue tracking system to

Table 5.1 Overview of change metrics of Kamei et al. (Kamei et al., 2013a).

Dimension	Name	Definition	Expected Rationale
Diffusion	NS	#modified subsystems	Modifying more subsystems increases the defect-proneness.
	ND	#modified directories	Modifying more directories increases the scattering of change and the probability of defects.
	NF	#modified files	Modifying more files increases the probability of defects.
	ENT	distribution of changes across all files	Changes affecting multiple files *equally* likely are more risky.
Size	LA	#lines of code added	Larger changes increase the probability of defects.
	LD	#lines of code deleted	Defect-proneness increases when more code is removed, since incorrect code could be deleted accidentally.
	LT	#lines of code in a file before the change	Larger code might be more complex to understand, hence, modifying it is more likely to be defect-prone
Purpose	FIX	is the change a defect fix?	Fixing a defect touches a buggy area of the code, hence the probability of introducing a new defect is higher.
History	NDEV	avg. #developers that changed the files before	Different developers modifying the same file may lead to misunderstanding.
	AGE	average time (#days) since the last change	Recently changed files are more defect-prone than stable code.
	NUC	#unique changes to modified files	The more files have been changed, the more opportunities for defects.
Experience	EXP	#prior commits by the developer	More experienced developers are less likely to introduce defects, unless they make more ambitious changes.
	REXP	#prior commits by the developer weighted by their age	Developers that recently modified a file have more fresh knowledge about the code base.
	SEXP	#prior commits by the developer on a subsystem	Developers that are dominant in a particular subsystem are less likely to introduce a defect there.

review patches and bug fixes (instead of using a dedicated reviewing environment like Gerrit), however we do not consider such projects in this paper.

According to Rigby et al (Rigby and Bird, 2013), modern software organizations have embraced lightweight processes for reviewing code changes, i.e., to decide whether a developer's change is safe to integrate into the official version control system. During such lightweight code review, assigned reviewers make comments on a code change or ask questions that can lead to a discussion of the change and/or different revisions of the code change, before a final decision is made about the code change. If accepted, the most recent revision of the code change can enter the version control system, otherwise the change is abandoned and the developer will move on to something else. Modern web apps like Gerrit are used to support this review process.

In this paper, we are interested in understanding whether the characteristics of collaborative group discussions of issue reports and code reviews, such as the discussion volume, intensity or positivity, can provide hints about the defect-proneness of the resulting accepted code change. For example, the behaviour of the discussion participants could show them being uncomfortable with a code change, even though it would pass through review. For this reason, we build explanatory models with issue and/or review metrics to understand why code changes that successfully passed reviewing turn out to be defect-introducing.

5.3 Case Study Setup

This section explains the methodology used to address our three research questions. We discuss the selection of case study systems, identification of defect-prone code changes, extraction of issue and review metrics, model building and model validation.

5.3.1 Selection of Case Study Systems

Our aim is to study whether human discussions of issue reports and code review can provide some indication that a code change that successfully passed review introduces a defect. This model would be used right after code review finished, as a final check before the code change would be integrated into the version control system. The only requirement is that the code change is linked to an issue report and code review, which is readily done by modern software organizations. Therefore, to conduct our empirical study, we required projects with a substantial number of commits linked to issues and reviews.

Openstack[1] and Eclipse[2] are two popular open source ecosystems featuring a large number of sizeable projects that have adopted code reviewing in the last couple of years (issue reports have been adopted by open source projects for a long time). They have adopted modern technology for version control systems (git), issue repositories (Bugzilla and Launchpad) and code review repositories (Gerrit). Furthermore, they are among the pioneers of pushing developers to explicitly link code changes to issues and reviews.

To obtain the version control, issue report and code review data of these ecosystems, we used the data set graciously provided by Gonzalez-Barahona et al. (Gonzalez-Barahona et al., 2015). They developed the MetricsGrimoire tool suite to mine the repositories of OpenStack and Eclipse, then store the corresponding data into a relational database. We used their version control, issue report and code review data sets[3] to perform our study.

5.3.2 Linking Commits to Bug Reports and Reviews

As mentioned in the previous section, by using Barahona et al.'s exposed databases we got access to the git version control, issue report and Gerrit review data of OpenStack and Eclipse. Then, we used heuristics to identify links between commits and reviews, and also between commits and issue reports.

In particular, links from an accepted review to its corresponding git commit can be identified by searching the Gerrit reviews for the commit identifier of the accepted revision of a patch. These commit identifiers had not been stored in Barahona's exposed databases, hence we modified MetricsGrimoire to download this additional information from the review repository, then updated Barahona's database with the extracted commit identifiers.

However, only for 60% of the accepted reviews, the commit identifier mentioned in the code

[1] http://openstack.org
[2] `http://eclipse.org`
[3] `http://gsyc.es/~jgb/repro/2015-msr-grimoire-data`

Table 5.2 Statistics of the Studied OpenStack and Eclipse Projects.

	Project	Total #commits (%defective)	Start Date	Total #reviews (%linked)	#commits linked (%total)	Total #issues (%linked)	#commits linked (%total)	#commits linked to both (%total)
Openstack	cinder	5013 (30%)	2012-05-03	4209 (65%)	2721 (54%)	2057 (78%)	1654 (33%)	1654 (33%)
	devstack	5515 (13%)	2011-09-11	3271 (77%)	2507 (45%)	1088 (70%)	827 (15%)	827 (15%)
	glance	4125 (28%)	2010-08-11	2385 (77%)	1844 (44%)	1521 (75%)	1073 (26%)	1155 (28%)
	heat	7341 (23%)	2012-03-13	5324 (72%)	3859 (52%)	1781 (83%)	1835 (25%)	1982 (27%)
	keystone	6654 (20%)	2011-04-14	4214 (73%)	3084 (46%)	1964 (73%)	1530 (23%)	1996 (30%)
	neutron	8512 (28%)	2010-12-31	6806 (64%)	4384 (51%)	3960 (70%)	2979 (35%)	2979 (35%)
	nova	32998 (25%)	2010-05-30	14699 (79%)	11583 (35%)	8210 (68%)	5940 (18%)	6600 (20%)
	...-manuals	8483 (26%)	2011-09-20	7356 (67%)	4951 (58%)	2867 (79%)	2121 (25%)	2290 (27%)
	swift	4333 (17%)	2010-07-12	2203 (72%)	1580 (36%)	1106 (46%)	953 (22%)	1043 (24%)
	tempest	5813 (18%)	2011-08-26	3875 (80%)	3105 (53%)	1619 (70%)	1337 (23%)	1453 (25%)
Eclipse	cdt	24025 (40%)	2002-06-26	1006 (92%)	721 (3%)	13576 (54%)	480 (2%)	9610 (40%)
	egit	4513 (48%)	2009-09-29	4270 (86%)	3655 (81%)	2200 (77%)	1624 (36%)	1850 (41%)
	jgit	3984 (36%)	2009-09-29	3736 (84%)	3187 (80%)	513 (41%)	598 (15%)	756 (19%)
	linuxtools	9712 (41%)	2007-02-02	3925 (86%)	3302 (34%)	1938 (56%)	485 (5%)	1456 (15%)
	scout.rt	3299 (39%)	2010-11-25	1668 (59%)	989 (30%)	1350 (80%)	825 (25%)	2045 (62%)

review corresponds to the actual commit in the official git repository of a project. The reason is that while a code change is being reviewed, other developers' code changes that are being reviewed in parallel, get accepted and entered in the version control system. Hence, by the time the former code change is accepted, it first needs to be integrated with these newly accepted commits ("rebasing" (Bird et al., 2009c)). As a result, a new git commit with new identifier is created that replaces the accepted code change. Since older versions of Gerrit did not update afterwards the code review with the new identifier, those reviews could not be directly linked with the version control system.

Fortunately, for more than 50% of the missing cases, we either found an additional review comment that has been added after rebasing mentioning the correct Git commit identifier, or we found the identifier of the review in the commit message of a git commit. Hence, using these 3 different techniques, we were able to retrieve the links in most of the Eclipse and OpenStack projects for at least 70% of the reviews.

In order to link commits to issue reports, we noticed after manual inspection of commit messages that Eclipse developers almost consistently mention the issue identifier in their commit message, following these regular expressions (for Openstack, we changed this format a bit, as issue numbers for Openstack just have 6 or 7 digits):

(bug|issue)[:#\s_]*[0-9]+

(b=|#)[0-9]+

[0-9]+\b

\b[0-9]+

However, after checking whether the identified numbers correspond to actual issue identifiers,

just 2% of related issues were found in this way. Instead, 35% of the links were found through issue report comments that mentioned a Git commit identifier, and the remaining links between issues and commits were identified through review info (since we had found sufficient links between reviews and commits). To detect the links between issues and reviews, we referred either to issue identifiers mentioned by review comments or the name of the branch on which a code change had been made, since some of them follow the naming convention "bug/1491511" with "1491511" an issue identifier. On average, more than 76% of the links (72% for Openstack and 81% for Eclipse) with issues to their commits were identified. To evaluate whether the identified links were correct, we manually examined a random number of them.

After linking commits to reviews and issues, and limiting the analyzed time period to those commits made starting from the time when Gerrit was introduced in a project, we eventually retained those projects with more than 3,000 commits, a reasonable amount of linkage (more than 500 linkages) and defect-prone commits (more than 10%). Out of 337 Openstack projects, 11 projects with good linkage and sufficient defect-prone commits were selected. For Eclipse, 10 projects out of 612 had more than 3000 commits and more than 1000 reviews, however most of them did not have enough issues. For example, *platform.ui* has more than 17,000 fixed issues, but most of them were reported before the project started using Gerrit (April 2013), and only 500 issues were reported afterwards. Therefore, we eventually selected only 5 Eclipse projects. Table 6.1 shows the statistics of our data set.

5.3.3 Identifying Defect-introducing Commits

Identifying which commits introduce a defect is not straightforward. Various algorithms and heuristics exist, with the SZZ algorithm (Śliwerski et al., 2005) still one of the most popular approaches. It automatically locates defect-introducing patches by linking information from a version control system like Git to a bug repository like Launchpad or Bugzilla. It consists of three steps: 1) finding all bug fix commits by identifying for each issue report linked to a commit whether it is a bug, in which case the commit likely is a bug fix; 2) identifying, using a standard command like "git blame" the most recent commits that changed the lines changed by the bug fix; and 3) tagging those commits as potentially defect-introducing.

As such, SZZ assumes that a bug fix only changes the source code lines that had a defect and that the most recent commit(s) that changed those lines were the defect-introducing commit(s). Typically, all commits pointed out by "git blame" are considered to be defect-introducing commits, although heuristics exist to filter out commits that appeared after the bug report was filed. We used the SZZ implementation of Kamei et al. (Kamei et al., 2013a)

to identify the defect-introducing commits in our case study systems. We customized this implementation to use the mapping from commits to issues obtained in Section 5.3.2 instead of a keyword-only approach.

5.3.4 Discussion Metrics

Table 5.3 Summary of issue and review discussion measures.

Dimension	Domain	Name	Rationale
Focus	both	commenter experience	The more experienced in leaving comments (for both reviews and issues), the more participative and helpful discussions could be, reducing the risk of defects slipping through.
	issue	reporter experience	The more experienced in issue reporting, the more accurate their reports could be, the more precise the issue could be solved.
	review	reviewer experience	The more experienced in reviewing, the more accurate they are expected to be, hence the higher the chance that any serious risk has been identified and remedied.
		#patch revisions	The more revisions a patch required, the more issues were detected and hence could still linger in the final patch revision.
		#inline comments	The number of comments reviewers made on specific lines of a patch instead of general reviews, as an indicator of the degree of detail of reviewing. The more detail, the lower the risk of remaining defects.
Length	both	#comments	The more comments are posted in a discussion, the more risk might be involved.
		comment length	The number of lines of comments on an issue or review, as a measure of the amount of discussion, may indicate that the discussed commit has a high likelihood of introducing a defect.
Time	review	review time	The total time spent on reviewing, might be related to the risk and defect-proneness of the issue.
	issue	fix time	The total time allocated to fix an issue might be related to the risk and defect-proneness of the issue.
	both	average discussion lag	The average time in between comments could be related to the risk of an issue or review, with risky ones seeing faster replies to comments.
Sentiment	both	Comment Sentiment (max/min/avg/extreme)	Negative sentiment of the participants during issue or review discussions may reveal doubt. Four variations of the sentiment metrics are provided.

In this section, we describe the issue and review discussion metrics that we use in our statistical models, complementing the change-level metrics of Table 5.1. The issue discussion metrics are mined from issue repository comments, hence we calculate them on a per-issue report (and hence per-commit) basis[4]. The review discussion metrics are mined from the Gerrit reviews and comments, and hence are calculated on a per-review basis. We grouped all discussion metrics into four categories, as Table 5.3 shows.

Focus. During review, reviewers can ask a developer to incorporate certain changes. If the developer agrees, these changes will result in a new revision of the patch that will undergo another round of reviewing by the same reviewers. When reviewers eventually are happy with a revision, that revision will be accepted for integration into the version control system (possibly with rebasing). The more revisions a patch had to go through before acceptance, the riskier, as it could indicate issues with a developer's mastery of or familiarity with the problem at hand, or could mean that the bug fix or feature being implemented is complicated. On the other hand, it could also indicate that the reviewers are serious about their work, as they are pointing out many problems.

Furthermore, Bacchelli et al. (Bacchelli and Bird, 2013) and McIntosh et al. (McIntosh et al., 2014) found that modern reviewing techniques, such as those supported by environments like

[4]For commits linked to multiple issue reports, we randomly select one of the issue reports.

Gerrit, do not imply high quality reviews. Indeed, one can "review" a patch by just pointing out typos or without checking how the changed lines integrate with existing code. More experienced reviewers or issue commenters might be more aware of these pitfalls and hence perform more focused reviews, reducing the risk of defects. This is also why we measure *#inline comments*, which corresponds to comments annotating specific code lines in a patch. For example, if line 12 of a patch should be improved, a reviewer could put a comment on that specific line, instead of posting a global review comment. More such comments again means more focused reviewing.

Length. Complex issues and patches can be more controversial and hence require more discussion than simple ones. We adopt this idea in our models by measuring the *#comments* and *comment length*. Initially, we also used *#authors*, but we found that this was too highly correlated with *#comments*, which is an easier metric to measure. Note that one cannot blindly count all comments, since some review or issue comments are automatically generated messages that do not constitute human discussion, and hence should be filtered.

Time. Through issue and review comments, people discuss, share their ideas and help each other regarding the issue or patch under consideration. Hence, the absence of communication for a prolonged time may indicate miscommunication or even indifference, which is a sign of risk. Hence, we capture these ideas by measuring the *fix time* and *average discussion lag* for issue reports and reviews. The former spans from creation date until the final comment, while the average discussion lag is the average of the time period between each subsequent pair of comments.

Sentiment. Until now, all selected metrics focused on quantitative aspects of the discussion, in the sense that they count a volume or measure time. However, issue report and review comments especially contain natural language content capturing the direct opinion of developers or other stakeholders on an issue or patch. Whenever people communicate, their words automatically incorporate certain feelings or sentiment (attitude of towards a subject (Mishra and Jha, 2012)) to convey their message or understand other people's reaction. For example, one developer or user might say "We are happy with a simple round robin distribution for new request", or "I am absolutely furious that the application did not say almost anything in its logs". Sentiments are not limited to verbal communication, but are also expressed when using computer-aided communication (Thelwall). Hence, the usage of friendly, positive words is a good indicator that a person is happy with a certain issue or patch, while angry expressions could indicate difficult interactions and stress that might reflect in the quality of the resulting work.

To capture such qualitative opinions, we extract and quantify the sentiments expressed by

issue and review comments. In sentiment analysis (Pang and Lee, 2008), the "polarity", i.e., positive or negative attitude, and "degree" of a document are measured quantitatively. A larger degree of sentiment represents more positive (or negative) sentiments and attitudes towards a subject, topic, idea or even a person. Most sentiment mining tools generate polarity and degree per sentence or paragraph. Since we need one sentiment score per issue report or review (not per sentence/paragraph), we are interested in the Max, Min and $Average$ of the individual sentiment scores to obtain one sentiment value. In addition, we also include the $Most\ extreme$ sentiment value, which is the sentiment value with the largest absolute value across all sentences/paragraphs of an issue report or review.

In practice, we first filter out any extra data other than human natural language, as comments often contain different kinds of information like source code snippets and stack traces. In our case, a lightweight method based on regular expressions turned out to be the most effective filtering approach (Bacchelli et al., 2010). We then applied the SentiStrength tool on the pre-processed comments to obtain sentiment scores from -5 to 5 for each paragraph. SentiStrength is one of the state-of-the-art lexical sentiment mining tool (Thelwall et al., 2010), which is easy to use and has been used successfully by several research projects (Thelwall et al., 2010). Alternatively, one could use machine learning-based sentiment mining (Socher et al., 2013) or, instead of sentiment, one could also measure other types of affect like emotions or polarity (Ortu et al., 2015b), however those tools are still in an early stage.

5.3.5 Building Explanatory Models

Similar to previous work (Basili et al., 1996; Cataldo et al., 2009; Kamei et al., 2013a), a logistic regression model is used to build an explanatory model of defect-prone code changes. For each commit, such a classification model returns a probability between 0 and 1 of defect-proneness. Based on a threshold value, one can then classify a commit as defect-prone (probability higher than threshold) or safe (probability lower than threshold). This paper uses the model building scripts of Kamei et al. (Kamei et al., 2013a), with a standard threshold of 0.5.

We use logistic regression to build explanatory models, i.e., models that explain defect-proneness of the data set on which they are trained rather than predict defect-proneness of a different data set (like on commits of a subsequent year). The evaluation of such models typically is conducted using 10-fold cross validation (Efron and Tibshirani, 1995). According to this technique, the dataset is randomly divided into 10 folds based on stratified partitioning such that each fold has the same proportion of defect-prone commits as the full data set. Then, one fold is picked as test data for model validation, while the rest of the data is used

as training data to build a model. This process is repeated for each of the 10 folds as test data, yielding a confusion matrix, based on which performance measures can be calculated.

Before training a model, we first perform three filtering steps. After collecting the required metrics, we first removed highly correlated factors. For this, we computed the variance inflation factors (VIF) for each metric and removed those metrics with variance inflation factor greater than 5. Then, we also applied Mallow's Cp criterion (Mallows, 1973) using a stepwise variable selection technique to remove the rest of the collinear metrics and those that do not affect the model. Step by step, this technique removes the worst metrics, i.e., metrics without any effect, until the deletion of the remaining metrics starts degrading the model.

Second, as Table 6.1 shows, our dataset is not balanced: the number of defect-introducing commits is much lower than the number of safe commits. According to Matsumoto et al. (Kamei et al., 2007), this degrades the performance of statistical prediction models. To avoid this problem, we resample the training set data, similar to previous work (Kamei et al., 2013a). With this approach, non-defect-introducing commits randomly are removed from the training set until we have the same number of defect-introducing and safe commits. Note that we cannot resample the test set, as this would bias the evaluation results (since real-life data is imbalanced, and hence we should evaluate our models as such).

Third, similar to Kamei et al. (Kamei et al., 2013a), we applied a standard log transformation to each metric with positive skew (i.e., having a long tail of values towards higher values), to even out the skewing effects on the model.

5.3.6 Validation of Model Performance

We validate both the models themselves as well as the importance (impact) of the metrics for the models.

Evaluating the Models.

To evaluate the defect-prone classifiers, we use the common *precision*, *recall*, *F1-measure* and *AUC* measures. The first three metrics are derived from a model's confusion matrix. Such a matrix summarizes the four different cases of a classification: a model can correctly classify a commit to be defect-prone (TP, true positive) or safe (TN, true negative), incorrectly classify a defect-prone commit to be safe (FN, false negative) or a safe commit to be defect-prone (FP, false positive).

Using these concepts, *precision* then is the percentage of commits classified as defect-prone by a model that is actually defect-prone, i.e., $\frac{TP}{TP+FP}$. This gives an indication of how often a

model's recommendation is correct (lack of false alarms). On the other hand, *recall* measures what percentage of the actual defect-prone commits in the data that can be found by the model, i.e., $\frac{TP}{TP+FN}$. Since there is a tradeoff between precision and recall, the *F1-measure* usually is computed, which is a weighted average of precision and recall: $\frac{(2\times Recall\times Precision)}{(Recall+Precision)}$.

Since the output of a logistic regression model is a probability value between 0 and 1, a threshold needs to be chosen to map the model value to 0 or 1. This means that a model's performance depends on the choice of this threshold. The AUC (Area Under the Curve) of the ROC (Receiver Operating Characteristics) curve is a measure that represents the overall performance of a logistic regression model across all thresholds (Lessmann et al., 2008). The larger the AUC, the better the classification performance. In particular, the larger the AUC is compared to 0.5, the better the model performs than a random classification model.

*.Evaluating the Metrics

To evaluate which metrics have the largest impact in the models, we compare our models to the code change-based baseline models incorporating the metrics of Table 5.1. For this comparison, we use a hierarchical analysis starting with the baseline model that uses software change measures only. Then, step by step, each dimension of issue or review discussion metrics is added to the model to study its importance in the model. To determine if a dimension provides new knowledge about defect-prone commits, we use an ANOVA test to compare a regression model with the previous one. Given an α value of 0.05, a $p-value$ lower than 0.05 rejects the null hypothesis that there is no significant difference in fit between both models, or in other words a $p-value < 0.05$ means that the new model performed significantly better than the old one. In contrast to the classifier performance using precision and other metrics, for model comparison we use the entire dataset to build one logistic regression model in each step.

After determining the issue or review discussion dimension with the highest impact, one can then analyze which individual metrics have the highest impact in the model using the effect size of Shihab et al. (Shihab et al., 2013). This approach first takes the full logistic regression model with the median value of each metric as input (the mode value for categorical variables like Change_Type). The model's output value for these inputs is recorded as the "base value". Then, in order to find the impact of each variable on the model, we replace for one metric at a time the median value by the median plus one standard deviation (for categorical values, the second most common value will be used to replace the mode), the rest of the variables stay on their median value. The effect size of this metric on defect-proneness can then be calculated as $\frac{newvalue-basevalue}{basevalue}$. We repeated this method for all significant variables to find their relative impact.

Since the median values represent "common" values for each metric, and adding a standard deviation is a "common" change in metric value, the effect sizes of different metrics can be compared directly to each other in order to find the metrics with the largest impact on defect-proneness. In particular, the effect sizes are independent of the unit of the metrics, in contrast to for example odds ratios (Bland and Altman, 1996). A positive effect size indicates an increase in defect-proneness for an increase in independent variable, while a negative effect size indicates a decrease in defect-proneness.

5.4 Case Study Results

RQ1. How well can issue discussion metrics explain defect-introducing changes?

Table 5.4 Performance of explanatory models with issue metrics. ^{b} means "base model", increases of $\geq 3\%$ in bold, and decreases of $\leq -3\%$ underlined.

project	prec^b	rec.^{b}	f^b	auc^b	prec	rec.	f	auc
cinder	0.82	0.49	0.62	0.53	0.83	**0.59**	**0.69**	**0.56**
devstack	0.41	0.64	0.50	0.63	0.39	0.66	0.49	0.62
glance	0.64	0.60	0.62	0.57	**0.67**	0.61	0.64	**0.60**
heat	0.73	0.54	0.62	0.58	0.75	**0.61**	**0.67**	**0.61**
keystone	0.61	0.68	0.65	0.70	0.59	0.68	0.64	0.69
neutron	0.65	0.73	0.69	0.71	0.64	0.74	0.69	0.70
nova	0.75	0.64	0.69	0.59	0.77	0.61	0.68	0.61
. . . -manuals	0.81	0.55	0.66	0.57	0.82	0.57	0.67	0.59
swift	0.46	0.67	0.54	0.64	0.45	0.66	0.54	0.63
tempest	0.58	0.56	0.57	0.53	**0.67**	0.58	**0.62**	**0.61**
cdt	0.73	0.70	0.72	0.72	0.73	0.70	0.72	0.71
egit	0.86	0.67	0.75	0.71	0.85	0.66	0.74	0.71
jgit	0.70	0.65	0.67	0.67	0.72	**0.68**	**0.70**	0.69
linuxtools	0.83	0.61	0.70	0.69	0.84	0.61	0.70	0.70
scout.rt	0.67	0.70	0.68	0.70	0.67	0.71	0.69	0.70

Approach. To understand how much information issue report discussions contain about the risk of their corresponding code change, we build explanatory models for the risk of a code change using the issue discussion metrics of Table 5.3, combined with the software change metrics of Table 5.1. Table 5.4 compares for all analyzed projects the performance metrics of the baseline model (only code change metrics) and the issue metrics model.

To identify the most important metrics in the models of each project, we calculated for all

Table 5.5 Performance of explanatory models with review metrics. ^{b} means "base model", increases of $\geq 3\%$ in bold, and decreases of $\leq -3\%$ underlined.

project	prec^b	rec.^{b}	f^b	auc^b	prec	rec.	f	auc
cinder	0.66	0.61	0.63	0.60	0.67	0.59	0.63	0.61
devstack	0.40	0.64	0.49	0.67	0.37	**0.69**	0.48	0.66
glance	0.56	0.61	0.58	0.58	0.57	0.59	0.58	0.59
heat	0.48	0.55	0.51	0.58	0.47	**0.60**	0.53	0.58
keystone	0.49	0.61	0.55	0.61	0.46	**0.64**	0.54	0.60
neutron	0.68	0.63	0.68	0.68	**0.72**	**0.71**	**0.71**	**0.71**
nova	0.61	0.65	0.63	0.64	0.61	0.65	0.63	0.64
. . . -manuals	0.51	0.51	0.51	0.56	**0.56**	**0.59**	**0.57**	**0.61**
swift	0.41	0.60	0.48	0.63	0.42	0.59	0.49	0.63
tempest	0.42	0.59	0.49	0.59	0.40	**0.64**	0.49	0.58
cdt	0.55	0.70	0.62	0.72	0.54	**0.73**	0.62	0.72
egit	0.76	0.65	0.70	0.68	**0.79**	**0.72**	**0.75**	**0.72**
jgit	0.63	0.66	0.64	0.66	**0.66**	**0.73**	**0.69**	**0.70**
linuxtools	0.76	0.67	0.71	0.69	0.76	0.64	0.70	0.69
scout.rt	0.54	0.63	0.58	0.65	0.55	0.65	0.59	0.66

metrics the effect size, as explained in subsection 6.3.5. Then, for each metric, we calculated the median effect size across all Openstack projects (and, separately, across all Eclipse projects) in order to globally rank the metrics from most extreme effect size (either positive or negative) to closest to zero. If a variable did not appear in a project's model, we gave it an effect size of zero for that model. Finally, Table 5.7 and Table 5.8 only show those metrics with median effect size different from zero, i.e., that appeared in the models of at least 50% of the projects.

Findings.

Five of the projects see an improvement in precision and/or recall of 3% up to 10%. ANOVA analysis showed that models augmented with metrics related to issue discussions significantly improves upon the baseline models (even though for some projects a decrease can be observed). The tempest project sees the largest improvements, with precision increasing by 9% and the AUC by 8% compared to the baseline model. Similarly, cinder and heat improve their recall by 10% and 7%, respectively. glance increases its precision by 3%, while jgit improves its recall by the same amount. On the other hand, we notice that nova loses 3% of precision and recall, respectively.

The most important metrics are FIX, LA, NDEV and Issue Commenter Experi-

Table 5.6 Performance of explanatory models with issue *and* review metrics. ^{b} means "base model", increases of $\geq 3\%$ in bold, and decreases of $\leq -3\%$ underlined.

project	prec^b	rec.^{b}	f^b	auc^b	prec	rec.	f	auc
cinder	0.83	0.49	0.62	0.54	**0.86**	**0.57**	**0.69**	**0.60**
devstack	0.41	0.59	0.48	0.64	0.39	0.56	0.46	0.63
glance	0.64	0.58	0.61	0.57	**0.76**	**0.71**	**0.74**	**0.71**
heat	0.72	0.51	0.60	0.56	**0.76**	**0.58**	**0.66**	**0.61**
keystone	0.72	0.61	0.66	0.63	0.73	0.60	0.66	0.64
neutron	0.64	0.74	0.69	0.71	**0.71**	**0.80**	**0.75**	**0.77**
nova	0.75	0.61	0.67	0.58	**0.78**	0.60	0.68	**0.62**
. . . -manuals	0.81	0.58	0.68	0.55	**0.85**	**0.62**	**0.72**	**0.62**
swift	0.46	0.67	0.55	0.64	**0.49**	0.62	0.55	0.65
tempest	0.60	0.44	0.50	0.53	**0.70**	**0.61**	**0.65**	**0.63**
cdt	0.59	0.68	0.63	0.70	0.58	0.66	0.62	0.69
egit	0.85	0.70	0.77	0.71	**0.89**	**0.75**	**0.82**	**0.77**
jgit	0.75	0.65	0.70	0.69	0.74	**0.71**	**0.73**	0.70
linuxtools	0.80	0.68	0.73	0.69	0.80	0.67	0.73	0.69
scout.rt	0.59	0.65	0.62	0.66	**0.62**	**0.70**	**0.66**	**0.69**

ence. The type of a change has a huge impact on the models of both Eclipse and OpenStack, twice with an effect size of -0.9. This negative value means that when the type of a commit changes from bug fix to no bug fix, the probability of defect-proneness increases, which might indicate that when developers fix bugs in those projects they pay more attention. The third most important metrics (Issue Commenter Experience and NDEV) also have a negative effect size, which means that the more experience the issue reporter has in commenting on issues (OpenStack) or the more developers have changed the modified files before (Eclipse), the lower the risk that the discussed code change will be defect-introducing. This seems intuitive for OpenStack, whereas for Eclipse the finding seems contradictory. The only metric in the top three with a positive effect size, is LA. Hence, the larger a code change is in terms of added lines of code, the higher the probability that it will be defect-introducing.

Models augmented with issue discussion metrics improve upon code-change baseline models. For 5 out of 15 projects, precision and/or recall improve 3% up to 10%.

Table 5.7 Metrics with most extreme effect size (Openstack). Issue metrics are underlined, review metrics in bold.

RQ1 Models		RQ2 Models		RQ3 Models	
Metric (Dimension)	Effect	Metric (Dimension)	Effect	Metric (Dimension)	Effect
FIX (Purpose)	-0.9	LA (Size)	0.21	FIX (Purpose)	-0.73
LA (Size)	0.16	**Reviewer Experience (Focus)**	-0.20	**Review Time (Time)**	0.11
Commenter Experience (Focus)	-0.15	LD (Size)	-0.13	Discussion Lag (Time)	0.08
Comment Length (Length)	-0.10	FIX (Purpose)	-0.12	Sentiment_Avg (Sentiment)	-0.05
Reporter Experience (Focus)	0.08	**Sentiment_Extreme (Sentiment)**	0.08	Sentiment_Extreme (Sentiment)	0.04
Discussion Lag (Time)	0.08	**Discussion Lag (Time)**	-0.05	**Discussion Lag (Time)**	-0.04
SEXP (Experience)	-0.08	NS (Diffusion)	0.05	Commenter Experience (Focus)	-0.02
Number of Comments (Length)	-0.07	**Review Time (Time)**	0.03	AGE (History)	0.01
LD (Size)	-0.05	ENT (Diffusion)	0.03		
ENT (Diffusion)	0.03				
Fix Time (Time)	0.03				
NFC (Diffusion)	0.03				
AGE (History)	0.02				

Table 5.8 Metrics with most extreme effect size (Eclipse). Issue metrics are underlined, review metrics in bold.

RQ1 Models		RQ2 Models		RQ3 Models	
Metric (Dimension)	Effect	Metric (Dimension)	Effect	Metric (Dimension)	Effect
FIX (Purpose)	-0.9	LA (Size)	0.27	LA (Size)	0.37
LA (Size)	0.39	AGE (History)	0.13	**Review Time (Time)**	0.14
NDEV (History)	-0.31	**ReviewTime (Time)**	0.12	AGE (History)	-0.11
AGE (History)	-0.11	ENT (Diffusion)	0.11	ENT (Diffusion)	0.08
Sentiment_Min (Sentiment)	0.08	**Reviewer Experience (Focus)**	-0.10	**Reviewer Experience (Focus)**	-0.07
ENT (Diffusion)	0.03	LT (Size)	0.07	**Discussion Lag (Time)**	-0.04
		LD (size)	0.04		

RQ2. How well can review discussion metrics explain defect-introducing changes?

Approach. We follow a similar approach as for RQ1, but this time use the review metrics of Table 5.3 instead of the issue metrics. The model performance metrics are shown in Table 5.5, while the most important metrics are shown in the second column of Table 5.7 and Table 5.8.

Findings.

9 projects improve their precision and/or recall by 3% to 8%. Except for heat, tempest and jgit, 6 of these 9 projects did not see an improvement for the issue metric models of RQ1, i.e., the improvements of review-based models seem complementary to those of issue-based models. The largest improvements can be found in terms of recall, with jgit, egit and openstack-manuals improving their recall by 6%, 7% and 8% respectively. horizon, despite its increase in precision by 4%, saw a drop in recall by 8%. Three other projects (devstack, keystone and linuxtools) saw a drop in precision or recall of 3%.

The most important variables are LA/LD, Reviewer Experience, AGE, FIX and Review Time. While LA again scores high (with positive effect size), LD also was found to be important (third place for OpenStack), but with a negative effect size. Hence, while

code changes that add many lines of code are more risky, so are code changes that do not remove too much code. Reviewer Experience, as expected, also has a negative impact (i.e., less risk with more experienced reviewers), similar to FIX (cf. RQ1). Eclipse has two other top metrics with positive effect size, i.e., the more time since the last change of the modified files or the longer reviewing takes, the higher the risk of a code change.

> *Review metrics also improve upon code change-based models, with 9 projects improving their precision and/or recall by 3% to 8%.*

Here, also we can see the models built based on the metrics related to both change metrics and issue discussion metrics, has better performance considering recall metric 5.5. While, on average there is around 2% improvement in recall has been observed, there is 6% increase for *heat* project (without any improvement in precision), and *tempest* also just achieve 4% increase for recall without any improvement for precision. Although an improvement with 6% happened for *devstack* recall, its precision decreases 3%. There are also some projects with rise in both precision and recall. *openstack-manuals* has 8% and 6% increase for recall and precision metrics respectively. *neutron* also gained 4% improvement in both of its precision and recall metrics in the enhanced model.

Other projects have almost the same performance for both baseline and enhanced models.

RQ3. How well can issue and review discussion metrics explain defect-introducing changes?

In this research question, we combined both metrics extracted from issue discussions and review discussions as representative metrics of human discussions attributes to see how they can totally impact on the impact of software quality by considering Just-In-Time defect prediction models.

Approach. To understand which of the issue and review models has the strongest link with defect-proneness of commits, RQ3 first builds the models of RQ1, then hierarchically adds the metrics of the review dimensions. The performance of the resulting models is shown in Table 5.6, while the most important metrics are listed in the third column of Table 5.7 and Table 5.8.

Findings. 8 of the projects improve both precision and recall by 3% up to 17%, while 2 more projects improve either precision or recall. Compared to RQ1 and

RQ2, we observe large improvements in precision of 12% (glance), 7% (neutron) and 10% (tempest), while for recall we observe improvements of 8% (cinder), 13% (glance) and 17% (tempest). Other improvements are from 3% to 5%. This results in increases in AUC of up to 6%, 7% or even 10% (tempest). In other words, the combination of issue report and review discussions metrics seems to have a major link with defect-proneness. That said, devstack and swift see a drop in recall of 3% and 5%, respectively.

Generally here we can see significant improvements for all metrics for Openstack projects 5.6. On average precision and recall increase 9% and 11% respectively, AUC and Accuracy with 13% enhancement reach 72%. *tempest* has highest increase for both precision and recall, with 17% and 15% rise, then *glance* with 12% and 13%. *Cinder* has good improvement, 8% increase in its recall (with 4% increase in precision), and for *neutron* also both precision and recall improve more than 5%. *Openstack-manuals* and *keystone* have better performance in a sense that their recall enhance with more than 2%. *Nova* has almost the same precision and recall. For *swift* and *devstack*, we can see a decrease in recall, with less than 5% decline, however *swift* has better performance precision-wise, with 3% growth.

The most important metrics overall are FIX, LA, Review Time, AGE and Issue Discussion Lag. Similar to the issue metric-based models of RQ1, FIX again has a large, negative effect size (but did not appear in the Eclipse models) and LA a large, positive effect. Hence, both top metrics are code-level metrics. The second (both projects) and third (for OpenStack) highest metrics, Review Time and Issue Discussion Lag, mostly have a positive effect size, indicating that the longer reviews have taken or the longer it took people to reply to each other's issue comments, the higher the risk of a code change. Both seem intuitive. In particular, a longer lag between issue discussions could be due to the complexity of the problem, or unavailable project members (too busy). Finally, AGE is the final top metric, this time surprisingly with a negative effect size, i.e., the longer since the last change to a file, the lower the risk of defects. We suspect this difference with the RQ2 model for Eclipse is either due to our calculation of median effect sizes or interaction with other metrics.

We note that the top 3 metrics in Table 5.7 and Table 5.8 contain 3 code change metrics, 1 issue metric and 1 review metric. This indicates that the performance of the final model depends on all metrics together, with different metrics of the three studied domains having an important link with defect-proneness of code changes. Given that the performance of the explanatory models also improved substantially by adding both review and issue discussion metrics, this suggests that one should consider adding their top metrics to any JIT defect model.

Combined models substantially improve precision and recall for half of the projects, with the most important metrics related to the type/size of a change, the time taken to review a change, the time since the last change and issue discussion lag.

5.5 Discussion

From the metrics related to the Focus dimension, especially the experience-based metrics turned out to have a major link with defect-proneness, either in the review (Reviewer Experience) or the issue (Issue Commenter Experience) domain. Their negative effect sizes indicate that the more experienced the person, the less risky the commit might be. These findings seem consistent with the results of Kononenko et al. (Kononenko et al., 2015). They also found reviewer experience to be a good indicator of review quality, with less experienced reviewers more likely to overlook potential problems.

Sentiment-related metrics seem to play a smaller role than experience, but still feature in more than half of the models of OpenStack and/or Eclipse. Minimum and extreme sentiment metrics tend to have a maximum effect size of 0.08. Given that the minimum sentiment is a negative metric, a positive effect size actually means that more negative sentiment has an increasing effect on defect-proneness. Extreme sentiment is harder to interpret, since it could be a very negative value or very positive value. Average sentiment (effect size of -0.05) tends to be positive, which again implies that higher sentiment is linked with lower defect-proneness. This seems intuitive, and somehow relates to the findings of Ortu et al. (Ortu et al., 2015a) that more positive emotions are linked with shorter issue fixing time, however, more research is needed to fully understand these observations.

We have not seen a significant impact between software quality and the number of comments (Length dimension), except for RQ1 (Number of Comments and Comment Length had negative effect size). These results support the observations of Bavota et al. (Bavota and Russo, 2015), as they also were not able to find any link between the number of comments and the chance of defects slipping through review.

5.6 Threats to Validity

Threats to internal validity concern confounding factors that might influence the obtained results. Although we studied both change measures and human discussion measures, there are likely other unknown factors that impact defect-inducing probabilities that we have not measured yet.

Due to the elaborate filtering that we performed in order to link three repositories (version control, bug repository, and code review) , we finally selected 15 huge projects from OpenStack and Eclipse projects, while for example Eclipse has around 549 projects. While just 10 Eclipse projects have more than 1,000 merged reviews in Gerrit, there were not enough links between issue reports and commits to use the projects for RQ1 and RQ3 (yielding 5 analyzed Eclipse projects in Table 6.1).

Construct validity considers the accuracy of observation measurements. First of all, the heuristics used to find the links between the three repositories are not 100% accurate, however we used the state-of-the-practice linking algorithms at our disposal. Recent features in Gerrit show that clean traceability between version control and review repositories is now within reach of each project, hence the available data for future studies will only grow in volume.

Furthermore, there are some inaccuracies and limitations related to the algorithms and tools that we have used. For example, we used the SZZ algorithm to identify defect-inducing changes. The SZZ algorithm has some limitations, since it looks for special keywords in commit messages to link bug fixing commit and to bug-introducing commits. If a fixing commit message does not contain the keywords used by the algorithm, that commit and its bug-introducing commit will be ignored. Similarly, for sentiment analysis of comments, we used SentiStrength, which is a lexical-based tool that has its own limitations and is not 100% precise. For example, it uses indirect affective terms that can increase inaccuracies. Indirect affective terms such as “feel” or “like” associate with sentiment, but do not directly express it. Hence, their sentiment value depends on the context (Thelwall).

Another threat to construct validity is that we assumed that defects had the same weight. This assumption is primarily because of unreliable assigned priorities and severities in issue tracking systems (Mockus et al., 2002; Herraiz et al., 2008). However, in reality, some defects are more severe and take more time to be resolved. Nonetheless, each defect that we considered was at least severe enough to be fixed and integrated into the system.

Threats to external validity correspond to the generalizability of our experimental results. We studied 15 large open source projects, but we have no evidence to claim that these results are representative of all projects out there. Hence, replication studies should confirm whether

our findings generalize to other similar open and closed source projects.

5.7 Related Work

There are many studies on defect prediction that use various metrics captured from version control systems and bug databases. Nagappan et al. (Nagappan and Ball, 2005) presented a set of related code churn measures like total lines of code, file churn, file count as highly effective predictors of defect density. Hassan (Hassan, 2009) showed in a large case study that change complexity metrics are better predictors in comparison to code complexity metrics. We refer to D'Ambros et al. (D'Ambros et al., 2012) for a detailed survey and evaluation of these models.

In contrast to prior work, for which the granularity were files or packages, Kamei et al. (Kamei et al., 2013a) proposed Just-In-Time(JIT) defect prediction models that predict "defect-prone" software changes using the 14 factors related to software changes described in subsection 5.2.1. Their study was conducted over six open source and five commercial projects and the proposed change risk model has an accuracy of 68 percent while it also may reduce the effort in resolving the most risky changes. Later, Fukushima et al. (Fukushima et al., 2014), through a case study on 11 open source projects, evaluated the performance of the JIT work in a cross-project context. They showed that strong within-project performance of a JIT model does not imply it also will perform well in a cross-project context. These studies build on earlier work by Kim et al. (Kim et al., 2008) and Mockus et al. (Mockus and Weiss, 2000).

In our study, we focused on the impact of human review and issue discussions on software quality. There are several other studies that investigated the impact of human factors on software quality. Here we briefly mention the most relevant ones for our work.

Wolf et al. (Wolf et al., 2009) conducted a study of the relation between communication, coordination and software quality integration. They used the IBM Jazz repository to investigate the relationship between communication structures and instances of coordination during code integration. By applying social network analysis metrics, measurable communication characteristics are computed. Results show that developer communication plays an important role in the quality of software integrations.

Bettenburg et al. (Bettenburg and Hassan, 2013) investigated how social interaction measures affect software quality in the form of post-release defects. They discovered that a consistent information flow in discussions decreases the probability of defects significantly and consequently increases software quality. Sliwerski et. al (Śliwerski et al., 2005) presented

the impact of work dependencies on software quality.

Graziotin et al. (Graziotin et al., 2014) conducted a study with 42 student participants and investigated the correlation of affect with creativity and analytical problem solving performance of software developers. Their results showed that happier software developers are more productive in problem solving performance. Murgia et al. (Murgia et al., 2014) found that software artifacts like issue repository comments can convey emotional information of developers. In this research, we use the sentiment of repository comments as one possible metric of emotions in issue or review discussions.

Tourani et al. (Tourani et al., 2014) studied the usage of automatic sentiment analysis on open source mailing lists and showed that development mailing lists carry both positive and negative sentiments. Guzman et al. also applied sentiment analysis in their work (Guzman et al., 2014) to analyze sentiment in topics extracted from software collaboration artefacts. In contrast to these studies, the work presented in this paper does not focus on sentiments of emails or comments, but rather on investigating the link between sentiment and defect-proneness. Lately, Ortu et al. (Ortu et al., 2015a) studied the impact of human emotions, sentiment and politeness on issue fixing time and found that extremely positive or negative issue comments are linked with lower issue fixing time. Hence, understanding the way in which human activities taking place before a code change relate to the defect-proneness (and hence quality) of that change, is important.

Bavota et al. (Bavota and Russo, 2015) empirically showed on three large systems that code reviewing significantly decreases the chance of defect introduction, while it also substantially increases the readability of the reviewed code. Similar to us, they first had to link commits to reviews to determine which commits were reviewed and which ones not. They used this data to compare defect-proneness of reviewed commits to non-reviewed commits. They also analyzed whether defect-proneness changes have lower review participation degree (i.e., number of reviewers and review comments).

Kononenko et al. (Kononenko et al., 2015) performed an empirical study on four large systems to quantitatively investigate factors that might affect code review quality. They analyzed technical properties of contributions, personal characteristics of developers, and some attributes of the team involved in review process. Finally, they found that some technical factors and developer-related factors like review experience and review load can influence code review quality. Note that the studied systems used the Bugzilla issue repository technology for reviewing, which means that some of the review metrics were impossible for us to obtain from the Gerrit review technology used by OpenStack and Eclipse.

Instead of explicitly evaluating the quality of reviews, we focus only on commits that are

linked to an issue report and/or review in order to build full-fledged models with 14 code change, 12 review and 10 issue metrics to understand the link between human discussions and defect-proneness of changes. Furthermore, we studied 15 large systems of 2 major ecosystems.

5.8 Conclusion

In this paper, we empirically studied the impact of human discussion metrics on "Just-In-Time" defect prediction models through an extensive study on 15 large open source projects. We categorized the identified human discussion metrics into four dimensions and measured them on both the issue report and/or review linked to a code change. Explanatory models built using these metrics show a strong connection between human discussion metrics and defect-prone commits, complementing traditional change-based metrics. In particular, while code change metrics related to the size, type (bug fix or not) and time since last change are the most impactful metrics, review time and issue discussion lag have shown to have a substantial positive (i.e., increasing) effect on defect-proneness. Some models also included the experience of issue reports and reviewers as important metrics, which have a negative (i.e., decreasing) effect on defect-proneness. We believe that our study opens up a variety of research opportunities to continue investigating the impact of collaborative characteristics on quality assurance.

CHAPTER 6 ARTICLE 3: ON ISSUE RESOLUTION AND REVIEW TIME – EMPIRICAL STUDY ON 10 OPENSTACK PROJECTS

Abstract

Researchers have started to explore the main factors impacting either the time taken for issue resolution or for code review, including measures of human collaboration and affectiveness, such as politeness, sentiment and emotions. This paper performs a large empirical study on 10 OpenStack projects that contrasts the factors playing a major role in issue resolution time to those playing a major role in review time to understand the differences between both. By studying 52 code-, review- and issue-related metrics across 10 dimensions, we find that different metrics impact issue resolution and review time, except for projects where the review time is relatively long. Apart from traditional churn metrics, the use of CI builds during review and positive interaction between reviewers impact review time, while the issue assignment effectiveness (tossing), amount of status changes and experience of commenters are the most influential on issue resolution time.

6.1 introduction

In today's competitive software industry, time-to-market is considered to be one of the most important attributes for a software organization to succeed (Wohlin and Ahlgren, 1995b). For example, large software companies like Facebook, Google, Mozilla and Netflix have streamlined their release engineering process to accelerate the delivery of releases to the end user, including new features and bug fixes (Adams and McIntosh, 2016). Of course, making only the release time faster is not sufficient, the whole software organization needs to improve its speed. In this context, two major responsibilities of developers, requiring them to be productive, are the resolution of issues, such as reported bugs or new features, and the peer-reviewing of other developers' code changes. Issues are typically reported to an issue tracking system like Bugzilla or Jira, after which a discussion amongst developers and reporter starts to better understand the issue at hand as well as start sketching out a patch. Once a developer has developed an initial patch, it is sent to the reviewing environment (e.g., Gerrit or Rietveld) for peer-review. Here, fellow developers analyze the quality of the patch, suggesting improvements and potentially asking for updated versions of the patch. Once the final version of the patch has been accepted, the issue is marked as resolved. If patch is not accepted, either because the issue is not feasible or nobody is able to work on it, the issue could be closed.

Recently, the issue resolution and review processes have seen a surge in research interest (Jiang et al., 2013; Kononenko et al., 2015; Bavota and Russo, 2015; Tourani and Adams, 2016), especially to understand the factors impacting them and to evaluate how well the processes are being executed. For example, modern code review has thrived thanks to the advent of lightweight reviewing environments (Rigby and Bird, 2013), and is not only used for detecting defects, but also to improve code style, increase team awareness and transfer knowledge within a team (Bacchelli and Bird, 2013). Other research instead focused on the time taken by code review (Jiang et al., 2013) or issue resolution (Weiss et al., 2007; Ortu et al., 2015a). By building models for code review and issue resolution time in terms of a variety of metrics, such studies are essential to identify ways in which developers can be made more productive.

So far, no study has contrasted the issue resolution and review processes within the same software organization. As such, it is unclear what factors impact only review or issue resolution time, and which factors are common. Worse, the actual interaction between issue resolution and review process is unknown. Recently Ortu et al. (Ortu et al., 2015a) have shown how affective factors like emotion and politeness of the participants in issue discussions have a non-negligible association with issue resolution time. This promising association has not been evaluated on review time, although similar to issue resolution the reviewing process is a collaborative activity. While, intuitively, reviewing should start at the end of the issue resolution period, it is unclear whether this is indeed the case in practice as well as what proportion of the issue resolution process is taken up by the review process.

In order to contrast the major factors impacting issues resolution and review time, including affective factors, we perform an empirical case study on 10 OpenStack projects, for whom we were able to link issue report data with patch and review data. Using a total of 52 metrics spread across 10 dimensions, we built models to address the following research questions:

RQ1) *What is the relation between the review and issue resolution periods?*

> We find that the issue resolution and code reviewing periods have different lengths that are not correlated to each other, which suggests that different factors are likely to influence both processes. The issue resolution process is not necessarily closed after successful patch reviewing.

RQ2) *How well can review discussion metrics explain reviewing time?*

> We obtained very good explanatory models based on review-related metrics, with CI

builds, non-bug fixes and positive interaction between reviewers as most influential review-related metrics (in addition to the traditional churn and age metrics).

RQ3) *How well can issue discussion metrics explain issue resolution time?*

Issue-related metrics dominate issue resolution time, except for projects with relatively long review time. Experience is linked with lower issue resolution time, while tossing, status changes, churn, CI builds and politeness/positive sentiment are linked with larger issue resolution time.

In the remainder of this paper, we first discuss the necessary background notions for our work (section 6.2). Next, we describe the case study setup (section 6.3), then present the results of the three research questions (section 6.4). After discussion and threats to validity (section 7.4), we discuss related work (subsection 7.2.3) and we finish with conclusions (section 7.5).

6.2 Background

6.2.1 Sentiment, Emotion, and Politeness

"Sentiment" refers to people's opinion, attitude, appraisal or opinion towards entities, events and their features (Mishra and Jha, 2012). We can measure sentiment within communication between people, which is not just limited to real world conversations but also includes virtual communication such as email, chat or twitter (Mishra and Jha, 2012). Comments left by developers during code reviews or issue resolution also apply, and hence may convey sentiments. We used the state-of-the-art SentiStrength tool [1], which uses a lexical approach (Thelwall) to estimate the sentiment score of informal English text and returns positive and negative scores ("polarization") from 1 to 5 or -5 to -1. The sign ("degree") of the sentiment score indicates a positive versus negative opinion or attitude.

On the other hand, an "emotion" is a psychological state of human feelings such as *sadness*, *happiness* or *anger*. This categorization can be done based on several frameworks, however Parrot's emotional framework is one of the more common ones in recent research (Murgia et al., 2014; Ortu et al., 2015a). Murgia et al.'s exploratory study showed that comments in issue reports express emotions (especially *joy*, *sadness*, *love*, and *anger*) towards different subjects, activities or even team members. Currently, the only tool able to automatically mine comments for emotions are the emotion classifiers of Ortu et al. (Ortu et al., 2015a).

[1] `http://sentistrength.wlv.ac.uk/`

Finally, politeness is "the practical application of good manners or etiquette[2]. It is related to the behaviour expected in a certain context, for example users who use rude language in a bug report might not get the same kind of attention as a friendly bug reporter. Danescu et al. in (Danescu-Niculescu-Mizil et al., 2013) built a classifier to automatically classify a text as (im)politeness and calculate a politeness probability. Scores lower than 0.5 typically are considered as neutral.

6.2.2 Issue Reports

An issue report is a report or description about a bug, new feature or an enhancement, recorded into an Issue Tracking System (ITS). Each issue report has a unique identifier and several attributes, including a subject briefly describing the issue report, an assignee responsible to resolve the issue, and several comments that represent discussions occurring during issue resolution. The status of an issue report indicates the state an issue report is in, allowing to track the issue's progress: *new* (when reported), *triaged* (when the issue has enough comments and analysis to be fixed), *in progress* (it has an assignee and work is going on for it), *fix committed* (the issue has had a patch accepted during the review process), and *fix released* (the accepted patch has made it into a new release). The corresponding issue life cycle is documented in the Openstack documentation[3]. In this paper, we consider the period from the *new* status until *fix committed* as the issue resolution period. As discussed elsewhere (da Costa et al., 2016; Jiang et al., 2013), the time between *fix committed* and *fix released* depends on management decisions outside the control of individual developers.

Typically, ITSes not only collect and manage issue reports, but they especially provide a shared environment for people to discuss the issues being resolved. Thus, as people discuss, help other team members, ask questions or give their opinions, their communications and interactions are tracked and hence can be studied . In our study, we mined the Launchpad issue repository of Openstack[4] to extract different metrics related to the issue resolution process.

6.2.3 Code Review

Code review is a crucial part of the quality assurance streategy of large and mature software projects. It aims at evaluating code changes, discovering and fixing defects and weaknesses before a submitted patch is integrated into the codebase. Modern code review technology

[2] `http://en.wikipedia.org/wiki/Politeness`
[3] `https://wiki.openstack.org/wiki/Bugs#Status`
[4] `https://launchpad.net/openstack`

has been widely adopted in both commercial and open source software projects, due to its lightweight nature and strong tool support (Rigby and Bird, 2013). Gerrit[5] is such a modern code review tool, providing a traceable code review process for git-based software projects (Bettenburg et al., 2015).

In a typical code review process recorded in Gerrit, there are reviewers and verifiers. Reviewers are responsible for checking proposed changes by leaving their comments and feedback for the author of the proposed patch or giving a score to indicate their agreement or disagreement. Verifiers further evaluate code changes by executing tests to ensure the correctness and consistency of the proposed patch. Apart from (human) team members, Continuous Integration (CI) tools are commonly used as verifiers, automatically compiling and testing newly submitted versions of a patch. The reports generated by these CI tools can then be appended to the code review.

6.3 Case Study Setup

This section discusses the data set used in our case study analysis to address our RQs, extracted metrics and finally our methodology.

6.3.1 Dataset

We used the OpenStack data set of 10 projects provided by Tourani et al. (Tourani and Adams, 2016), with the statistics shown in Table 6.1. Openstack[6] is a popular open source ecosystem with a large number of projects that use a modern code review repository (Gerrit), issue repository (Launchpad), and version control system (Git). Developers typically follow guidelines to make explicit links between the code reviews, issues and commits, enabling researchers to link a substantial number of commits to reviews and issues.

The data set originally was mined by Gonzalez-Barahona et al. (Gonzalez-Barahona et al., 2015) in the form of a relational database. Tourani et al. (Tourani and Adams, 2016) then used a variety of heuristics to recover the links between commits, reviews and issues. Because of rebasing (Bird et al., 2009c), only for 60% of the reviews the corresponding Git commit could be found. For more than 50% of the missing cases, the correct Git commit identifiers were mentioned in review comments or in some other field, eventually leading to at least 70% of the reviews being linked to commits.

To link commits to issue reports, issue identifiers mentioned in commit messages or commit

[5] `https://code.google.com/p/gerrit/`
[6] http://openstack.org

identifiers mentioned in issue reports were used. Finally, the links of 72% of issue reports were identified. By combining the links between commits and reviews, and commits and issue reports, a data set of commits linked to both review and issue reports was obtained. The properties of this data set are shown in Table 6.1.

6.3.2 Independent Metrics

As independent variables for our study, we considered the set of code change metrics proposed by Kamei et al. (Kamei et al., 2013a), and the sets of metrics related to human discussions introduced by Ortu et al. (Ortu et al., 2015a) and Tourani et al. (Tourani and Adams, 2016). We also introduced additional metrics.

Change-level metrics were extracted from the Git version control repository. For each commit, its issue and review discussion metrics subsequently were mined from the Launchpad issue repository and Gerrit review repository. For commits linked to several issue reports, one issue report randomly was selected . Table 6.9 shows the resulting list of metrics, categorized per domain (code change, review, issue or both) and dimension. In the following, we briefly describe these metrics.

Code Change (First 5 dimensions) The complexity of a patch proposed to resolve an issue dictates not only the time to develop the patch for the code change, but likely also its review time. For example, the more diffused a code change is, the more effort and hence time is required from developers to analyze the patch . A code change has several attributes relevant to the diffusion, size, purpose, history and experience involved in a code change, as proposed by Kamei et al. (Kamei et al., 2013a).

Focus More experience at certain tasks make people confident and likely faster at reviewing or resolving issues. However, such people might also be more susceptible to resolving more difficult issues. For this reason, we included focus-related metrics like experience into our study. Apart from experience-related metrics, other metrics in this dimension are indicators of the complexity and accuracy of the reviewing/resolution work.

Integration The easier and smoother a proposed patch integrates into the existing code base (i.e., the lower NegativeCI and workflow, and the larger PositiveCI), the shorter review time might be. Note that "pausing" a review means that the reviewers can continue providing review comments, but the review cannot be flagged as accepted (i.e., be integrated into the code base), effectively blocking a patch from progressing.

Quantity The more people contributing during the review process, the more difficulties might have arised. For each review, we counted the number of times a review score was

Table 6.1 Statistics of the studied OpenStack projects.

Project	Start Date	# commits linked to issues and reviews (%total)
cinder	2012-05-03	1654 (33%)
devstack	2011-09-11	827 (15%)
glance	2010-08-11	1155 (28%)
heat	2012-03-13	1982 (27%)
horizon	2010-07-12	1043 (24%)
keystone	2011-04-14	1996 (30%)
neutron	2010-12-31	2979 (35%)
nova	2010-05-30	6600 (20%)
openstack-manuals	2011-09-20	2290 (27%)
tempest	2011-08-26	1453 (25%)

given.

Length The length of comments and the number of comments are two indicators of the amount of discussion. More discussions may reveal a controversial problem that might take longer to be solved. Other length-related metrics like *#sentences* or *#authors* were highly correlated with *#comments*, and hence not included Comments generated automatically by the version control servers or other tools were excluded.

Affect Every piece of text consists of two aspects of information: facts (objective expression) and opinion (subjective expression) (Thelwall). Facts are about various entities, their attributes and events, while opinion expresses feelings, emotions or sentiments towards a subject. In the context of our study, we consider the text inside issue or review comments, which information, have been shown to carry emotions and sentiments (Murgia et al., 2014; Ortu et al., 2015a; Tourani and Adams, 2016).

To extract such affective metrics, first we filtered out automatically generated comments , as we are interested only in affective metrics of human conversations. Furthermore, each comment can have non-human language parts like code snippets or stack traces. By applying a lightweight method using regular expressions, we were able to filter out such parts of comments.

To calculate the sentiment of comments, we applied SentiStrength, as explained in section 6.2. We calculated the average sentiment value for review and issue comments, as well as the sentiment of the first and last comment only, as Ortu et al. showed that the sentiment and politeness of the last comment of an issue report may have a strong impact on the issue fixing time (Ortu et al., 2015a).

To measure the politeness of comments we adopted the tool of Danescu et al. (Danescu-Niculescu-Mizil et al., 2013), and for emotions the emotion classifier of Ortu et al. (Ortu et al., 2015a).

6.3.3 Preliminary Analysis of Review Discussion Metrics

Handling Collinearity After collecting the required metrics, we observed correlations between metrics, due to redundancy, for example between comment sentiment and the last comment's sentiment. As a matter of fact, we expected some of these correlations while designing and selecting the different metrics, but still put these metrics in our initial list to analyze these correlations in more detail. To remove collinearity, we calculated the variance inflation factors (VIF) for each metric, then metrics with VIF greater then 5 will be removed (Fox and Weisberg, 2010).

Handling Skew Positive skew, i.e., a long tail of values towards higher values, is a threat for the performance of models. Here, we dealt with the effect of skew by using standard log transformation.

6.3.4 Building Explanatory Models

According to Tantithamthavorn et al. (Tantithamthavorn et al., 2016), classification techniques like logistic regression models produce defect prediction models that outperform models trained using clustering, rule-based, SVM, and neural network techniques. Hence, we applied logistic regression models to investigate influential factors on the issue resolution or review time. Similar to Ortu et al. (Ortu et al., 2015a), we mapped issue resolution and review time, which are numeric values, to a binary value using their median value as a criterion. A review or issue resolution time larger than the median was considered as "long" (value 1), and a shorter time as "short" (value 0). Using this binary dependent variable, the output of each logistic regression model then reveals the probability of the issue resolution or review time spent being long (probability of at least 0.5).

To remove metrics without significant impact on the model, we applied Mallow's Cp criterion (Mallows, 1973) in a stepwise variable selection. The final model just contains variables with significant impact, such that the deletion of the filtered out variables would not improve the model.

6.3.5 Evaluating the Models

To validate the models, we use the following common performance measures: *precision*, *recall*, *F1-measure* and *AUC*. Precision measures the percentage of cases where the model correctly classifies something as "long", whereas recall measures the percentage of all "long" cases successfully found by the model. The F1-measure is the arithmetic mean of precision and recall. Finally, the AUC value provides an overall performance of a model across all possible probability thresholds (not just 0.5). The higher the AUC compared to 0.5, the better the model performs compared to a random model.

These validation metrics are calculated using 10-fold cross validation technique (Efron and Tibshirani, 1995) . Using this technique, the data set is divided into 10 folds (each having a similar proportion of "long" review of issue resolution time). Then, one at a time, each fold is selected as test data, with the rest of the data used as training data for building a model. After ten iterations, this process finally yields a so-called confusion matrix from which precision, recall, F1-measure and AUC can be calculated.

6.3.6 Identifying Significant Metrics

To determine the metrics with the strongest impact on issue resolution or review time, we used the technique by Shihab et al. (Shihab et al., 2013). This approach calculates a baseline output of the model with all input variables set to their median value. Then, one metric at a time, 1) we change the value of that metric by adding one standard deviation to its median value, while the values of the other metrics remain unchanged on their median values; then 2) for the given metric, we compute $\frac{outputofchangedvalue-basevalue}{basevalue}$, the so-called called *effect size* of that metric, which is the relative increase or decrease for the corresponding output.

The effect size value is independent of the type or unit of the corresponding metric, which facilitates comparisons among different metrics. This value also has a direction indicating either an increase or a decrease in output variable and enables us to identify metrics with decreasing (negative, i.e., shorter time) or increasing (positive, i.e., longer time) impact. For each metric, we analyze the distribution of its effect size across the 10 studied projects.

6.4 Case Study Results

RQ1. What is the relation between the review and issue resolution periods?

Motivation Intuitively, one would expect the work on a particular issue (such as a bug fix or new feature) to start from the time the issue is reported, going through a period of issue

discussions to understand the requirements and start sketching an initial design, until a first patch version is proposed by a developer. This patch then would enter the reviewing process, where reviewers might ask subsequent patch versions to address errors, omissions or wrong decisions. If the patch is accepted, it would be integrated into the version control system of the project and the issue report marked as resolved. If the patch is rejected, the issue discussions could continue until a new patch is proposed and successfully passes review, or the issue is closed without resolution.

Before analyzing and comparing the factors that impact issue resolution and reviewing time, this research question empirically studies whether the above interaction between issue and review process matches reality.

Approach In order to analyze the relationship between the issue resolution and reviewing process, we analyze the overlap between the issue resolution and reviewing period. To do this, we first extract, for each issue report and its corresponding review, the issue's and review's start and end time. For issues, we consider as start time the moment a particular issue is reported, and as end time the moment the issue's status changed to *fix committed*. For reviews, the start time is the moment on which a patch has been submitted and the end time is the moment on which the final version of the patch has been merged. Note that our data set only comprises issues with accepted patches, and ignores rejected patches.

Before comparing issue resolution and review periods, we had to check the timezone used for the recorded issue (Launchpad) and review (Gerrit) dates. In contrast to the Gerrit review dates, which are stored in the Eastern time zone[7], Launchpad issue dates are stored in the GMT time zone [8]. Therefore, by converting the review dates to GMT using R's built-in functions (which take into account leap years and other exceptional circumstances), we could compare the start and end time of issues and reviews to each other.

Using this data, we studied the three theoretically possible overlaps between the issue resolution and review periods graphically shown in Figure 6.1.

[7]https:en.wikipedia.orgwikiEastern_Time_Zone
[8]https:en.wikipedia.orgwikiGreenwich_Mean_Time

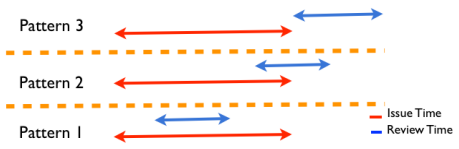


Figure 6.1 Patterns of overlap between issue resolution and review periods.

Findings. A median of 95% of issues in the studied systems have the reviewing period embedded inside the issue resolution period (pattern1). As shown in Table 6.2, the theoretically expected interaction between issue resolution and reviewing time by far is the most popular, followed by pattern2 (issue resolution process is stopped before the reviewing process finishes). This means that the rest of our study will focus only on the issues/reviews that follow pattern1.

Despite the reviewing period being embedded into the issue resolution period (pattern1), there is only a weak correlation between issue resolution and reviewing time. Indeed, the Spearman correlation between issue resolution and review time is shown in Table 6.4. As is commonly done, we considered weak a correlation less than 0.4, moderate a correlation from 0.4 to 0.7, and strong a correlation greater than 0.7. Only for horizon and tempest, the correlation reaches 0.4, but even these correlations could not be considered as moderate. This low correlation suggests that the lengths of both periods are mostly independent, and hence are impacted by different factors.

One possible explanation for this is shown on Figure 6.4 and 6.5a, which show respectively the distribution of the lengths of issue resolution/reviewing periods and the distribution of the ratio of review time to the issue resolution time. Only for cinder, devstack, horizon and nova, the issue resolution and reviewing periods have similar lengths. For the other projects, the length of the reviewing period drops down to a median of 8% (openstack-manuals), indicating that the issue resolution time is an order of magnitude larger in several of the projects.

In a non-negligible number of cases, the issue resolution process is not immediately closed after patch acceptance. Figure 6.5b shows on the one hand that the median time between the end of the review and issue resolution period is 8.74 hours, since various projects automatically close an issue upon patch acceptance, but that the 75th percentile can go up to 40% of the total issue resolution time. We manually analyzed such cases sampling 99 issues. This number of issues in purpose of obtaining a confidence level of 90% and confidence interval of 10%. This means that a proportion of X%in our sample of issues with certain characteristics actually corresponds to $X \pm 10\%$ in the population of issues. In Table 6.3, you can see various reasons observed in our sample, mostly the importance of a bug is set to low or medium. Another major reason is reopening of bugs, which also observed and investigated in detail by An et al. (An et al., 2014). Holiday (new year holiday) is another reason, also while many discussions happened around the bug report or the bug report is not complete and it takes some time to be completed, for instance to be reproduced. For some bug reports their states changed to “fix committed” not immediately after the review and it takes time. Finally, frequent changes of assignees or setting them late, different milestones, integration

Table 6.2 Popularity of the three patterns.

Project	Pattern3	Pattern2	Pattern1
nova	92%	6%	2%
glance	96%	2%	2%
openstack-manuals	96%	4%	0%
keystone	92%	7%	1%
cinder	95%	4%	1%
tempest	94%	5%	1%
neutron	95%	4%	1%
devstack	96%	3%	1%
horizon	95%	2%	3%
heat	92%	7%	1%

and merge new changes to main branch are also other factors that might contribute to longer issue resolution time length.

Table 6.3 Reasons for delayed issue times.

Cause of delay	Count
Low importance	16
Medium importance	14
Bug Reopening	14
No reason (for one month or more wihout any changes)	13
Related bugs (even in other systems)	10
Holiday	7
Long discussions (around the bug report)	6
Incomplete bug report	5
Late change state to fix committed	4
Milestone changes (5 times)	2
Making bug report as wish list	2
Assignee changes (3 times or more before review)	2
Setting assignee late	2
Merged to master branch late (from stable branch)	2

The issue resolution and code reviewing periods have different lengths that are not correlated to each other, which suggests that different factors are likely to influence both processes.

Table 6.4 Correlations between issue resolution and review time, for the studied projects.

cinder	devstack	glance	heat	horizon	keystone	neutron	nova	...-manuals	tempest
0.58	0.23	0.24	0.59	0.36	0.57	0.38	0.47	0.12	0.33

RQ2. How well can review discussion metrics explain reviewing time?

Motivation.

Reviewing has become one of the cornerstones of modern software quality assurance (Bacchelli and Bird, 2013). As the main quality gate between software developers and a project's code repositories, reviews aim to intercept errors, coding style violations or bad practices in general in submitted patches before they pollute the main code base and might impact the whole development organization. Given their central role, reviews should not only be thorough, but also be as fast as possible, in order to not slow down regular development and not generate more severe merge conflicts (as the patch under review gradually would become less compatible with the main code base (Bird et al., 2009c)). In this research question, we study which factors impact review time.

Approach.

For each project separately, we build an explanatory logistic regression model of review time based on the metrics discussed in subsection 6.3.4 to analyze how the factors measured by these metrics can explain review time. The model explains the probability that a given review would belong to the 50% slowest reviews. Since in some projects review time can represent a median of up to 50% of issue time, we also considered issue-related metrics in our models, even though we expect issue-related metrics to play a minor role compared to the review-specific metrics.

We first built baseline models involving only the change-related metrics. Then, we added either review-related metrics or issue-related metrics, and finally we built models with all three groups of metrics. For each kind of model, we evaluated the performance using 10 fold cross validation, yielding the results in Table 6.5.

Since some metrics might have a larger influence on the dependent variable, i.e., the probability that review time belongs to the 50% slowest reviews, we then calculated the impact of each metric in the generated logistic regression model as explained in subsection 6.3.5. We then studied the resulting effect sizes. Metrics not appearing in the models were given an

effect size of zero.

Findings. The models using review-related metrics obtain an average precision and recall of 0.78 and 0.79. For these models, glance has the largest improvements in performance compared to the baseline model based solely on change-related metrics, i.e., an increase of 23% in precision, 21% in recall and 21% in AUC. Next up is horizon, with a 22% increase in precision, 21% in recall and 21% in AUC. Among all projects, openstack-manuals and tempest have the lowest increase in recall with 7% and 10%, and lowest improvement in precision with 7% and 8%. Generally, on average, we can see 16% increase in precision, 14% in recall, and 14% across all projects when review metrics are added to the change metrics.

Combining issue metrics to change metrics does not lead to a significant improvement in the models' performance, which confirms our findings from RQ1 about the large difference in length between review and issue process. The improvements obtained when combining all metrics together all are due to the review-related metrics. The issue resolution process does not play a large role in the reviewing process, since considering just issue related metrics, the model performance does not improve significantly Table 6.5.

Reviews involving continuous integration builds, changes with high number of votes tend to have a longer review time.

Figure 6.2 shows number of votes has most impact on the review time. Reviews investigated by more reviewers may take a longer time, which seems intuitive. The total number of continuous integration checks has a large, positive median effect size and hence are associated with longer review times. One hypothesis is that many reviews with passing builds also have earlier failing builds, although we did not find strong correlations between both metrics. Alternatively, it is clear that due to the traffic of submitted patches, the CI environment cannot be kicked off immediately for each new patch version. The resulting delay could be one of the reasons for the strong link with long review times . This is confirmed by OpenStack's efforts to improve the scheduling algorithm of their CI system (Carrez, 2014). Length of comments during reviews is the third metric with the most positive impact size indicating reviewing with more explanations will take longer time, probably they are more complicated.

Apart from these review-related metrics, the amount of churn (ADD) also has a large, increasing link with review time. This seems intuitive, as larger changes to lesser known code indeed require more effort by reviewers.

Reviews of non-bug fixes, involving positive interaction between reviewers or more modified subsystems tend to have a shorter review time.

Table 6.5 Performance of review time models with only "change metrics" ^{c}, "change metrics" and "review metrics" $^{c+r}$, "change metrics" and "issue metrics" $^{c+i}$, and all metric categories.

project	p^{c}	r^{c}	f^{c}	auc^{c}	p^{c+r}	r^{c+r}	f^{c+r}	auc^{c+r}	p^{c+i}	r^{c+i}	f^{c+i}	auc^{c+i}	p	r	f	auc
cinder	0.66	0.69	0.67	0.66	0.82	0.78	0.80	0.81	0.69	0.66	0.67	0.68	0.82	0.80	0.81	0.81
devstack	0.57	0.58	0.58	0.62	0.67	0.73	0.70	0.73	0.54	0.58	0.56	0.60	0.67	0.71	0.69	0.72
glance	0.62	0.65	0.63	0.66	0.85	0.86	0.85	0.87	0.56	0.65	0.60	0.61	0.87	0.86	0.87	0.88
heat	0.58	0.64	0.61	0.62	0.78	0.79	0.79	0.80	0.61	0.64	0.62	0.64	0.78	0.80	0.79	0.80
horizon	0.63	0.60	0.61	0.60	0.82	0.76	0.79	0.78	0.65	0.63	0.64	0.63	0.82	0.75	0.79	0.79
keystone	0.54	0.74	0.63	0.66	0.70	0.88	0.78	0.81	0.57	0.84	0.68	0.71	0.67	0.89	0.77	0.80
neutron	0.63	0.63	0.63	0.62	0.85	0.85	0.85	0.85	0.63	0.63	0.63	0.62	0.84	0.84	0.84	0.84
nova	0.60	0.68	0.64	0.65	0.77	0.79	0.78	0.80	0.63	0.69	0.66	0.67	0.77	0.79	0.78	0.80
...-manuals	0.68	0.65	0.66	0.66	0.76	0.71	0.73	0.73	0.66	0.64	0.65	0.64	0.76	0.74	0.75	0.74
tempest	0.65	0.66	0.66	0.66	0.76	0.75	0.75	0.76	0.68	0.68	0.68	0.69	0.76	0.78	0.77	0.78
Average	0.62	0.65	0.63	0.64	0.78	0.79	0.78	0.79	0.62	0.66	0.64	0.65	0.78	0.80	0.79	0.79

Table 6.6 Performance of issue time models with only "change metrics" ^{c}, "change metrics" and "review metrics" $^{c+r}$, "change metrics" and "issue metrics" $^{c+i}$, and all metric categories.

project	p^{c}	r^{c}	f^{c}	auc^{c}	p^{c+r}	r^{c+r}	f^{c+r}	auc^{c+r}	p^{c+i}	r^{c+i}	f^{c+i}	auc^{c+i}	p	r	f	auc
cinder	0.73	0.65	0.69	0.67	0.76	0.64	0.69	0.69	0.82	0.66	0.73	0.73	0.83	0.70	0.76	0.75
glance	0.59	0.58	0.58	0.56	0.60	0.61	0.60	0.57	0.75	0.71	0.73	0.72	0.76	0.72	0.74	0.73
heat	0.67	0.60	0.63	0.60	0.71	0.59	0.64	0.63	0.76	0.57	0.65	0.67	0.77	0.62	0.68	0.69
horizon	0.65	0.58	0.61	0.60	0.74	0.63	0.68	0.68	0.72	0.58	0.65	0.65	0.77	0.66	0.71	0.71
keystone	0.69	0.49	0.57	0.54	0.71	0.50	0.59	0.56	0.82	0.64	0.72	0.69	0.80	0.63	0.71	0.67
neutron	0.53	0.54	0.53	0.51	0.52	0.49	0.51	0.50	0.72	0.65	0.68	0.69	0.72	0.64	0.68	0.69
nova	0.69	0.61	0.65	0.63	0.71	0.64	0.67	0.66	0.75	0.64	0.69	0.68	0.77	0.69	0.73	0.72
...-manuals	0.76	0.50	0.60	0.60	0.75	0.49	0.59	0.59	0.81	0.61	0.70	0.67	0.81	0.59	0.68	0.66
tempest	0.69	0.65	0.67	0.66	0.71	0.66	0.68	0.68	0.79	0.68	0.73	0.74	0.82	0.75	0.78	0.78
devstack	0.57	0.58	0.58	0.62	0.67	0.73	0.70	0.73	0.54	0.58	0.56	0.60	0.67	0.71	0.69	0.72
Average	0.66	0.58	0.61	0.60	0.69	0.60	0.64	0.63	0.75	0.63	0.68	0.68	0.77	0.67	0.72	0.71

The metric with the most negative median effect size is Change_Type, indicating that changes other than bug fixes tend to take the least amount of time to review. This makes sense, as bug fixes, although users wish them to be delivered faster, need to be checked in more detail to ensure that the bug is really fixed and no further damage is made to the organization's reputation.

Interestingly, the average of love emotion (which means gratitude) in review comments has a decreasing effect, i.e., reviews with comments having a kinder tone seem to take less time to review. Of course, we cannot make any causality claims, but intuitively this observation seems to make sense. More kindness expressed among reviewers may imply better communications, leading to more constructive and effective collaboration. This result is consistent with the impact of other affective metrics in our models, with firstComment_sentiment also having a decreasing effect. Conversely, firstComment_politeness was seen with an increasing impact on reviewing time, again suggesting that positivity and openness seems to be more interesting during reviews.

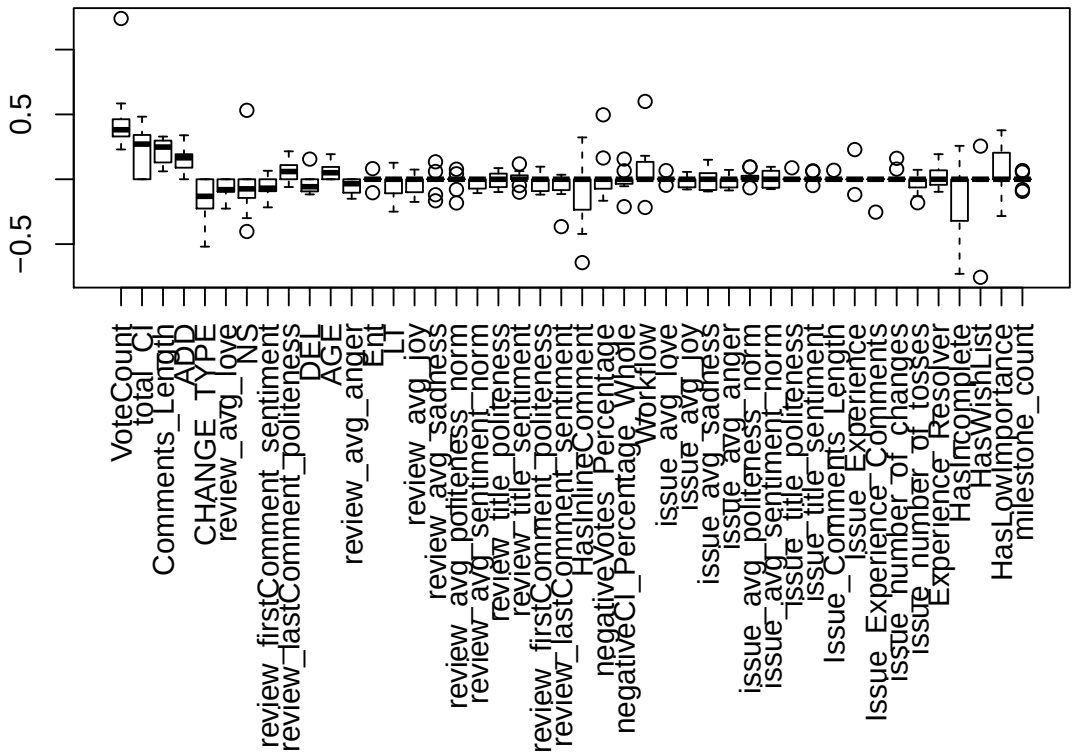


Figure 6.2 Distribution of effect size for metrics in the reviewing time models, ordered from highest to lowest median effect size.

Finally, the more modified subsystems (NS), the shorter the change's reviewing time. This was a surprise to us, since modifying more subsystems probably require more time to investigate different aspects and impacts of a commit on various subsystems.

We obtained very good explanatory models based on review-related metrics, number of votes, CI builds, non-bug fixes and positive interaction between reviewers as most influential review-related metrics (in addition to the traditional churn metric).

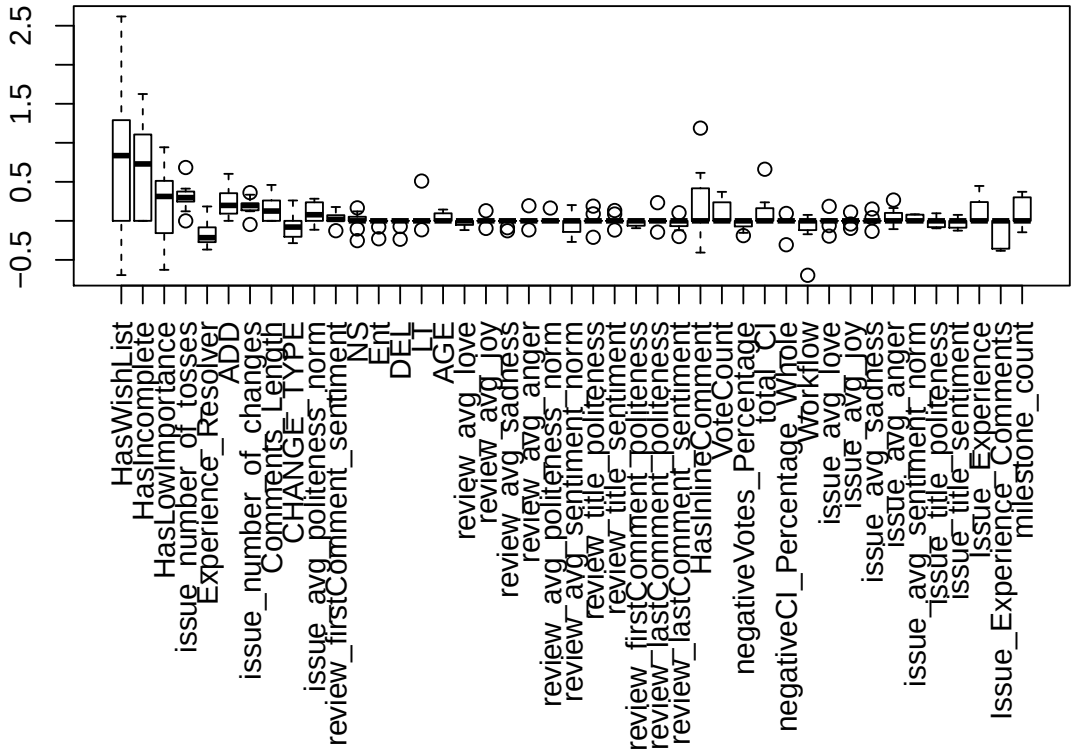


Figure 6.3 Distribution of effect size for metrics in the issue resolution time models, ordered from highest to lowest median effect size.

RQ3. How well can issue discussion metrics explain issue resolution time?

Motivation.

Issue resolution time can be interpreted as one of the major measures of development effort (Weiss et al., 2007), providing an indicator of the productivity of developers in completing their tasks, including fixing defects and developing new features. Faster issue resolution not only is a good sign in terms of developer productivity, but also is well regarded by the end user. Hence, this question aims to understand what factors tend to be related to faster or slower issue resolution time.

Approach. We take a similar approach as for RQ2, but with as dependent metric the probability that issue time is longer than the median issue time. We again consider change-, review- and issue-related metrics, but this time we expect the issue-related metrics to

dominate (with some review-related metrics in the projects with longer review time). The models' resulting performance metrics are shown in Table 6.6, while the distribution of effect sizes is shown in Figure 6.3.

Findings. The models combining issue and review metrics obtain average precision and recall of 77% and 67%, with review metrics only contributing in the projects with long review time. Compared to the baseline models containing only change-related metrics, the combined models' precision of 7 projects improved more than 10%, and for 5 projects recall improved with more than 10%. While all projects saw both precision and recall increase, the precision and recall of neutron increased the most, with 19% and 10% respectively. Generally, we can see significant improvement in both precision and recall, with increases of 16% (glance), 13% (tempest), 12% (horizon) and 10% (heat and devstack) compared to recall increases of 14% (glance and keystone), 12% devstack, 10%(neutron and tempest). Finally, the AUC of 5 projects increased by more than 10%.

As suggested by the findings in terms of the ratio between review and issue resolution time (Figure 6.5a), the projects with proportionally longer review time have models in which the review-related metrics play a larger role. In particular, we can see that the recall of the review-related models for devstack, heat, horizon and nova is at least as high as the recall for the issue-related models, while the same holds for the AUC values of heat, horizon and nova. In other words, one first needs to check the interaction between issue resolution and reviewing period before deciding which metrics to consider in the issue resolution time models.

Incomplete issues and issues with low importance, also with insubstantial tossing, status changes, churn added take longer until resolution.

If an issue has low importance or changed to a wish list, it is likely to be resolved later, also incomplete issue reports are more probable to fix in a longer time. These observations intuitively seem correct as developers naturally will address critical and high importance issues first. When a bug report is not complete, it takes extra time and effort to be understood. Another influential metric is the number of times an issue is tossed around to different developers, i.e., the initial person assigned to resolve the issue was not able to do so and tossed the issue to someone else, until the right person was found. Similarly, the more the status of an issue is changed, the larger the indication that an issue is being tossed around, more information is required from the issue reporter to for example reproduce a bug, or some other event happened involving a delay in resolution. Surprisingly, the length of review comments also shows increasing impact on issue resolution time, by indicating that the corresponding issues is more complicated and requires more explanations or discussions. Also, similar to the review time models of RQ2, the amount of churn added is related to longer issue resolution

time. This effect especially was observed in the projects with longer review time.

More experienced issue fixer is linked with faster issue resolution time.

This observation intuitively makes sense as the more experienced someone is in resolving issues, the more familiar he or she is with the software system and issue resolution process, likely leading to faster resolution.

Issue-related metrics dominate issue resolution time, except for projects with relatively long review time. Experience is linked with lower issue resolution time, while low importance, incomplete bug report, tossing, status changes, and churn are linked with larger issue resolution time.

6.5 Related Work

Regarding review practices, Bavota et al. (Bavota and Russo, 2015) found that reviewed code has a significantly higher readability and less chance of defect-introducing commits with respect to non-reviewed code, while review participation degree (i.e., the number of reviewers and review comments) may affect this chance. Kononenko et al. (Kononenko et al., 2015) showed that code review quality is affected by both technical and personal attributes, like review experience In their later study (Kononenko et al., 2016), they surveyed 88 Mozilla core developers to understand their perception of code review quality. As factors affecting review time, the survey identified technical ones like change complexity and patch size, and human or social ones like experience of reviewer and patch writer, personality of reviewer and his personal priorities. Our study also adopted human-related factors like sentiment, politeness and emotions of discussion participants, apart from more traditional code change-related factors. Jiang et al. (Jiang et al., 2013) built a model of review time in the Linux kernel, mainly using patch-, email- and developer-related metrics. The above work is the latest in a long line of review research (Bacchelli and Bird, 2013; McIntosh et al., 2014; Rigby and Bird, 2013). With respect to issue fixing time, Weiss et al. (Weiss et al., 2007) presented an approach that automatically predicts the person-hours spent on resolving an issue. Given a new issue report, they identify a sufficient number of issue reports that are textually most similar in terms of their title and description, then take their average resolution time as prediction for the new issue. Their predictions were promising and performed well. Giger et al. (Giger et al., 2010) also studied issue reports of three open source projects to propose a

model that can predict a new issue to be resolved fast or slow. This model used attributes of a newly reported issue, such as the reporter, assignee, milestone and severity . Our study includes the same metrics, but complements them with many other dimensions and two other domains of metrics, i.e., code change- and review-related metrics.

Wolf et al. (Wolf et al., 2009) found that there is a relation between communication structures and software quality. They defined and calculated social network metrics and communication characteristics that have an important impact on the quality of software integration. Bettenburg et al. (Bettenburg and Hassan, 2013) measured the impact of social interaction measures on software quality . Tourani et al. (Tourani and Adams, 2016) studied the impact of human discussion metrics on JIT (Just-In-Time) defect prediction models using 15 large open source projects. Strong connections between human discussion metrics and defect-inducing commits were observed. In this study, we extended those metrics and studied their impact on the length of time spent for reviewing or issue fixing by developers and reviewers.

Graziotin et al. (Graziotin et al., 2014) showed that studying the human factors of software engineering is necessary. They found a correlation between affect and analytical problem solving performance of software developers in their experiments. Based on their results, happier software developers are more productive specially in problem solving performance. Wagner et al. (Wagner and Ruhe, 2008) conducted a systematic literature review in order to extract a list of major effective factors for software development productivity. They assigned an importance score to each factor based on the number of papers that refer to that factor. They found that special considerations go to human-related (soft) factors like Communication, Team Cohesion (cooperativeness of software stakeholders), Respect, and Fairness. Technical factors such as Software Size, Product Complexity, and Development Flexibility also play a role in the productivity of developers.

De Choudhury et al. (De Choudhury and Counts, 2013) investigated emotions expressed by employees in internal microblogging tools (considering 500 corporations). Guzman et al. (Guzman et al., 2014) studied sentiments expressed in commit comments, then analyzed the extracted sentiments based on different factors like programming language, team distribution, and time of the commit. Their results show promise for understanding the factors that affect developers' sentiments, or the role that developers' sentiments may play in the developers' outcome. Tourani et al. (Tourani et al., 2014) showed that both positive and negative sentiments are expressed in open source software development mailing lists, and they identified different categories for both positive and negative sentiments presented in software mailing lists. Murgia et al. (Murgia et al., 2014) found that software artifacts like issue repository comments can also convey emotional information of developers. Later, Ortu

et al. (Ortu et al., 2015a) showed that extremely positive or negative sentiments presented in issue comments are linked with lower issue fixing time. Here, we also focused on sentiments, emotions and politeness of issue and review comments (besides other metrics) to identify their impact on review or issue fixing time.

6.6 Discussion

6.6.1 Influential metrics

As suggested by our findings in RQ1 that issue resolution and review period are mostly independent, we found in RQ2 and RQ3 that review time and issue resolution time models are impacted by different metrics, except for those projects with longer review times. Churn and review comment length are the only major metrics in common between both kinds of models. Hence, one of the main take-home messages of this work is that prior to selecting factors for use in models to explain review and/or issue resolution time, one should first analyze the characteristics of both processes.

Affective-related metrics seem to play a smaller role than experience or churn, but still appeared in the top 10 major influential attributes on both models explaining reviewing time and issue resolution time factors. Review_comment_politeness, and issue_comment_politeness have increasing impact on review time and issue resolution (that are partly backed up by related work (Ortu et al., 2015a)), while the love emotion has contradictory impact, i.e. the more love emotions expressed in review comments, the less time will be spent on reviewing.

6.6.2 Issue resolution models

To compute issue fixing time as an output parameter in section 6.4, we calculated the time span between the time an issue status changed to *fix committed* and the time it was reported. In other words, we considered the last *fix committed* as status that a particular issue was addressed. However, as described in subsection 6.2.2, the lifetime of an issue is composed of *fix released* which is the final status of an issue when it is integrated into the release. Based on the metrics obtained from the issue repository, we assumed those metrics mainly can explain the time span before issue fixing, since between *fix committed* and *fix released*, there might be other external factors related to the integration phase and testing that were not recorded in the issue repository. Here, we change our assumption and consider *fix released* as the end point for computing issue time and rebuild the models to examine the power of our models in explaining issue releasing time.

First, like section 6.4, we computed different patterns shown in Figure 6.1, and built the models and evaluated their performance.Table 6.7 illustrates how pattern 1 is still the dominant one and here we continue our study with pattern1 and excluded patern2 and pattern3 from our data.

The results are depicted in Table 6.8 computed using the same approach as discussed in section 6.4 except that the output parameter was computed differently i.e. *fix released* assumed as final status of an issue report. We can still see improvements by adding issue and review metrics. However, issue metrics and review metrics nearly have the same impact on the performance (average precision and recall of models^{c+i} are just 5% higher than models^{c+r}), which might indicate that selected issue metrics are not very impactful on the output parameter and there might be other significant factors not considered in this study. Combining all metrics, the performances of the models are still lower than the results achieved in section 6.4, specially regarding precision, which is above 75% in section 6.4 and here is almost less than 70%.

These results can confirm our assumption that there are metrics influencing issue release time than those studied in this paper. Recent study by Alencar da Costa et al. (da Costa et al., 2016) reveals that *queue rank* and *cycle queue rank* are the two most important factors affecting integration delay in traditional and rapid releases respectively. Although they studied a different project, i.e. Firefox, their result also is an indicator of the presence of some impactful factors on issue release time that are beyond the studied metrics.

6.7 Threats to Validity

We expect that the different proportion of review versus issue resolution time also holds for other systems, but generalizability of our case study results is a threat to the external validity of this study. Although we studied 10 large open source projects, they all belonged to the same OpenStack ecosystem. Studies on other open and closed-source systems are necessary.

Regarding threats to internal validity, i.e., other factors that could explain our results, there are likely other unknown and hidden factors that could influence the review or issue resolution time that we have not measured yet, either because we are not aware of them, or they are not extractable from the available repositories and artifacts. Despite this threat, we studied a wide range of metrics covering ten different dimensions (including code change and human discussion measures).

One major threat to construct validity is the accuracy of the affective metrics. Sentiment, emotion and politeness are measured based on textual analysis of human discussions in re-

Table 6.7 Statistics of the defined patterns across several projects with regard to issue released time

Project	Pattern1	Pattern2	Pattern3
nova	98%	2%	0%
glance	99%	1%	0%
openstack-manuals	97%	3%	0%
keystone	98%	2%	0%
cinder	99%	1%	0%
tempest	96%	4%	0%
neutron	98%	2%	0%
devstack	98%	2%	0%
horizon	99%	1%	0%
heat	99%	1%	0%

Table 6.8 Performance of explanatory models for issue time. ^{c} means model based on "change metrics",$^{c+r}$ means model based on "change metrics" and "review metrics", $^{c+i}$ means model based on "change metrics" and "issue metrics"

project	p^{c}	r^{c}	f^{c}	auc^{c}	p^{c+r}	rl^{c+r}	f^{c+r}	auc^{c+r}	p^{c+i}	r^{c+i}	f^{c+i}	auc^{c+i}	p	r	f	auc
cinder	0.54	0.54	0.54	0.53	0.60	0.59	0.59	0.59	0.59	0.60	0.59	0.59	0.61	0.62	0.61	0.61
devstack	0.51	0.59	0.55	0.56	0.55	0.61	0.58	0.60	0.66	0.63	0.65	0.68	0.70	0.69	0.69	0.72
glance	0.66	0.66	0.66	0.66	0.76	0.69	0.72	0.73	0.75	0.76	0.76	0.75	0.76	0.78	0.77	0.76
heat	0.62	0.61	0.61	0.61	0.63	0.63	0.63	0.63	0.62	0.57	0.59	0.60	0.64	0.61	0.63	0.63
horizon	0.59	0.58	0.58	0.57	0.63	0.58	0.60	0.61	0.63	0.59	0.61	0.60	0.65	0.62	0.63	0.63
keystone	0.67	0.54	0.60	0.55	0.71	0.50	0.59	0.59	0.71	0.57	0.63	0.60	0.74	0.54	0.63	0.62
neutron	0.52	0.49	0.51	0.52	0.55	0.37	0.44	0.53	0.67	0.60	0.63	0.65	0.68	0.62	0.65	0.66
nova	0.56	0.58	0.57	0.55	0.62	0.63	0.63	0.61	0.63	0.62	0.62	0.62	0.65	0.65	0.65	0.64
...-manuals	0.64	0.63	0.63	0.62	0.63	0.60	0.61	0.61	0.76	0.72	0.74	0.73	0.74	0.71	0.73	0.72
tempest	0.61	0.60	0.60	0.60	0.61	0.58	0.60	0.60	0.65	0.63	0.64	0.64	0.66	0.61	0.63	0.64
Average	0.59	0.58	0.58	0.58	0.63	0.59	0.60	0.61	0.69	0.63	0.65	0.65	0.68	0.65	0.66	0.66

view and issue comments. In this paper, we adopted state-of-the-art tools like SentiStrength, Danescu et al.'s politeness tool (subsection 6.2.1), and the emotion classifier tool proposed by Murgia et al. (Murgia et al., 2014). However, due to the ambiguity in natural language, the affective measures are approximations and there are some exceptions like sarcasm that can not be identified by tools. Elfenbein and Nalini also claim that to precisely identify emotions, sentiments or politeness in comments, one should understand the developers' culture, including their dictionary and slang (Elfenbein and Ambady, 2002). Tools clearly are not able to do this, yet the fact that we confirmed Ortu et al.'s findings regarding positive sentiment gives us confidence that the tools and measures used convey a trend.

Apart from the impreciseness of natural language analysis, our study assumes that the textual write-up of issue and review comments clearly convey the emotions, sentiment and politeness of the developers involved. Based on empirical evidence (Pang and Lee, 2008), we assume

that there is indeed a causal relationship between developer's feelings and his or her writings in comments.

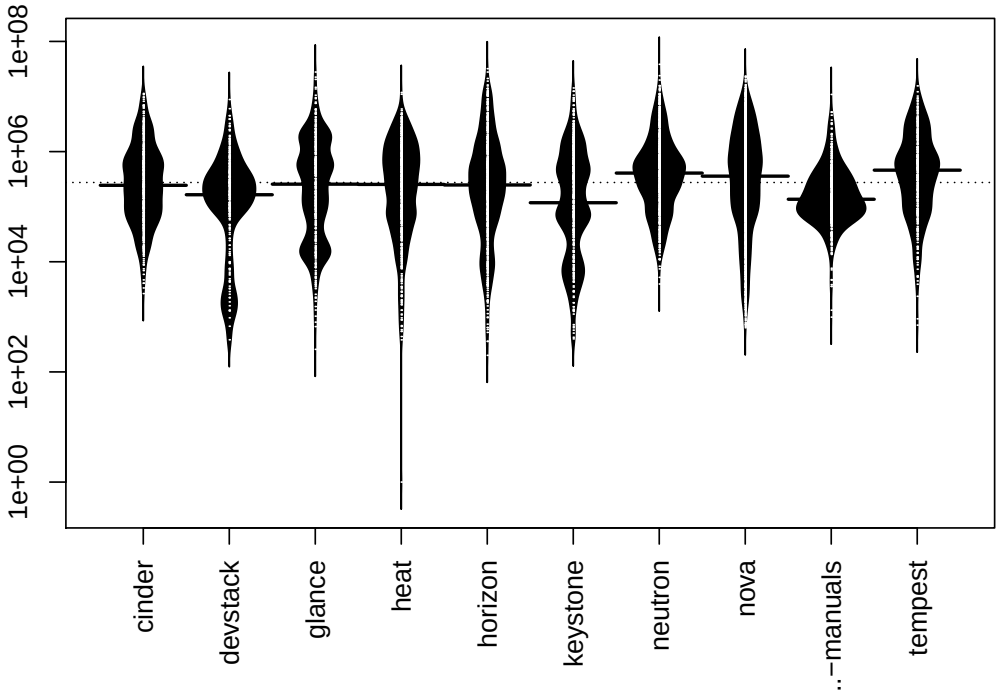
A final threat to construct validity are the heuristics and data filtering techniques used to extract the different metrics, requiring us to link three kinds of repositories (version control, bug repository, and code review). We mitigate such a threat by using a well-known outlier filtering procedure and by testing our scripts. In particular, we had to ensure that the extracted dates had the right time zone information and no inconsistencies existed. As mentioned in section 6.2, there are commits with more than one issue report, which might impact our results. To mitigate this threat, we computed the number of commits with more than one issue reports and found out that except for "swift", 70% of commits in the projects corresponded to only one issue report.

Moreover, we assumed that code reviews and issue resolutions are fully documented and communicated in the Gerrit and Launchpad tools. While these assumptions hold in most cases, some issue resolution and code review activities could be performed via other channels such as in person meetings, emails, and so on.

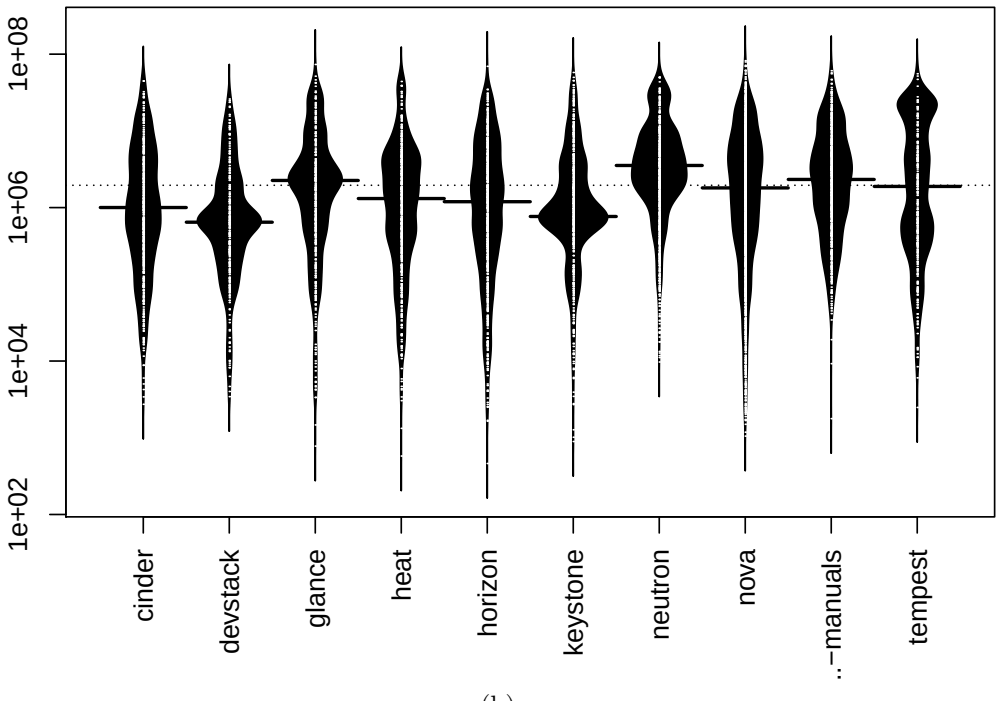
6.8 Conclusion

Given the growing pressure in today's software companies to decrease time-to-market, this paper contrasts the factors impacting time of two main tasks of developers, i.e., issue resolution and peer reviewing. We found that, although related, both processes are only loosely correlated, suggesting that different factors impact the time taken by issue resolution and review. We indeed found that the review time models were dominated by review-related metrics, and issue resolution models by issue-related metrics, except for those projects where review time was a large enough part of the issue resolution process. CI builds, non-bug fixes and positive interaction between reviewers were the most influential review-related metrics impacting review time, while tossing, status changes, and experience are the most influential issue-related metrics.

While our findings regarding the impact of CI builds and positive interaction between discussion participants could be interesting, at a higher level, the take-home message of this paper is that, although relatively accurate models of review and issue resolution time can be built, preliminary empirical analysis is essential to determine the categories of metrics that make sense in such a model. In particular, differences between projects in the proportion of time taken up by reviews relative to the time taken for issue management explain important differences in the impact of metrics.

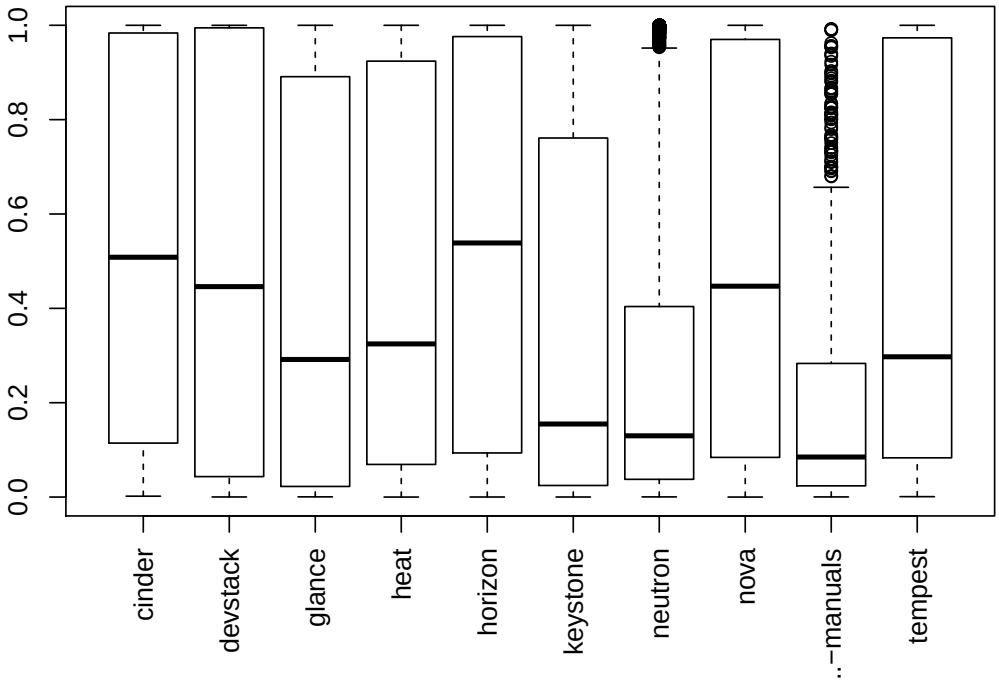


(a)

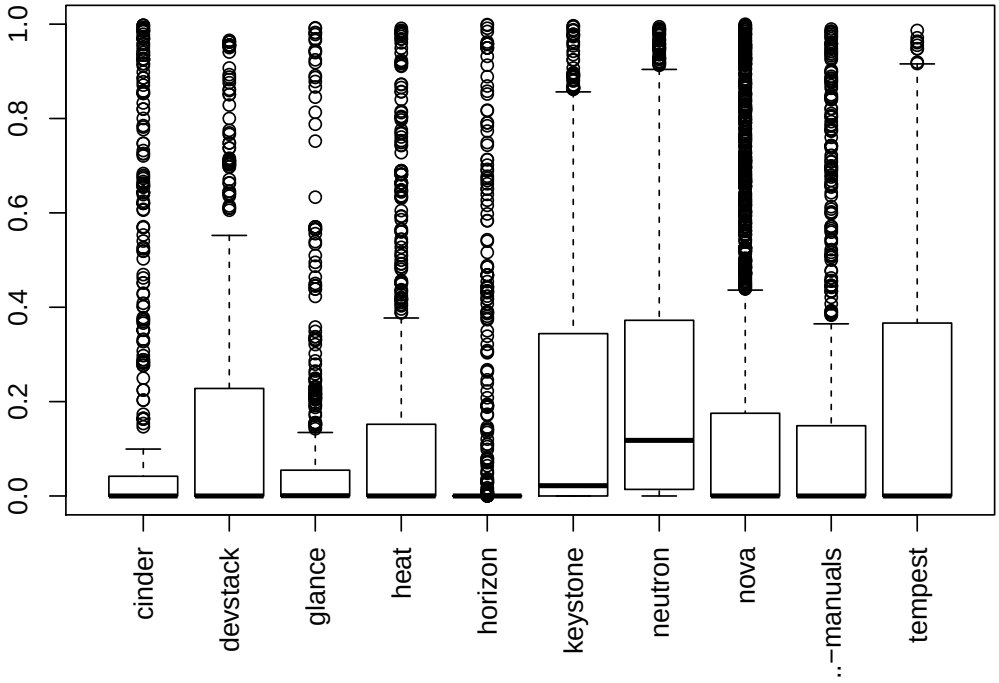


(b)

Figure 6.4 Distribution of (a) Review and (b) Issue time (#hours).



(a)



(b)

Figure 6.5 Proportion of issue resolution time spent (a) during and (b) after review.

Table 6.9 Summary of change-related (c) (Kamei et al., 2013a), issue-related (i) (Ortu et al., 2015a) and review-related (r) (Tourani and Adams, 2016) metrics used in our study. Metrics applicable both to review and issue resolution are marked as "b".

Dimension		Name	Expected Rationale
Diffusion	c	NS (#)	More modified subsystems likely means longer review/resolution.
		ND (#)	More modified directories likely means longer review/resolution.
		NF (#)	More modified files likely means longer review/resolution.
		ENT (#)	Changes affecting multiple files *equally* might need more time to review/resolve.
Size	c	LA (#)	Changes adding more code probably require more time.
		LD (#)	Changes removing more code probably require more time.
		LT (#)	Larger code files might be harder to review/resolve
Purpose	c	FIX (boolean)	Bug fixes might require more thought to review/resolve.
History	c	NDEV (#)	The higher the avg. #developers that changed a file before, the more time.
		AGE (#)	The higher the average time (#days) since the last change to a file, the more time.
		NUC (#)	The more #unique changes happend to the modified files before, the more time.
Experience	c	EXP (#)	Developers with more prior commits might be faster at reviewing/resolving.
		REXP (#)	Developers that recently modified a file have more fresh knowledge.
		SEXP (#)	Developers dominant in a subsystem might be faster to review/resolve.
Focus	b	commenter_experience (#)	The more experienced in leaving review/issue comments, the more participative and helpful in discussions .
	i	issue_reporter_experience (#)	The more experienced in issue reporting, the more accurate the information provided, the faster review/issue resolution can be done.
		issue_fixer_experience (#)	The more experienced in issue fixing, the more skillful, the faster issue resolution can be done.
		issue_changes (#)	The more often an issue changes its status, the longer the resolution time.
		issue_tosses (#)	The more time lost finding the right person, the longer review/resolution takes.
	r	reviewer_experience (#)	The more experienced in reviewing, the faster reviewing will be.
		patch_Revisions (#)	The more revisions a patch required, the more time it needs to be reviewed.
		inline comments (#)	The more comments reviewers provide on specific lines of a patch, the more detailed and hence slower the reviewing process.
Integration	r	PositiveCI (#)	The more successful verifications done by CI, the less build and test problems, and hence the shorter the review process.
		NegativeCI (#)	The more failing verifications done by CI, the more build and test problems, and hence the longer the review process.
		workflow (#)	The more the review of a patch is paused, the longer the review time.
Quantity	r	votes (#)	The more review scores, the more reviewers and hence longer review time.
		negative_votes (%)	Larger percentage of negative review scores may indicate longer review time.
Length	b	comments (#)	The more comments are posted in a discussion, the longer the review/resolution.
		comment length (#)	The more number of lines a review/issue comment takes, the more discussion, indicating that the patch's review/resolution might take longer.
Affect	b	Comment Sentiment (#) (first/last comment, title, avg)	Negative sentiment of the participants during issue or review discussions may indicate an unsatisfied situation and longer resolution/review time.
		Comment Politeness (#) (first/last comment, title, avg)	Low politeness during discussions might be an indicator of aggression in communication, resulting in longer time for resolution or reviewing.
		love/sadness/anger/joy (prob.) (avg)	More love and joy during discussions may be indicators of effective and constructive communication, while sadness and anger may disclose hidden problems or obstacles during development, impacting review/resolution time.

CHAPTER 7 ARTICLE 4: CODE OF CONDUCT IN OPEN SOURCE PROJECTS

Abstract

Open source projects rely on collaboration of members from all around the world using web technologies like GitHub and Gerrit. This mixture of people with a wide range of backgrounds including minorities like women, ethnic minorities, and physically challenged people may increase the risk of offensive and destroying behaviours in the community, potentially leading affected project members to leave towards a more welcoming and friendly environment. To counter these effects, open source projects increasingly are turning to codes of conduct, in an attempt to promote their expectations and standards of ethical behaviour. In this first of its kind empirical study of codes of conduct in open source software projects, we investigated the role, scope and influence of codes of conduct through a mixture of quantitative and qualitative analysis, supported by interviews with practitioners. We found that the top codes of conduct are adopted by hundreds to thousands of projects, while all of them share 5 common dimensions.

7.1 Introduction

Since technologies have enabled new forms of open collaboration on the Internet by adopting tools like version control, mailing lists, chat systems and wikis, a large crowd of project members with diverse experiences and interests are able to collaborate in open source projects to produce large, complex and successful systems (Gutwin et al., 2004). Indeed, according to the Open Source Initiative, "In order to get the maximum benefit from the [open source] process, the maximum diversity of persons and groups should be equally eligible to contribute to open sources" (ope). Furthermore, Vasilescu et al. have shown that gender diversity is beneficial for productivity of GitHub teams (Vasilescu et al., 2015a).

However, a mixture of people with different cultures, personalities and interests may increase the risk of offensive behaviours happening. For example, on June 18 2015, one of the core maintainers of OpalRB, a Ruby-to-Javascript transpiler, left transphopic comments on Twitter[1]. The resulting pile-on of responses ranged widely from those discussing that publicly stated opinions of a member have no bearing on the community, to those who expected consequences for the offensive person (e.g., excluding him from the community) or those

[1]https://twitter.com/krainboltgreene/status/611569515315507200

thinking that as long as the contributed code was good, working with the offensive member was acceptable[2]. Similarly, gender-related incidents have been reported as cause for leaving a project (Vasilescu et al., 2015b).

Therefore, it seems essential for open source communities to protect their members from these kinds of unacceptable, destroying behaviours and provide a welcoming, safe, friendly, and inclusive environment in which people can collaborate effectively towards presenting successful products. So far, the concept of code of conduct has been emerging as possible solution, since e.g. in OpalRB the above discussion has led to a discussion on adopting a code of conduct for this project[3]. Such a code of conduct basically establishes ground rules for communications between participants, outlines enforcement mechanisms for violations and tries to codify the spirit of a community, such that anyone can contribute comfortably regardless of gender, ethnicity, physical challenges or sexual orientation.

However, some communities find codes of conduct repressive and a threat for open source communities. One common argument is that participants in open source communities are mature enough to deal with debates and differences, and hence it should be obvious for members how to behave. Some strong opponents explicitly picked a "No Code of Conduct" for their communities[4]. Furthermore, even in projects that do adopt a code of conduct, the adoption process suscitates substantial discussion because of doubts whether codes of conducts work and disagreement about what should go into them and the exact wording to use[5].

Since there is no empirical evidence regarding the status, nature of, and procedure for establishing codes of conduct in open source projects, the primary purpose of this paper is to empirically examine codes of conduct in open source projects, identifying the procedures followed in their implementation and monitoring, as well as understanding its scope and impact in open source communities. To this end, we address the following research questions:

RQ1) *What are the major codes of conduct in open source projects?*

There are seven common codes of conduct used by more than 51,000 open source projects.

RQ2) *What do major codes of conduct stipulate?*

[2] `https://github.com/opal/opal/issues/941`
[3] `https://github.com/opal/opal/issues/942`
[4] `https://github.com/domgetter/NCoC`
[5] `https://github.com/opal/opal/issues/942`

Codes of conduct outline OSS community's expectations and values against members' behaviours to create friendly and inclusive community, while, violations of them have consequences. Their scope is all spaces of the community, either online or offline, however, it can be broader.

RQ3) *How are codes of conduct used in open source projects?*

Community's concerns, needs and history play an important role in their code of conduct design. But, similarities among communities may lead communities to reuse same code of conduct.

In the remainder of this paper, we first describe the necessary background notions and related work (section 7.2). Then, we investigate the first research question, present its approach and results(section 7.3), this section will be followed with the second and third research questions (section 7.3, section 7.3). After mentioning the threats to validity (section 7.4), we finish with conclusions (section 7.5).

7.2 Background and Related Work

7.2.1 Open Source Projects and Diversity

Eric Raymond (Raymond, 1999) summarized open source software (OSS) differences and properties in comparison to other types of software development. More specifically, the open access model of OSS development encourages participants with maximum ability and skill, specific expertise and minimum restrictions in geographical issues to participate. This leads to a high degree of diversity amongst project members in terms of gender, ethnicity, religion and age, which is likely to be an influential factor impacting OSS project success (Colazo and Fang, 2010). Bazile-Jones et al. (Renee Bazile-Jones, 1996) stated that managing and valuing diversity in workplaces, as one intellectual asset, can bring long-term wealth. Valuing diversity refers to recognizing individual differences and dissimilarities, and respecting them by considering everyone's needs and expectations. Sherae et al. expanded on the theoretical understanding of diversity and its implications in OSS projects (Daniel et al., 2013). They defined three types of diversity, i.e., disparity (based on contribution reputation), separation (based on culture) and variety (reputation) diversity. They discussed how each of these types plays a role in the project success from a community engagement point of view or market success. Vasilescu et al. in (Vasilescu et al., 2015a), discussed about the various aspects of diversity in open source projects. Using GitHub, the largest public code repository for OSS projects for data extraction, they finally showed more diversity in gender and tenure

is associated with higher productivity and turnover. In other words diverse teams which consist of both men and women while some having more experience and some less, show better performance.

7.2.2 Code of Conduct

According to wikipedia [6], a code of conduct generally is a set of rules articulating standard behaviour and responsibilities for an individual, party or group. It is commonly written for employees of a company in order to 1) protect the business and 2) inform the employees of the company's expectations. The International Federation of Accountants (IFA), IFAC, provided a more precise definition. According to them, a code of conduct comprises principles, values, standards and rules that act as guidelines that have the overall benefit of the stakeholders in mind and at the same time respect the rights of members.

Recently, codes of conduct have been adopted in OSS communities, as open and welcoming communities, to deal with diversity problems. One of the oldest codes of conduct in OSS communities is Ubuntu designed for Ubuntu community more than 10 years ago, newer versions of it published later. Some OSS codes of conduct like Ubuntu are being used by other OSS projects. We will discuss about famous OSS codes of conducts and their prevalence in section 7.3.

It is worth mentioning that a code of conduct is different from a code of ethics. A code of ethics is adopted to clarify for members the meaning of "right" and "wrong" based on the business of the company, and therefore is applied to make decisions (IFA) about their actions and manners. A code of conduct on the other hand is confined to actions or behaviours of employees and is usually intended for them only (instead of for stakeholders).

Workplace Harassment

Most codes of conduct aim to protect members from harassment, thus it seems important to understand what workplace harassment means. According to the Oxford dictionary, harassment is "Aggressive pressure or intimidation". As such, workplace harassment is any offensive, belittling or threatening behaviour toward an individual worker or group of workers. It results in an unpleasant, humiliating or intimidating environment employees feel uncomfortable in and consequently damages effective work and productivity of employees[7].

In OSS communities, just like any other workforce, workplace harassment may include, but is

[6] https://en.wikipedia.org/wiki/Code_of_conduct

[7] `https://web.archive.org/web/20120328034350/http://apsc.gov.au/publications01/harassment.htm`

not limited to, online or offline harassing behaviours such as verbal comments, sexual jokes or insults, sexual images in public spaces, intimidation, stalking, inappropriate physical contact, bullying.

7.2.3 Related Work

The subject of the use of codes of conduct in business is not new, as Heermance discussed similar concepts in 1924 in his book (Brogan, 1925). According to J. White et al. (White and Montgomery, 1980), a number of proposals for general and industry-related codes of conduct appeared, with the business communities also showing interest in codes of conduct to establish standards of behaviours in their corporation. J. White et al. also mentioned the need for codes of conduct increased in the late 1970s due to many corporations, all over the world, getting decentralized, geographically dispersed, since their management wanted to share their core philosophy among the various offices. A code of conduct was the corporations' primary means to ensure and promote appropriate behaviour.

White et al. conducted a survey study of 673 financial corporations in that period. According to their study, corporations then would call a code of conduct differently, such as "ethical practices statement", "guidelines for employee conduct" or "policy statement on corporate ethics". They analyzed the contents of these codes of conduct and found that most of them contained a general statement expressing the philosophy of the company's management, employees' responsibility to comply with laws and regulations, and also the employees' responsibility to avoid conflict-of-interest situations with competitors, suppliers or customers.

According to Brenner et al. (Brenner and A.Molander, 1971), corporations apply a code of conduct to achieve goals such as fostering positive values, resolving ethical problems, communicating and promoting a company's moral values. Codes of conducts can also provide a basis for ethical behaviour in other domains like academia (Rezaee et al., 2001). Rezaee et al. discussed that such codes of conduct in colleges and universities should highlight some key issues, such as preventing financial and scientific fraud.

Rashid et al. (Rashid et al., 2015) considered ethical concerns of a software engineering society from a different perspective. They discussed how contemporary software systems increasingly are getting more open while managing a large portion of people's daily lives. Moreover, these systems may lead to crime and jeopardize people's safety and sustainable living. These concerns should be tackled by software engineers during the design and implementation of the system. Although software engineering codes of ethics like the one of ACM (ACM) give some guidelines for ethical decision making throughout the software engineering process, they are not practical enough to be applied in the software construction and development process.

Instead, Rashid et al. proposed a framework based on Boehm's Spiral model of software engineering (Boehm, 1988) to decompose ethical concerns and needs in different steps of the framework to be tackled in each stage. Even though their research deals with ethical concerns in software engineering, it mostly focuses on the final software products, not the software community and teams developing a software product.

Becker et al. (Becker et al., 2015) provided a common perspective for researchers and practitioners of the role of software engineering in sustainability, to enable effective communication about this concern. Similar to Rashid et al., they discussed how the software engineering discipline can impact sustainability challenges related to society and natural environment. Existing codes of ethics for computing professional are not extensive enough to cover these issues and the software engineering discipline lacks a framework to incorporate sustainability thinking into the design of software. Hence, they identified five sustainability dimensions. We conjecture that codes of conduct can cover parts of the social (equity, justice and democracy) and individual (comfort and welfare) dimensions in the software engineering discipline.

Tourani et. al in (Tourani and Adams, 2016), empirically studied the impact of human discussion metrics such as developers' sentiments expressed in their discussions, beside change-metrics, on the quality of the resulting work in terms of the number of defect-introducing commits. Their results showed that sentiment-related metrics play role in more than half of their models, i.e., they have impact on product quality. Their results revealed the importance of developers' sentiments on the quality of the project. Previously, Ortu et al. showed human affective metrics of developers, like politeness, emotion and sentiment can affect issue fixing time, e.g. positive emotions like happiness are linked with shorter issue fixing time (Ortu et al., 2015a).

7.3 RQ1. What are the major codes of conduct in open source projects?

Motivation This first research question is concerned with finding the frequently used codes of conduct as well as with determining their prevalence in open source projects. In addition to showing the relevance of this paper and motivating the need for further research on codes of conduct, a list of popular codes of conduct, with the high number of usages, will be used to answer RQ2, i.e., to understand the common ingredients of codes of conduct.

Approach

We used a two-pronged approach. First, we used a list of seven codes of conduct, consisting of the Open Code of Conduct of the ToDo group[8], a well-known organization in the field, as

[8] `http://todogroup.org/opencodeofconduct/`

well as six other codes claimed by the ToDo group to be the giants on whose shoulders Open Code of Conduct stands. We performed a brute-force search on GitHub with the names of these codes of conduct to determine the order of magnitude of their popularity of these codes. GitHub is a popular code repository site used by millions of popular and active open source projects (Dabbish et al., 2012). Since this first approach only aims to provide an order of magnitude for the prevalence of the seven suggested codes of conduct, and given the number of search results, we did not eliminate duplicate search results or false positives. Furthermore, searching only by name ignores projects that just mention the URL of the code of conduct used.

While the above limitations are acceptable to obtain an order of magnitude, our second approach instead uses a second group of data sets and involves manual analysis to get an accurate set of mappings between projects and the codes of conduct that they use. To do this, we used the principles of systematic reviews proposed by Kitchenham (Kitchenham et al., 2009): our population is "Open Source communities", our intervention is "Code of Conduct", and two electronic databases are used, i.e., Github[9] and Google. The latter allows to find projects not hosted on GitHub or whose code of conduct is not stored in their version control system.

We used the same queries for both GitHub and Google: 1) *"code of conduct" "open source"* and 2) *"code of conduct" software community*. Quotation marks were explicitly added to reduce the number of false positive hits. Furthermore, we added "open source" and software community to filter out codes of conduct for conferences and other events as opposed to codes of conduct for software projects, which are the focus of this paper.

Although the Google search engine initially claimed that 57,000 and 2,110,000 hits were found, respectively, for the two search queries mentioned above, in reality, after Google's own duplicate filtering, the number of hits turned out to be around 500 and 700. After further manual removal of additional duplicates and incomplete matches of the query, the results yielded only 395 and 324 actual hits, respectively. Finally, after filtering out search results related to schools and conferences, we obtained 306 and 241 hits, respectively. Manual analysis of the entire collection of $547 = 306 + 241$ links yielded 108 unique open source communities using a code of conduct, including well-known ecosystems such as Apache, Eclipse, Openstack, Debian, and Scala.

For GitHub, the two queries yielded 17,498 and 2,417 textual files in markdown format (mostly README files), respectively. The top 200 hits of the first query (based on GitHub's "best match"), and top 40 hits for the second query (idem) were manually checked to see

[9] `https://github.com/search`

if they belong to codes of conduct of open source projects. Out of these 240 links, 184 corresponded to actual open source projects using a code of conduct. The false positive links include duplicate links across different versions of a project, or irrelevant links of non-software communities (e.g., Software Carpentry[10]).

Finally, as an additional data source, we also studied the top GitHub projects to analyze whether they use a code of conduct, and if so, which one. We used the number of watchers of a GitHub project as a measure of project popularity, similar to other work (Vasilescu et al., 2015c). We studied the 150 most watched projects and found that 52 of them have a code of conduct, including such projects as Linux, JQuery, Angular and Swift.

In the obtained search results in Google and GitHub, we then compared the number of occurrences of each identified code of conduct to the order of magnitude numbers for the initial seven codes of conduct, and also tried to identify any missing major codes of conduct. However, since some projects may focus more on GitHub, while others do not, we considered each data source separately. In addition, we noticed that many Google results correspond to ecosystems instead of to individual projects, while the GitHub results mostly correspond to individual projects. As such, the Google ecosystem results actually imply that a larger number of existing projects are using the same code of conduct Table 7.2.

Results

Eleven codes of conduct are commonly used in open source projects, with seven of them ranging from 500 up to several thousands of adopting projects.

RQ2. What do major codes of conduct stipulate?

Motivation In the previous research question, we identified seven common codes of conduct used either directly or as the basis for a custom code of conduct. Here we are interested in understanding the content of such a code of conduct, i.e., the basic elements, measures or other guidelines, as well as the way in which these are written up. In particular, we are interested in understanding the kinds of behaviour addressed by codes of conduct, potential measures taken, but also whether codes of conduct are written up as suggestions versus stringent rules (i.e., the style of writing).

Approach To answer this research question, we manually studied the seven popular codes

[10] `http://software-carpentry.org`

Table 7.1 Popularity numbers (order of magnitude) for codes of conduct obtained from GitHub (first approach).

Code of Conduct	Number of Hits on GitHub	Examples
Contributor Covenant	43,681	Molajo/Molajo, trevororeilly/dotfiles, SecComm/Crayon, yuluyi/Isomorphic-React-Seed, tweetstockr
Open Code of Conduct	2,167	wildland/cli-tools, KineticCafe/code-of-conduct, Tacklr/CacheManager, spotify/ios-ci, PearlCast/PearlCast
Python	2,025	PyDiff/PyDiff.github.io, 18F, brettcannon/oplop, link39/205-pi, roadcap/homebrew,sfdevs/sdcodecamp
Citizen	1,253	npr/npr-one-api-js-sdk, cworth-gh/stony, gulpjs/gulp, lkodai/Design-LK, ojs/ojs, ctdk/goiardi
Ubuntu	1,180	goodeggs/format-location, Star2Billing/cdr-stats-docs, garyjs/Newfiesautodialer,Alamofire/Foundation, Trustroots/trustroots
Django	1,054	jrief/django-angular, DBCboots, Pythonkc, Calagator, ordergroove/check_mariadb_slaves
Geek Feminism	544	nzruby, brave/chromium, crosswalk-project/chromium-crosswalk, javascripthers/javascripthers.github.io, openSNP/snpr

of conduct obtained in RQ1. For each of them, we read and identified the underlying components, then looked for similarities and differences. Whereas most of the seven codes are independent, we found considerable overlap in terms of their major components and ingredients. In particular, each code of conduct included the following five components in one form or the other:

- Purpose: the rationale for the code of conduct, typically the desire to obtain a certain kind of environment for project members to work and collaborate in.
- Honorable behaviour: behaviour that is valuable for and accepted by the community.
- Unacceptable Behaviour: negative behaviour that should be avoided.
- Enforcement: mechanisms for reporting and punishing violations of the code of conduct.
- Scope: the online and offline spaces where the code of conduct applies, for example only in mailing list or any online discussion forum.

Below, we detail each of these components in more detail, while Table 7.1 summarizes how the above components and other dimensions apply to each of the seven codes of conduct. Note that for codes of conduct with more than one version, we referred to the latest one.

Table 7.2 Popularity numbers for codes of conduct obtained via Google (second approach). Codes of conduct from Table 7.1 have been emphasized.

Code of Conduct	#Google Results (out of 108)
Ubuntu	20
Contributor Covenant	20
Django	13
Python	9
Citizen	9
Open Code of Conduct	7
Geek Feminism	7
Mozilla	6
Twitter	5
Rust	5
Ada Initiative	4
KDE	4
SpeakUP	3
Apache	2
Thoughtbot	3
Openstack	3
Debian	2
Puppet	2
PyCon	2

Results

Purpose: All codes stress the desire of diversity and of a welcoming community, while some explicitly list the desired diversity attributes (e.g. gender, sexual orientation and disability). Since OSS project communities consist of professionals and volunteers from around the world, the seven codes of conduct all promote an inclusive and safe environment to everyone, for the sake of sustainability of the community. For example, the *Contributor Covenant* refers especially to personal characteristics like gender, age, size, body, religion, ethnicity, and sexual orientation. *Python* and *Ubuntu* just generally refer to diverse groups, without explicitly naming them, while *Citizen* also mention socio-economic status and *Django* adds political belief and family status to the list of known diversities. The *Open Code of Conduct* contains a separate section, i.e., "diversity statement", which explicitly deals with diversity and encourages members towards some expected behaviours in different situations like when a participant has made a mistake. Furthermore, they also refer to language and technical abilities as diversity axes.

Honorable behaviour: codes of conduct tend to pinpoint general positive be-

Table 7.3 Popularity numbers for codes of conduct obtained via GitHub queries (second approach). Codes of conduct from Table 7.1 have been emphasized.

Code of Conduct	#GitHub Results (out of 184)	#top GitHub Results (out of 52)
Ubuntu	42	3
Twitter	42	3
Django	38	2
SpeakUP	36	0
Apache	35	0
Python	32	0
Contributor Covenant	15	19
Mozilla	11	3
Open Code of Conduct	7	2
Thoughtbot	5	0
Citizen	4	2
Rust	3	4
Ada Initiative	3	0
Geek Feminism	0	1

haviours like being respectful, patient, kind, focusing on the best for community, being considerate and collaborative. The *Python* code of conduct is the least specific about positive behaviours, only mentioning being open, considerate and respectful, while *Geek Feminism* does not mention any accepted or encouraged behaviour. *Ubuntu* listed more detailed positive actions such as encouraging community members to ask questions in case of doubt, stressing the responsibility of everyone to answer such questions. They also encourage any members who want to leave the community to do so with minimal interruption for the project. The *Ubuntu* code of conduct also has a special part about leadership and responsibility, for instance pointing out that leadership can be taken up by anyone competent in the community, basically declaring themselves as meritocracy in their code of conduct. Expected behaviours for leaders are also named, such as highlighting and rewarding great work of others, or being courageous to take bold decisions. Such leadership-related clauses seem specific to ecosystem-related codes of conduct.

Unacceptable behaviour: most codes of conduct denounce sexist/racist language, contempt and jokes that harass marginalized people, as well as violence and threats. The *Contributor Covenant* explicitly mentions sexualized language or imagery, trolling, insulting and publishing of private information of others as unexpected behaviors. *Django* adds discriminatory jokes and violent threats to this list. *Geek Feminism* and the *Open code of conduct* provide a more detailed list, pointing out additional issues such as

deliberate misgendering, physical contact, stalking, following, harassing photography and recording, and threats of violence. *Python* and *Ubuntu* do not refer to unexpected or negative behaviors in their codes of conduct at all.

The studied codes of conduct differ in the style used in their description, i.e., some like *Geek Feminism* just mention negative behaviours to discourage community members from exhibiting those, while on the opposite side of the spectrum *Python*, for instance, just states desired and valued behaviours to reinforce such positive behaviours. Finally, some codes like *Citizen* and *Ubuntu* refer to both unacceptable and honorable behaviours. The styles of the analyzed codes of conduct, either positive or negative, are shown in Table 7.1.

Furthermore, while expressing positive and negative behaviours, some codes of conduct like *Citizen* are phrased in the form of rules like "Refrain from demeaning, discriminatory, ...", while others like *Ubuntu* instead state their intent by listing the expected values of the community like "Be respectful". In other words, codes of conduct can be phrased in two different ways (IFA): a policy-based (or rule-based) approach or a values-based (or principles-based) approach. The former are very detailed and provide rules and policies in the form of dos and don'ts, while the latter are expressed by examples and principles rather than exhaustive policies and rules. In between these two extremes of codes of conduct, some codes of conducts can mix both approaches. For instance, *Django* is rule- and values-based at the same time, since it discusses both the value of "Being respectful", but also states "Do not insult or put down other participants". Table 7.1 shows the approach taken by the analyzed codes of conduct.

Enforcement: In communities with a code of conduct, unacceptable behaviors typically are reported to a specific group of team members with the power to decide about the appropriate actions to take. In the *Contributor Covenant*, violations can be reported via a specified email address. The reporter should be treated confidentially, while the report must be investigated and followed up appropriately. In addition, if the code of conduct is not enforced correctly, the project leadership can repress the corresponding maintainers. The *Open code of conduct* specifies a more detailed process, such as the information required in a report and who will be responding in special cases (e.g., when the respondent himself did the harassment). *Django* promises to answer reports within a specific period (one week) but they prioritize ongoing situations and threats to physical safety as distinguished incidents to be addressed immediately.

Geek Feminism proposes a responsible team called the "Geek Feminism Anti-Abuse Team" that promises to not name the victim publicly and respect her confidentiality. *Citizen* just introduces a specific group of members with their contact information for receiving the vio-

lation reports, claiming that respondents will help victims. Surprisingly, *Ubuntu* and *Python* do not mention any reporting or enforcement guidelines in their codes of conduct.

Since some codes of conduct, especially those intended to be customized by other open source projects, express honorable and unacceptable behaviours in a generic way, declaring concrete enforcement and punishment mechanisms is not possible for them as these would differ based on the specific project that adopts them. However, even in those case, some codes of conduct like *Django* and *Geek Feminism* still provided boundaries for any punishments. For example, *Django* proposes a list of punishments ranging from "Nothing" to "A request for a public or private apology" that will be performed in response to violations, while *Geek Feminism* warns to exclude offenders from the community as response to a violation, or to publicly identify the harasser to the project's (or even general) community. *Citizen* briefly mentions the consequences of unaccepted behaviors like permanent expulsion from community as worst case penalty.

Scope: codes of conduct apply to all community members, i.e., both paid and volunteering contributors, in all community spaces (online or offline). The *Contributor Covenant* and *Citizen* define their scope not only as the communication spaces within the project, such as mailing lists, but also as any outside space a community member is representing the project, for example when using an official project email address. The enforcement scopes of *Django* and *Geek Feminism* seem wider, as they cover not only the community spaces, but also any intervention of a member outside the community (either when representing the community or not). In case of any violation, the responsible committee should be informed and this may have consequences for that member. *Python*, *Ubuntu*, and the *Open Code of Conduct* do not explicitly state the scope of their codes of conduct.

The different scopes of codes of conduct are related to the intention behind the code. For example, *Geek Feminism*, as a community supporting women who do geeky activities, has as goal to support women within these activities everywhere, with as consequence that their code of conduct ranges even outside the community spaces. This contrasts with the goal in some communities like OpalRB that aim at representing the more successful projects, and only care about things that are strictly related to their projects' community[11]. This is why the Opal community, after the notorious incident discussed in section 7.1, adopted[12] *Contributor Covenant* v1.0[13], a version of this code of conduct where the boundary of enforcement is not specified and can be restricted to the inside of the community.

[11] https://github.com/opal/opal/issues/942
[12] https://github.com/opal/opal/pull/947
[13] http://contributor-covenant.org/version/1/0/0/

Table 7.4 Comparison of the characteristics of the seven studied codes of conduct.

Code of Conduct	Length (pages)	Phrasing	Style	Scope	> 1 Version?	Dependencies
Citizen	3	Rule	Both	All community venues online or offline. Outside the scope of community when violations adversely affect well-being of community.	Yes	Django, Geek Feminism
Contributor Covenant	2	Rule	Both	Within project spaces and in public spaces when representing the community, for example by email.	Yes	
Django	4 (with reporting guidelines)	Both	Positive	All spaces online or offline, outside spaces may affect person's participation	Yes	Speak UP!
Geek Feminism	3	Rule	Negative	All Geek Feminism sponsored spaces, both online and offline, and also outside.	No	
Open Code of Conduct	3	Rule	Both	Not mentioned	No (stopped)	
Python	1	Value	Positive	Not mentioned	No	
Ubuntu	4	Both	Positive	Not mentioned	Yes	

Codes of conduct outline OSS community's expectations and values against members' behaviours to create friendly and inclusive community, while, violations of them have consequences. Their scope is all spaces of the community, either online or offline, however, it can be broader.

RQ3. How are codes of conduct used in open source projects?

Motivation So far, we focused on the popularity of codes of conduct in open source projects, and tried to understand the basic elements and attributes of codes of conduct. In this research question, we investigate why and how they emerged in the open source world, which problems they are dealing with, the process and thought behind them, and their influence and limitations in open source communities.

Approach

Since no quantitative data is available about the adoption of codes of conduct, nor their enforcement (e.g. there is no such thing as a code of conduct complaint repository), we opted for interviews with leaders and creators of codes of conduct in open source communities. Creators of 6 of the top codes of conduct in Table 7.2 were contacted and invited to a Skype call interview[14]. Of the 5 positive reactions, we were able to perform 4 interviews, two via

[14]Creation of a code of conduct usually is a collaborative process.

email and two via Skype (each interviewee belonged to a different open source community). We refer to the anonymous interviewees as A, B, C, and D in the rest of the paper.

Skype interviews were held in semi-structured form and email interviews in structured form (Cohen and Bailey, 1997), both based on our list of questions in Table 7.5. In the semi-structured interviews, interviewees could also bring up new ideas to complement our predetermined questions, while in our email interviews they could just answer the predetermined questions. One Skype call took half an hour, the other took one hour (based on the interviewees' availability).

The interviews were audio-recorded and afterwards we prepared a transcription of the talk. Again by applying an *open coding process*, we analyzed the interviews and decomposed them into distinct themes, which approximately mapped to the questions, such as *motivation*, *evolution*, and *strengths*, while labeling them with any important information from the transcripts. Finally, a table was built with the major identified themes as rows, and the different viewpoints of the four interviewees in the different columns.

Results

Motivation for creating a code of conduct (even early on in a project): observations of negative behaviour in previous communities. All interviewees stated that they reflected on their own personal experience of negative behaviours in the past, in different spaces of the community such as on mailing lists or IRC. Therefore, they were motivated to create a dedicated code of conduct for their community. For instance, interviewee B said they wanted to create a new community around some desired behaviours and attitudes that were uncommon in their previous community.

A and B have created their own codes of conduct from the beginning of the project when their communities were still small. They stated that the code of conduct acted as ground rules for the project helping the community to grow, without it they would not be able to tolerate working in the community.

C stated that arguments happen all the time in open source communities, like the typical argument between people who implemented a product and pursue its stability in contrast to members who are looking to grow the product by adding new features. Hence, open source communities are contentious sometimes, and a code of conduct could be one solution to protect the community, retaining a safe and friendly environment.

Ingredients of code of conduct: concerns, expectations, priorities and even history of a community. Similar to the aforementioned motivation for creating a code of conduct, the ingredients to add to those codes of conduct also relate back to the experiences of the creators in previous communities. Two interviewees, B and C, both pointed out that

Table 7.5 Prepared questions for interviews with open source practitioners.

1-	What motivated your community to create a code of conduct?
2-	To create your project's code of conduct, how were the needs of the community gathered?
3-	Did you reuse an existing Code of Conduct (from other industries or software industry)?
4-	How did your project's community react to the idea of having a Code of Conduct?
5-	How is your Code of Conduct enforced? Have any violations been reported already?
6-	What are the strengths and weaknesses of a Code of Conduct for open source projects?

these needs can also be based on the histories of certain kinds of behaviours in the community, the priorities of the community, their products and consequently situations the community might be running into. One interviewee also affirmed that during the writing phase of a code of conduct, the corresponding community should be understood well and should not be dictated by other communities on how they are supposed to behave. Interviewee C also indicated that even culture and geography play a role. For instance, they got feedback from their project members in India and Middle East stating that their code of conduct reflected issues that are too specific to the United States.

Corporate-supported open source projects[15] encountered a duality, as Interviewee C mentioned. On the one hand, they have to comply with their company's policies about campus, central buildings and so on. On the other hand, they also want to offer a welcoming and protective environment for their projects. Thus, there is a gap between existing codes of conduct and the typical issues those organizations have to deal with as corporations, which should be handled in their specific code of conduct too.

However, interviewee A believed that codes of conduct will converge to some extent, because communities already copy text from one another, and he expects this to continue. For instance interviewee A's community copied some sentences from another code of conduct, which were acknowledged explicitly in the text. His opinion was confirmed by interviewee C, since A realized that some communities have similarities that can lead to reuse of codes of conduct. In section 7.3, we also observed that many projects reused another code of conduct.

Evolution: Similar to software artefacts, codes of conduct evolve as well. Interviewee B explained that their code of conduct has been updated five or six times in 10 years.

[15] I.e., major commercial software companies like Microsoft that manage their own open source projects.

Starting from a short document that did not seem to be that serious, gradually several elements were changed, for example the phrasing changed from rule-based to value-based, and they added significant details on the leadership of the community, whereas leadership was not a big deal for them at the beginning. He also stated that throughout all of these versions, they only changed the textual expressions, but not the intention of the code of conduct.

According to interviewee A, every new suggestion about the code of conduct from their community members is welcomed until there are enough arguments and justifications. In interviewee C's community, the discussion for the code of conduct is started by the board, then passed around to all members via the member mailing list. This list is where community needs were addressed and implemented as necessary. Afterwards, changes were voted on by board membership.

As such, all interviewees experienced or expected some changes to their codes of conduct across time. This is because new situations come up, new members are invited, communities are expanded and new or more concrete needs are raised that need to be covered by codes of conduct. From the interviews, it followed that, similar to regular code changes, such changes to a code of conduct should be revised and investigated by a committee before applying them, although different communities might use different processes to apply changes. For instance, in community D, voting among the board membership happens.

Reaction: especially people outside a project complain about the code of conduct. Since most of the interviewed communities grew around their codes of conduct, their projects considered a code of conduct to be a good thing and codes have been adjusted as needed by the community across time. Interviewee A said that the code largely served to attract like-minded people and repel those who did not agree, which caused reinforcement of the norms embedded within the code of conduct. B also confirmed that his community rejected those who wanted to join the community but did not like their code of conduct. Based on B's opinion, a code of conduct is a means for "retention of newcomers but it does not attract any newcomers".

Interviewee C said that none of the communities he runs reacted negatively, i.e., the code of conduct was a non-issue to them. However, according to him, from outside the community, several criticisms have been voiced, mostly like "heterosexual white man can not write a code of conduct and can not understand how to do it". However, in his opinion the criticism that someone is not capable to do something because of his/her color in fact goes against the code of conduct.

Enforcement policy: varies from signing the code by all members to a responsibility for project leaders to protect the community. In B's community, members

should sign their code of conduct, while in communities of other interviewees, there was no such need and the code is adopted implicitly by any contributor. However, interviewee C believed that the code of conduct is a declaration on the part of the project leader, not the community members. It should say that someone as a leader will take the responsibility to address all complaints and will not harass members for complaining.

Interviewee B stated that in their community, a code of conduct is mostly enforced in the sense that people refer to it whenever someone is being disrespectful, for example on a chat channel or mailing list. Just the act of mentioning the code suffices in most of the cases to calm down a conflict. In a select number of cases, when a difficult situation occurred, it had to be raised to a public community counselor. Interviewee A talked about actions that his community chose to enforce a code of conduct: first a polite notice, later a stern notice or warning, then a request to leave, finally a technical measure to prevent participation. During his tenure of around 3 years, only a handful of people had to be asked to leave. Interviewee D had experienced two reports during his tenure.

Strength and weaknesses: codes of conduct promote a friendly and inclusive environment, but may induce a policing environment. Finally, all interviewees believed that a community with a code of conduct has an advantage to foster clear, explicit norms that make the social environment more tolerable, friendly and welcoming to many individuals, while empowering diversity. Interviewee A mentioned that "it helps a community grow in a way that reinforces those norms, by articulating common reference points. It encourages development of social awareness, in this way reflecting on appropriate behaviour, i.e., people skills." Interviewee C believed that having a code of conduct means that leaders make a commitment that they care about their community members.

Interviewee A mentioned that it is impossible to exhaustively enumerate all situations or resolutions for conflicts in a code of conduct, which he identified as a drawback or inability of codes of conduct. He also believed that a code of conduct may embed and enforce norms more specific or larger than some community members desire. The interviewee also stated that some community members may find a code of conduct intolerable, as it may imply a degree of behavioural scrutiny or "policing". Interviewee D confirmed this issue as a weakness and mentioned that some people feel that a code of conduct prohibits them from speaking freely.

About the difference between *rule-based* and *value-based* phrasing of a code of conduct, interviewee B found that both have their benefits, such as signaling a safe and welcoming environment. Rule-based phrasing is more clear and allows to easily react when a conflict happens, whereas value-based phrasing is less likely to deter people from joining the com-

munity, which is an advantage. Since in rule-based phrasing, misbehaviours are mentioned, they may induce to others that the related community is an environment with hostile and unfriendly issues.

> *A community's concerns, needs and history play an important role in the design of its code of conduct. Yet, similarities among communities may lead communities to reuse an existing code of conduct.*

7.4 Limitations and Threats to Validity

Threats to construct validity focus on how accurately our observation measurements are done. To measure the number of open source projects with a code of conduct and also the most popular codes of conduct, we performed queries on Google and GitHub. Certainly, there were false positive hits among these results, as discussed before, while the applied search queries might not return all possible cases. To mitigate these risks, we complemented a rough name-based search approach with 3 smaller, manually analyzed data sets that were used together to evaluate the most popular codes of conduct.

Regarding external validity, for RQ2 we considered seven codes of conduct as representative codes, however these might be different from other existing codes of conduct, or not comprehensive enough to cover others. We reduced this risk by considering seven codes of conduct that seem to be among the most popular codes according to Google queries. However, further studies need to confirm our findings on other codes of conduct. Similarly, while the practitioners invited for the interviews were creators of actual codes of conduct, only 4 of the invited people could be interviewed. Given their experience with the creation and adoption of a code of conduct, and the overlap between their answers, we are confident that our findings cover a large spectrum of codes. Of course, interviews with other practitioners as well as a larger-scale survey are necessary to generalize our findings.

Threats to internal validity concern confounding factors that can influence the results. For example, the interviewees may be biased, since they were the driving force behind the creation and adoption of a code of conduct. We tried to lessen this risk by designing structured and semi-structured interviews, preparing unbiased questions that also encompass different aspects of codes of conduct. Again, more interviewees should be considered to further reduce this risk.

7.5 Conclusion

In this paper, we found that adopting codes of conduct in open source projects is an emerging phenomenon in order to deal with diversity issues and provide a safe and inclusive community. The phrasing of a code of conduct, enforcement mechanism used, scope and other properties might vary depending on the code of conduct and community. We obtained these insights through a combination of manual analysis of the codes themselves, as well as through interviews with creators and adopters of codes of conduct.

Although our study is the first step showing the role of codes of conduct in open source projects, we believe it opens up a variety of research opportunities, since it is one of the first mainstream solutions to deal with conflicts in a software project. Indeed, recently research on detection of emotions, politeness and sentiment in software engineering interactions has taken off (Ortu et al., 2015a; Tourani and Adams, 2016; Tourani et al., 2014). Except for Dullemond et al. (Dullemond et al., 2013), most of this work focuses on measuring the presence of some kind of conflict or negative feelings, with the aim of informing managers about these. Codes of conduct are a concrete tool to act on such information.

However, since codes of conduct are relatively young, more detailed empirical evaluation of their effectiveness and of best practices is required. More qualitative and quantitative studies should be done to provide recommendations and guidelines for the design and improvement of codes of conduct in this domain. Once codes of conduct will be more mature, more data will be available for quantitative studies, especially regarding enforcement and effect of the code of conduct in different stages and processes of the software development process.

CHAPTER 8 GENERAL DISCUSSION

In this thesis, we conducted a series of empirical studies to first identify the presence of affective metrics and next investigate their impact. To investigate the impact of affective metrics, we considered two major output parameters, quality in terms of the number of buggy commits and time in terms of the issue resolution time and code reviewing time. Quality is one of the most important concerns in software engineering, received huge attention in recent years, for instance bug prediction models have gained importance. Time and the performance of the process are other important factors contributing to software success. Finally, we explored one general solution adopted in many open source projects providing a healthier work environment regarding emotional and affective status of participants.

8.1 Detecting sentiment from software engineering artifacts and evaluating automatic sentiment analysis tool

We evaluated the adoption of an automatic sentiment analysis in mailing lists to identify positive or negative sentiments (happiness or distress) and their evolution. This study was backed with our previous study (Murgia et al., 2014), which revealed developers do express their emotions in issue reports towards different matters such as design choices, maintenance activity or colleagues. Moreover, mailings lists are common for discussion in distributed software engineering. Sentiment scores were computed for each email of Apache Software Foundation mailing lists, both developer and user mailing lists. In order to evaluate the adopted tool and also observe the evolution of sentiment across different months we did sampling in a special way described in section 4.4. Two raters separately scored the selected sample. Raters reached agreement of 76.62% while in the beginning they have less agreement and after discussions they obtained this percentage of agreement. To continue the rest of study we discarded disagreement cases.

However, the evaluation results, precision and recall, were low. We identified two problems cause these low performance and tried to conduct another experiment to measure them more accurately. Those problems, explained in detail in section 4.4 of chapter 3, were related to 1) the way of sampling 2) usage of technical keywords with polarities in terms of SentiStrength's dictionary meanings. We overcame these problems and conducted further experiment with SentiStrength explained in the following.

8.2 An Empirical Comparison of Different Sentiment Analysis and Aggregation Methods

Knowing the sentiment of each sentence, finally we need to find out the sentiment of the whole email composed of several sentences. In this section, we empirically compare the two major approaches for sentiment analysis, machine learning and lexical-based, on developer and user mailing lists. We again study the same two large, successful open source software projects: Ant and Tomcat. For machine learning methods, we chose state-of-the-art tool, Stanford Sentiment Analysis[1], while for lexical-based approaches we used the Sentistrength tool. To reach one score for each email, we evaluated different possible ways of aggregating scores of email sentences.

8.2.1 Experimental Setup

We randomly picked for each mailing list of each project 200 sample emails in contrast to chapter 4 which used a conditional sampling over most positive and most negative months. Moreover, this number of sample emails was adequate as it leads a confidence level of 95% and confidence interval of 5%. Providing the appropriate inputs for sentiment analysis from emails, we adopted the same approach described in section 4.4, i.e., automatic emails were filtered out and any non-natural language text inside the rest of emails were removed too. Then, Sentistrength and Stanford tool separately run on the extracted texts from emails.

Stanford Sentiment analysis tool scores based on 5 sentiment classes from "very negative" to "very positive" (very negative, negative, neutral, positive, very positive) for each sentence. SentiStrength based on its 4 configuration can return several values as sentiment score. By default, its reported score ranged in the following values:

- -1 (not positive) to -5 (extremely negative)
- 1 (not negative) to 5 (extremely positive)

While binary (positive/negative), trinary (positive/negative/neutral) and single scale (-4 to +4) are other configurations of SentiStrength. Therefore, both tools return a range of values, since our raters evaluate the polarity of each email and ignored the amplitude (because of simplicity), we need to map those ranges to sign, i.e., negative, positive, and neutral. Therefore, we discretized different possible scores in various ways. For instance, the -4 to +4 values returned by Sentistrength tool in single scale mode can be mapped to the combinations of positive, negative and neutral values depicted in Table 8.1.

[1]http://nlp.stanford.edu/sentiment/

Table 8.1 Sample of discretizations for Sentistrength returned scores in single scale mode

Negative	**Neutral**	**Positive**
[-4,-1]	0	[1,4]
[-4,-2]	[-1,0]	[1,4]
[-4,-3]	[-2,0]	[1,4]
-4	[-3,0]	[1,4]
[-4,-1]	[0,1]	[2,4]
[-4,-2]	[-1,1]	[2,4]
[-4,-3]	[-2,1]	[2,4]
-4	[-3,1]	[2,4]
[-4,-1]	[0,2]	[3,4]
[-4,-2]	[-1,2]	[3,4]
[-4,-3]	[-2,2]	[3,4]
-4	[-3,2]	[3,4]
[-4,-1]	[0,3]	4
[-4,-2]	[-1,3]	4
[-4,-3]	[-2,3]	4
-4	[-3,3]	4

Running the tools, we achieved one sentiment score for each sentence, however, we need the one sentiment score for each email. To this end, several aggregation methods can be adopted to combine sentiment scores of sentences: minimum, first quartile, mean, median, third quartile, and max.

To evaluate the results, like section section 4.4 of chapter 3, two separate raters read the emails and manually scored them while their scores were compared. First, there was an agreement for 75% of emails between raters. However, raters discussed about their discrepancies and for 95.34% of sample emails they reached equal scores.

Results

Max and Mean Methods performed better than other aggregation methods. For both Ant and Tomcat systems, Sentistrength appeared with better performance regarding the following discretizations: [-5,-2][-1,0][1,5] (negative, neutral and positive ranges), [-5,-1][0,1][2,5], [-5,-1]0[1,5] and so on as depicted in Table 8.2. However, we found that the Stanford tool shows higher performance for user emails with negative sentiment, which also observed for the Tomcat system. Mean seems the best aggregation method for Stanford toolTable 8.3. However, for Tomcat, binary Sentistrength with Max method has better performance considering positive sentiments emails for both user and developer mailing lists.

To conclude, we found out machine-learning based approaches applying Mean method show better performance for user emails having negative sentiment, while lexical approaches with Max method work better for emails, either developers or users, having positive sentiment. Our results have no impact on results achieved in chapter 4, as there we conducted our study to investigate the existence of sentiment in mailing lists and its evolution. However, we explained in chapter 4 the performance calculated for SentiStrength tool was not accurate enough, while through this additional study, we compared SentiStrength with Stanford and calculated its performance more precisely.

Table 8.2 Sentistrength tool best results for developer and user emails with positive and negative sentiments

Approach	**Metric**	**Discretization**	**Sentiment**	**Mailing-list Type**	**Fmeasure**	**System**
Sentistrength	MaxS	0,[-1,0],[0,1], [-2,0],[-2,1],[-3,0], [-3,1],[-1,1]	Positive	Developer	0.8	Ant
Sentistrength	MaxS	0,[-1,0],[-1,1] ,[0,1],[-1,2],[0,2] ,[-1,3],[0,3]	Negative	Developer	0.68	Ant
Sentistrength-Binary	Mean	0	Positive	User	0.72	Ant
SentiStrength	MaxS	0,[-1,0],[0,1] ,[-1,1],[0,2],[-1,2] ,[-1,3],[0,3]	Negative	User	0.48	Ant
Sentistrength-Binary	Mean Median	0	Positive	Developer	0.78	Tomcat
Sentistrength Sentistrength-Scale	Mean	0,[-1,0],[-2,0] ,[-3,0],[-4,0]	Negative	Developer	0.59	Tomcat
Sentistrength-trinary	Mean	0	Positive	User	0.71	Tomcat
SentiStrength	MaxS	0,[0,1],[-1,0] ,[-1,1],[-1,2],[0,2]	Negative	User	0.53	Tomcat

Table 8.3 Stanford tool best results for developers and users emails with positive and negative sentiments separately

Approach	**Metric**	**Discretization**	**Sentiment**	**Mailing-list Type**	**Fmeasure**	**System**
Stanford	Max	0	Positive	Developer	0.54	Ant
Stanford	Mean	[1,0]	Negative	Developer	0.51	Ant
Stanford	ThirdQ	[-1,0],[0,0]	Positive	User	0.43	Ant
Stanford	Mean	[-1,0]	Negative	User	0.63	Ant
Stanford	Max	0	Positive	Developer	0.62	Tomcat
Stanford	Mean	0	Negative	Developer	0.58	Tomcat
Stanford	Max	0	Positive	User	0.53	Tomcat
Stanford	Median Mean	0	Negative	User	0.78	Tomcat

8.3 Understanding positive and negative sentiments in OSS mailing lists

As explained before, two raters scored sample emails while obtained an agreement of 76.62%, therefore, we found out there are positive and negative sentiments in emails. We investigated those sample to figure out the meaning of positive and negative sentiment of them, i.e. in which cases they are positive, and when they are negative. We found out several categories for both emails with positive sentiment and with negative sentiment. Therefore, we obtained more insights about the meaning of positive and negative sentiments in emails so that we could compare user mailing list and developer mailing list regarding their sentiment in our selected OSS projects, i.e. Ant and Tomcat. Certainly, there is no evidence to support the claim that our results are generalizable to other OSS projects.

8.4 Explanatory Models for Just-In-Time Quality

We studied Just In Time (JIT) quality prediction models in chapter 4. While previous studies proposed this model using change metrics obtained from source code such as churn, our work was the first to link change (technical) data, reviewing discussions data and data of issue fixing discussions together to analyze the overall models. It is worth mentioning that recent work by Tim Menzies et al. (Menzies et al., 2016) shows the effort to resolve delayed issues are not greater than when issues were resolved immediately after introduction. In this chapter, we built the explanatory models to understand the relation between human discussions and bug proneness of a patch, which provides clues for understanding the causes of defects. For half of the studied projects, the precision and recall of the proposed models improve substantially.

The major contribution of chapter 4 is involving feelings of stakeholders in the proposed models through computing the sentiments of corresponding review comments and issue comments which is so unique in the domain of study. Our previous studies in chapter 3 and in (Murgia et al., 2014) showed the presence of affects in software artifacts, while by performing further study explained in section 8.2, we observed good performance for SentiStrength tool applying for developer and user mailing lists, i.e., texts used for communication in software development environment.

We have other observations like when the type of a commit changes from bug fix to no bug fix, the probability of defect-proneness increases. These observations need more investigations to find out to what extent they are general and also understand the reason behind them.

Our final results showed sentiment-related metrics also played role in quality explanation while featured in more than half of the models. However, more research is needed to fully

explain how they impact the quality.

8.5 Explanatory Models for Code Reviewing Time and Issue Resolution Time

In chapter 5, we studied issue resolution time and code reviewing time, investigating factors impacting these times. This is the first study that examine issue resolution and code reviewing time while contrasting and comparing their scale, proposing reasons for their differences.

Wide range of metrics spread across 10 different dimensions proposed with the rational behind them covering various aspects of issue resolution and code reviewing process, including metrics related to a code change, or human discussion related metrics. We specifically showed extra metrics rather than metrics related to code change improved the performance of the proposed models significantly. Our models for code reviewing time reach high performance up to 78% precision and recall, and our models for issue resolution time have an accuracy of up to 70%.

Although these explanatory models are not able to predict future issue resolution time or code reviewing time, they can explain existing metrics and provide key insights for obstacles and problems delaying issue resolution or code reviewing time. Moreover, such a good performance of models paves the way for building prediction models with reasonable performance.

8.6 Identifying affective-related metrics influencing quality and Time

In chapter 4, we considered feelings expressed through review discussions using sentiment analysis tools. However, in chapter 5, to analyze the factors influencing time of the code reviewing and issue resolution, we extended affect related metrics, i.e., politeness, emotion, and sentiment of comments computed using appropriate tools. As discussed in both of these studies, chapter 4 and chapter 5, integrating these metrics into other metrics to build explanatory models was unique. However, for both of them, a list of influential metrics were also computed so that we were able to show affective-related metrics are among influential metrics impacting quality and time. However, there were more impactful metrics from other dimension. Our work is the first empirical study that measures the impact of affective-related metrics on quality and time (time of code reviewing and issue resolution), in software engineering, using quantitative analysis.

8.7 Understanding the role and characteristics of codes of conduct in open source software projects

Our previous results showed the importance of affective-related metrics in open source projects. Recently, as one solution for providing more friendly and inclusive environment, codes of conduct are being adopted in open source communities. Therefore, we were motivated to find out why they appeared in open source communities, and how they can impact interactions of members and provide a healthier and more friendly environment.

We figured out the role of codes of conduct in open source projects and conducted qualitative study interviewing different famous creators of codes of conduct in open source projects to obtain more fundamental insight about them. So far, no study has been performed about codes of conduct in open source projects.

Codes of conduct may indicate that open source communities understood the risks of offensive behaviours for their projects and tried to prevent those threats by applying appropriate solution. In other words, open source communities realized that providing comfortable atmosphere for participants is one of the success keys for their project, so that participant can freely and regardless of their gender, religion, ethnicity, and so on can work together to succeed the project. We conjecture that codes of conduct consequently influence affects of members so that they feel better participating in safe and protective teams, however, certainly further studies needed to evaluate this assumption.

8.8 Overview of the obtained results and their impacts

In Table 8.4, insights of our obtained results and their impacts in software engineering context have been listed.

Table 8.4 Studies and their impacts

Concern	Results and Their Impact
Feasibility of affect detection from software collaborative artifacts	Our results in chapter 4 (motivated by (Murgia et al., 2014)) revealed the presence and evolution of affects in the collaborative software media like issue reports or mailing lists. This may provide solutions for software managers and practitioners to monitor their teams' environment regarding the affects of members and notice problems in proper time. Consequently, they can avoid serious problems by taking appropriate measurements.
Evaluating tools measuring affects in software engineering context	Results observed in chapter 4 and subsection 8.2.1 showed tools designed for sentiment analysis like SentiStrength or Stanford have reasonable performance in software engineering context in comparison with their performance in contexts they have been designed for them initially. However, for SentiStrength, we observed that tuning the tool before application (e.g. by removing technical terms) beside preparing the text (cleaning the text from non-human texts) lead to higher performance. This also may provide guides for practitioners showing them how to measure affects using software textual collaborative artifacts used in software engineering process. In addition, limitations and deficiencies of the tools have been discussed too.
Investigating the link between affective-related metrics and quality	Results in chapter 5 showed that sentiment-related metrics featured among the most influential metrics in defect prediction models. Based on our studied projects, healthier and happier software environment teams have less defect-prone commits. Our approach also help researchers how to investigate the impact of collaborative characteristics on software quality.
Investigating the link between affective-related metrics and development time	Results in chapter 6 revealed affective-related metrics appeared among the 10 most influential attributes on both models explaining code reviewing time and issue resolution time. While, more investigation in our studied projects showed that happier environment leads to faster code reviewing. We also proposed prior to selecting factors for use in explanatory models, one should first analyze the features of output parameters as we did for review and issue resolution time.
Examining one current solution applied in open source projects to improve affects of community	Codes of conduct has been adopted in open source projects as a solution to provide a healthier and more friendly environment. Performing quantitative and qualitative studies, we investigated open source codes of conduct to understand how this solution may lead to happier or more positive environment. Our study is the first study in software engineering domain presenting fundamental insights about codes of conduct in open source projects such as understanding their popularity, the requirements behind their design, their scope, enforcement, and limitations.

CHAPTER 9 CONCLUSION AND FUTURE WORK

On the one hand, the software engineering process as cognitive and collaborative process is being performed by human beings. On the other hand, affects (emotions, moods, and feelings) influence cognitive processing activities and the productivity of individuals. The affects of software stakeholders may impact software engineering process in different aspects.

As a preliminary step towards investigating our hypothesis, we analyzed the feasibility of automatically measuring affects including emotions and sentiments from software artifacts, since emotional awareness can help management to better understand various characteristics of their team environments, foreseeing some problems and obstacles. Then, we ran several empirical studies to examine the relation of affect-related metrics on quality of commits and the time taken for issue resolution and code reviewing.

Our results indicated strong link between affective-related metrics and two important variables, i.e., the quality of the product, and time spent for code reviewing and issue resolution. Finally, we studied codes of conduct, a solution adopted by open source projects to improve the affect so that participants feel more comfortable and friendly environment despite the existing diversity in open source communities.

9.1 Extracting affective metrics from software artifacts

Our first preliminary study (Murgia et al., 2014), towards evaluating the feasibility of a tool for automatic emotion mining from software artifacts, revealed that issue reports do carry emotions of developers during their discussions. This motivated us for our next study in chapter 4, investigating the presence and evolution of positive and negative sentiment in the email communication of users and developers of two large open source projects, Tomcat and Ant. We observed that developer and user mailing lists do contain sentiment, and we identified 6 categories of positive sentiment in emails, and 4 categories of negative sentiment. We also conducted additional experiments to study and compare two state-of-the-art sentiment analysis tools, one (Stanford) as representative for machine-learning-based tools and one (SentiStrength) as representative for lexical-based tools. SentiStrength shows better performance for developer mailing list (FMeasure higher than 0.68). Later, in chapter 5 and 6, we adopted SentiStrength to compute sentiment score of issue comments and review discussion comments. Also, for each comment, in addition to sentiment score, we determined emotion and computed politeness too using proper tools.

9.2 Investigating the role of affective-related metrics

In chapter 5 we defined human discussion metrics across various categories including the sentiment category consisting of comments sentiments to study the link between these metrics and the "Just-In-Time" defect prediction models. Our results indicated that human discussion metrics improve models and show strong linkage between them and buggy commits. Although, among human-discussion metrics, those related to the experience of people have stronger link with defect-proneness, sentiment related metrics play role in the models. According to sentiment-related metrics, higher sentiments indicating higher quality by decreasing impact on the bug-proneness commits.

In chapter 6, we focused on metrics that impact the time taken for code reviewing and issue resolution by considering a wider range of metrics. Those metrics were obtained from 3 domains: source code, review or issue discussions. We found the code reviewing time and issue resolution time are mostly independent variables, suggesting that different factors are influencing them. Our models reach high precision and recall, specially for code reviewing time. By computing influential metrics, we found that affective-related metrics appeared in the top 10 most powerful metrics. We observed that the more love(gratitude) expressed through comments, the lower the total time spent on code reviewing, while higher politeness is linked with longer code reviewing or issue resolution time.

9.3 Understanding the role of codes of conduct in open source projects

Finally, in chapter 7, we conducted an empirical study to obtain insights from codes of conduct in open source projects, since their purpose is to provide more friendly and welcoming environment to every participants. Affect detection from software artifacts, as discussed in this thesis, measures presence of affects and can be used to help managers to gain more emotional-awareness about their teams and working environment while codes of conduct can be used to act based on such awareness.

9.4 Hypothesis Revisited

Based on our results in chapter 5 and 6, we can accept part of our hypothesis subsection 1.3.3 as there is a strong relation between affects of participants in open source projects and software quality and time of their work. For quality, we considered buggy commits as a measure of quality, while it is possible to extend our study to other definitions of quality explained in section 2.2.5. Also, time of code reviewing and issue resolution as two major

activities were considered as dominant time.

Our results in chapter 7 revealed that open source communities, by applying codes of conduct, are trying to improve friendly and comfortable environment and consequently bring about more positive affects in community. However, due to the lack of enough data, we were not able to measure the affect improvement resulting of adopting codes of conduct in open source projects.

9.5 Future Work

9.5.1 Adopting more accurate affect measurement

In our empirical studies, we applied different tools for computing affects (including sentiment, politeness, emotion). These tools all have enough accuracy that does not affect the correctness of our conclusion, especially by applying aggregation. However, their accuracy might be increased adopting new techniques. For instance, one can upgrade machine-learning based tools like Stanford by feeding new training data, i.e., software engineering corpus of data with known affects. Certainly, we would like to adopt such more accurate tools to redo the empirical analysis to find out more precise findings.

In our empirical studies to investigate the impact of affects, issue and review comments were adopted as sources for extracting the affects of participants during their discussions. However, other communication tools like IRC chat messages or mailing lists can be added for finding members' affects. However, one essential challenge for using these repositories is providing the linkage between their data and commits as needed in our conducted empirical studies.

9.5.2 Individual vs. group study

Since software engineering is a discipline consisting of different stages like design, development, deployment, and maintenance, there are also specific roles such as programmers, systems analysts, and project managers. Distinguishing between these roles and investigating the affect-related metric of each group separately, can bring about new findings which is more precise. In this way, their interactions can be analyzed to find out what types of interactions can increase the positivity of working environment.

9.5.3 Feature-based sentiment analysis

Sentiment analysis can be run on the contents of software artifacts towards various features including specific patch, new release or new technology. In this way, finer granularity will be achieved for the obtained sentiments and more particular study and analysis can be done based on various features. Other affect metrics, emotions and politeness also can be computed based on specific features.

9.5.4 Affect evolution in critical periods of time

Affect related metrics can be computed in different periods of time based on specific events of the project like release time or particular milestones of the projects. One can first check the presence of affect evolution, and secondly, study the impact of affect evolution on different parameters of software process like its quality.

9.5.5 Empirical study on the relation between affect and quality defined from different perspective

Quality measurement used in our study in chapter 4 of this thesis was based on the defect density definition regarding commits, however, we can consider quality from different viewpoints like customer satisfaction as explained in section 2.2.5. For instance, customer rating information can be applied to measure software quality, then investigate how various metrics including affect-related metrics of developers may impact customer satisfaction. Quality also can be measured by amount of changes that each developer has submitted.

9.5.6 Can affects of developer reveal their loyalty towards organization?

In order to identify the risk of project members leaving a project, we conjecture that the affects of stakeholders in project communication media can be used as an indicator of their intent to leave. In particular, we conjecture that when stakeholders leave because of disinterest, conflicts or other issues, their negative affect (related to sadness) should be much more prevalent in their communications. Hence, by measuring the affects of developers, it seems possible to detect and predict symptoms of leaving developers. Similarly, the affect of new developers during their ramp-up process might be identified, making it possible to identify frustration or other growing pains.

9.5.7 Measuring the adoption of codes of conduct and their impact

It would be interesting to investigate the evolution of affect-related metrics in open source projects after adopting codes of conduct, in other words studying the impact of codes of conduct on affect-related metrics. This may also lead to some guidelines for improving codes of conduct.

REFERENCES

"Predicting Turnover of Employees from Measured Job Attitudes." *Organizational Behavior & Human Performance*, vol. 13, no. 2, pp. 233–243, Apr. 1975.

"Software engineering code of ethics and professional practice," available at http://www.acm.org/about/se-code.

"Ifac," available at http://www.ifac.org.

"Open source initiative," available at https://opensource.org/osd-annotated.

B. Adams and S. McIntosh, "Modern release engineering in a nutshell – why researchers should care," in *Leaders of Tomorrow: Future of Software Engineering, Proceedings of the 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Osaka, Japan, March 2016.

T. M. Amabile, S. G. Barsade, J. S. Mueller, and B. M. Staw, "Affect and Creativity at Work," *Administrative Science Quarterly*, vol. 50, no. 3, pp. 367–403, 2005.

S. Ambler, *"Agile modeling: effective practices for extreme programming and the unified process"*. John Wiley & Sons, Inc. New York, 2002.

L. An, F. Khomh, and B. Adams, "Supplementary bug fixes vs. re-opened bugs," in *Proceedings of the 14th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)*, Victoria, BC, Canada, September 2014, pp. 205–214.

A. Aue and M. Gamon, "Customizing sentiment classifiers to new domains: A case study," in *Proceedings of Recent Advances in Natural Language Processing (RANLP)*, Borovets, Bulgaria, 2005.

A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 712–721.

A. Bacchelli, M. Lanza, and R. Robbes, "Linking e-mails and source code artifacts," in *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 375–384.

A. Bacchelli, T. D. Sasso, M. D'Ambros, and M. Lanza, "Content classification of development emails," in *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, 2012, pp. 375–385.

C. Barron and A. Barron, *The Creativity Cure: How to Build Happiness with Your Own Two Hands*. Scribner, 2013.

V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Transactions on Software Engineering*, vol. 22, no. 10, pp. 751–761, Oct. 1996.

C. D. Batson, L. L. Shaw, and K. C. Oleson, *Differentiating Affect, Mood, and Emotion: Toward Functionally-based Conceptual Distinctions*. Sage, 1992.

G. Bavota and B. Russo, "Four eyes are better than two: On the impact of code reviews on software quality," in *Proceedings of 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. Los Alamitos, CA, USA: IEEE, 2015, pp. 81–90.

B. Bazelli, A. Hindle, and E. Stroulia, "On the personality traits of stackoverflow users," in *Proceedings of 2013 IEEE International Conference on Software Maintenance*, Eindhoven, The Netherlands, 2013, pp. 460–463.

C. Becker, R. Chitchyan, L. Duboc, S. Easterbrook, B. Penzenstadler, N. Seyff, and C. C. Venters, "Sustainability design and software: The karlskrona manifesto," in *Proceedings of the 37th International Conference on Software Engineering*, ser. ICSE '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 467–476.

N. Bettenburg and A. E. Hassan, "Studying the impact of social interactions on software quality," *Empirical Software Engineering*, vol. 18, no. 2, pp. 375–431, Apr. 2013.

N. Bettenburg, A. E. Hassan, B. Adams, and D. M. German, "Management of community contributions," *Empirical Software Engineering*, vol. 20, no. 1, pp. 252–289, Feb. 2015.

C. Bird, T. Menzies, and T. Zimmermann, *The Art and Science of Analyzing Software Data*. Elsevier Science, 2015.

C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu, "Fair and balanced?: Bias in bug-fix datasets," in *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ser. ESEC/FSE '09. New York, NY, USA: ACM, 2009, pp. 121–130.

C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. German, and P. Devanbu, "The promises and perils of mining git," in *Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories*, ser. MSR '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–10.

S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*, 1st ed. O'Reilly Media, Inc., 2009.

J. M. Bland and D. G. Altman, "Statistics notes: Transformations, means, and confidence intervals," *BMJ*, vol. 312, no. 7038, p. 1079, 1996.

B. W. Boehm and P. N. Papaccio, "Understanding and controlling software costs," *IEEE Transactions on Software Engineering*, vol. 14, no. 10, pp. 1462–1477, Oct. 1988.

B. W. Boehm, "A spiral model of software development and enhancement," *Computer*, vol. 21, no. 5, pp. 61–72, May 1988.

M. M. Bradley, P. J. Lang, M. M. Bradley, and P. J. Lang, "Affective norms for english words (anew): Instruction manual and affective ratings," 1999.

S. N. Brenner and E. A.Molander, "Is the ethics of business changing," *Harward Business Review*, vol. 55, no. 1, pp. 57–71, 1971.

J. Brodkin, "Linus Torvalds defends his right to shame Linux kernel developers," http://www.webcitation.org/6O2zErgzE, July 2013.

A. P. Brogan, "Codes of ethics: A handbook. edgar l. heermance," *The International Journal of Ethics*, vol. 36, no. 1, pp. 102–103, 1925.

G. Canfora, L. Cerulo, M. Cimitile, and M. D. Penta, "How changes affect software entropy: an empirical study," *Empirical Software Engineering*, vol. 19, no. 1, pp. 1–38, 2014.

T. Carrez, "Preventing craziness: a deep dive into openstack testing automation," Presentation at FOSDEM 2014, February 2014.

M. Cataldi, A. Ballatore, I. Tiddi, and M.-A. Aufaure, "Good location, terrible food: detecting feature sentiment in user-generated reviews." *Social Netw. Analys. Mining*, vol. 3, no. 4, pp. 1149–1163, 2013.

M. Cataldo, A. Mockus, J. A. Roberts, and J. D. Herbsleb, "Software dependencies, work dependencies, and their impact on failures," *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp. 864–878, Nov. 2009.

J. Cohen, "A coefficient of agreement for nominal scales," *Educational and psychological measurement*, vol. 20, no. 1, pp. 37–46, 1960.

S. Cohen and D. Bailey, "What makes teams work: Group effectiveness research from the shop floor to the executive suite," *Journal of Management*, vol. 23, no. 3, pp. 239–290, 1997.

J. A. Colazo and Y. Fang, "Following the sun: Temporal dispersion and performance in open source software project teams," *Journal of the Association for Information Systems*, vol. 11, no. 11, 2010.

R. Colomo-Palacios, A. Hernández-López, Á. García-Crespo, and P. Soto-Acosta, *A Study of Emotions in Requirements Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 1–7.

P. Crosby, *Quality is free: the art of making quality certain*, ser. Mentor executive library. New American Library, 1980.

D. A. da Costa, S. McIntosh, U. Kulesza, and A. E. Hassan, "The Impact of Switching to a Rapid Release Cycle on Integration Delay of Addressed Issues: An Empirical Study of the Mozilla Firefox Project," in *Proceedings of the International Conference on Mining Software Repositories (MSR)*, Austin, Texas, 2016, pp. 374–385.

L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in github: Transparency and collaboration in an open software repository," in *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, ser. CSCW '12. New York, NY, USA: ACM, 2012, pp. 1277–1286.

M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: A benchmark and an extensive comparison," *Empirical Software Engineering*, vol. 17, no. 4-5, pp. 531–577, Aug. 2012.

C. Danescu-Niculescu-Mizil, M. Sudhof, D. Jurafsky, J. Leskovec, and C. Potts, "A computational approach to politeness with application to social factors," *Computing Research Repository*, vol. abs/1306.6078, 2013.

S. L. Daniel, R. Agarwal, and K. J. Stewart, "The effects of diversity in global, distributed collectives: A study of open source project success," *Information Systems Research*, vol. 24, no. 2, pp. 312–333, 2013.

S. R. Das and M. Y. Chen, "Yahoo! for amazon: Sentiment extraction from small talk on the web," *Management Science*, vol. 53, no. 9, pp. 1375–1388, 2007.

S. C. de Barros Sampaio, E. A. Barros, G. S. de Aquino Junior, M. J. C. e Silva, and S. R. de Lemos Meira, "A review of productivity factors and strategies on software development," in *Proceedings of the Fifth International Conference on Software Engineering Advances, ICSEA*, Nice, France, 2010, pp. 196–204.

M. De Choudhury and S. Counts, "Understanding affect in the workplace via social media," in *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, ser. CSCW '13. New York, NY, USA: ACM, 2013, pp. 303–316.

K. Dullemond, B. v. Gameren, M.-A. Storey, and A. v. Deursen, "Fixing the out of sight out of mind; problem: One year of mood-based microblogging in a distributed software team," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, ser. MSR '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 267–276.

B. Efron and R. Tibshirani, "Cross-validation and the bootstrap: Estimating the error rate of a prediction rule," Division of Biostatistics, Stanford University, Stanford, California, Tech. Rep., May 1995.

H. A. Elfenbein and N. Ambady, "On the Universality and Cultural Specificity of Emotion Recognition: A meta-analysis," *Psychological Bulletin*, vol. 128, pp. 203–235, 2002.

M. E. Fagan, "Design and code inspections to reduce errors in program development," *IBM Systems Journal*, vol. 38, no. 2-3, pp. 258–287, Jun. 1999.

R. Feldman, "Techniques and applications for sentiment analysis," *Commun. ACM*, vol. 56, no. 4, pp. 82–89, Apr. 2013.

J. L. Fleiss, "Measuring nominal scale agreement among many raters." *Psychological bulletin*, vol. 76, no. 5, p. 378, 1971.

J. Fox and S. Weisberg, *An R Companion to Applied Regression*. SAGE Publications, 2010.

B. L. Fredrickson, "The role of positive emotions in positive psychology: The broaden-and-build theory of positive emotions." *American psychologist*, vol. 56, no. 3, p. 218, 2001.

T. Fukushima, Y. Kamei, S. McIntosh, K. Yamashita, and N. Ubayashi, "An empirical study of just-in-time defect prediction using cross-project models," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 172–181.

D. Galin, *Software Quality Assurance: From Theory to Implementation*. Pearson, June 2003.

E. Giger, M. Pinzger, and H. Gall, "Predicting the fix time of bugs," in *Proceedings of the 2Nd International Workshop on Recommendation Systems for Software Engineering*, ser. RSSE '10. New York, NY, USA: ACM, 2010, pp. 52–56.

J. M. Gonzalez-Barahona, G. Robles, and D. Izquierdo-Cortazar, "The metricsgrimoire database collection," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, ser. MSR '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 478–481.

D. Graziotin, X. Wang, and P. Abrahamsson, "Happy software developers solve problems better: psychological measurements in empirical software engineering," *PeerJ*, p. e289, 3 2014.

D. Graziotin, X. Wang, and P. Abrahamsson, "Do feelings matter? on the correlation of affects and the self-assessed productivity in software engineering," *Journal of Software: Evolution and Process*, vol. 27, no. 7, pp. 467–487, 2015.

H. P. Grice, "Logic and conversation," in *Syntax and Semantics: Vol. 3: Speech Acts*, P. Cole and J. L. Morgan, Eds. New York: Academic Press, 1975.

J. Guaspari, *I Know It When I See It*. AMACOM Books, 2004.

C. Gutwin, R. Penner, and K. Schneider, "Group awareness in distributed software development," in *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, ser. CSCW '04. New York, NY, USA: ACM, 2004, pp. 72–81.

E. Guzman and B. Bruegge, "Towards emotional awareness in software development teams," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2013. New York, NY, USA: ACM, 2013, pp. 671–674.

E. Guzman, D. Azócar, and Y. Li, "Sentiment analysis of commit comments in github: An empirical study," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 352–355.

A. Guzzi, A. Bacchelli, M. Lanza, M. Pinzger, and A. v. Deursen, "Communication in open source software development mailing lists," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, ser. MSR '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 277–286.

T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Transactions on Software Engineering*, vol. 31, no. 10, pp. 897–910, Oct. 2005.

A. E. Hassan, "Predicting faults using the complexity of code changes," in *Proceedings of the 31st International Conference on Software Engineering*, ser. ICSE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 78–88.

A. E. Hassan and T. Xie, "Software intelligence: The future of mining software engineering data," in *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, ser. FoSER '10. New York, NY, USA: ACM, 2010, pp. 161–166.

S. Henry and D. Kafura, "Software structure metrics based on information flow," *IEEE Transactions on Software Engineering*, vol. 7, no. 5, pp. 510–518, Sep. 1981.

I. Herraiz, D. M. German, J. M. Gonzalez-Barahona, and G. Robles, "Towards a simplification of the bug report form in eclipse," in *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, ser. MSR '08. New York, NY, USA: ACM, 2008, pp. 145–148.

M. Hu and B. Liu, "Mining and summarizing customer reviews," in *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '04. New York, NY, USA: ACM, 2004, pp. 168–177.

Y. Jiang, B. Adams, and D. M. German, "Will my patch make it? and how fast?: Case study on the linux kernel," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, ser. MSR '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 101–110.

R. Jongeling, S. Datta, and A. Serebrenik, "Choosing your weapons: On sentiment analysis tools for software engineering research," in *Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, ser. ICSME '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 531–535.

F. Jurado and P. Rodriguez, "Sentiment analysis in monitoring software development processes," *Journal of Systems and Software*, vol. 104, no. C, pp. 82–89, Jun. 2015.

J. Juran and A. Godfrey, *Juran's quality handbook*, ser. Juran's quality handbook, 5e. McGraw Hill, 1999.

R. Kabacoff, *R in Action: Data Analysis and Graphics With R*. Manning Pubn, 2013.

E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining github," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 92–101.

Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, "A large-scale empirical study of just-in-time quality assurance," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 757–773, 2013.

Y. Kamei, A. Monden, S. Matsumoto, T. Kakimoto, and K. ichi Matsumoto, "The effects of over and under sampling on fault-prone module detection." in *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement (ESEM2007)*. ACM / IEEE Computer Society, 2007, pp. 196–204.

Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, "A large-scale empirical study of just-in-time quality assurance." *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 757–773, 2013.

S. Kan, *Metrics and Models in Software Quality Engineering*. Addison-Wesley, 2003.

I. A. Khan, W.-P. Brinkman, and R. M. Hierons, "Do moods affect programmers debug performance?" *Cognition, Technology and Work*, vol. 13, no. 4, pp. 245–258, Nov. 2011.

S. Kim and E. J. Whitehead, Jr., "How long did it take to fix bugs?" in *Proceedings of the 2006 International Workshop on Mining Software Repositories*, ser. MSR '06. New York, NY, USA: ACM, 2006, pp. 173–174.

S. Kim, E. J. Whitehead, Jr., and Y. Zhang, "Classifying software changes: Clean or buggy?" *IEEE Transactions on Software Engineering*, vol. 34, no. 2, pp. 181–196, Mar. 2008.

B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering - a systematic literature review," *Information and Software Technology*, vol. 51, no. 1, pp. 7–15, Jan. 2009.

O. Kononenko, O. Baysal, L. Guerrouj, Y. Cao, and M. W. Godfrey, "Investigating code review quality: Do people and participation matter?" in *Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, ser. ICSME '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 111–120.

O. Kononenko, O. Baysal, and M. W. Godfrey, "Code review quality: How developers see it," in *Proceedings of the 38th ACM/IEEE International Conference on Software Engineering*, ser. ICSE '16. Austin, TX, USA: ACM, 2016.

O. Kucuktunc, B. B. Cambazoglu, I. Weber, and H. Ferhatosmanoglu, "A large-scale sentiment analysis for yahoo! answers," in *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, ser. WSDM '12. New York, NY, USA: ACM, 2012, pp. 633–642.

A. Landowska, "Affect-awareness Framework for Intelligent Tutoring Systems." in *Proceedings of 6th International Conference on Human System Interaction.* Sopot, Poland: IEEE Computer Society, 2013.

T. Lee, J. Nam, D. Han, S. Kim, and H. P. In, "Micro interaction metrics for defect prediction," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11. New York, NY, USA: ACM, 2011, pp. 311–321.

J. Leiman, "Quantitative measurement of emotion," Morpace Inc., Tech. Rep., April 2013.

M. Lerner, "Software maintenance crisis resolution," *The New IEEE standard, IEEE Software Development*, pp. 65–72, August 1994.

S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, Jul. 2008.

B. Liu, *Sentiment Analysis and Opinion Mining*, ser. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2012.

B. Liu, M. Hu, and J. Cheng, "Opinion observer: Analyzing and comparing opinions on the web," in *Proceedings of the 14th International Conference on World Wide Web*, ser. WWW '05. New York, NY, USA: ACM, 2005, pp. 342–351.

C. L. Mallows, "Some comments on Cp," *Technometrics*, vol. 15, no. 4, pp. 661–675, 1973.

D. R. Matsumoto, *The Cambridge Dictionary of Psychology.* New York: Cambridge University Press, 2009.

T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 308–320, Jul. 1976.

S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 192–201.

I. Md Rakibul and F. Z. Minhaz, "Towards understanding and exploiting developers' emotional variations in software engineering," in *Proceedings of 2016 IEEE 14th International Conference on Software Engineering Research, Management and Applications (SERA).* Los Alamitos, CA, USA: IEEE Computer Society, 2016, pp. 185–192.

T. Menzies, W. Nichols, F. Shull, and L. Layman, "Are delayed issues harder to resolve? revisiting cost-to-fix of defects throughout the lifecycle," *Computing Research Rrepositry*, vol. abs/1609.04886, 2016.

T. Miller, S. Pedell, A. A. Lopez-Lorca, A. Mendoza, L. Sterling, and A. Keirnan, "Emotion-led modelling for people-oriented requirements engineering," *Journal of Systems and Software*, vol. 105, no. C, pp. 54–71, Jul. 2015.

N. Mishra and C. K. Jha, "Article: Classification of opinion mining techniques," *International Journal of Computer Applications*, vol. 56, no. 13, pp. 1–6, October 2012, published by Foundation of Computer Science, New York, USA.

I. Mistrìk, J. Grundy, A. Hoek, and J. Whitehead, Eds., *Collaborative Software Engineering*. Springer, 2010.

A. Mockus and D. M. Weiss, "Predicting risk of software changes," *Bell Labs Technical Journal*, vol. 5, pp. 169–180, 2000.

A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and mozilla," *ACM Transactions on Software Engineering and Methodology*, vol. 11, no. 3, pp. 309–346, Jul. 2002.

I. Moura, G. Pinto, F. Ebert, and F. Castor, "Mining energy-aware commits," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, ser. MSR '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 56–67.

S. C. Müller and T. Fritz, "Using (bio)metrics to predict code quality online," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. New York, NY, USA: ACM, 2016, pp. 452–463.

S. C. Müller and T. Fritz, "Stuck and frustrated or in flow and happy: Sensing developers' emotions and progress," in *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ser. ICSE '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 688–699.

J. C. Munson and T. M. Khoshgoftaar, "The detection of fault-prone programs." *IEEE Transactions on Software Engineering*, vol. 18, no. 5, pp. 423–433, 1992.

A. Murgia, P. Tourani, B. Adams, and M. Ortu, "Do developers feel emotions? an exploratory analysis of emotions in software artifacts," in *Proceedings of the 11th IEEE Working Conference on Mining Software Repositories (MSR)*, Hyderabad, India, May 2014.

N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *Proceedings of the 27th International Conference on Software Engineering*, ser. ICSE '05. New York, NY, USA: ACM, 2005, pp. 284–292.

N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in *Proceedings of the 28th International Conference on Software Engineering*, ser. ICSE '06. New York, NY, USA: ACM, 2006, pp. 452–461.

M. Ortu, B. Adams, G. Destefanis, P. Tourani, M. Marchesi, and R. Tonelli, "Are bullies more productive? empirical study of affectiveness vs. issue fixing time," in *Proceedings of the 12th IEEE Working Conference on Mining Software Repositories (MSR)*, Florence, Italy, May 2015.

M. Ortu, G. Destefanis, M. Kassab, S. Counsell, M. Marchesi, and R. Tonelli, "Would you mind fixing this issue? an empirical analysis of politeness and attractiveness in software developed using agile boards," in *XP2015, Helnsiki*. Springer, 2015, p. in press.

C. Osgood, G. Suci, and P. Tenenbaum, *The Measurement of meaning*. Urbana:: University of Illinois Press, 1957.

A. J. Oswald, E. Proto, and D. Sgroi, "Happiness and productivity," Bonn, IZA discussion papers 4645, 2009.

A. Pak and P. Paroubek, "Twitter as a corpus for sentiment analysis and opinion mining," in *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, N. C. C. Chair), K. Choukri, B. Maegaard, J. Mariani, J. Odijk, S. Piperidis, M. Rosner, and D. Tapias, Eds. Valletta, Malta: European Language Resources Association (ELRA), may 2010.

B. Pang and L. Lee, "Opinion Mining and Sentiment Analysis," *Foundations and Trends in Information Retrieval*, vol. 2, no. 1-2, pp. 1–135, Jan. 2008.

S. Rao, *Happiness At Work*. McGraw-Hill Education (India) Pvt Limited, 2010.

A. Rashid, K. Moore, C. May-Chahal, and R. Chitchyan, "Managing emergent ethical concerns for software engineering in society," in *Proceedings of 37th IEEE/ACM International Conference on Software Engineering, ICSE*, vol. 2, Florence, Italy, 2015, pp. 523–526.

E. S. Raymond, *The Cathedral and the Bazaar*, 1st ed., T. O'Reilly, Ed. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 1999.

B. L. Renee Bazile-Jones, "Diversity in the workplace: Why we should care," *CMA - the Management Accounting Magazine*, vol. 70, no. 5, pp. 9–12, 1996.

Z. Rezaee, R. C. Elmore, and J. Z. Szendi, "Ethical behavior in higher educational institutions: The role of the code of conduct," *Journal of Business Ethics*, vol. 30, no. 2, pp. 171–183, 2001.

P. C. Rigby and C. Bird, "Convergent contemporary software peer review practices," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2013. New York, NY, USA: ACM, 2013, pp. 202–212.

P. C. Rigby and A. E. Hassan, "What can oss mailing lists tell us? a preliminary psychometric text analysis of the apache developer mailing list," in *Proceedings of the Fourth International Workshop on Mining Software Repositories*, ser. MSR '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 23–.

P. C. Rigby, D. M. German, and M.-A. Storey, "Open source software peer review practices: A case study of the apache server," in *Proceedings of the 30th International Conference on Software Engineering*, ser. ICSE '08. New York, NY, USA: ACM, 2008, pp. 541–550.

I. Robertson and C. Cooper, *Wellbeing: Productivity and Happiness at Work*. Palgrave Macmillan, 2011.

J. A. Russell, A. Weiss, and G. A. Mendelsohn, "Affect Grid: A Single-Item Scale of Pleasure and Arousal," *Journal of Personality and Social Psychology*, vol. 57, no. 3, pp. 493–502, Sep. 1989.

J. Russell, "A circumplex model of affect," *Journal of personality and social psychology*, vol. 39, no. 6, pp. 1161–1178, 1980.

V. Sehgal and C. Song, "Sops: Stock prediction using web sentiment," in *Proceedings of the Seventh IEEE International Conference on Data Mining Workshops*, ser. ICDMW '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 21–26.

W. R. Shadish, T. D. Cook, and D. T. Campbell, *Experimental and Quasi-Experimental Designs for Generalized Causal Inference*, 2nd ed. Houghton Mifflin, 2001.

E. Shihab, A. E. Hassan, B. Adams, and Z. M. Jiang, "An industrial study on the risk of software changes," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, ser. FSE '12. New York, NY, USA: ACM, 2012, pp. 62:1–62:11.

E. Shihab, Y. Kamei, B. Adams, and A. E. Hassan, "Is lines of code a good measure of effort in effort-aware models?" *Information and Software Technology*, vol. 55, no. 11, pp. 1981–1993, Nov. 2013.

J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" in *Proceedings of the 2005 International Workshop on Mining Software Repositories*, ser. MSR '05. New York, NY, USA: ACM, 2005, pp. 1–5.

R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Stroudsburg, PA: Association for Computational Linguistics, October 2013, pp. 1631–1642.

M. Taboada, J. Brooke, M. Tofiloski, K. Voll, and M. Stede, "Lexicon-based methods for sentiment analysis," *Computational Linguistics*, vol. 37, no. 2, pp. 267–307, Jun. 2011.

C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Automated parameter optimization of classification techniques for defect prediction models," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. New York, NY, USA: ACM, 2016, pp. 321–332.

M. Thelwall, "Heart and soul: Sentiment strength detection in the social web with sentistrength," http://sentistrength.wlv.ac.uk/documentation/SentiStrengthChapter.pdf.

M. Thelwall, K. Buckley, G. Paltoglou, D. Cai, and A. Kappas, "Sentiment in short strength detection informal text," *Journal of the Association for Information Science and Technology*, vol. 61, no. 12, pp. 2544–2558, Dec. 2010.

Y. Tian, J. Lawall, and D. Lo, "Identifying linux bug fixing patches," in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE '12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 386–396.

P. Tourani and B. Adams, "The impact of human discussions on just-in-time quality assurance," in *Proceedings of the 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Osaka, Japan, March 2016.

P. Tourani and B. Adams, "On issue resolution and review time-empirical study on 10 openstack projects," *Empirical Software Engineering*, 2017, submitted.

P. Tourani, Y. Jiang, and B. Adams, "Monitoring sentiment in open source mailing lists -– exploratory study on the apache ecosystem," in *Proceedings of the 2014 Conference of the*

Center for Advanced Studies on Collaborative Research (CASCON), Toronto, ON, Canada, November 2014, pp. 34–44.

P. Tourani, B. Adams, and A. Serebrenik, "Code of conduct in open source projects," in *Proceedings of the 24rd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Klagenfurt, Austria, February 2017.

P. D. Turney, "Thumbs up or thumbs down?: Semantic orientation applied to unsupervised classification of reviews," in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ser. ACL '02. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, pp. 417–424.

B. Vasilescu, V. Filkov, and A. Serebrenik, "Perceptions of diversity on GitHub: A user survey," in *8th International Workshop on Cooperative and Human Aspects of Software Engineering*, ser. CHASE. IEEE, 2015, pp. 50–56.

B. Vasilescu, D. Posnett, B. Ray, M. G. van den Brand, A. Serebrenik, P. Devanbu, and V. Filkov, "Gender and tenure diversity in github teams," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, ser. CHI '15. New York, NY, USA: ACM, 2015, pp. 3789–3798.

B. Vasilescu, A. Serebrenik, and V. Filkov, "A data set for social diversity studies of github teams," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, ser. MSR '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 514–517.

S. Wagner and M. Ruhe, "A systematic review of productivity factors in software development," State Key Laboratory of Computer Science, Institute of Software, Tech. Rep., 2008.

S. Wagner, T. U. München, and M. Ruhe, "A systematic review of productivity factors in software development," xx, Tech. Rep., 2010.

C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?" in *Proceedings of the Fourth International Workshop on Mining Software Repositories*, ser. MSR '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 1–.

H. M. Weiss and R. Cropanzano, "Affective events theory: A theoretical discussion of the structure, causes and consequences of affective experiences at work." *Staw B. M., Cummings L. L. (eds.), Research in Organizational Behavior*, vol. 18, pp. 1–74, 1996.

B. J. White and B. R. Montgomery, "Corporate codes of conduct," *Scholarly Journals*, vol. 23, no. 4, pp. 80–87, 1980.

C. Wohlin and M. Ahlgren, "Soft factors and their impact on time to market." *Software Quality Journal*, vol. 4, no. 3, pp. 189–205, 1995.

C. Wohlin and M. Ahlgren, "Soft factors and their impact on time to market." *Software Quality Journal*, vol. 4, no. 3, pp. 189–205, 1995.

T. Wolf, A. Schroter, D. Damian, and T. Nguyen, "Predicting build failures using social network analysis on developer communication," in *Proceedings of the 31st International Conference on Software Engineering*, ser. ICSE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–11.

T. Zimmermann, "Mining workspace updates in cvs," in *Proceedings of the Fourth International Workshop on Mining Software Repositories*, ser. MSR '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 11–.

T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, ser. PROMISE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 9–.