Ph.D. in Electronic and Computer Engineering
Dept. of Electrical and Electronic Engineering
University of Cagliari

Mining Software Repositories: Measuring Effectiveness and Affectiveness in Software Systems.

Marco Ortu

Advisors: Michele Marchesi and Roberto Tonelli
Curriculum: ING-INF/05 Informatica

Cycle XXVII
2013 - 2014

Ph.D. in Electronic and Computer Engineering
Dept. of Electrical and Electronic Engineering
University of Cagliari

Mining Software Repositories: Measuring Effectiveness and Affectiveness in Software Systems.

Marco Ortu

Advisors: Michele Marchesi and Roberto Tonelli
Curriculum: ING-INF/05 Informatica

Cycle XXIV
2013 - 2014

Dedicated to my wife.

In memory of Giulio.

Contents

List of Figures

*

List of Tables

*

Chapter 1

Introduction

Software Engineering field has many goals, among them we can certainly deal with monitoring and controlling the development process in order to meet the business requirements of the released software artifact. Software engineers need to have empirical evidence that the development process and the overall quality of software artifacts is converging to the required features. Improving the development process's *Effectiveness* leads to higher productivity, meaning shorter time to market, but understanding or even measuring the software development process is an hard challenge. Modern software is the result of a complex process involving many stakeholders such as product owners, quality assurance teams, project manager and, above all, developers. All these stakeholders use complex software systems for managing development process, issue tracking, code versioning, release scheduling and many other aspect concerning software development.

In Open Source System the situation is complicated by the complex structures of open source communities, often spread around the globe, with different time shifts, cultures, languages and environments. Beside their complexity, Open Source Communities provide valuable empirical data that can help researchers in their work. Source code repositories, issue tracking systems, mailing list etc., represent an immense golden mine containing valuable data that can help researchers understanding the process behind the manufacturing of a software artifact. Tools for project management and issues/bugs tracking are becoming useful for governing the development process of Open Source software. Such tools simplify the communications process among developers and ensure the scalability of a project. The more information developers are able to exchange, the clearer are the goals, and the higher is the number of developers keen on joining and actively collaborating on a project.

By analyzing data stored in such systems, researchers are able to study and address questions such as: Which are the factors able to impact the software productivity? Is it possible to improve software productivity shortening the time to market?.

The present work addresses two major aspect of software development process: *Effectiveness* and *Affectiveness*. By analyzing data stored in project management and in issue tracking system of Open Source Communities, we measured the *Effectiveness* as the time required to resolve an issue and analyzed factors able to impact it.

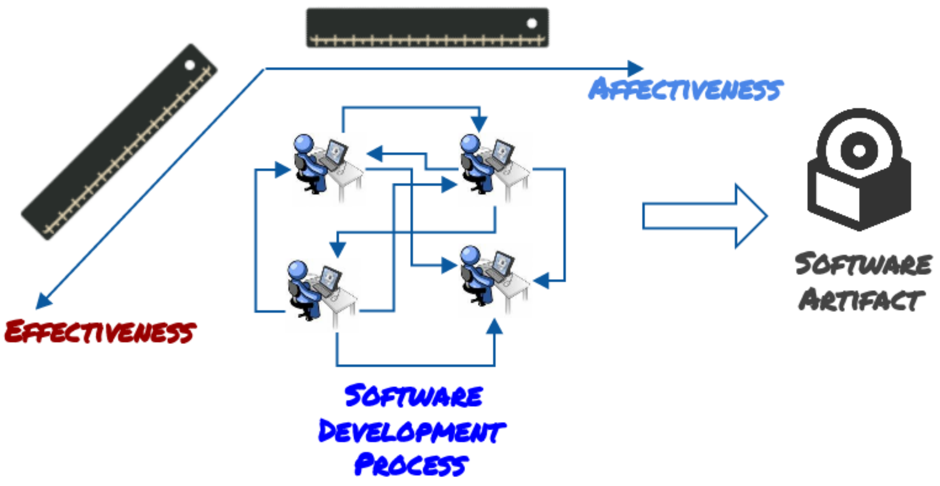


Figure 1.1: Measuring Software

Figure 1.1 depict the main goal of this work. In the first part of this thesis, we investigate, to which extends, the time required to resolve an issue can be influenced by the maintenance type and by the development team that solved the issue, showing that the type of maintenance plays a major role. In the second part, we first introduce the concept of *Affectiveness* in software engineering, then we provide a set of tools for measuring *Affectiveness* expressed by developers and finally we analyze the impact of *Affectiveness* on the time required to fix an issue, showing that it plays a major role.

1.1 Thesis Overview

This thesis is organized as follows: in Chapter 2 we present the main concepts that are extensively used in the rest of this thesis.

Chapter 3 presents a study of the *Effectivenss* of the software development process by analyzing the Fixing Time of the maintenance activities performed on software artifacts. This study shows that the Fixing Time of a maintenance activity is influenced by the type of the maintenance activity. Then this chapter presents a study of the *Effectiveness* of developers working teams in Open Source Software Systems showing that developers in OSS systems are organized in communities and that these communities have different *Effectiveness* as measured by the average Fixing Time and that *Effectiveness* is independent by the community size.

Chapter 4 presents a novel point of view of analyzing the software development process: the *Affectiveness* expressed by developers. *Affectivenss* is a general term for those aspects able to represent the “’emotional of a person”. *Affectivenss* represents the second dimension of software measure analyzed in this thesis and we measure it by measuring *Emotion*, *Sentiment* and *Politeness* expressed by developers in text written during development process. This study first analyses the feasibility of detecting *Affectiveness* in software showing that although developers more often express neutral

Chapter 5 presents a literature review of Affectiveness and Effectiveness as treated in this thesis.

Chapter 6 finally draws the conclusions of this thesis highliting results and contributions and presenting futures works.

Chapter 2

Background

2.1 Mining Software Repositories

2.1.1 Software Repositories

Jira

An Issue tracking systems (ITS) is a golden mine for MSR research activity. ITS store project's development activities related to project's issues like bugs, tasks, enhancements and feature requests. ITS records the history of these reports providing information related to their status (e.g. opened, closed, fixed), severity, priority and much more. As far as the authors know, there is not other dataset publicly available which offers a so high number of developers comments. To achieve such a big number or artifacts we have selected all projects hosted by four well know open source communities: Apache Software Foundation, JBoss, Spring and Codehaus communities. These big open source communities are indeed rich of issue reports with relevant information about issues' resolution discussion. We experimented the data mining on Jira since it offers more valuable data than traditional ITS such as Bugzilla. Indeed, it records more development information and supports explicit cross-linking from different repositories (e.g. SCM, ITS). This open new research opportunities based on more reliable dataset.

Issues in Jira are classified in several categories such as bugs, improvements, feature requests or tasks.

The database schema is shown in Fig. 2.1.
Our database has four main tables:

Issue Report. It stores the information extracted from the issue reports.

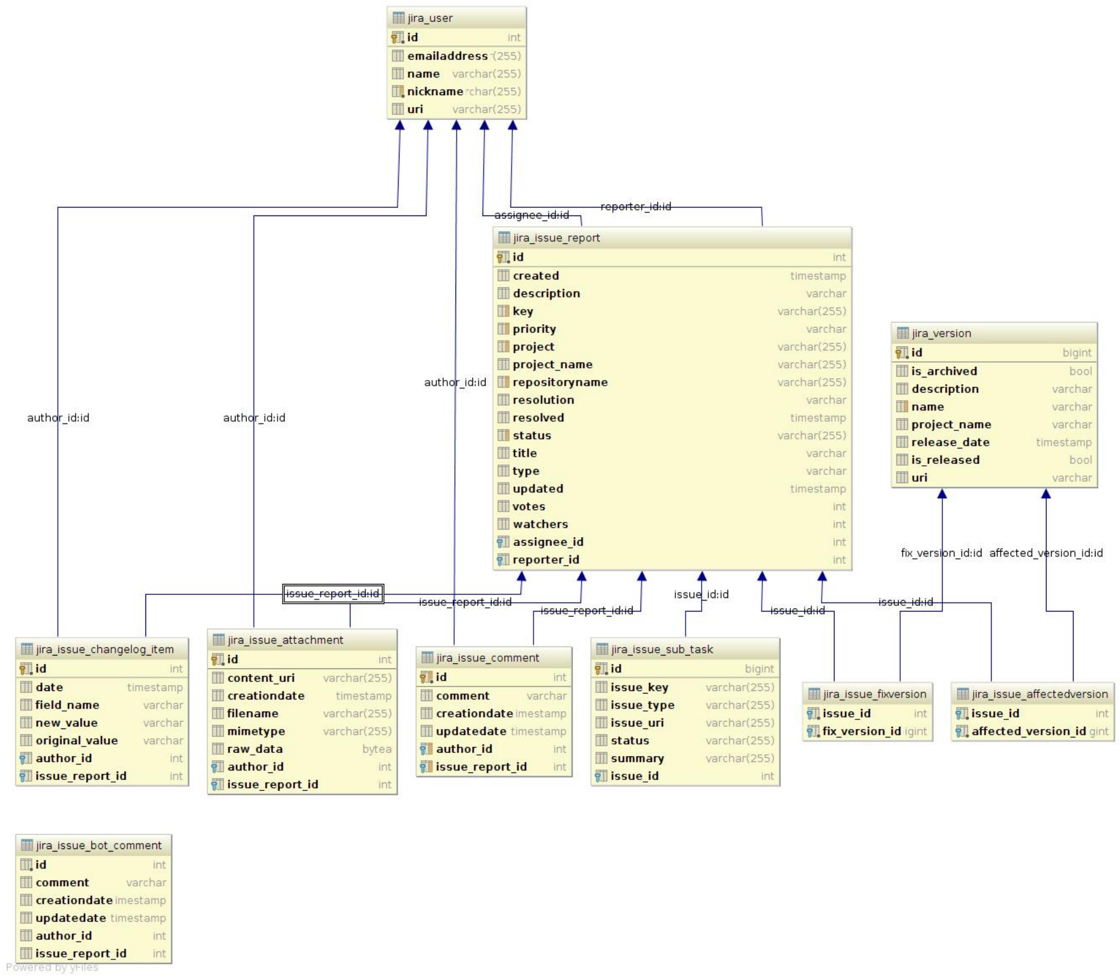


Figure 2.1: Data base schema

Issue Comment. It represents all the comments posted by users and developers in a Jira issue report. Is is associated to issue report table.

Jira User. It records jira user information.

Version.

Issue Affected Versions. It represents the software versions affected by an issue.

*Issue Fixed Versions*It represents the software versions that fixed an issue.

Issue Attachment It contains all files attached on an issue report.

Issue Change Log History It stores all the changes made on an issue, i.e., changing the resolution, changing the priority etc.

The dataset contains issues from 2002 to 2013 and was collected during the research of this thesis. I contains about 700k issue reports along with about 2M comments and about 1k open source projects.

GitHub

GitHub is a web-based code repository service providing a collaborative software development environment and a social network for developers repository. We use this repository since it host many open source projects, it is documented[1] and it is publicly available for replication studies [48]. In Figure 2.2, we report the portion of interest of dataset schema. Issue reports are characterized by two elements:

- Events. Events have *action* and *action specific* fields used to describe respectively the type of event (e.g. the time the issue was closed or the time a pull request pointed by the issue was merged).
- Label. Users may add a label to an issue report *to signify priority, category, or any other information that you and your fellow maintainers find useful.* From labels like "bug", "feature" it is possible to infer the type of maintenance performed (e.g.; bug fixing, feature introduction). We assume that these labels are accurate since they are *freely* provided by the developer who was aware of the activity performed. This labels are not attached with default values (like priority normal in Bugzilla) but are spontaneously added by developers whenever they want to add a relevant information for the issue handling.

2.2 Distribution Functions

In this section, we present the lognormal and Weibull distribution functions. Such distributions are suited to model sample data presenting leptokurtic behavior (a "fat tail" distribution) and were already used for modeling software metric distributions [25, 26].

Equations 2.1 and 2.2 describe the mathematical function of lognormal and Weibull distributions respectively. Here, the variable x represents the issue resolution time of a particular maintenance activity. Using these functions we can

[1] http://ghtorrent.org/relational.html

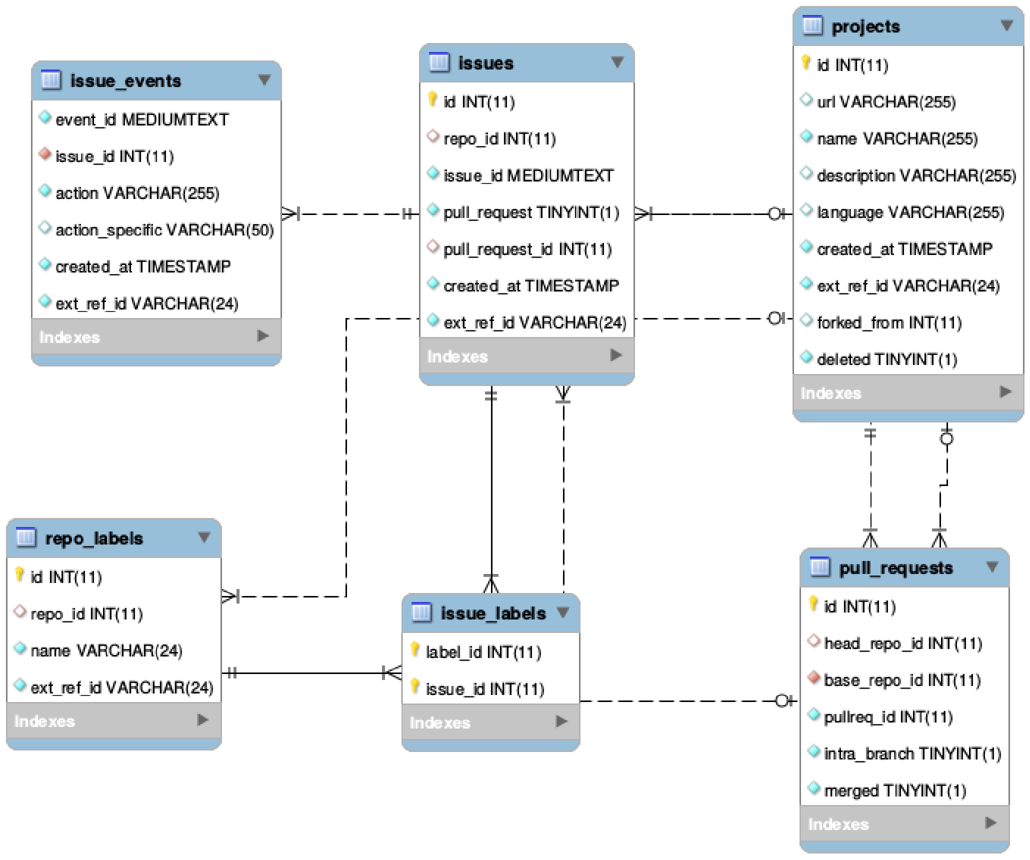


Figure 2.2: Github dataset schema.

model the distribution of the issue resolution time for any maintenance type independently.

In the rest of paragraph we briefly discuss how these distributions might be linked to the issue resolution time generation.

$$F(x) = \frac{1}{\sqrt{2\pi}\sigma x} \cdot e^{-\frac{lnx-\mu}{2\sigma^2}} \tag{2.1}$$

$$F(x) = 1 - e^{(\frac{x}{\lambda})^k} \tag{2.2}$$

2.2.1 Lognormal

The lognormal distribution has been used to analyze the distribution of lines of code in software systems which presents a fat tail [124]. The associated generative process considers *units* with a *property* characterized by a value, like classes with a certain number of lines of code. The units are randomly selected to increment their property value, and such increment is directly proportional to the actual property value. It can be demonstrated that such process produces a statistical distribution with a fat tail, which can appear as a power law with a cut-off at large values.

In our context, the process leading to a lognormal distribution of the issue resolution time may be obtained considering issues as units and the property as the fixing time. Developers start working for addressing an issue when it enters the issue tracking system, in the meanwhile new issues are introduced. Issues are handled independently and by different developers, and it is likely that difficult issues are managed in multiple working sessions, so that when an issues is tackled, developers work on it for a certain time, and then stop, leaving the remaining work to another session. The time needed to work on an issue in the following working session can be considered roughly proportional to the time accumulated in the previous sessions, since it is proportional to the overall complexity and difficulty associated to the issue. Furthermore issues can be considered as randomly selected by developers, since they are randomly introduced in the issue tracking system as they are discovered. This rough hypothesis could explain the good fitting provided by the lognormal distribution.

2.2.2 Weibull

The Weibull distribution models a system with an initially fixed number of *components* with a certain *failing rate*. Eventually, the fraction of failed components saturates to one. Also this process can generate a statistical distribution providing a power law fat tail.

For what concerns the process leading to a Weibull distribution in our dataset, the issues represent the components and the fixing time represent the average failing time. Since this may vary randomly, there will be issues quickly solved, corresponding to components failing rapidly, and there will be long lasting issues, corresponding to more robust components, whose number decreases as the duration increases, satisfying the hypotheses of the Weibull model.

2.2.3 Cumulative Complementary Distribution Function

In probability theory and statistics, the cumulative distribution function (CDF) describes the probability that a real-valued random variable X with a given probability distribution will be found at a value less than or equal to x. In the case of a continuous distribution, cumulative complementary distribution function (CCDF) is its complementary defined as 1-CDF. The plot of CCDF is useful whenever we are dealing with distributions right-skewed and with fat tail. Indeed, such distributions cannot be characterized by statistics like mean and standard deviation [75].

2.3 Affectivenss

In this section, we describe the three kinds of affective metrics studied : politeness, sentiment and emotion. These three metrics have been used by other researchers, i.e., politeness [77] and [30], sentiment [52] and [88], and emotion [73].

2.3.1 Sentiment

We measured sentiment using the state-of-the-art SentiStrength tool[2], which is able to estimate the degree of positive and negative sentiment in short texts, even for informal languages. SentiStrength by default detects two sentiment polarization:

- Negative: -1 (slightly negative) to -5 (extremely negative)
- Positive: 1 (slightly positive) to 5 (extremely positive)

It uses a lexicon approach based on a list of words in order to detect sentiment. SentiStrength was originally developed for English and was optimized

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

for short social web texts. We used SentiStrenght to measure the sentiment of developers in issue comments (which often are short).

2.3.2 Politeness

Politeness is "the ability to make all the parties relaxed and comfortable with one another[3]." Danescu et al. [30] proposed a machine learning approach for evaluating the politeness of Wikipedia[4] and Stackoverflow[5] requests. Since Stackoverflow is well-known in the software engineering field and is largely used by software practitioners, the model that Danescu et al. used [30] is suitable for our domain, i.e., Jira [6] issues, where developers post and discuss about technical aspects of issues. The authors provide a Web application[7] and a library version of their tool.

Given some text, the tool calculates the politeness of its sentences providing as a result one of two possible labels: *polite* or *impolite*. Along with the politeness label, the tool provides a level of confidence related to the probability of a politeness class being assigned. We thus considered comments whose level of confidence was less than 0.5 as neutral (namely the text did not convey either politeness or impoliteness). Table 2.1 and 2.2 show some examples of polite and impolite comments as classified by the tool[8].

2.3.3 Emotions

Emotion mining tries to identify the presence of human emotions like `joy` or `fear` from text, voice and video artifacts produced by humans. As such, it is different from sentiment analysis, which instead evaluates a given emotion as being positive or negative [80]. The field of sentiment analysis as a whole is currently moving towards emotion mining, since this provides more detailed insights into the behavior of people [123]. Since these research areas affect the decision-making process of people [80], a diverse range of actors, from marketing departments and investors to politicians make use of their techniques. In software engineering, emotion mining applied to text artifacts could be used to provide hints on factors responsible for `joy` and satisfaction among developers (e.g., new release), or `fear` and anger (e.g., deadline or a recurring bug). Moreover, it provides a different perspective to interpret productivity and job sat-

[3]http://en.wikipedia.org/wiki/Politeness
[4]https:en.wikipedia.orgwikiMain_Page
[5]http:stackoverflow.com
[6]Jira Issue Tracking System https://www.atlassian.com/software/jira
[7]http://www.mpi-sws.org/cristian/Politeness.html
[8]User's names are reported as <*dev_name_a*> for the sake of privacy.

Comment	Confidence Level
Can you put more detail in description ? If you can attach what was done in 0.89-fb branch, that would be nice. Thanks, <dev_name_b>	0.83
<dev_name_a>, can you open a new Jira for those suggestions? I'll be happy to review.	0.919
<dev_name_a>, can you submit a patch against trunk? (Sorry, thought I tagged this 0.7 to begin with.)	0.8

Table 2.1: Examples of polite comments.

Comment	Confidence Level
Why are you cloning tickets? Don't do that.	0.816
- why blow away rack properties? - how does this allow talking to non-dynamic snitch?	0.85
<dev_name_a>, What is the point of doing that?	0.81

Table 2.2: Examples of impolite comments.

isfaction. Since several studies show that it is possible to "contract" emotions from others through computer-mediated communication systems [50, 56], development artifacts like mailing lists or the discussion board of an issue tracking system could be a promising source for mining developer emotions during software evolution.

Primary emotions	Secondary emotions	Tertiary emotions
love	Affection	Compassion, Sentimentality, Liking, Caring, ...
	Lust/Sexual desire	Desire, Passion, Infatuation
	Longing	
Joy	Cheerfulness	Amusement, Enjoyment, Happiness, Satisfaction, ...
	Zest	Enthusiasm, Zeal, Excitement, Thrill,Exhilaration
	Contentment	Pleasure
	Optimism	Eagerness, Hope
	Pride	Triumph
	Enthrallment	Enthrallment, Rapture
Surprise	Surprise	Amazement, Astonishment
Anger	Irritability	Aggravation, Agitation, Annoyance, Grumpy, ...
	Exasperation	Frustration
	Rage	Outrage, Fury, Hostility, Bitter, Hatred, Dislike, ...
	Disgust	Revulsion, Contempt, Loathing
	Envy	Jealousy
	Torment	Torment
Sadness	Suffering	Agony, Anguish, Hurt
	Sadness	Depression, Despair, Unhappy, Grief, Melancholy, ...
	Disappointment	Dismay, Displeasure
	Shame	Guilt, Regret, Remorse
	Neglect	Embarrassment, Humiliation, Insecurity, Insult, ...
	Sympathy	Pity, Sympathy
Fear	Horror	Alarm, Shock, Fright, Horror, Panic, Hysteria, ...
	Nervousness	Suspense, Uneasiness, Worry, Distress, Dread, ...

Table 2.3: Parrott's emotion framework.

Parrott's Framework

Emotion is a "psychological state that arises spontaneously rather than through conscious effort and is sometimes accompanied by physiological changes" [58]. General types of emotions are `joy`, `sadness`, `anger`, `surprise`, `hate` and `fear`. However, many other categories and sub-categories can be identified. Since there is not one standard emotion word hierarchy, many studies in the cognitive psychology domain [101] have focused on research about emotions, resulting in various proposals for cathegorizing emotions [96, 89, 84].

One of the more recent classifications of emotions is Parrott's framework [84], which classifies human emotions into a tree structure with 3 levels, as is shown in Table 2.3. Each level refines the granularity of the previous level, making abstract emotions more concrete. For example, level-1 of this classification consists of six primary-emotions, i.e., `love`, `sadness`, `anger`, `joy`, `surprise` and `fear`. The concise and intuitive nature of the primary emotions makes Par-

```
[...] I'm not so convinced that moving all
the static methods out is useful (Fear).

How is a bunch of static methods on a
utility class easier than a bunch of static
methods within the HtmlCalendarRenderer
better? (Anger)

[...] the risk of introducing new bugs for
no great benefit (Fear).

Specific feedback regarding this specific
patch: (1) There is significant binary
incompatibility (Neutral).

[...] Previously almost all these helper
methods were private; this patch makes them
all public [...] (Neutral)
```

Figure 2.3: Example of issue comments with identified emotions for each sentence

rott's classification easy to understand by different stakeholders. In particular, the classification is not just aimed at the people rating a particular artifact as describing a particular emotion, but also appeals to people like team leads trying to benefit from the emotional classification to understand the emotions of their team members. In our study, we only consider the six primary emotions, but in future work we plan to extend our results to secondary and tertiary emotions for the most popular primary emotions.

Chapter 3

Software Effectiveness

3.1 On the influence of maintenance activity types on the issue resolution time

The ISO/IEC 14764 standard specifies four types of software maintenance activities spanning the different motivations that software engineers have while performing changes to an existing software system. Undoubtedly, this classification has helped in organizing the workflow within software projects, however for planning purposes the relative time differences for the respective tasks remains largely unexplored.

In this empirical study, we investigate the influence of the maintenance type on issue resolution time. From GitHub's issue repository, we analyze more than 14000 issue reports taken from 34 open source projects and classify them as corrective, adaptive, perfective or preventive maintenance. Based on this data, we show that the issue resolution time depends on the maintenance type. Moreover, we propose a statistical model to describe the distribution of the issue resolution time for each type of maintenance activity. Finally, we demonstrate the usefulness of this model for scheduling the maintenance workload.

3.1.1 Introduction

Software maintenance is a key ingredient of any successful software project, certainly in modern development processes with their emphasis on iterative and incremental development. Already in 1976, Swanson introduced the first classification of software maintenance types to aid researchers and practitioners in describing the activities they are performing [107]. Swanson's classification has later been extended and today is incorporated into the ISO/IEC 14764 standard which defines four types of activities: (a) corrective maintenance is devoted to removing bugs; (b) adaptive maintenance is related to adding new features; (c) perfective maintenance deals with activities to enhance performance; (d) and preventive maintenance takes into account changes on software finalized to avoid future bugs.

In this study, we investigate the relationship between the type of maintenance activity (as defined by the ISO/IEC 14764) and the time required for finishing work items. For our empirical study, we rely on GitHub's issue repository that hosts issues that are ascribable as requiring corrective, adaptive, perfective or preventive maintenance. We try out several statistical distributions to see which ones are suitable for describing the distribution of the issue resolution time for each type of maintenance activity.

This work follows the Goal-Question-Metric paradigm [112]. The *goal* of this study is to evaluate the impact of the type of maintenance activity in the issue resolution process. The *focus* is to evaluate how the metric issue resolution time changes for the corrective, adaptive, perfective and preventive maintenance. For this purpose, we investigate which statistical model is suitable for describing the resolution time distribution. The *viewpoint* is that of issue triager and researchers. The first one, scheduling maintenance workload, is interested in evaluating the resolution time per maintenance activity. The latter, studying software maintenance based on data mined from software repositories, is interested on how to exploit the maintenance type for building better predictive models. The *environment* of this study regards the issue tracking repository with 14000 issues taken from 34 open source projects. To achieve our purpose, we pursue the following research questions:

- *RQ1: Is the issue resolution time dependent on maintenance type?*

- *RQ2: Is it possible to model the distribution of the issue resolution times with respect to the type of maintenance considered?*

This section is organized as follow. In section 2.2, we provide the required background related to distribution functions. In section 3.1.2 we describe the data used for the empirical study. In section 3.1.6 we present our results and in section 3.1.7 we provide an operative example on how to use these findings. The threats to validity are reported in section 3.1.8.

3.1.2 Experimental Setup

This section presents in 3.1.3 the dataset we use, in 3.1.4 how we map issues to maintenance activity and finally in 3.1.5 how we measure the issue resolution time.

3.1.3 Dataset

For our empirical study we use the issue reports recorded in the GitHub repository as described in 2.1.1.

Since we are interested in maintenance activity that are complete, we select only closed issues, namely we removed every issue report without a closed event. Table 4.14 shows the dataset statistics: almost 50% of issues are labeled, within this 50%, almost 30% are labeled with keyword we can relate to the maintenance activity that was performed. For the empirical study, we use 34 (out of 90) projects since they have at least one issue with a label related to a maintenance activity. Our sample take into account projects developed with different programming languages (e.g., C, Java, ...) and with few or lot of developers involved (up to 95). From this projects we collect more than 14,000 issues.

Table 3.1: *GitHub dataset statistics.*

Statistic	value
Number of projects	90
Number of projects with labeled issues	34
Number of different languages per projects with labeled issues	12
All Issues	≃ 100000
Issues with at least one label	≃ 50000
Issues related to *maintenance*	≃ 39000
Issues related to *maintenance* with status *closed*	14298
Mean of labeled issues per projects	44
Standard Deviation of labeled issues in projects	72
Min labeled issues per projects	1
Max labeled issues per projects	345
Min developers per projects with labeled issues	1
Max developers per projects with labeled issues	95

3.1.4 Mapping issue to the maintenance type

Issue reports in GitHub may refer to any type of maintenance. Classification of these issues is not easy since none of the authors contributed to any of these projects. For this reason, we relied on our developer experience to classify the issue according to the label they have. We believe that labels used by developers in GitHub are reliable since assigned by the person who actually performed the maintenance activity. The first two authors manually inspected all issue labels in the dataset and, where feasible, they mapped the label to a particular maintenance activity. As guideline for the labeling we refer to several studies presented in literature [1, 71, 107]. We used this approach since it was already successfully adopted by Mockus and Votta and Purushothaman and Perry [71, 91]. In these cases the authors determine the type of maintenance analyzing the text message submitted by the developer.

We did not map any label that was not clear or was not ascribable to a maintenance type. For example, labels like *GUI* and *Mobile* are generic an do not give hints on the type of maintenance performed, hence were classified in a separate category. For the same reason, we did not consider issues that use labels ascribable to different type of maintenance. Mapping of these labels as either a corrective, perfective, adaptive and preventive would have introduced a threats to validity in our analysis.

Table 3.2 shows the mapping with the percentages of maintenance types.

3.1.5 Issue Resolution Time

Figure 3.1 shows the typical issue timeline in GitHub:

- T_{cr} represents the time an issue is created.
- T_{cl} represents the time an issue is closed.
- T_{a} represents the time an issue is assigned to a developer.
- T_{s} is the time a developer subscribe that an issue as been assigned to him.
- T_{m} represent the time when an issue is merged in the repository, namely the local commit is merged in the remote repository.

In our analysis we are interested in analyzing the *working* time spent by the developer to resolve the issue. For this reason, we do not consider the triaging steps needed to assign the issue T_{a} - T_{cr} nor the steps between an issue is merged into the code base and confirmed as being closed T_{cl} - $T_{\mathrm{m}}\dot{\mathrm{W}}$e compute the issue resolution time as the difference between T_{m} and T_{s} namely we compute the difference between the first time the issue is subscribed and the last

Maintenance type	Labels	Samples
Corrective	Bug, Bugs, Bug Report, non critical bug, Type-Bug,Critical,bug, Defect, Framework bug, Rails bug, browser bug, Type-Defect,Crashes - 500 errors	6754 (47.9%)
Perfective	Enhancement, Type-Enhancement, Improvement,Type-Improvement,Cleanup refactoring, code refactoring	2397 (17%)
Adaptive	Feature, Feature Request, New feature,Approved feature,Type-New Feature	4723 (33.5%)
Preventive	Test, Testing Framework	226 (1.6%)
Not Mapped (due to doubtful label)	Task, Suggestion, ...	3149 (8% of all labeled issues)
Not Mapped (due to multi type of maintenance performed)	Third party issue, Status-Started, Cleaning, ...	467 (1 % of all labeled issues)

Table 3.2: Table of mapping issue-labels/maintenance-type

time the issue is merged in the repository. In this way we take into account also supplementary fixes due to re-opened bugs. Our assumption is that the developer subscribes for issue resolution when she is ready for the maintenance activity; whereas she merges the code change only when the maintenance activity is complete. We assume that such time-checkpoints are representative of the time spent to resolve the issue [16, 81].

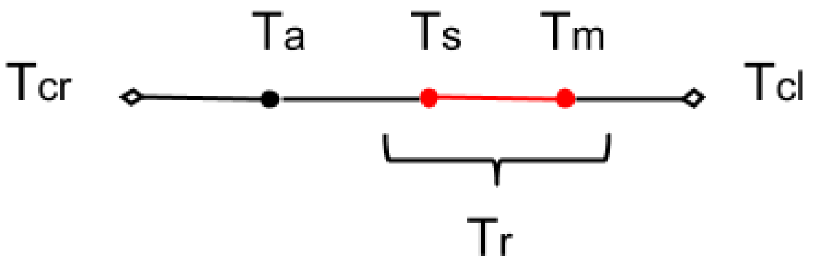
3.1.6 Results and Discussion

For each research question, we first discuss its motivation, followed by the approach we used and finally we present our findings.

RQ1. Is the issue resolution time dependent on maintenance type?

Motivation. Weiss et al. reveal that JBoss's bugs are on average addressed quicker than new features [115]. Assuming that bugs and features are addressed with different type of maintenances, we may hypothesize that issue resolution time is influenced by the maintenance type.

A similar hint is provided by the relationship between code change and



$$Tr = Tm - Ts \tag{3.1}$$

Figure 3.1: Example of timeline for GitHub issue.

maintenance type. Hindle et al. show that large commits — commits that involve more than thirty files — are related more likely to perfective than to corrective maintenance [60]. Purushothaman et al. show that even in each file the percentage of lines of code modified is not equally distributed among the different types of maintenance activities [91]. Having evidence that type of maintenance "correlates" with the amount of files and lines of code handled, we hypothesize that the type of maintenance “correlates" with issue resolution time as well.

Approach. In order to detect differences among the resolution times belonging to different maintenance activities we adopted the Wilcoxon test [117, 102], which is non-parametric and thus can be used with no restrictions nor hypotheses on the statistical distribution of the sample populations. The test is suitable for comparing differences among the averages or the medians of two populations when their distributions are not gaussian. For the analysis, we use the one sided Wilcoxon rank sum test using the 5% significance level (i.e., p-value < 0.05) and we compare each resolution time dataset with all others datasets.

Findings. The issue resolution time depends on maintenance type. Figure 3.2 reports resolution times boxplots in a logarithmic scale (time is expressed in days). The time distributions are right-skewed with values ranging across different orders of magnitudes, suggesting a non gaussian distribution of the data.

Table 4.11 reports the result of the Wilcoxon test. The p-values obtained for all the couples, which are all below 0.005, indicating that issue fixing times are significantly different for different maintenance types. In particular the test performed for each couple shows to a high significance level that: **perfective**

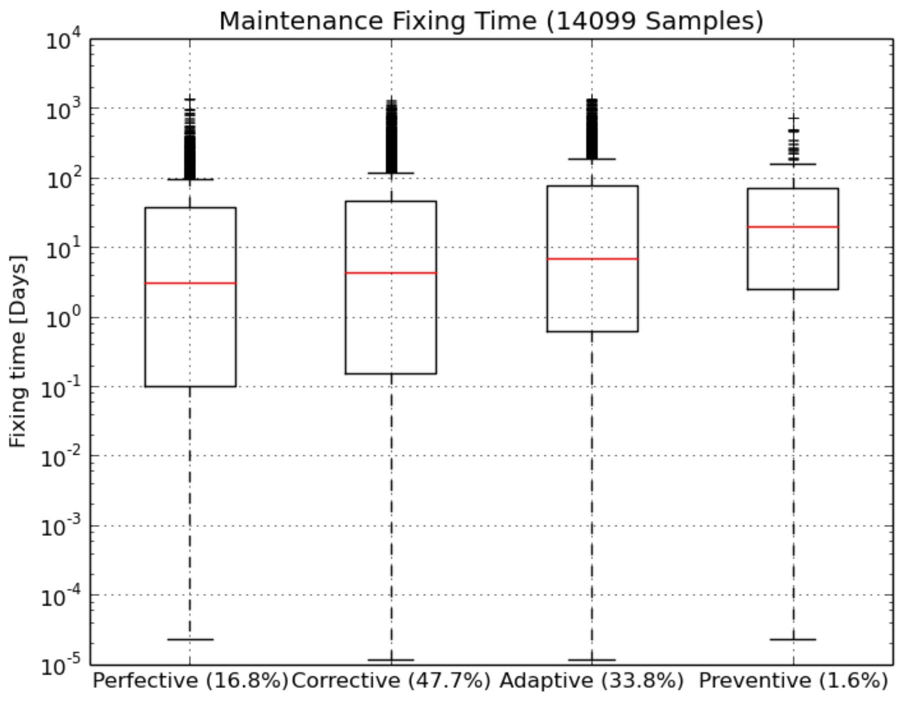


Figure 3.2: Resolution time in days grouped by maintenance type

Pair of Groups Compared	Test	p-value	Effect Size
Corrective vs Perfective	greater	2.3E-5	0.024
Corrective vs Adaptive	less	1.9E-5	0.096
Corrective vs Preventive	less	2.4E-5	0.076
Adaptive vs Preventive	less	4.3E-3	0.049
Adaptive vs Perfective	greater	n.s	0.124
Perfective vs Preventive	less	n.s	0.144

Table 3.3: Wilcoxon test for groups split by maintenance, test column indicates if the median of the first group is greater or less than the second type(n.s stands for not statistically significant.)

maintenance is on average faster than corrective maintenance; corrective is on average faster than adaptive; adaptive is on average faster than preventive.

We conclude that different maintenance activities do have different issue resolution times.

RQ2. Is it possible to model the distribution of the issue resolution times with respect to the type of maintenance considered?

Motivation. RQ1 exhibits that the issue resolution time depends on the type of maintenance handled. Here, we investigate how to describe the distribution of the issue resolution time to highlight the differences among the maintenance types. A correct modeling is indeed crucial for scheduling the development activities and estimate them.

Approach. We analyze the lognormal and Weibull statistical distributions models. These distributions have been proved to be suitable to model software metric distributions [25, 26]. Moreover, in section 2.2 we explained which process may lead to such distributions in the context of the issue resolution time.

Findings.

In tables 3.4, 3.5 and 3.6 we report the lognormal and Weibull best fitting parameters with the significance levels, represented by the fitting coefficient R^2, along with the size of dataset used for the fitting. We do not report a similar table for preventive maintenance because the distribution consists of only 200 sample, thus it is not possible to repeat the same analysis with different sample size as for the other maintenance activities. We compute these parameters taking into account issues handled with corrective maintenance, but the same behavior is exhibited by the best fitting parameters associated to other maintenance types; here skipped for the sake of simplicity. These tables show that lognormal and Weibull have good fitting for the raw data in all cases, with a level of significance higher than 90%. The best fitting parameters are in agreement with the results of the Wilcoxon rank sum test (table 4.11). Moreover, such parameters are quite stable and do not vary with the considered sample size. This is a relevant point since it enables these models to be used also in the early stages of the project development when few hundreds of resolved issues are available. In our empirical study, just using a random sample of 3% (200 out of 6754 - number of issues for corrective maintenance) of the issues, we can predict the issue resolution time. In figures 3.3c, 3.3b, 3.3d and 3.3a we plot the Cumulative Complementary Distribution Function (CCDF) of issue resolution time. As we can see, the first 200 points are practically overlapping all other points as well as the Weibull curve.

For all these reasons, the **statistical distribution models are suitable to quantify how much the issue resolution times is influenced by the type of maintenance.**

	Lognormal			Weibull		
Sample Size	μ	σ	R^2	λ	k	R^2
100	11.186	4.9773	0.95285	1.0561e06	0.3502	0.9902
100	11.349	5.0921	0.93528	0.8970e06	0.2852	0.9863
200	11.77	4.5228	0.9271	0.8843e06	0.3128	0.9858
200	11.267	5.023	0.94094	1.1848e06	0.3479	0.9915
500	11.877	4.4152	0.92547	1.0019e06	0.3192	0.9891
500	11.554	4.7862	0.92842	0.8924e06	0.2883	0.9767
1000	11.716	4.605	0.9249	0.9475e06	0.3030	0.9839
1000	11.71	4.4949	0.93989	0.9095e06	0.3033	0.9921
All	11.743	4.5949	0.93899	0.9279e06	0.3008	0.9859

Table 3.4: Lognormal and Weibull fitting parameters for different sample size for corrective maintenance type

From the comparison of tables 3.4, 3.5 and 3.6, we can highlight some differences between Weibull and lognormal models. The Weibull distribution shows less stable best fitting parameters compared to lognormal. However, it has a better fitting as confirmed by the coefficient R^2. This fact is highlighted also in figures 3.3c, 3.3b, 3.3d and 3.3a where the Weibull distribution exhibits a better overlapping with raw data. We must underline that our data is limited to few years. This introduces a upper cut off on resolution times, which cannot be

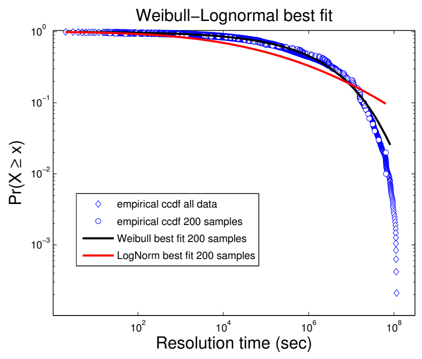
	Lognormal			Weibull		
Sample Size	μ	σ	R^2	λ	k	R^2
100	11.313	4.0977	0.9495	5.364e05	0.3151	0.984
200	11.751	4.4427	0.9449	9.7704e05	0.2958	0.9839
500	11.447	4.3648	0.9554	7.1057e05	0.294	0.9875
1000	11.562	4.2992	0.9533	7.6992e05	0.2990	0.9891
All	11.587	4.2370	0.9476	7.6707e05	0.3038	0.9886

Table 3.5: Lognormal and Weibull fitting parameters for different sample size for perfective maintenance type.

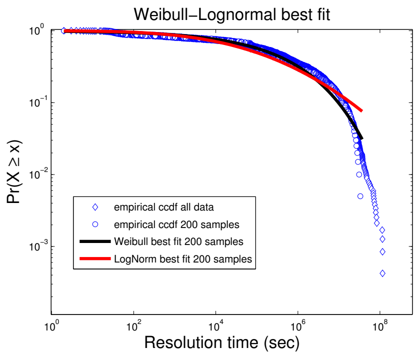
	Lognormal			Weibull		
Sample Size	μ	σ	R^2	λ	k	R^2
100	13.073	3.6821	0.9574	2.2962e06	0.3917	0.9875
200	13.148	3.2827	0.9688	2.1859e06	0.4101	0.9963
500	12.611	4.0179	0.9592	1.7755e06	0.4308	0.9895
1000	12.747	3.8036	0.9623	1.8473e06	0.3602	0.9904
All	12.776	3.6721	0.9700	1.8448e06	0.3615	0.9945

Table 3.6: Lognormal and Weibull fitting parameters for different sample size for adaptive maintenance type

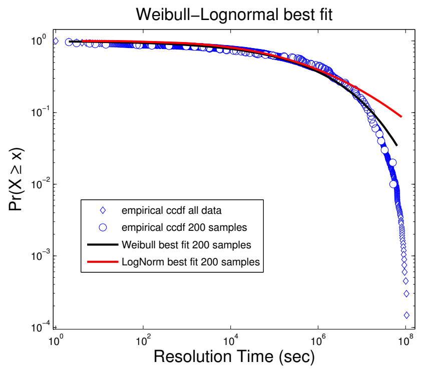
longer than the time span analyzed. This may be the reason why the lognormal fails to fit well in the tail of the distribution, even if the fitting parameters are stable. We believe that in the case of data without such upper cut off the lognormal would provide a better fit and more accurate estimates. In fact the lognormal curve well overlaps the bulk of the distribution and mainly fails in the tail, for larger values, where it remains much higher than the data curve. But in this region data are sparse and rare, because of the finiteness of the dataset. In principle, with an infinite amount of data, the fat tail would present arbitrarily large values. Namely, if one could consider arbitrarily large datasets, there will be maintenance operations lasting an arbitrarily large amount of time. In case of finite dataset, data in the tail become sparse and the related complementary cumulative distribution will drop to zero faster than in the case if theoretically infinite dataset. This explain why the lognormal distribution best fitting, which suits for a theoretically infinite dataset with no cut off in the tail, fails in fitting properly the data in the tail while is very good for data in the bulk.



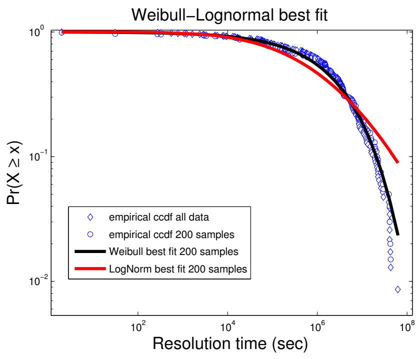
(a) Lognormal and Weibull CCDF fitting for corrective maintenance.



(b) Lognormal and Weibull CCDF fitting for perfective maintenance.



(c) Lognormal and Weibull CCDF fitting for adaptive maintenance.



(d) Lognormal and Weibull CCDF fitting for preventive maintenance.

3.1.7 How to use Statistical Distribution Models

We present two practical examples to show the effectiveness of our model. In the first example we tackle the problem of estimating the time effort needed to resolve accumulated issues over time. Suppose a company started working on solving issues and collected issue resolution times. After (say) six months of work it accumulated a queue of unsolved issues during the last two weeks or the last month. We simulated this scenario using our data, considering the six months commits divided in two groups: "solved issues" (the set SI), and "accumulated issues" during last two weeks (the set AI), still to be solved. Using the set SI we estimated the Weibull parameters for the various kinds of maintenance. The set AI is composed by:

- Last two weeks: 8 corrective, 7 adaptive, 3 perfective, 0 preventive
- Last month: 15 corrective, 15 adaptive, 7 perfective, 0 preventive

From the knowledge of the Weibull CCDF we partitioned the set AI into bins, corresponding to percentage of issues multiple of ten, namely 10%, 20%, and so on. For each percentage we determined the issue resolution times for each kind of maintenance. For example, from the Weibull parameters for corrective maintenance we have that 10% of issues are solved in less than 540 seconds, 20% in less than 6400 seconds, and so on. This procedure corresponds to solve the inverse transform of the CCDF for issue resolution times at discrete value, which can be very practical. Next we used these data to statistically infer times needed for solving the issues. For example, 10% of the 8 corrective maintenance operations, will be solved in 540 seconds, so that this amount can be estimated multiplying 0.8 by 540 seconds. Then another 0.8, which together with the first makes the 20% of data, takes less than 6400 seconds. Thus we multiply 0.8 by 6400 and sum it up with the estimate obtained for the first 10%, and so on. We stop the count when we reach the 90%, since the Weibull CCDF goes to 100% when time is infinite.

For the last two week data, the lower estimate provides 207 days for corrective maintenance, 251 days for the adaptive, 62 days for the perfective. Since we stop the count when we reach the 90%, these can be considered lower estimates. Next we randomly selected the issue resolution time for 8 corrective maintenance operations, for 7 adaptive maintenance operations, and 3 perfective maintenance operations from our dataset and summed up all the resolution times, and repeated the procedure three times. The results of the trials are in table 3.7.

In the second example the company wants to commit with customers requiring a certain amount of improvements, and needs to estimate the fraction

Maintenance type	TRT[days] 1° trial	TRT[days] 2° trial	TRT[days] 3° trial
8 corrective maintenance	523	313	316
7 adaptive maintenance	437	307	683
3 preventive maintenance	47	119	120

Table 3.7: Issue resolution time estimations (TRT stands for Total Resolution Time)

of improvements required by customers that can be solved in a given amount of time, before committing itself. We simulated the number of possible improvements required by customers using our data, estimating the perfective maintenance operations requested in a month. We averaged this number over 10 months, and obtained 20 requests, which is suitable for a medium size company. Again we used the Weibull CCDF with parameters estimated by the set SI, and partitioned the number of perfective maintenance operation requested into bins corresponding to percentage of issues multiple of ten. We suppose the company wants to estimate the time for solving 70% of requests. From the Weibull CCDF best fitting we obtain, using the same procedure as above, an estimate of 43 days for solving 14 issues. Next we randomly selected the issue resolution time for 14 perfective maintenance operations from our dataset and summed up all the resolution times, and repeated the procedure three times. The results of the trials are reported in table 3.8.

# issues	TRT[days] 1° trial	TRT[days] 2° trial	TRT[days] 3° trial
14	532	304	167

Table 3.8: Issue resolution time estimations for preventive maintenance (TRT stands for Total Resolution Time)

These three trials display a large variability which is intrinsic of data distributed according to a power-law in the tail. In fact, there are many issues with relatively low fixing times, but there are also issues with a very large fixing time, even if these latter are much less. Thus, the result of summing up all resolution times for 14 issues randomly selected can largely vary according to the possibilities of selecting 14 issues all with low resolution time, or 14 issues containing even one single issue with a very large resolution time.

3.1.8 Threats to Validity

In this section we present the threats to validity of our study according to the guidelines reported in [122].
Threats to internal validity concern confounding factors that can influence the obtained results. We assume a causal relationship between the issue resolution time and the type of maintenance performed [60, 71, 91]. However, several factors can influence the issue resolution time like the complexity of the issue, type of project, project deadline etc.. Due to the large number of samples, projects and time frame analyzed, we assume that these factors may compensate each other. We hypothesize that our dataset is big enough to be avoid the bias related to specific factors. **Threats to construct validity** focus on how accurately the observations describe the phenomena of interest. In this study, the elements of interest are the issue resolution time and the type of maintenance. The first element represents the *working* time spent to address a maintenance activity (stripped by the triaging times not devoted to code maintenance). We compute this value as the difference between (1) the time when the issue is subscribed by the developer and (2) the time when the issue is merged in the repository. We use these time-checkpoints assuming that the developers subscribe an issue when they ready to change the maintain the code; whereas they merge the code change only when the maintenance activity is completed. The adoption of issue fields to determine the resolution time has been already used in literature [16, 81]. The second element represents the type of activity performed by the developer to address the issue. We use the labeling provided by developer to classify the issue and then the type of maintenance performed. Since the labeling was done by people aware of the activity performed, we consider accurate and reliable such labels. To limit the impact of a subjective interpretation of the labels, the first and the second author referred to definition of maintenance types provided in literature [1, 71, 107]. We rely on this approach since it was already successfully adopted in literature [71, 91]. We decided to do not consider the issue for the analysis whenever the classification of its labels was not possible or doubtful. We assume that these labels (e.g.; *GUI*, *Mobile*) do not belong to specific maintenance category, namely they do not introduce a bias (due to a maintenance type underrepresented).

Threats to external validity correspond to the generalizability of our experimental results. In this study, we use more than 30 projects that are representative of open source domain. To generalize our findings we should extend the analysis to industrial projects.

Threats to reliability validity correspond to the degree to which the same data would lead to the same results when repeated. We address this threat describing all steps of our experiments and using a dataset freely available [48].

Repeating our analysis may easily lead other researchers to the same results.

3.2 Measuring and Understanding the Effectiveness of JIRA Developers Communities

Tools for project management and issues/bugs tracking are becoming useful for governing the development process of Open Source software. Such tools simplify the communication process among developers and ensure the scalability of a project. The more information developers are able to exchange, the clearer are the goals, and the higher is the number of developers keen on joining and actively collaborating on a project. In this study we present a preliminary empirical analysis of the communities-structure of developers in JIRA by analyzing 7 popular projects hosted in the repository. We analyze how these communities perform in terms of issue-resolution time of any given issue. The main contributions of this work are the confirmation of the existence of communities in developer networks, and the empirical finding that the issue resolution-time of any given issue is not correlated with the dimension of a developer community.

3.2.1 Introduction

During the last decade, researchers have been exploring the effectiveness of Open Source software communities, and today, Open Source systems are no longer considered to be children of a lesser God. The quality reached by such systems is well known and recognised [33]. Faster Internet connections, smartphones, tablets and devices always connected to the Internet help open source developers to stay in touch with each other in order to generate software. Communication processes are a key-factor for open source paradigm development, and tools able to manage the communication within a group of people developing and creating something together are therefore vital. In this study we analysed the developers' structure for seven projects hosted in JIRA[1], a proprietary issue tracking product developed by Atlassian. JIRA provides bug tracking, issue tracking, and project management functions,and includes tools allowing migration from competitors. According to Atlassian [2], it is used for issue tracking and project management by over 25,000 customers around the world. An Issue Tracking System (ITS) is a repository used by software developers as a support for the software development process. It supports corrective maintenance

[1]https://www.atlassian.com/software/jira

[2]http://blogs.atlassian.com/2013/05/why-people-choose-jira-6/

activity like Bug Tracking systems, along with other types of maintenance requests.

Since JIRA is becoming more and more popular among developers, it is worthwhile to understand the impact of such a product on the structure of developer communities. Several studies have demonstrated that developers involved in the process of creating new open source software are organised, and that a clear structure exists. Linux, for example, is not the result of a disorganised process executed by disorganised developers. Their success in developing a very complex system, used even by NASA[3], was not simply based on luck. However, this doesn't mean that open source paradigms always lead to the development of quality products. Questions such as the following arise about open source communities: "Can I trust something produced by people working for free, driven only by passion and pleasure?" [57] "Can software developed without a commercial plan and strict deadlines be of high quality?" "How can developers work efficiently without central project coordination?" [28].

Diseconomies of scale can affect the communication process of a group of developers working together. When the number of developers increases (e.g. when newcomers join a project), the structure becomes more difficult to manage, and it may be difficult to keep track of who is doing what. Tools such as JIRA help to reduce co-ordination problems, to increase the level of communication and to scale up the project by reducing releases' time. The main goal of this study was to provide empirical evidence showing whether developers are organised with defined structures/teams, and if such teams perform differently in terms of productivity. We define the productivity of a team as the average fixing time for any given issue. We answer the following research questions: **RQ1:** Does the open source software developer's network graph contain communities? **RQ2:** Are there differences in fixing time between work teams? The rest of the section is organized as follows: in Section 5.1.2 we give an overview of the related works, in Section 3.2.2 we explain the dataset we used and the methodology, in Section 3.2.5 we discuss our results and we conclude with Section 3.2.8 and VI by analyzing the threats to validity.

3.2.2 Experimental Setup

3.2.3 Dataset

We built our dataset by collecting data from the Apache Software Foundation Issue Tracking system, JIRA[4] as described in Sec. 2.1.1. We mined the projects

[3]http://www.linux.com/news/featured-blogs/191-linux-training/711318-linux-foundation-training-prepares-the-international-space-station-for-linux-migration

[4]https://www.atlassian.com/software/jira

of the Apache Software Foundation collecting issues from 2002 to December 2013, Table 4.14 shows the corpus of the 7 projects selected for our analysis, highlighting the number of comments recorded for each project and the number of developers involved. We selected projects with the highest number of comments.

Project	# of comments	# of developers
HBase	91016	951
Hadoop Common	61958	1243
Derby	52668	675
Lucene Core	50152	1107
Hadoop HDFS	42208	850
Hive	39002	757
Hadoop Map-Reduce	34793	875

Table 3.9: Selected Projects Statistics

3.2.4 Developer's Network

We extracted the developer network based on the data contained in JIRA. JIRA allows users to post issues (with a set of properties such as maintenance type, priority, etc.) and to comment on them. We built developer network modeling nodes, which represent developers, and edges from *node A* to *node B*, which represent when *developer A* was commenting on *developer B's issue*. In this manner, we obtained a directed network.

We then used Gephi[5] [5] to analyze the obtained network. Gephi is an interactive visualization and exploration tool. We ran the modularity algorithm, based on the algorithms developed by Blonde [13] and Lambiotte[67], in order to obtain the network communities. We finally measured the obtained networks.

Project	Modularity	Avg. Degree	Avg. Clustering Coeff.	# of Communities
HBase	0.283	5.552	0.459	8
Hadoop Common	0.334	5.617	0.296	15
Derby	0.193	5.170	0.484	11
Lucene Core	0.269	3.493	0.339	14
Hadoop Map-Reduce	0.332	5.040	0.242	12
Hive	0.333	4.358	0.287	16
Hadoop HDFS	0.284	5.275	0.311	8

Table 3.10: Selected Projects Network Statistics

[5]http://gephi.github.io/

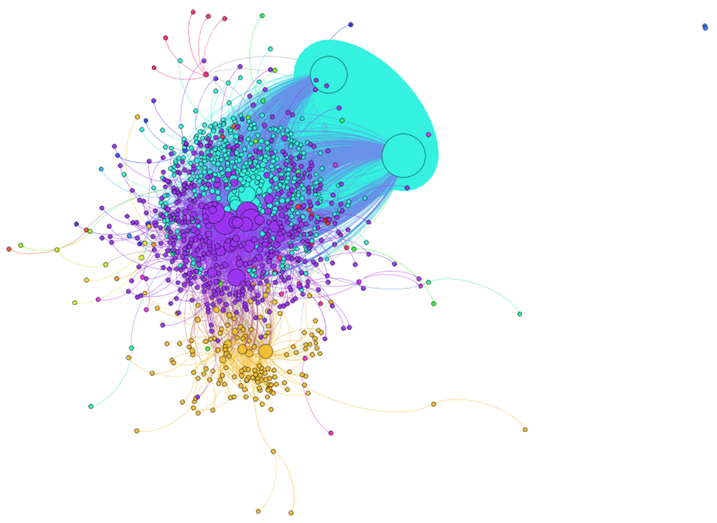


Figure 3.4: Example of Developer's Graph Extracted From Lucene-Core Project.

Figure 3.4 shows an example of network graph we obtained. Node size represents the number of issues posted by a developer, and the edge size from *node A* to *node B* represents the number of comments posted by *node A* in response to issues posted by *node B*.

3.2.5 Result and Discussion

3.2.6 Does the open source software developer's network graph contain communities?

Motivation. Understanding developer structure in open source projects allows both work teams and management to have better control over the whole project. Both issue triage and workload scheduling may benefit from having a clear view of how the developers are organized and how productivity is spread across work teams. For this reason our first research question aims to explore the presence of communities in JIRA developer networks.

Approach. We created the developer network graph as described in 3.2.4. Each Node represents a developer who posted/commented on an issue, and each Edge represents a developer who commented on another developer's issue. We applied the modularity algorithm [67] to obtain developer communities.

Findings. Open Source Projects hosted in JIRA do have communities. Table 3.10 shows the network metrics for the analyzed projects. The Modularity metric represents the fraction of the edges that fall within the given groups minus the expected such fraction if edges were distributed at random. It is positive if the number of edges within groups exceeds the number expected on the basis of chance. For a given division of the network's vertices into some modules, modularity reflects the concentration of edges within modules compared with the random distribution of links between all nodes regardless of modules. Avg. Degree represents the average node degree (as the sum of in-degree and out-degree). The Avg. Clustering Coeff. metric quantifies on average how close node neighbors are to being a clique (complete graph) [114]. The last column represents the number of communities found by the algorithm [67]. Results show that the 7 open source projects analyzed have a number of communities, ranging from 8 to 16.

3.2.7 Are there differences in fixing time between work teams?

Motivation. Based on the findings related to the first research question, we know of the presence of communities in JIRA developer networks. It is also of interest to analyze these communities in order to understand if and how the workload is distributed. This kind of information can be useful during issue triaging and scheduling of workload distribution. For example if we find a community resolving a high percentage of maintenance issues, then it is likely that a maintenance issue will be assigned to them, or if there is a community with a fast average issue resolution time, then this community can be assigned bug issues when the release date is imminent.

Approach. We used the developer network obtained in the first research question. For each project, we analyzed its developer communities by evaluating the average issue resolution time, the number of issues resolved, and the distribution of maintenance type and priority of fixed issues.

Findings. While JIRA Developer Communities have different average issue fixing times, the distribution of issue types and priority is similar across the communities.

Tables 3.11 to 3.17 show, for each project, the number of developers belonging to a particular community, the number of issues resolved, and the average issue fixing time.

Community Id	Community Size	# of Fixed Issues	AVG Fixing Time [Days]
6	2	1	5.6
4	20	186	159.4
13	6	2	175
3	245	2919	214.7
2	109	879	216.3
1	143	2498	242.1
12	3	2	258.2
0	90	372	260.8

Table 3.11: Derby Communities Statistics.

The first result is related to Pareto's law (20% of developers doing 80% of the issue resolution) [63]. There are only a few communities taking care of the majority of issues. These communities have a different average resolution time, and this number is independent from the community size and from the number of fixed issues.

Community Id	Community Size	# of Fixed Issues	AVG Fixing Time [Days]
9	2	1	0.7
10	5	1	10.9
6	26	221	52.3
3	141	2359	71.8
2	502	2162	86.1
0	293	2992	120.9
1	60	207	122.2
4	84	215	158.8
5	31	363	193.7

Table 3.12: Hadoop Common Communities Statistics.

We evaluated the Pearson's correlation coefficient between the average issue fixing time and the number of issues resolved, and between the average issue fixing time and the community size. We found a rather weak correlation (<0.3) with the only exception being for Hadoop Map/Reduce, in which the correlation between average issue fixing time and the community size was 0.68. This result indicates that the average issue resolution time is a property of a developer community, and it does not depend on the community size.

There is a great difference in the number of issues resolved by the communities and the average issue resolution time per community. This fact is consistent across all of the 7 projects analyzed. In order to understand how the productivity is distributed across the developer communities, we calculated, for each community, the fixed issue distribution of maintenance type and priority. Figures 3.5 and 3.6 show, for each project, the distribution of maintenance type and the priority of the issues fixed by a community. For each project, the bar-chart on the left represents the distribution of fixed issue maintenance type, and the bar-chart on the left represents the distribution of fixed issue priority.

For almost all the projects, these two distributions are similar; namely, there are no specialized communities, for example, communities solving mostly Bug with Critical priority. We can then conclude that the average issue resolution time is property of the community and does not depend on the community size, the number of fixed issues, or the maintenance type and priority.

3.2.8 Threat to validity

We now discuss the threats to validity of our study, following common guidelines for empirical studies [121]. Construct validity threats concern the relation between theory and observation. For the calculation of the issue resolution time, we have not taken into account the complexity of a given software,

Community Id	Community Size	# of Fixed Issues	AVG Fixing Time [Days]
4	3	1	10.7
6	113	1621	45.1
2	199	1803	66
1	146	528	218.5
0	234	595	317.7

Table 3.13: Hadoop HDFS Communities Statistics.

Community Id	Community Size	# of Fixed Issues	AVG Fixing Time [Days]
5	108	519	66.6
2	152	1862	70.8
7	57	84	119.4
0	113	412	128.8
1	169	1049	152.6
3	209	445	471.8

Table 3.14: Hadoop Map/Reduce Communities Statistics.

Community Id	Community Size	# of Fixed Issues	AVG Fixing Time [Days]
0	62	517	45.5
2	153	1629	47.8
6	43	378	60.2
5	91	578	64.3
1	143	1494	69.7
3	286	3215	85.3
4	163	1707	90.2

Table 3.15: HBase Communities Statistics.

Community Id	Community Size	# of Fixed Issues	AVG Fixing Time [Days]
14	2	1	0.09
9	2	1	0.9
12	2	1	3.2
2	2	1	7.3
8	4	1	14
6	2	2	14.9
1	126	1105	39.4
0	229	967	96.9
7	267	2221	97.9
3	92	383	98.1
10	2	2	100.9
5	50	113	133.2
11	2	1	379.4

Table 3.16: Hive Communities Statistics.

Community Id	Community Size	# of Fixed Issues	AVG Fixing Time [Days]
7	94	1	2.9
14	2	89	28.4
17	211	608	112.7
8	8	7	147.1
13	275	377	208.5
11	86	9	251.6
10	95	8	300.6
20	70	3	323.5
19	193	7	436.9
16	79	46	549.7

Table 3.17: Lucene - Core Communities Statistics.

or the complexity of a specific sub-system of a software. This simplification could affect our findings. Our model is a starting point, but in order to obtain more precise it would be necessary to insert in the model a complexity factor. Developers working on the core part of a e-commerce system, would need more time to solve a given issue related to a part of the system that performs payment-operations, than developers involved in the same project, but working on a web-page that visualizes an item in the basket.

Threats to internal validity concern our selection of subject systems, tools, and analysis method. With respect to the system studied in this work we considered only 7 systems hosted in JIRA.

Threats to external validity are related to generalisation of our conclusions. Our results are not meant to be representative of all projects hosted in the repository and we have analyzed only the JIRA issue-traking system.

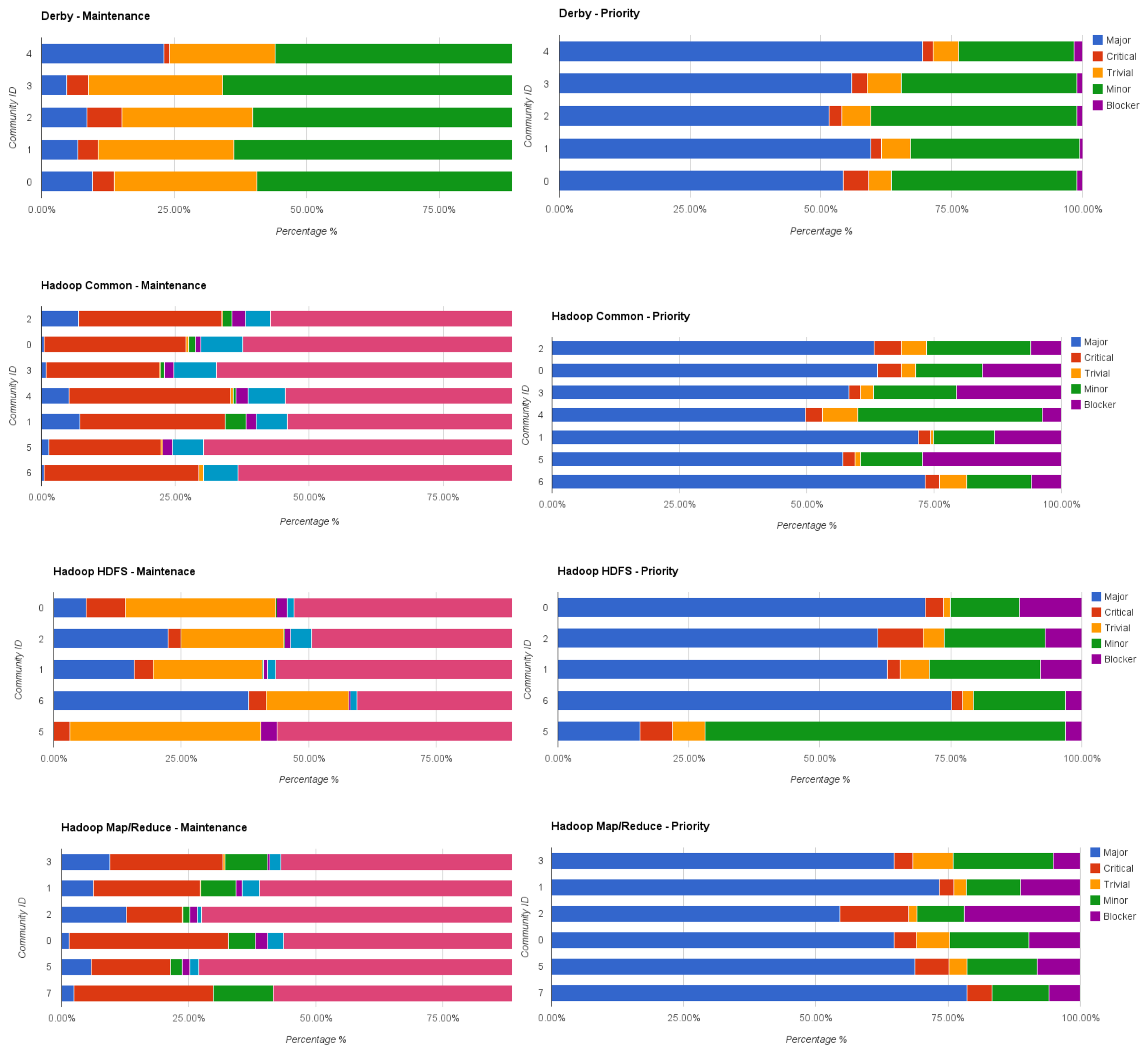


Figure 3.5: Examples of Communities Distributions of Issue's Maintenance Types and Priorities

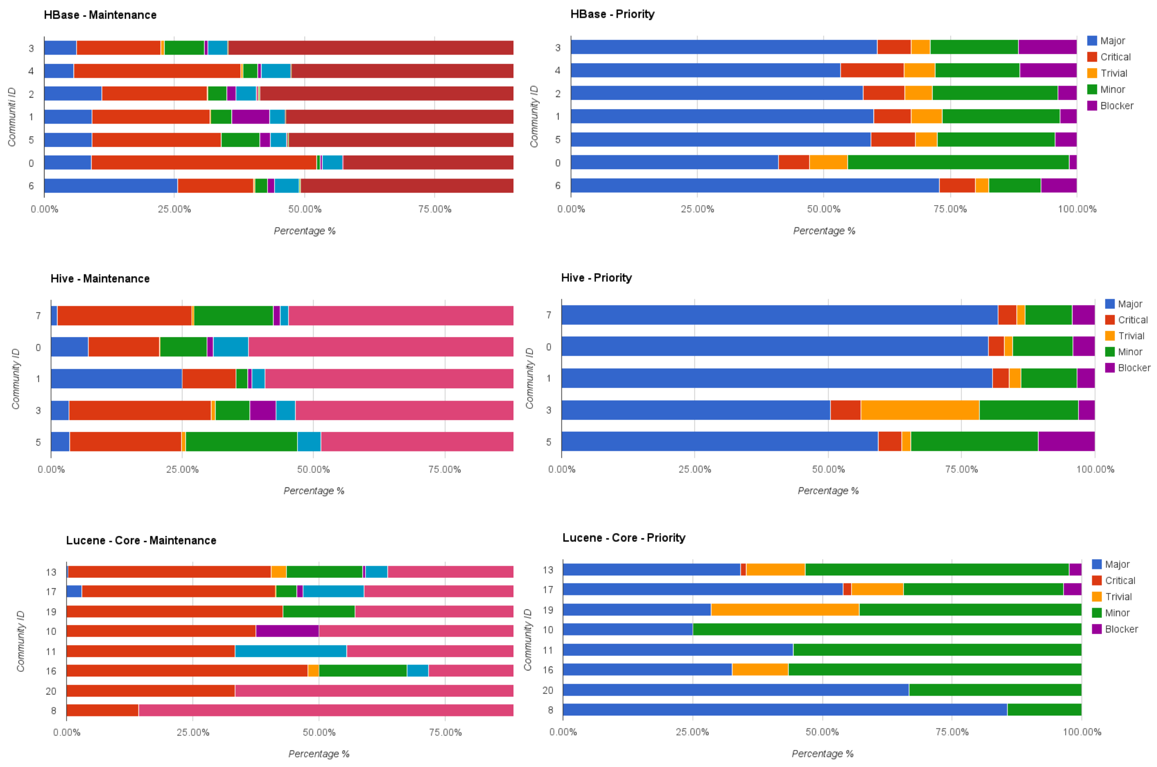


Figure 3.6: Examples of Communities Distributions of Issue's Maintenance Types and Priorities

Chapter 4

Affectiveness In Software Development

4.1 Do developers feel emotions? an exploratory analysis of emotions in software artifacts

Software development is a collaborative activity in which developers interact to create and maintain a complex software system. Human collaboration inevitably evokes emotions like joy or sadness, which can affect the collaboration either positively or negatively, yet not much is known about the individual emotions and their role for software development stakeholders. In this study, we analyze whether development artifacts like issue reports carry any emotional information about software development. This is a first step towards verifying the feasibility of an automatic tool for emotion mining in software development artifacts: if humans cannot determine any emotion from a software artifact, neither can a tool. Analysis of the Apache Software Foundation issue tracking system shows that developers do express emotions (in particular gratitude, joy and sadness), which can be identified with a certain amount of training by humans. However, the more context is provided about an issue report, the more human raters start to doubt and nuance their interpretation of emotions. Hence, more investigation is needed before building a fully automatic emotion mining tool.

4.1.1 Introduction

In July 2013, the Linux kernel mailing list was shaken up by an agitated discussion between Linus Torvalds and a senior developer [18]: "I am serious about this. Linus, you're one of the worst offenders when it comes to verbally abusing people and publicly tearing their emotions apart.” Other people joined her, noting “scolding people [...] is not likely to encourage people to want to become senior developers” and “Thanks for standing up for politeness/respect. If it works, I’ll start doing Linux kernel dev. It’s been too scary for years.” On the other hand, Linus defended himself, claiming “not telling people clearly enough that I don’t like their approach, they go on to re-architect something, and get really upset when I am then not willing to take their work.”

This example suggests that a rational view of software development only provides a partial picture of how stakeholders really behave: the developers may underperform if they do not feel safe and happy. Positive emotions like happiness help people to be more creative [42], which is essential for successful software design [19]. If not, fear, or absence of courage, could refrain developers from changing or refactoring their code [2]. These effects of emotions are similar to other domains, where people have found that feelings and emotions dictate to a large extent our actions and decisions [90]. For example, consumer opinions on retailer sites influence potential buyer decisions [87]. The mood of people, evaluated through tweets, correlates with changes in the activity of buying or selling in the stock market [14]. Since even the most talented developer could underperform and eventually leave the project just because she is unhappy with her environment or colleagues it is important to support managers and project leads in detecting emotions in their team. Mining emotions from discussion boards is relevant when face-to-face meetings are not feasible or efficient. For example, in distributed development (both open and closed source), projects have almost no personal interaction except for sporadic conference calls. In such environments, gauging emotions across geographical locations is essential for managers to become immediately aware of new problems and be able to take typical managerial action to defuse the situation. They can then organize conference calls focused on the reasons of a developer’s unhappiness (e.g., sexual harassment), or schedule a special meeting if developers suddenly became anxious (e.g., for issue due to an OS update).

Since awareness of emotions in a team currently is a manual activity, we want to use messages posted on a project’s public discussion boards (i.e., one of the major means of communication) to mine developer emotions. In particular, as a first step towards building a tool for automatic emotion mining, we performed a pilot study (with the four authors) and a full user study (with 16 participants) to determine whether emotions can actually be detected from typical software maintenance artifacts like issue reports, and, if so, whether humans can actually agree on the emotions identified and how much information (context) they need for that. Without such agreement, automated tools would not make sense to build. In particular, we analyzed a significant sample of 792 developer comments (400 in the pilot study, 392 in the full study) of the Apache projects using Parrott's emotional framework [84] to answer the following research questions:

RQ1) *Can human raters agree on the presence or absence of emotions in issue reports?*

We found that raters agree the most on the absence of an emotion, followed by the presence of `love` (i.e., gratitude) and (less strongly) `joy` and `sadness`. Having more than two raters does not significantly improve agreement.

RQ2) *Does context improve the agreement of human raters on the presence of emotions in issue reports?*

We found that providing human raters with more context about an issue seems to cause doubt (i.e., nuances) instead of more confidence in the identified emotions. Having more than two raters makes ratings more robust to this.

To our knowledge, this is the first feasibility study of emotion mining in development artifacts like issue reports. Based on our findings, issue comments have potential as data source for emotion mining, yet more work is needed to fully understand the role of context on the identification of emotions.

In the remainder of this section, we first describe the the experimental setup (Section 4.1.2), followed by a discussion up front of mined developer emotions (Section 4.1.5). We then address the two research questions (Section 4.1.6) and discuss our findings (Section 4.1.7). After a discussion of the threats to validity (Section 4.1.10) and related work (Section 5.2.1).

4.1.2 Experimental Setup

In our experiment, we analyze emotions reported in issue reports extracted from open source systems. This section discusses the data set used in our analysis, the general procedure used to rate issue report comments, and finally how pilot and full study where organized.

Dataset

We mined the issue repository of the Apache software foundation[1] as described in Sec 2.1.1, since such a type of repository contains a significant amount of

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

Projects	Issues	Comments	Users	Start Date	End Date
117	81,523	271,416	20,537	10/2000	07/2013

Table 4.1: Apache Software foundation statistics.

information related to a project's dynamics [55]. The mined issue reports belong to 117 open source projects, ranging from large, long-lived projects such as Tomcat [2], and Lucene[3], to smaller projects like RAT[4]. For the experiment, we use a random sample of the project's comments, so the larger the project the higher the chance that we analyzed some of its comments. Given this wide range of projects, we believe that this data source provides a representative overview of open source issue reports.

We parsed Apache's Jira-based repository in July 2013, fetching all the issue reports since the 19th of October 2000. For each issue report, we extracted the developers' comments, as well as the standard issue report fields mentioned in Section 4.1.2. Table 4.1 reports the statistics for our dataset. Since an issue comment can consist of multiple sentences, and an issue report of multiple issue comments, we decided to perform our analyses at the level of issue comments to avoid a too coarse granularity. Given the large number of issue comments, we sampled enough issue comments to obtain a confidence level of 95% and confidence interval of 5%. This means that a proportion of X% in our sample of issue comments actually corresponds to $X \pm 5\%$ in the population of issue comments. For this reason, we obtained samples of 392 or more comments (out of 271,416 comments). Note that we only focus on publicly available communication data (in particular, issue reports). Even in an industry setting, similar publicly available data can be found to use as input for emotion mining, without privacy concerns.

Emotion Mining

In order to rate a particular issue comment as having a particular emotion, each rater identified the emotions associated to each sentence of the comment. We use Parrott's six primary emotions, i.e., `love`, `joy`, `surprise`, `anger`, `sadness` and `fear`, since we did not know the distribution of emotions across comments beforehand. Future work could specialize emotions like `sadness` towards deeper levels. The rating process was based on (1) each rater's personal interpretation of emotions, and (2) a common understanding of Parrott's

[2]http://tomcat.apache.org/
[3]http://lucene.apache.org/
[4]http://creadur.apache.org/rat/

Table 4.2: Interpretation of Cohen and Fleiss κ values.

κ value	interpretation
<0	poor
0–0.20	slight
0.21–0.40	fair
0.41–0.60	moderate
0.61–0.80	substantial
0.81–1.0	almost perfect

framework. For the latter, we first explained and illustrated such a framework (Section 2.3.3) to all participants. Figure 2.3 shows an example comment belonging to issue 1235 of the Tomahawk project where a developer reveals his opinions about the risk of moving towards static methods (which he believes would be useless) To show his dislike, he uses wordings associated with `anger` and `fear`, interspersed with neutral phrases where the author expresses an objective evaluation of the patch. Although not shown, a sentence can express more than one emotion. Based on the sentences' individual emotions, a rater would mark the entire issue comment as containing `anger` and `fear`. We ignore Neutral annotations, since those correspond to absence of emotions.

Since we have no ground truth, each research question considers agreement on a particular comment's emotions as a "correct" classification. Agreement boils down to a majority vote of two raters out of two or three or more raters out of four agreeing on the presence or absence of a particular emotion for a given issue comment. This is of course a threat to validity, but the only objective way to decide which emotions are "correct", since retrospectively asking the author of a comment about a potentially identified emotion is bound to generate unreliable results. To measure the degree of inter-rater agreement on identified emotions, we calculate either Cohen's κ value [24] (two raters) or Fleiss' κ value [40] (more than two raters). Both values can be interpreted according to Table 4.2. In order to determine whether inter-rater agreement values differ statistically significantly, we also provide the values' corresponding confidence interval (with α value of 0.05). If this interval overlaps with another value's interval, we cannot conclude that the two agreement values are significantly different. In addition to these statistical agreement values, we also provide the more basic percentage of cases for which raters agree on a particular emotion or set of emotions.

Table 4.3: Comments assigned to person 1 of Group A (P1A) and person 1 of Group B (P1B). Their assignments for round 2 switch presence/absence of context.

ID	Group A		Group B		Round 1 (A)	Round 1 (B)
1	P1A	P2A	P1B	P2B	context	no context
2	P1A	P2A	P1B	P2B	no context	context
...	...	...	...	...	...	...
14	P1A	P2A	P1B	P2B	no context	context
15	P1A	P3A	P1B	P3B	no context	context
16	P1A	P3A	P1B	P3B	context	no context
...	...	...	...	...	...	...
28	P1A	P3A	P1B	P3B	context	no context
29	P1A	P4A	P1B	P4B	no context	context
...	...	...	...	...	...	...
98	P1A	P7A	P1B	P7B	no context	context

4.1.3 Pilot Study

The pilot study is used in RQ1 to investigate (1) the type of emotions in issue reports and (2) the level of agreement that human raters can achieve on the identified emotions. We randomly selected 400 issue report comments from the Apache issue reports. Then, we arbitrarily assigned each comment to two authors of the study, randomly making sure that authors are not paired up with the same person all the time (contrary to the full study, we did not enforce that everyone shares exactly the same number of comments with every other rater). Eventually, each author received a file containing 200 issue report comments. Each author then went through his or her list of comments to mark all Parrot's emotions that he or she was able to identify.

4.1.4 Full Study

In addition to the pilot study, we also performed a larger ("full") study to verify the consistency of the RQ1 results, as well as to investigate the influence of context on emotion rating in RQ2. A comment's context in an issue report is the list of comments of that report that were submitted before that comment. We are interested in validating whether or not knowing the context of a comment makes rating the emotions in that comment easier. If it is, then one should always consider this context when rating emotions. From the Apache issue reports we randomly selected 392 issue comments (confidence interval of

5%) that have at least one comment of context. Contrary to the pilot study, we only analyzed the closing comments of issue reports, since those have a higher chance of having context. We then selected 4 Master's students, 10 PhD students and 2 research associates from Polytechnique Montréal and University of Antwerp. We organized two groups A and B, both with the same number of master and PhD students. Table 4.3 illustrates how we assigned comments to group members. First of all, each group should rate each comment twice (both groups together rate each comment four times), while we also wanted to limit the bias caused by the wide variety in experience, nationalities and culture of participants. For these reasons, each group member rated 14 comments in common with each other group member. Second, in order to compare both groups' ratings, we mapped each member in group A to a member in group B with similar experience (for example, person 1 of group A (p1A) and person 1 of group B (p1B)). In order to compare the ratings between two groups , each couple like (p1A,p1B) received the same assignment (modulo random reordering). Third, since we want to verify the influence of context on emotion rating, we divided the experiment in two rounds where the participants rate each of the comments assigned to them twice: once without its context and once with. So, given a particular couple's assignment, we randomly added context for some of the comments in one round, while we added context for the other comments in the second round, as shown in Table 4.3. Fourth, to reduce the impact of seeing first a comment with or without context, we made it such that the assignment of p1A (after randomly adding context) for round 1 corresponds to the assignment of p1B in round 2, while the assignment of p1A in round 2 corresponds to the assignment of p1B in round 1. In each round, all participants rate 98 comments (and each group 392 comments). Finally, to counter the learning effect and at the same time obfuscate the goal of the study, the two rounds where separated by a time gap of at least 6 days in between submitting the results of the first round and starting the second round. Similar to the pilot study, in each round each participant received a file with comments. As a means of training, he or she received an explanation about the Parrot framework as well as examples of each emotion based on the results of the pilot study (Section 4.1.5). Then, the participants analyzed the list of comments to mark all Parrot emotions that they were able to identify. For comments with context, the participants rate only the emotions in the comment (not the other comments of the context). The dataset of all experiments will be available by contacting the authors[5].

[5]http://bit.ly/1g1olgq

4.1.5 Developer Emotions on Issue Comments

This section presents the typical emotions that we identified in issue reports during the pilot case. We opted to discuss these emotions up front in order to provide a better understanding of the emotional content of issue reports. For each of the six primary Parrott emotions, we report the most representative text snippets as well as an explanation of why the snippet contains that emotion. Whenever appropriate, we also report the secondary (e.g., `shame`) or tertiary (e.g., `guilt`) emotion.

`Love`

1. Thanks for your input! You're, like, awesome
2. Thanks very much! I appreciate your efforts
3. [I] would love any

`Love` is generally presented in sentences that express gratitude, i.e., a developer exhibits that he "likes" the person (example 1), the person's activity (example 2) or the software artifact delivered (e.g., patch). `Love` can be expressed also as a kind of `desire` (example 3). In issue comments, `love` is oriented primarily towards co-workers.

`Joy`

1. I'm happy with the approach and the code looks good
2. great work you guys!
3. Hope this will help in identifying more usecases

`Joy` is normally associated to positive achievements, in the form of `satisfaction` (example 1) or `enthusiasm` (example 2). In the first case, the text reports keywords like "good" or "great". In the second case, the phrase ends with a "!". A less common case of `joy` is `optimism` expressed by a positive outlook for a successful achievement (example 3). `Joy` is expressed towards software artifacts or co-workers.

`Surprise`

1. I still question the default, which can lead to surprisingly huge memory use

2. I also documented an unexpected feature with the SlingServletResolver
3. Oops. It needs to be added to Makefile

`Surprise` is expressed for unexpected, generally negative, behavior of the software (example 1 and 2). A second case is represented by mistakes introduced accidentally by a developer and discovered later on (example 3). We did not document any case where `surprise` referred to co-workers.

`Anger`

1. I will come over to your work and slap you
2. WTF, a package refactoring and class renaming in a patch?
3. This is an - ugly - workaround

`Anger` generally goes along with menaces (e.g., "slap" or "kill"), negative adjectives (e.g., "ugly") or profanity (e.g., "WTF"). These emotions reveal `hostility` and bullying towards co-workers (example 1) or `dislike` towards software artifacts (examples 2 and 3).

`Sadness`

1. Sorry for the delay Stephen.
2. Sorry of course printStackTrace() wont work
3. wish i had pay more attention in my english class .... now its pay back time .... :-(
4. Apache Harmony is no longer releasing. No need to fix this, as sad as it is.

`Sadness` is generally expressed by developers that feels `guilty`, i.e., they apologize for a delay (example 1) or for the unsatisfactory code produced (example 2). `Sadness` can be expressed also for reasons not dependent on the issue handled (example 3), or on the developer actions (example 4).

`Fear`

1. I'm worried about some subtle differences between char and Character
2. I'm most concerned with some of the timeouts
3. I suspect that remove won't work either in this case.

`Fear` is expressed by a developer in a state of `worry` or `anxiety`. This emotion is expressed explicitly using the keyword "worry" or its synonyms like "concern" (example 1 and 2). Another common case is to express a negative outlook with respect to a particular development choice (example 3). We observed that `fear`, differently from other emotions, is limited to software artifacts only.

Issue comments contain emotional content of developers. Some emotions can refer to software artifacts and co-workers (e.g., `joy`, `anger` and `sadness`), while others target only software artifacts (e.g., `surprise` and `fear`) or co-workers (e.g., `love`).

4.1.6 Experimental Results

For each research question, we first discuss its motivation, followed by the approach we used and our results.

RQ1.

Motivation. Emotion mining from software development artifacts like issue reports, emails or change logs is not trivial, since such artifacts consist of unstructured data [3, 4]. I.e., they are relatively short, written in an informal way with emoticons, and, contrary to regular text parsed in sentiment analysis, they typically contain technical content like stack traces or code snippets interleaved with regular text. Because of this, it could prove to be difficult to glance emotional content from software development artifacts, let alone agree between different human raters.

Approach. We use the comment ratings of the pilot study and full study to address this research question. As a first step, we measured the percentage of agreement on the presence and absence of emotions. As a second step, we used Cohen's κ to calculate agreement across all raters for each comment. For the full study, we measured the agreement (a) for each pair of (round, group), and (b) across both groups. In the latter case, we merge the corresponding ratings of *(round 1, group A)* with *(round 2, group B)*, and of *(round 1, group B)* with *(round 2, group A)*. Comparing cases (a) and (b) allows to evaluate whether agreement changes when having two, three or four raters for a comment.

Findings. Only in on average 46.11±5% of the comments, both raters had the same rating for all 6 emotions. Table 4.4 shows for each study the number and percentage of comments for which both raters assigned to the same

	pilot	round 1		round 2	
	study	group A	group B	group A	group B
#common	165	215	193	127	207
%common	41.25±5	54.85±5	49.23±5	32.40±5	52.81±5
#common present	20	36	36	16	40
%common present	5.00±5	9.18±5	9.18±5	4.00±5	10.00±5

Table 4.4: Percentage of comments in which raters agreed on presence or absence of all 6 emotions, as well as the number of those comments with at least one emotion present

comment agreed on all 6 emotions. The highest number of such agreement occurred for group A in round 1, while the same group obtained the lowest agreement in round 2, statistically significantly lower than in the four other cases (except for the pilot study). Since (round 1, group A) and (round 2, group B) both considered the same comments (and context), that configuration of comments and context seems more easy for raters to agree on.

Furthermore, on average for 7.47±5% of the comments for which raters agreed on all 6 emotions, at least one emotion was present. Typically, raters agreed on absence of emotions for dry comments like “committed” and “done”.

Only for `love`, the raters achieved moderate agreement, while `joy` and `sadness` obtained fair agreement. Table 4.5 and Table 4.5b show the percentage and Cohen κ values (with confidence interval) of agreement for each emotion individually, for the pilot and large study respectively. `Love` clearly obtains the highest κ agreement, corresponding to a moderate value. Except for the pilot study, `joy` and `sadness` have a strong fair agreement. `Fear`, `anger` and (especially) `surprise` only obtained poor/slight agreement. These numbers are more or less stable across the five cases, with some fluctuations. For example, group A did not have any agreement at all for `surprise` in round 2, contrary to

	%agreement (#)	%agreed presence (#)	%agreed absence (#)	lower κ	Cohen κ	upper κ
`love`	90.75±5 (363)	5.75±5 (23)	85.00±5 (340)	0.38	0.51	0.64
`joy`	69.75±5 (279)	6.50±5 (26)	63.25±5 (253)	0.11	0.19	0.27
`surprise`	96.75±5 (387)	0.00±5 (0)	96.75±5 (387)	-0.02	-0.01	0.00
`anger`	90.75±5 (363)	0.50±5 (2)	90.25±5 (361)	-0.07	0.06	0.19
`sadness`	80.75±5 (323)	3.50±5 (14)	77.25±5 (309)	0.06	0.18	0.29
`fear`	93.25±5 (373)	0.50±5 (2)	92.75±5 (371)	-0.07	0.10	0.26

(a) (round one, group A)

	%agreement (#)	%agreed presence (#)	%agreed absence (#)	lower κ	Cohen κ	upper κ
`love`	89.03±5 (349)	7.91±5 (31)	81.12±5 (318)	0.40	0.53	0.65
`joy`	86.48±5 (339)	3.06±5 (12)	83.42±5 (327)	0.10	0.24	0.38
`surprise`	89.80±5 (352)	1.28±5 (5)	88.52±5 (347)	-0.00	0.15	0.30
`anger`	90.82±5 (356)	1.28±5 (5)	89.54±5 (351)	0.00	0.17	0.33
`sadness`	93.62±5 (367)	2.04±5 (8)	91.58±5 (359)	0.16	0.36	0.55
`fear`	93.11±5 (365)	1.28±5 (5)	91.84±5 (360)	0.05	0.24	0.43

(b) Percentage of agreement (absolute number in parentheses) and Cohen κ values (with confidence intervals) for each emotion in RQ1 (full study)

	%agreement (#)	%agreed presence (#)	%agreed absence (#)	lower κ	Cohen κ	upper κ
`love`	92.35±5 (362)	5.61±5 (22)	86.73±5 (340)	0.41	0.55	0.69
`joy`	82.40±5 (323)	3.83±5 (15)	78.57±5 (308)	0.07	0.20	0.33
`surprise`	89.54±5 (351)	0.77±5 (3)	88.78±5 (348)	-0.06	0.07	0.21
`anger`	90.56±5 (355)	0.26±5 (1)	90.31±5 (354)	-0.10	0.00	0.10
`sadness`	91.58±5 (359)	3.57±5 (14)	88.01±5 (345)	0.25	0.41	0.58
`fear`	87.76±5 (344)	1.02±5 (4)	86.73±5 (340)	-0.05	0.08	0.21

(c) (round one, group B)

	%agreement (#)	%agreed presence (#)	%agreed absence (#)	lower κ	Cohen κ	upper κ
`love`	85.71±5 (336)	6.38±5 (25)	79.34±5 (311)	0.27	0.40	0.52
`joy`	82.91±5 (325)	4.08±5 (16)	78.83±5 (309)	0.11	0.23	0.36
`surprise`	90.82±5 (356)	0.00±5 (0)	90.82±5 (356)	-0.05	-0.03	-0.01
`anger`	92.86±5 (364)	0.77±5 (3)	92.09±5 (361)	-0.04	0.14	0.32
`sadness`	93.88±5 (368)	1.79±5 (7)	92.09±5 (361)	0.14	0.34	0.54
`fear`	93.62±5 (367)	0.51±5 (2)	93.11±5 (365)	-0.06	0.11	0.28

(d) (round two, group A)

	%agreement (#)	%agreed presence (#)	%agreed absence (#)	lower κ	Cohen κ	upper κ
`love`	92.35±5 (362)	3.57±5 (14)	88.78±5 (348)	0.27	0.44	0.61
`joy`	82.40±5 (323)	5.61±5 (22)	76.79±5 (301)	0.16	0.29	0.41
`surprise`	89.29±5 (350)	0.77±5 (3)	88.52±5 (347)	-0.06	0.07	0.21
`anger`	92.86±5 (364)	0.51±5 (2)	92.35±5 (362)	-0.07	0.09	0.25
`sadness`	88.78±5 (348)	3.32±5 (13)	85.46±5 (335)	0.17	0.32	0.47
`fear`	91.58±5 (359)	1.02±5 (4)	90.56±5 (355)	-0.01	0.15	0.32

(e) (round two, group B)

Table 4.5: Percentage of agreement (absolute number in parentheses) and Cohen κ values (with confidence intervals) for each emotion in RQ1 (pilot study).

most of the other cases.

The percentage of agreement for `joy` in the pilot study was the lowest, with 30.25±5% of the comments containing disagreement. However, all other emotions and cases had less disagreement than 19.25±5% (`sadness` in pilot study).

At most 7.91±5% (`love`) of the comments agreed on the presence of a particular emotion, whereas up to 96.75±5% (`surprise`) agreed on the absence of a particular emotion. Table 4.5 and Table 4.5b indeed show that most of the comments were rated as not having a particular emotion (an agreed presence of 0% means that there were no comments where an emotion was present). This is the reason why, despite the high percentage of general agreement, the corresponding κ values are low. The emotions with the lowest κ values (`fear`, `anger` and `surprise`) sometimes have only 0, 1 or 2 agreed occurrences, while the most frequently agreed emotion (`love`) had up to 31 occurrences (group A, round 1).

Only for `joy`, three raters agree significantly more on an emotion than only two raters. Table 4.6 breaks these percentages down for each individual emotion, together with the Fleiss κ value of agreement across the four raters. Agreement between all four raters obtains lower percentage values than requiring at least three raters to agree. In fact, the Fleiss κ values for four-rater agreement are in the same ballpark as for the case of two raters (Table 4.5 and Table 4.5b). However, the agreement between at least three raters overall is higher than in the case of two (or four) raters, but only in the case of `joy` there really is

(a) (round 1, group A) and (round 2, group B)

	%agreement	%agreement	lower	Fleiss	upper
	of 4 (#)	of ≥3 (#)	κ	κ	κ
love	82.91±5 (13)	94.39±5 (24)	0.48	0.49	0.50
joy	69.90±5 (5)	93.88±5 (19)	0.23	0.25	0.26
surprise	78.83±5 (0)	97.19±5 (2)	0.05	0.06	0.07
anger	85.20±5 (2)	96.94±5 (2)	0.15	0.16	0.17
sadness	84.18±5 (7)	96.17±5 (10)	0.34	0.35	0.36
fear	85.97±5 (1)	98.21±5 (7)	0.22	0.23	0.24

(b) (round 1, group B) and (round 2, group A)

	%agreement	%agreement	lower	Fleiss	upper
	of 4 (#)	of ≥3 (#)	κ	κ	κ
love	79.59±5 (9)	95.41±5 (32)	0.45	0.46	0.47
joy	68.37±5 (6)	92.86±5 (17)	0.22	0.23	0.24
surprise	81.63±5 (0)	98.21±5 (2)	0.04	0.05	0.06
anger	84.18±5 (0)	97.70±5 (1)	0.06	0.07	0.08
sadness	85.71±5 (5)	97.19±5 (11)	0.35	0.36	0.37
fear	82.91±5 (1)	97.96±5 (4)	0.12	0.13	0.14

Table 4.6: Percentage of agreement for the full study between four (2nd column) and at least three (3rd column) raters, together with the Fleiss κ inter-rater agreement and confidence intervals (4th to 6th column). In parentheses the number of analyzed cases of a particular emotion in which raters agreed on the presence of the emotion.

a significant improvement. Hence, having more than two raters does not seem to make a significant difference.

While some emotions obtain higher agreement than others, only `love`, `joy` and `sadness` obtained at least fair agreement. Although comments clearly contain emotions, raters agree the most on the absence of an emotion. Having more than two raters does not change the agreement significantly.

RQ2.

Motivation. Rating a comment without its context can be compared to eavesdropping on a group conversation and only catching the last phrase of the conversation. It is likely that the interpretation of that phrase depends on the previous discussion (i.e., context) of the conversation. For example, the sentence “yeah, right” can have a different meaning (both sarcastically and otherwise) [109], when following a sentence like “with java 8 we fix all problems” than when following “breaking backward compatibility is risky”. However, due to the technical and unstructured nature of software development artifacts, the impact of context might be different than in literary English documents. Here, we want to analyze the impact of context on agreement between raters.
Approach. This research question only considers the full study. Since each group considers 392 comments once without and once with context (they were randomly distributed across two rounds), here we merge the results of both

(a) Group A

	%agreement (#)	%agreed presence (#)	%agreed absence (#)	lower κ	Cohen κ	upper κ
love	88.78±5 (348)	7.91±5 (31)	80.87±5 (317)	0.40	0.52	0.64
	85.97±5 (337)	6.38±5 (25)	79.59±5 (312)	0.27	0.40	0.53
joy	86.22±5 (338)	3.32±5 (13)	82.91±5 (325)	0.11	0.25	0.39
	83.16±5 (326)	3.83±5 (15)	79.34±5 (311)	0.10	0.22	0.35
surprise	91.07±5 (357)	1.28±5 (5)	89.80±5 (352)	0.01	0.18	0.34
	89.54±5 (351)	0.00±5 (0)	89.54±5 (351)	-0.06	-0.04	-0.02
anger	92.86±5 (364)	1.02±5 (4)	91.84±5 (360)	0.00	0.19	0.37
	90.82±5 (356)	1.02±5 (4)	89.80±5 (352)	-0.02	0.13	0.29
sadness	95.41±5 (374)	1.53±5 (6)	93.88±5 (368)	0.15	0.38	0.60
	92.09±5 (361)	2.30±5 (9)	89.80±5 (352)	0.15	0.33	0.50
fear	93.62±5 (367)	1.02±5 (4)	92.60±5 (363)	0.02	0.21	0.41
	93.11±5 (365)	0.77±5 (3)	92.35±5 (362)	-0.02	0.15	0.33

(b) Group B

	%agreement (#)	%agreed presence (#)	%agreed absence (#)	lower κ	Cohen κ	upper κ
love	93.11±5 (365)	4.85±5 (19)	88.27±5 (346)	0.40	0.55	0.70
	91.58±5 (359)	4.34±5 (17)	87.24±5 (342)	0.31	0.46	0.62
joy	83.16±5 (326)	4.85±5 (19)	78.32±5 (307)	0.14	0.27	0.40
	81.63±5 (320)	4.59±5 (18)	77.04±5 (302)	0.10	0.23	0.35
surprise	89.80±5 (352)	0.51±5 (2)	89.29±5 (350)	-0.08	0.04	0.16
	89.03±5 (349)	1.02±5 (4)	88.01±5 (345)	-0.04	0.10	0.24
anger	92.60±5 (363)	0.26±5 (1)	92.35±5 (362)	-0.10	0.03	0.15
	90.82±5 (356)	0.51±5 (2)	90.31±5 (354)	-0.08	0.05	0.18
sadness	91.07±5 (357)	3.32±5 (13)	87.76±5 (344)	0.22	0.38	0.54
	89.29±5 (350)	3.57±5 (14)	85.71±5 (336)	0.20	0.35	0.50
fear	90.05±5 (353)	1.02±5 (4)	89.03±5 (349)	-0.03	0.12	0.27
	89.29±5 (350)	1.02±5 (4)	88.27±5 (346)	-0.04	0.10	0.25

Table 4.7: Percentage of agreement and Cohen κ values (with confidence intervals) for comments without and with context (RQ2). The percentages are relative to the 392 comments without and with context, respectively odd and even rows.

rounds such that, for each group, we can compare the ratings without and with context. For this comparison, we calculate similar agreement percentage and Cohen κ values as for RQ1. Furthermore, we measure how often raters made a different decision for a particular emotion when seeing context or not, and whether such different decisions led from agreement to disagreement, disagreement to agreement or did not have any net effect.

Findings. Adding context slightly reduces rater agreement, but not significantly. Table 4.7 compares, for each group, the agreement amongst the rating results of the comments without context (odd rows) and those with context (even rows). Except for `surprise` in group A, the κ agreement is not significantly different (the confidence intervals still overlap) with or without context, even though the actual κ values seem lower with context than without. Similarly, the percentages of agreement seem lower with context, but not in a significant way. Sometimes context finds more evidence of the presence of emotion than without context, while sometimes the inverse situation holds. Similar to RQ1, both groups have similar results, except for `anger`, for which group B had a much lower agreement (since less occurrences were agreed upon).

Most of the raters pick the same answer without or with context, yet they tend to switch more from absence to presence of an emotion than the other way around. Table 4.8 shows for both groups how many raters picked a different answer for an emotion in the absence or presence of context. Clearly, in most cases (between 90.3% to 95.7% of the time) raters did not change their rating, which suggests that (1) ratings for a particular comment are fairly stable, and (2) context does not add substantially new information for the interpretation of a particular comment. At the same time, we can also see that if a rater changes his or her mind, he or she rather tends to mark a previously (i.e., with-

context		love		joy		surprise		anger		sadness		fear	
		0	1	0	1	0	1	0	1	0	1	0	1
A	0	650	28	658	46	711	28	719	29	722	32	730	21
	1	29	77	30	50	32	13	21	15	13	17	21	12
B	0	701	18	644	36	712	28	730	23	691	32	712	25
	1	16	49	32	72	21	23	14	17	23	38	22	25

Table 4.8: How often raters changed their rating from the one in a row (comment without context) to the one in a column (comment with context). 0 means that a particular emotion was not selected, while 1 means that it was selected.

		love		joy		surprise		anger		sadness		fear	
		d	a	d	a	d	a	d	a	d	a	d	a
group A	d	26	18	26	28	9	26	13	15	5	13	6	19
	a	29	319	40	298	32	325	23	341	26	348	21	346
group B	d	15	12	42	24	19	21	16	13	17	18	19	20
	a	18	347	30	296	24	328	20	343	25	332	23	330
3-rater	d	9	8	9	8	1	9	3	8	4	8	1	7
	a	14	361	26	349	7	375	7	374	10	370	6	378

Table 4.9: How often raters went from disagreement (d) to agreement (a) or vice versa when comparing the set of comments without context (rows) to the set of comments with context (columns), for groups A, B, and when combining both groups (at least three raters agreeing).

out context) absent emotion as present, than the other way around (except for `love`, `surprise` and `fear` in group A). In the case of `sadness` the relative difference between both cases goes from 50% to more than 100%. This would suggest that although context does not play a major role in agreement, in cases when it does, raters become less sure and tend to mark an additional emotion as being present. **The change of mind due to context pushes more pairs of raters from agreement to disagreement than the other way around.** Table 4.9 shows for each comment and emotion whether the raters' change of mind has an impact on the agreement between the raters . Even though the vast majority of cases did not change agreement/disagreement, the results also show for all emotions and both groups that more raters went from agreement to disagreement when showing context (row “a”, column “d”), than the other way around (row “d”, column “a”). Again, context does not seem to have a major impact, but when it does, it causes more uncertainty (disagreement) than agreement. This observation is less pronounced when using agreement between at least three raters, as shown in the bottom two rows of Table 4.9. Only for `love` and `joy`, there is still more agreement turning into disagreement than the inverse, but for the other four emotions, the usage of three or more raters makes the results more robust to fluctuations introduced by context. Hence, even though more raters do not significantly improve agreement (RQ1), they make ratings more robust.

Context does not play a significant role in the rating of emotions in issue comments, but when it does, it seems to cast more doubt than confidence, unless more raters are used.

4.1.7 Discussion

This section discusses our findings in more detail.

4.1.8 Impact of Context

At first sight, our findings for RQ2 seem counter-intuitive: while one would expect that the addition of context strengthens agreement due to the availability of more information, we seem to observe quite the opposite, i.e., raters starting to doubt and changing their mind. Although more experiments are needed to confirm and understand this phenomenon, we briefly discuss a couple of hypotheses. The worst case scenario would be that emotion mining is so subjective and nuanced that even for humans it is impossible to correctly determine the presence of a specific emotion in an issue report. However, we believe that the truth is more subtle. For example, in RQ2 we only rated the last comment of an issue report, and reports with context contain (by definition) the viewpoint of multiple commenters, for which it is not always clear how they relate to the last commenter’s viewpoint.Consider a hypothetical example of the following three comments by three different commenters: “Class FooBar is a total waste of time, just nuke it!”, “We do have users relying on its features, I’m afraid we should fix this bug” and “I share your view, working on it”. Although the first comment clearly contains `anger` and the second one `sadness`, the third one is quite ambiguous regarding which view is shared. Without context, the comment might be neutral, while with context it might be `neutral`, `anger`, `sadness` or a combination. As such, context does not necessarily filter the set of possible emotions. On the contrary, it enriches the nuances on the emotions perceived by a rater and can lead to different interpretations. Another hypothesis is that using a simple yes/no decision as rating is too large a simplification. Maybe one should provide multiple ratings, which would allow to model uncertainty in a rating.

4.1.9 Do Emotions Matter for Issue Reports?

Our premise was that, similar to other domains, emotions could have an impact on software development activities like bug fixing or development of new features. To get an initial impression of whether this claim contains a grain of truth, we performed a short analysis with data stemming from this study and from a smaller, preliminary one made internally. Here, we check whether reports with different emotions tend to be fixed faster, have more comments or have more people following ('watching') the issue report. Our analysis uses the three most frequent emotions (`love`, `joy` and `sadness`) on which raters agree. For the full study, we included emotions with at least three agreeing raters. This yields a total of 207 comments: 73 for `love`, 62 for `joy` and 72 for `sadness` (note that the same report can feature in multiple emotions). We then looked up the corresponding issue reports' fix time, number of comments and number of watchers. We check the null hypothesis that the reports for the three emotions either have the same average fix time, number of comments or number of watchers. For this reason, we performed (non-parametric) Kruskal-Wallis tests: if the null hypothesis was rejected (α value of 0.05), i.e., at least one emotion has a different average value for one of the three measured attributes, we performed post hoc tests to determine the emotion with significantly different property values. We found a significant difference for the number of comments, i.e., reports with a comment rated as `love` tend to have a lower number of comments (median value of 5) than `joy` (median value of 7.5) or `sadness` (median value of 12). Similarly, the number of watchers of reports with a comment rated as `love` has a median value of 0 whereas for `sadness` the median value is 1, i.e., less people monitor the former reports. Although not strictly significantly different, the Kruskal-Wallis test for the fixing time of reports obtained a low p-value of 0.057, with reports containing a `love` comment taking a median number of 20 days to be resolved, compared to 53.5 for `joy` and 68.5 for `sadness`. Of course, more analysis is needed to fully investigate the link between emotions on software development, however our initial findings for number of comments, watchers and (to a lesser degree) fixing time suggest that there might indeed be a link.

4.1.10 Threats to Validity

Threats to internal validity concern confounding factors that can influence the obtained results. We assume a causal relationship between a developer's emotions and what he or she writes in issue report comments, based on empirical evidence (in another domain) [80]. Moreover, since developer communication has as first goal information sharing, removing or disguising emotions may make comments less meaningful and cause misunderstanding. Since the comments used in this study were collected over an extended period from developers not aware of being monitored, we are confident that the emotions we mined are genuine. This is also why we could not involve the authors of the comments in our study. Threats to construct validity focus on how accurately the observations describe the phenomena of interest. Rating of emotions from textual issue report comments presents some difficulties due to ambiguity and subjectivity. To reduce these threats, the authors adopted Parrott's framework as a reference for emotions. Finally, to avoid bias due to personal interpretation, in all experiments each commit was analyzed by at least two participants. Threats to external validity correspond to the generalizability of our experimental results [20]. In this study, we manually analyze a sample of issue reports belonging to 117 open source projects. We chose the projects as a representative sample of the universe of open source software projects, with different development teams and satisfying different customers' needs. Replications of this work on other open source systems and on commercial projects are needed to confirm our findings. Threats to reliability validity correspond to the degree to which the same data would lead to the same results when repeated. This research is the first attempt to manually investigate emotions of developers from issue reports, hence no ground truth exists to compare our findings. We defined the ground truth through agreement or disagreement of the raters. Since we involved 16 people with a wide variety in experience, nationalities and culture, other groups of raters might obtain agreement on different emotions and comments, possibly leading to different results. However, RQ2 showed that both groups of the full study and, to some extent, the pilot study obtained similar levels of agreement. This study is focused on text written by developers *for* developers. To correctly depict the emotions embedded in such comments, it is necessary to understand the developers' dictionary and slang. This assumption is supported by Elfenbein and Nalini's work that provided evidence that for members of the same cultural and social group it is easier to recognize emotions than for people belonging to different groups [37]. Since all the participants of this study have a background in computer science, we are confident that participants may interpret the issue comments in the same manner as the developers. We did not involve raters with different background (such as linguists or psychologists), because they may make oversights or misinterpret the terms used by developers.

4.2 Would you mind fixing this issue? An Empirical Analysis of Politeness and Attractiveness in Software Developed Using Agile Boards

A successful software project is the result of a complex process involving, above all, people. Developers are the key factors for the success of a software development process and the Agile philosophy is developer-centred. Developers are not merely executors of tasks, but actually the protagonists and core of the whole development process. This study aims to investigate social aspects among developers working together and the appeal of a software project developed with the support of Agile tools such as Agile boards. We studied 14 open source software projects developed using the Agile board of the JIRA repository. We analysed all the comments committed by the developers involved in the projects and we studied whether the politeness of the comments affected the number of developers involved over the years and the time required to fix any given issue. Our results show that the level of politeness in the communication process among developers does have an effect on the time required to fix issues and, in the majority of the analysed projects, it has a positive correlation with attractiveness of the project to both active and potential developers. The more polite developers were, the less time it took to fix an issue, and, in the majority of the analysed cases, the more the developers wanted to be part of project, the more they were willing to continue working on the project over time.

4.2.1 Introduction

According to the 8th Annual State of Agile survey report[6], "more people are recognising that agile development is beneficial to business, with an 11% increase over the last 2 years in the number of people who say agile helps organisations complete projects faster." A main priority reported by users was to accelerate time to market, more easily manage changing priorities, and to better align IT and business objectives. Agile project management tools and Kanban boards experienced the largest growth in popularity of all the agile tool categories, with use or planned use increasing by 6%. In addition, one of the top

[6]http://www.versionone.com/pdf/2013-state-of-agile-survey.pdf

five ranked tools was Atlassian JIRA[7], with an 87% recommendation.

How does one classify a software as agile? The process of defining a software as "Agile" is not simple. Over the years, a variety of tools have been developed in order to help developers, team managers and other parties involved in the development process of a software system. These tools each constitute a specific aspect of the Agile world. The Agile boards, for example, represent the central aspect of communication in the Agile philosophy. As Perry wrote [85] "the task board is one of the most important radiators used by an agile team to track their progress." The communication aspect is central and is the key to fast development. When a new developer joins a development team, the better the communication process works, the faster the new developer can become productive and the learning curve can be reduced. The know-how and the shared-knowledge of a project should always be easily accessible for the development team during the development process. Fast releases, continuos integration and testing activities are directly connected to the knowledge of the system under development and hence the communication process is crucial. Tools such as the JIRA board are a good solution to bridge the gap between open source software development and the Agile world. It is the view of many that agile development requires a physical aspect, i.e. developers working together in the same room or building, or at the same desk because the pair programming paradigm requires at least two people working simultaneously on the same piece of code, but can the developers work remotely? Is it possible to apply Agile methodologies even for open source software developed by a community which is spread out around the globe?

By using tools such as the JIRA board, it is indeed possible to apply the theoretical approach of the Agile board for a software project being developed by developers working in different physical places.

Working remotely, in different time zones and with different time schedules, with developers from around the world, requires coordination and communication. The communication process in this context becomes more difficult (if compared to the communication process used by developers sharing the same office) and the politeness, the mood and the social dynamics of the developers are important factors for the success of the project.

These days, even in the software development process, the social and human aspects of the development process are becoming more and more important. The Google style has become a model for many software start-ups. A pleasant work environment is important and affects the productivity of employees. Is politeness important in a software development process? "Politeness is the practical application of good manners or etiquette. It is a culturally

[7]https://www.atlassian.com/software/jira

defined phenomenon and therefore what is considered polite in one culture can sometimes be quite rude or simply eccentric in another cultural context. The goal of politeness is to make all of the parties relaxed and comfortable with one another." [8] The last part of the definition is what we are considering in our analysis. In this specific work we did not take different cultures into account (although developers involved in a specific project could be from all around the world); we focused on the politeness of the comment-messages written by the developers.

This study aims to show how project management tools such as Agile boards can directly affect the productivity of a software development team and the health of a software project. We studied the relationship among global project metrics (magnetism and stickiness) and affective metrics (politeness) by analysing the communication among developers. We considered 14 open source projects from the Apache Software Foundation's JIRA repositories.

This study aims to answer the following research questions:

- **Does the politeness among developers affect the issues fixing time?**

- **Does the politeness among developers affect the attractiveness of a project?**

4.2.2 Experimental Setup

Dataset

We built our dataset collecting data from the Apache Software Foundation Issue Tracking system, JIRA [9]. An Issue Tracking System (ITS) is a repository used by software developers as a support for the software development process. It supports corrective maintenance activity like Bug Tracking systems, along with other types of maintenance requests. We mined the ITS of the Apache Software Foundation collecting issues from 2002 to December 2013. In order to create our dataset, since the focus of our study was about the usefulness of Agile boards, we selected projects for which the JIRA Agile board contained a significant amount of activity. Table 4.14 shows the corpus of 14 projects selected for our analysis, highlighting the number of comments recorded for each project and the number of developers involved. We selected projects with the highest number of comments.

[8] en.wikipedia.org/wiki/Politeness
[9] https://www.atlassian.com/software/jira

Project	**# of comments**	**# of developers**
HBase	91016	951
Hadoop Common	61958	1243
Derby	52668	675
Lucene Core	50152	1107
Hadoop HDFS	42208	757
Cassandra	41966	1177
Solr	41695	1590
Hive	39002	850
Hadoop Map/Reduce	34793	875
Harmony	28619	316
OFBiz	25694	578
Infrastructure	25439	1362
Camel	24109	908
ZooKeeper	16672	495

Table 4.10: Selected Projects Statistics

Magnet and Sticky Metrics

Yamashita et al. [120] introduced the concepts of magnetism and stickiness for a software project. A project is classified as *Magnetic* if it has the ability to attract new developers over time. *Stickiness* is the ability of a project to keep its developers over time. We measured these two metrics by considering an observation time of one year. Figure 4.1 shows an example of the evaluation of Magnet and Sticky metrics. In this example, we were interested in calculating the value of Magnetism and Stickiness for 2011. From 2010 to 2012 we had a total of 10 *active*[10] developers. In 2011, there were 7 active developers and 2 of them (highlighted with black heads) were new. Only 3 (highlighted with grey heads) of the 7 active developers in 2011 were also active in 2012. We can then calculate the Magnetism and Stickiness as follows:

- *Magnetism* is the portion of new active developers during the observed time interval, in our example 2/10 (*dev_6* and *dev_7* were active in 2011 but not in 2010).
- *Stickiness* is the portion of active developers that were also active during next time interval, in our example 3/7 (*dev_1*, *dev_2*, *dev_3* were active in 2011 and in 2012).

[10]We consider active all developers that posted/commented/resolved/modified an issue during the observed time (from dev_1 to dev_10)

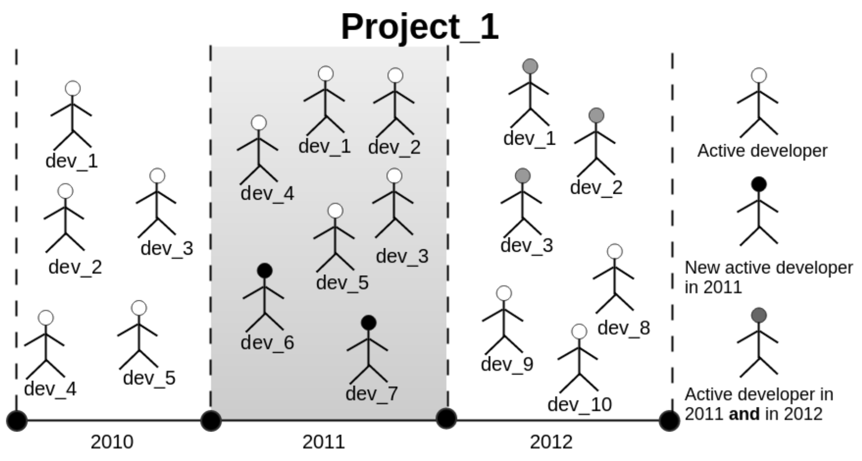


Figure 4.1: Example of Magnet and Sticky in 2011

Politeness

We measured politeness as described in Section 2.3.2 and we evaluated the average politeness per month considering all comments posted in a certain month. For each comment we assigned a value according to the following rules:

- Value of +1 for those comments marked as polite by the tool;
- Value of 0 for those comments marked as neutral (confidence level<0.5);
- Value of -1 for those comments marked as impolite.

We finally averaged the assigned values for a certain month. We analyzed the politeness of about 500K comments.

4.2.3 Result And Discussion

Does the politeness among developers affect the issues fixing time?

Motivation. Murgia et al. [72] demonstrated the influence of maintenance type on the issue fixing time, while Zhang et al. [125] developed a prediction model for bug fixing time for commercial software. There are many factors able to influence the issues fixing time; in this case we were interested in finding out if politeness expressed by developers in comments had an influence on the issues fixing time.
Approach. In order to detect differences among the fixing time of polite and

impolite issues, we used the Wilcoxon rank sum test. Such a test is non parametric and unpaired, and [102] [117] [116]. The test is non-parametric and can be used with no restrictions or hypotheses on the statistical distribution of the sample populations. The test is suitable for comparing differences among the averages or the medians of two populations when their distributions are not gaussian. For the analysis, we used the one-sided Wilcoxon rank sum test using the 5% significance level (i.e., p-value<0.05) and we compared issue fixing time between polite and impolite issues.

We grouped issues together as follows:

- we first divided comments in two sets: polite and impolite, ignoring neutral comments;

- we divided issues in two sets: polite issues, commented only with polite comments, and impolite issues, commented only by impolite comments.

- we ignored issues with both polite and impolite comments, and ignored issues with neutral comments.

For each issue we evaluated the politeness expressed in its comments (removing neutral comments as discussed in section 4.2.2) and we then divided issues in two groups: polite issues containing polite comments and impolite issues containing impolite comments. For each of this two groups of issues we evaluated the issue fixing time as the difference between resolution and creation time. **Findings**. **Issue fixing time for polite issues is faster than issue fixing time for impolite issues for 10 out of 14 analysed projects.**

Figure 4.2 shows the box-plot of the issues fixing time for the two groups of issues considered (polite and impolite) in four projects *Harmony,Derby, Hadoop HDFS* and *Hadoop Common.* The issues fixing time is expressed in hours on a logarithmic scale. As we can see for the four projects in the example, the median of the issues fixing time for polite issues is smaller than that for impolite issues.

Table 4.11 shows the Wilcoxon test results. Test's column indicates if the median of the first group (group of polite issues containing polite comments) is greater or lesser than the second group (group of impolite issues containing impolite comments).

Table 4.11 shows that for 10 of the 14 projects analysed the issues fixing time of polite issues is faster than the issue fixing time of impolite issues. *Camel* behaved differently, in this case the issues fixing time for impolite issues is faster than the issues fixing time of polite issues. Furthermore for *Infrastructure, Lucene Core* and *Cassandra* projects the *Test* value indicates that polite

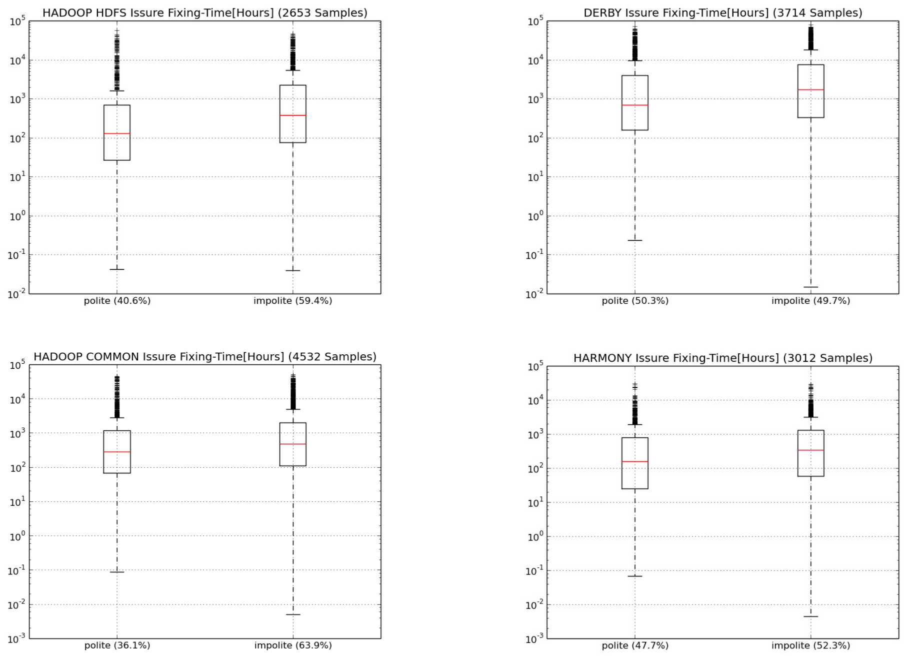


Figure 4.2: Box-plot of the fixing-time expressed in Hours. The number in parenthesis next to polite/impolite indicates the percentage of impolite and polite issues

issues fixing time is still lesser than the impolite issues fixing time but the *p-value*>0.05 and thus for these projects we cannot conclude that the two distribution are statistically different. We can see that the size effect is generally small with a maximum of 0.19 for Hadoop HDFS and a minimum of 0.007 for Infrastructure.

Project	**Test**	**p-value**	**effect size**
ZooKeeper	lesser	***	0.14
Camel	greater	***	0.089
Infrastructure	lesser	0.67	0.007
OFBiz	lesser	***	0.15
Harmony	lesser	***	0.133
Hive	lesser	***	0.061
Solr	lesser	***	0.089
Cassandra	lesser	0.51	0.012
Hadoop HDFS	lesser	***	0.192
Lucene Core	lesser	0.492	0.01
Derby	lesser	***	0.15
Hadoop Common	lesser	***	0.11
HBase	lesser	***	0.144
Hadoop Map/Reduce	lesser	***	0.11

Table 4.11: Wilcoxon test results

Figure 4.3 shows the the average politeness per month, calculated as described in section 4.2.2. We used the same four project depicted in Figure 4.2. It is interesting to note that there are variations in the average politeness over time. This is by no mean a representation of a time dynamics, but simply the representation of random variation of average politeness over time. In Hadoop HDFS for example, we can see how the average politeness is negative (namely majority of comments are impolite) for some time interval and positive of some others. As we have seen, for those projects polite issues are solved faster, so monitoring the average politeness over time can be helpful during software development. If there is a time period with a negative politeness, then the community may take action to drive the average politeness back to positive values.

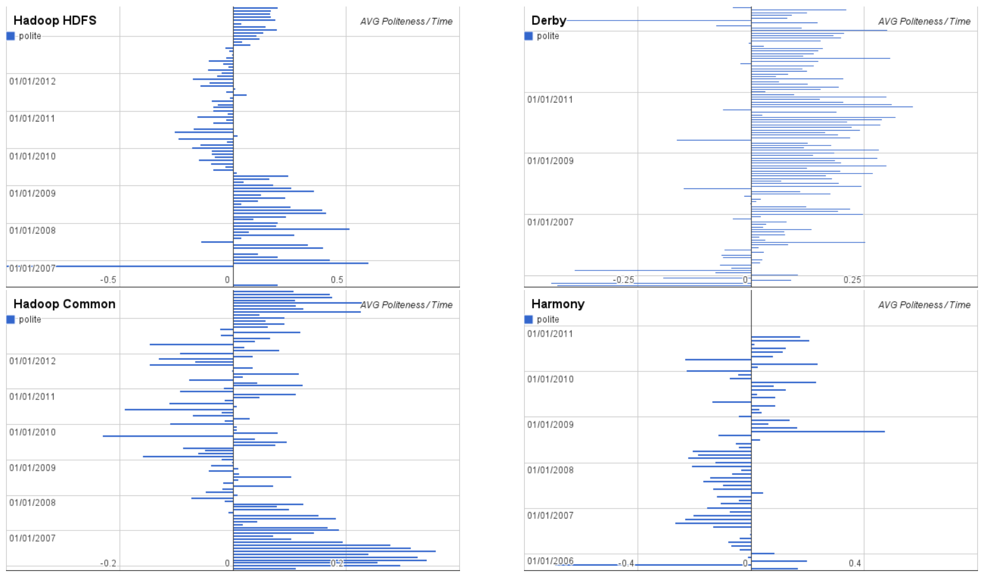


Figure 4.3: Average Politeness per month

4.2.4 Does the politeness among developers affect the attractiveness of a project?

Motivation. Magnetism and Stickiness are two interesting metrics able to describe the general health of a project; namely, if a project is able to attract new developers and to keep them over time we can then conclude that the project is healthy. On the contrary, if a project is not magnetic and is not sticky we can conclude that the project is losing developers and is not attracting new developers over time. Although there may be many factors influencing magnetism and stickiness, we were interested in analysing the correlation between politeness expressed by developers in their comments and these two metrics.
Approach. In order to detect if there was a direct correlation between magnetism and stickiness of a project and politeness, we considered an observation time of one year. During this time interval we measured magnetism, stickiness and percentage of comments classified as polite by the tool. Since we had no evidence that the politeness in the observed time could affect magnetism and stickiness in the same time interval or in the next observation time, we evaluated the Pearson's correlation coefficient and the cross-correlation coefficient.
Findings. **In the majority of projects Magnet and Sticky are positively correlated with Politeness.** Table 4.12 shows the Pearson's correlation and cross-correlation coefficient between the percentage of polite comments and magnetism and stickiness during an observation time of one year. The first two columns represent Pearson's correlation coefficient between Magnetism and

Stickiness and the percentage of politeness comments during the same observation time (one year in our case). The second two columns represent the cross-correlation coefficient between the same metrics. The Pearson's correlation revealed that 9 out of 14 project have a positive correlation between Magnetism, Stickiness and Politeness. In the 5 projects where Pearson's correlation is negative we can see that when considering the cross correlation coefficient is positive in all cases. **Although Pearson's correlation is not always positive, we can conclude that Politeness is positively correlated with Magnetism and Stickiness metrics in the subsequent years.**

Project	**Pearson's Correlation**		**Cross-Correlation**	
	Magnet	**Sticky**	**Magnet**	**Sticky**
HBase	0.672	0.667	0.581	0.667
Hadoop Common	0.848	0.641	0.848	0.641
Derby	-0.830	-0.804	0.126	0.240
Lucene Core	-0.399	0.705	0.494	0.705
Hadoop HDFS	0.716	0.526	0.716	0.627
Cassandra	0.876	0.631	0.876	0.631
Solr	0.602	0.773	0.602	0.773
Hive	0.372	0.802	0.714	0.802
Hadoop Map/Reduce	0.631	0.697	0.631	0.697
Harmony	-0.730	-0.784	0.142	0.372
OFBiz	0.692	0.498	0.692	0.498
Infrastructure	0.1	-0.112	0.479	0.610
Camel	-0.576	-0.67	0.120	0.293
ZooKeeper	-0.535	0	0.319	0.497

Table 4.12: Politeness Vs Magnet and Sticky Pearson's and Cross-Correlation Coefficient

4.2.5 Threats To validity

Threats to external validity are related to generalisation of our conclusions. With regard to the system studied in this work we considered only open source systems and this could affect the generality of the study; our results are not meant to be representative of all environments or programming languages. Commercial software is typically developed using different platforms and technologies, with strict deadlines and cost limitation and by developers with different experiences.

4.3 Are Bullies more Productive? Empirical Study of Affectiveness vs. Issue Fixing Time

Human Affectiveness, i.e., the emotional state of a person, plays a crucial role in many domains where it can make or break a team's ability to produce successful products. Software development is a collaborative activity as well, yet there is little information on how affectiveness impacts software productivity. As a first measure of this impact, this study analyzes the relation between sentiment, emotions and politeness of developers in more than 560K Jira comments with the time to fix a Jira issue. We found that the happier developers are (expressing emotions such as *JOY* and *LOVE* in their comments), the shorter the issue fixing time is likely to be. In contrast, negative emotions such as *SADNESS*, are linked with longer issue fixing time. Politeness plays a more complex role and we empirically analyze its impact on developers' productivity.

4.3.1 Introduction

Team sports like soccer [49] are a primary example that the productivity of an organization is not only a product of the talent in a team, but depends heavily on human affectiveness, i.e., the way in which individuals feel and how they perceive their colleagues [41]. A rude coach without people management skills will only alienate his team, prompting them to just do anything to avoid his scorn rather than focusing on winning the next game. Highly talented players with family issues likely have difficulties to focus on their job, while selfish, greedy or opportunistic players disrupt the harmony in a team. On the other hand, a group of medium-level players could grow into a winning squad if they enjoy working together and form a cohesive team.

Similar to sports teams, human affectiveness in software engineering has a huge impact on the abilities of a software organization [29] [53], yet the need to collaborate with remote teams (both in closed and open source development) makes the situation even more challenging [9] [22]. The fact that people do not work physically in the same location not only makes coordination of tasks more difficult, it requires them to align with colleagues and interpret colleagues' feelings through emails, discussion boards (e.g., issue tracking systems) and conference calls. The exclusive use of such systems and the absence of face to face communication could encourage developers in pursuing impolite communicative behaviour [97], which is known to detract newcomers from a project [104]. Many famous examples of this exist on the Linux kernel mailing list, for example in exchanges between the creator of the Linux kernel and some of the Linux developers [11].

In previous research [73], the authors manually analyzed whether discussion boards like bug repositories contain emotional content. They indeed found evidence of gratitude, joy and sadness, and also weak evidence that the presence of emotions like gratitude was related with faster issue resolution time. However, due to the manual nature of the analysis, the data sample was relatively limited. Furthermore, emotions are but one of the possible human affectiveness measures, and might not have the strongest relation with issue resolution time.

In this study, we empirically analyze more than 560K comments of the Apache projects' Jira issue tracking system to understand the relation between human affectiveness and developer productivity. In particular, we extract affectiveness metrics for emotion, sentiment and politeness, then build regression models to understand whether these metrics can explain the time to fix an issue. We aim to address the following research questions:

RQ1: ***Are emotions, sentiment and politeness correlated to each other?*** The considered affective metrics have a weak correlation with each other. **RQ2:** ***Can developer affectiveness explain the issue fixing time?*** Affective metrics are significant for explaining the issue fixing time. Our logistic regression model has a Precision of 0.67 and a Recall of 0.671 against 0.319 and 0.565 for a Zero-R baseline model. **RQ3:** ***Which affective metrics best explain issue fixing time?*** Sentiment and emotions such as *JOY* and *LOVE* have a positive effect on the issue resolution time (i.e., they take longer) whereas negative sentiment and emotions such as *SADNESS* have a negative impact on the issue resolution time (i.e., they take less time). Issue average politeness has a negative impact on the issue fixing time.

The rest of the section is organized as follows: we first discuss related work (Section 5.2.3). In Section **??**, we describe how we measure affectiveness by measuring emotions, sentiment and politeness in developers' comments. Section 4.3.4 introduces the Apache projects' Jira Issue Tracking System dataset and our methodology. In Section 4.3.5 we present and discuss our findings, followed by a discussion of threats to validity in Section VI.

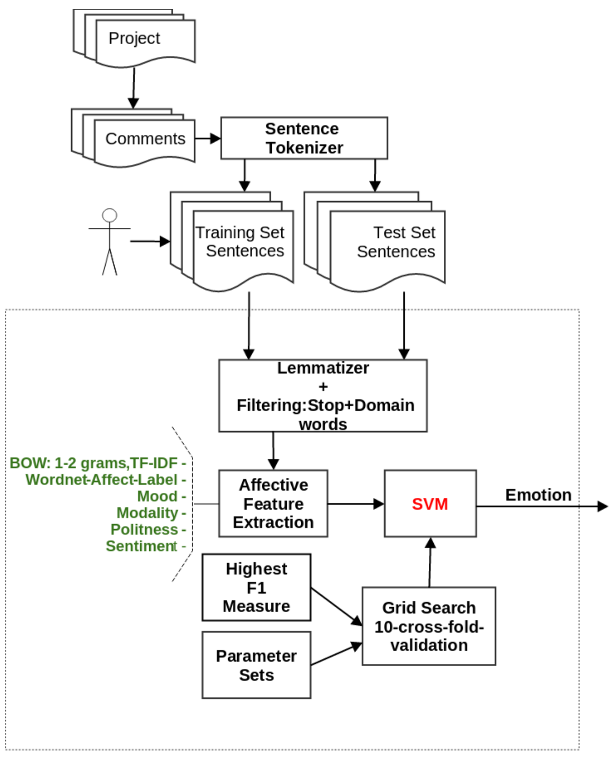


Figure 4.4: Emotion Classifier Architecture

Emotion	Accuracy	Precision	Recall	F1
ANGER	0.770	0.746	0.737	0.736
JOY	0.892	0.788	0.733	0.746
SADNESS	0.855	0.847	0.798	0.812
LOVE	0.881	0.798	0.772	0.775

Table 4.13: Emotion classifier performance

4.3.2 Measuring Emotions

While sentiment is a measure of positive or negative emotion expressed in a given text relative to some topic, emotions are more fine-grained and relate to a particular *emotional state*. This corresponds to a variety of human feelings such as *LOVE* or *ANGER*. Different emotion framework exists, which decompose emotions into a basic set of emotions. Similar to Murgia et al. [73], we used Parrott's emotional framework, which consists of six basic emotions: joy, sadness, love, anger, sadness, and fear. Despite conceptual frameworks like Parrott's Framework, to the best of our knowledge there is no available emotion analysis tool such as the ones available for measuring sentiment and politeness. For this reason, we built a machine learning classifier able to identify the presence of four basic emotions: *JOY*, *LOVE*, *ANGER* and *SADNESS* (these are the most popular emotions identified by Murgia et al. [73] in issue comments). Figure 4.4 shows the emotion classifier's architecture.

As input, the classifier requires all comments posted on a project's issue tracking system. For each comment, we used a sentence tokenizer [12] that divides a comment into sentences. For each sentence, we applied a classic text preprocessing approach, removing all the stop words and the domain words. Developers' comments often contain code, such as code snippets or stack traces, and in order to remove this text (which is irrelevant for emotion detection), we filtered out non-English words within a sentence using Wordnet[13]. The output of the *Lemmatizer* block is a vector containing all the words of a sentence. We enhanced each sentence vector considering the bi-grams (all individual words and all pairs of consecutive words) before performing the affective feature extraction. Using bi-grams is useful for considering negation such as *"don't like"*, which would not be considered using single words.

The *Affective Feature Extraction* block then extracts the following affective

[11]http://arstechnica.com/information-technology/2013/07/linus-torvalds-defends-his-right-to-shame-linux-kernel-developers/

[12]http://nlp.stanford.edu/software/tokenizer.shtml

[13]http://wordnet.princeton.edu/

features:

- Affective labels: we used the Wordnet Affect label [106] to obtain an affective label [14] for each sentence's words.
- Mood: we used the tool based of De Smedt et al. [31] to measure the grammatical mood, i.e., the presence of auxiliary verbs (e.g., could, would) and adverbs (e.g., definitely, maybe) that express uncertainty.
- Modality: we used the same tool to measure the degree of uncertainty expressed in a whole sentence.
- Sentiment: the sentence's sentiment measured using Sentistrength.
- Politeness: the sentence's politeness measured using Danescu et al.'s tool [30].

For each of the four emotions, we built a dedicated Support Vector Machine classifier, since this kind of classifier has proven to be particularly suitable for text classification. It has several parameters and we used a grid search algorithm [15] using the F1 score [16] in order to find the optimum tuning configuration. We used a manually annotated corpus of comments and their emotion for training the machine learning Classifiers, one for each emotion. The training set consisted of 4000 sentences (1000 for each emotion), which was manually annotated by three raters having a strong background in computer science (Elfenbein et al. [37] provided evidence that for members of the same cultural and social group it is easier to recognize emotions than for people belonging to different groups).

A sentence was marked as containing a particular emotion if at least two out of three raters marked the presence of that particular emotion. If not, the sentence was marked as not having that emotion (and also added to the training set). We validated our emotion classifier using Bootstrap validation with 1000 iterations [17]. Bootstrap validation splits a dataset in training and test set according to a given ratio (we used 90% training - 10% testing) and generates N sets (1000 in our case) uniformly sampled with replacement from the initial dataset. This technique yields more stable measures of accuracy precision and recall, compared to other validation techniques such as cross-validation or

[14]An affective-label is a label assigned to a word and its synonyms that indicates the emotional state of that word. For example, the word "sad" has X and Y as affective label, see http://wndomains.fbk.eu/wnaffect.html

[15]http://en.wikipedia.org/wiki/Hyperparameter_optimization

[16]http://en.wikipedia.org/wiki/F1_score

[17]http://en.wikipedia.org/wiki/Bootstrapping_(statistics)

leave-one-out validation. Table 4.13 shows the performance obtained during bootstrap for each of the four machine learning classifiers. The models obtained a very high performance on the annotated corpus of comments. Given the (still) limited size of the training set, this may be due to some degree of overfitting. However, for emotions like LOVE and SADNESS, the most influential words used by the classifiers are "thanks" and "sorry", which are extremely common words across issue comments. In that sense, the models are relatively general. Since these models are a first attempt to design an emotion classifier, we decided to adopt the models in our study. Future research should focus on enhancing emotion classification.

4.3.3 Case Study Setup

Dataset

We used the Jira dataset described in Sec. 2.1.1, selecting issue reports from projects of the Apache Software Foundation, since Apache is one of the most studied software ecosystems [110]. An Issue Tracking System (ITS) is a repository used by software developers as support for corrective maintenance activities like Bug Tracking, along with other types of maintenance requests. We mined the ITS of the Apache Software Foundation, collecting issues from 2002 to December 2013. Table 4.14 shows the corpus of 14 projects selected for our analysis, highlighting the number of comments recorded for each project and the number of developers involved. We chose the top 14 projects with the highest number of comments since our focus is to measure the affectiveness expressed in developers' comments. However, our corpus still contains popular projects such as Lucene and Hadoop.

4.3.4 Experiment Design

In order to evaluate the impact of affective metrics on the issue fixing time we designed our experiment as follows. We built a logistic regression model[18] for classifying the issue fixing time as short or long based on a set of independent variables characterising Jira issues [100]. The output of the logistic regression model, given the metric values of a particular issue, is the probability of the issue to be fixed in a short or long time. One then needs to select a threshold probability above which the logistic outcome is interpreted as "long fixing time". Since the logistic regression model has a binary output, we had to transform the numeric issue fixing times of Jira into a binary value, with 1 meaning

[18]http://en.wikipedia.org/wiki/Logistic_regression

Project	# issues	# comments	# developers	issues' average # comments	issues' average # commenters
HBase	9353	91016	951	9.73	2.93
Hadoop Common	7753	61958	1243	7.99	2.98
Derby	6101	52668	675	8.63	2.74
Lucene Core	5111	50152	1107	9.81	2.96
Hadoop HDFS	4941	42208	757	8.54	2.9
Cassandra	6271	41966	1177	6.69	2.54
Solr	5086	41695	1590	8.19	3.18
Hive	5124	39002	850	7.61	2.8
Hadoop Map/Reduce	4747	34793	875	7.32	2.74
Harmony	6291	28619	316	4.54	2.22
OFBiz	5098	25694	578	5.04	2.23
Infrastructure	6804	25439	1362	3.60	1.95
Camel	6147	24109	908	3.92	1.76
ZooKeeper	1606	16672	495	3.32	1.87

Table 4.14: Statistics of the selected projects (developers correspond to the Jira users that are involved in a project, i.e. *committers*, *issue reporters* and *comment posters.*)

that the issue fixing time will be *longer* than the issue fixing time median, and zero meaning *shorter* than the median.

As independent variables, we considered a set of *control metrics* as control variables for our case study, and a set of *affective metrics* as controlled variables. Table 4.15 shows the considered metrics. The controlled variables are the issue characteristics proposed by Giger et al. [44] as listed in the first half of Table 4.15. These control metrics cover all dimensions of Giger et al.'s work [44]. In particular, Giger et al. found that assignee and reporter experience have the strongest influence on bug fixing time. The second set of independent variables, i.e., the controlled variables, are different variations of the three affectiveness metrics of Section **??** that we deemed related to issue fixing time (these variations are non-exhaustive).

Instead of building one model with all metrics at once, we used a hierarchical modelling approach where one metric at a time is added, a model is built, then the model is compared using an ANOVA test to the previous model (without that metric) to check whether the addition of the metric leads to a statistically significant improvement of the model. We then considered in our final model, only those metrics that were significant, i.e., those metrics with a *p-value* <0.01 (marked with ** or ***). The significant metrics are shown in bold in Table 4.16.

Finally, we evaluated the impact of each metric in the model as shown in Figure 4.5, using the general approach proposed by Shihab et al. [100]:

- First, we gave as input to the logistic regression model the median values of each metric, since those values represent a "common" value for the metric. The corresponding output probability is called *baseline output.*

		Control Metrics	
metric	Type	Range	Description
reporter previous # comments	Number	>=0	# comments previously posted by the issue reporter
assignee previous # comments	Number	>=0	# comments previously assigned to the issue assignee
issue priority	Category	TRIVIAL..CRITICAL	The priority assigned to the issue (Major, Minor, Critical etc.)
issue type	Category	BUG..NEW_FEATURE	The issue maintenance type (Bug, New Feature, Task etc.)
issue # watchers	Number	>=0	The number of Jira users watching the issue
issue # developers	Number	>=0	The total number of Jira users that commented on an issue, including reporter and assignee
issue # status changes	Number	>=0	The total number of times an issue has been changed (such as changing status, resolution, type, priority etc.)
issue # comments	Number	>=0	The total number of comments posted on an issue report
		Affective Metrics	
metric	Type	Range	Description
issue avg sentiment	Number	[0,1]	The average sentiment expressed in the issue comments
issue avg politeness	Number	[0,1]	The average politeness expressed in the issue comments
issue love comments proportion	Proportion	%	The percentage of issue comments expressing love emotion
issue joy comments proportion	Proportion	%	The proportion of issue comments expressing joy emotion
issue sadness comments proportion	Proportion	%	The proportion of issue comments expressing sadness emotion
issue anger comments proportion	Proportion	%	The proportion of issue comments expressing anger emotion
issue title sentiment	Number	[0,1]	The sentiment expressed in an issue's title
issue title politeness	Number	[0,1]	The politeness expressed in an issue's title
issue first comment sentiment	Number	[0,1]	The sentiment expressed in the issue's first comment
issue first comment politeness	Number	[0,1]	The politeness expressed in the issue's first comment
issue last comment sentiment	Number	[0,1]	The sentiment expressed in the issue's last comment
issue last comment politeness	Number	[0,1]	The politeness expressed in the issue's last comment

Table 4.15: Metrics used in our study

Feature	**z-value**	**p-value**
assignee # previous comments	-19.322	<2e-16 ***
reporter # previous comments	-0.933	<2e-16 ***
issue priority:Critical	7.194e-02	5.94e-09 ***
issue priority:Major	12.263	< 2e-16 ***
issue priority:Minor	14.200	< 2e-16 ***
issue priority:Trivial	6.687	2.28e-11 ***
issue type:Bug	-1.230	0.218550
issue type:Improvement	-0.872	0.383073
issue type:New Feature	-0.415	0.677798
issue type:Sub-task	-1.050	0.293538
issue type:Task	-0.621	0.534872
issue type:Test	-1.277	0.201539
issue type:Umbrella	1.136	0.256108
issue type:Wish	0.049	0.961256
issue # watchers	3.590	0.000330 ***
issue number of developers	27.559	< 2e-16 ***
issue number of changes	40.329	< 2e-16 ***
issue avg sentiment	-5.594	2.22e-08 ***
issue avg politeness	11.485	< 2e-16 ***
issue avg love	-16.329	< 2e-16 ***
issue avg joy	-9.099	< 2e-16 ***
issue avg sadness	14.388	< 2e-16 ***
issue avg anger	-0.212	0.831741
issue title sentiment	2.884	0.003922 **
issue title politeness	3.512	0.000444 ***
issue first comment sentiment	1.676	0.093723 .
issue first comment politeness	2.108	0.035053 *
issue last comment sentiment	4.839	1.30e-06 ***
issue last comment politeness	-9.843	< 2e-16 ***

Table 4.16: Coefficient and p-values for the metrics of the logistic regression model. Metrics in bold are significant to the model.

- One metric at a time, we add one a standard deviation to the considered *metric k* leaving all other metrics unchanged on their median values. This yields a probability that we call *metric k output*.
- For each *metric k*, we calculated the relative increase of the *metric k output* relative to the *baseline output*, i.e., (*metric k output*–*baseline output*)/*baseline output*.
- We can then compare the relative increase of each metric to determine the metric with the largest impact (relative increase), as well as the sign of the increase (positive/negative), independent of the unit/type of the metric. For categorical metrics, we used the mode (most frequently used value) instead of the median.

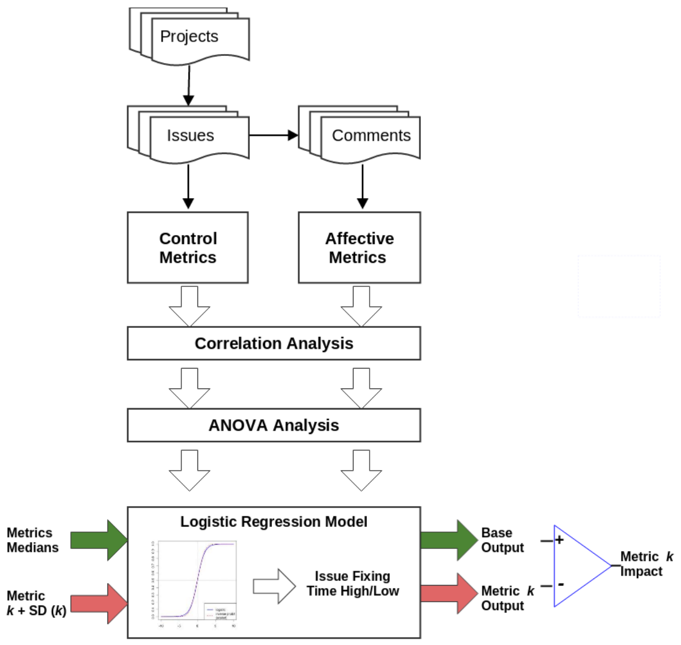


Figure 4.5: Experiment Schema

4.3.5 Results

RQ1: Are emotions, sentiment and politeness correlated to each other?

Motivation. Our final goal is to understand the impact of affectiveness on the issue fixing time. For this purpose, we build a regression model using affective metrics in RQ2. However, since all affective metrics measure something

about the feelings of stakeholders we first need to understand whether sentiment, emotion and politeness are really independent measures, or if there is overlap between them, in which case we should filter out some of the metrics.
Approach. In order to evaluate the correlation between the considered affective metrics, we measured the sentiment, emotions and politeness of developer comments using metrics in Table 4.15, considering only issues with at least two comments. For each issue, we used the love/joy/sadness/anger comment proportion, average politeness and sentiment per issue considering all comments posted on the same issue. We first calculated for each issue comment a politeness value according to the following rules:

- Value of +1 for those comments marked as polite by the tool;
- Value of 0 for those comments marked as neutral (confidence level<0.5);
- Value of -1 for those comments marked as impolite.

Then we averaged the assigned politeness across all comments, obtaining a number in a range from -1 to 1. We finally normalize the average issue politeness in a range from 0 to 1.

Similar to the average issue politeness, we evaluated the average issue sentiment measuring for each comment of an issue, the sentiment using SentiStrenght. As described in Sec. 2.3.1, SentiStrenght yields a value in a range from -5 to 5. Averaging all comments' sentiments we obtain the issue average sentiment as a number in the range from -5 to 5, which we normalize again in a range from 0 to 1. After normalization, issue with average sentiment and politeness 0 means respectively extremely impolite and negative (sentiment), 0.5 means neutral politeness and sentiment and 1 extremely polite and positive (sentiment).

We calculated the emotion proportions, average sentiment and politeness of about 560K comments (about 68K issues) then computed the Pearson correlation coefficient among all the considered metrics, except for the non-numeric issue type and priority [44]. 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.
Findings. Weak correlation exists between *issue average politeness* and *issue first comment politeness*, and between *issue last comment politeness* and *issue last comment sentiment* . Table 4.17 shows the correlations larger than 0.3. The affective metrics have a maximum weak correlation of 0.36 between the *issue average politeness* and *issue first comment politeness*. Some of the control metrics instead have a moderate to strong correlation with a maximum value of 0.7 between *issue # developers* and *issue # comments*. Given the strong correlation between *issue # developers* and *issue # comments*, we considered all metrics except *issue # comments* in the remainder of our analysis.

	issue average politeness	reporter # previous comments	issue # watchers	issue last comment sentiment	issue # changes	issue # developers
assignee # previous comments	n.s	**0.49**	n.s	n.s	n.s	n.s
issue first comment politeness	**0.36**	n.s	n.s	n.s	n.s	n.s
issue last comment politeness	n.s	n.s	n.s	**0.36**	n.s	n.s
issue # developers	n.s	n.s	**0.55**	n.s	**0.48**	n.s
issue # comments	n.s	n.s	**0.48**	n.s	**0.67**	**0.7**

Table 4.17: Weak and moderate correlations in our dataset (RQ1)

RQ2: Can developer affectiveness explain the issue fixing time?

Motivation. Productivity is an important factor for a software organization to be successful, i.e., achieving shorter time to market, for this reason understanding the factors that impact software productivity is crucial during software development. Although there are many factors that impact the issue fixing time [44], there is little information about the impact of developers's affectiveness on the issue fixing time. In this RQ, we investigate a possible relation between the affective metrics for emotions, politeness, and sentiment with issue fixing time.

Approach. As explained in Section 4.3.4, we used the metrics in Table 4.15 to build a logistic regression model for explaining the issue fixing time.

Findings. Affective metrics are significant for the explanation of the issue fixing time. Our logistic regression model has a Precision of 0.67 and Recall of 0.671 against respectively 0.319 and 0.56 for the ZeroR model. Table 4.16 shows how significant the metrics are for the logistic regression model. We considered significant all metrics with a *p-value*<0.01. As expected, the control metrics such as the *issue priority, issue reporter/assignee previous comments* and the *issue number of developers/changes* are significant. However, more interesting is that affective metrics such as the *issue percentage of emotion x* and *issue average politeness/sentiment* are significant.

To calculate the total performance of the model, we chose only the metrics from Table 4.16 that are significant (*p-value*<0.01), then built a final logistic regression classifier. Table 4.18 shows a comparison between the classification performance of our logistic regression model and a ZeroR classifier. The latter is a baseline model that always answers the same output ("long"), and often is used as a baseline to compare a model to (models performing worse are not worth the effort). By definition, the ZeroR model has perfect recall for "Long", but its precision suffers, and recall for the "Short" class is zero, which results in an average weighted precision and recall (across both classes) of 0.319 and 0.565 respectively. On the other hand, our model obtains good precision and recall for both classes, resulting in a much higher average precision and recall. The precision, recall and AUC of our model are comparable to those obtained by Giger et al. [44] and are better than the precision and recall of the ZeroR classifier. AUC is the area under the receiver operating characteristic curve. It can be interpreted as the probability that, when randomly selecting a positive ("Long") and a negative ("Short") example the model assigns a higher score to the positive example [74]. For a random model, this probability would be 0.5, which is the AUC obtained for the ZeroR model in our case. Our logistic model obtains an AUC value higher than 0.5, better than random. Furthermore, we can see how adding the affective metrics to our model, precision, recall and

Classifier	Class	Precision	Recall	F1	AUC
ZeroR	Short	0	0	0	0.5
	Long	0.565	1	0.722	
	Weighted Avg.	0.319	0.565	0.408	
Logistic **without** affective metrics	Short	0.602	0.6	0.601	0.715
	Long	0.69	0.7	0.695	
	Weighted Avg.	0.655	0.656	0.655	
Logistic **with** affective metrics	Short	0.626	0.607	0.616	0.734
	Long	0.704	0.72	0.712	
	Weighted Avg.	0.67	0.671	0.67	

Table 4.18: Logistic regression model performance

AUC are all increased.

4.3.6 RQ3: Which affective metrics best explain issue fixing time?

Motivation. We found that the affective metrics are significant for the logistic regression model that we built, as shown in Table 4.16. Since not all are equally influential in a regression model, we now are interested in quantifying which metrics have the strongest link with issue fixing time. In particular, are affectiveness measures as important as traditional issue-related measures?

Approach. In order to understand the impact of affective metrics, we evaluated the impact of each metric on the logistic regression model as described in Sec. 4.3.4.

Findings. Positive sentiment and emotions such as *JOY* and *LOVE* have a negative effect (i.e., reduce issue fixing time) on the issue resolution time whereas negative sentiment and emotions such as *SADNESS* have a positive impact on the issue resolution time. Issue average politeness has a positive impact on the issue fixing time.

Table 4.19 shows the relative increase in the logistic regression *baseline output* when fixing all metrics but one on their median values and adding one standard deviation to one metric's median value. The two control metrics *issue number of developers* and *issue number of changes* have the highest impact (>100%): the more developers involved or changes being made, the longer the fixing time. In contrast, the *issue assignee/reporter previous comments*, which are a measure of developer experience, have a negative impact on the issue fixing time, i.e., the more the issue's assignee or reporter is experienced the more likely the issue fixing time will be shorter.

Apart from the above control variables, some affective metrics also have a significant impact. The more polite an issue's last comment is, the more likely the issue fixing time was shorter. Similarly, the issue average sentiment impact is -10.52%, which means that the more positive the average sentiment is, the faster an issue is fixed. *JOY* and *LOVE* have a negative impact of -26.42% and -50.19% respectively, whereas the *SADNESS* emotion has a positive impact of 38.49%. In other words, SADNESS is linked with longer issue fixing time, whereas JOY and LOVE are linked to shorter fixing times.

Feature	**% of increment of logistic reg. output when the adding one SD**
issue # changes	192.09%
issue # developers	134.23%
issue average politeness	49.76%
% sadness comments	38.49%
issue last comment sentiment	13.72%
watchers	10.92%
issue reporter prev. comments	-9.18%
issue avg sentiment	-10.52%
% joy comments	-26.42%
issue last comment politeness	-29.10%
% love comments	-50.19%
assigne # previous comment	-54.45%

Table 4.19: Metrics impact on issue fixing time. Affective metrics are highlighted in bold.

Similar to the *% of sadness comments*, the *issue's average politeness* increases the likelihood of a long issue fixing time by 49.76%. This result is somehow unexpected. One would expect that the more developers communicate in a polite way, the more they are able to be productive. We discuss the impact of politeness in the next section.

4.3.7 Discussion

This section investigates in more detail the role played by the *issue's average politeness*, since it is somehow unexpected that the issue average politeness is related to longer issue fixing time. To enable a deeper analysis, we distinguished between three groups of issues:

- *High-Politeness*: issues with average politeness 1.
- *Medium-Politeness*: issues with average politeness in the range]0,1[. This category corresponds to issues that are more or less neutral.
- *Low-Politeness*: issues with average politeness 0.

We use box plots and hexbin plots [19] to understand how the issue fixing time is distributed across these three categories.

Figure 4.6 shows the box plot in logarithmic scale of the issue fixing time for the three categories of average politeness considered. Issues with *Low-Politeness* and *high-Politeness* have the shortest fixing time, containing respectively 38.8% and 10.4% of the total number of issues. This finding is further confirmed by the hexbin plot of Figure 4.7, where we can see that for *Medium-Low-Politeness* the majority of issues are shifted up towards higher values of issue fixing time compared to Low- and High-Politeness. In other words, the extreme cases of politeness, both in positive and negative sense, are linked with faster fixing time compared to more neutral cases. Such a non-linear link between an independent variable and the dependent variable cannot be captured by a logistic model, which is why the model suggested in RQ2 that higher politeness is linked with longer issue fixing time (since the median fixing time of High-Politeness is slightly higher than for Low-Politeness). This finding for High-Politeness confirms the findings of Ortu et al. [77].

What is still unclear is why the extreme cases have lower fixing times. One plausible reason for Low-Politeness issues (which captures 38.8% of all issues, i.e., the majority of extreme politeness cases) is such issues quickly conclude an issue because of the negative or positive tone of the comments. Alternatively, issues of the extreme politeness cases (positive and negative) might have attracted more participants, resulting in more discussion and hence longer fixing time.

Figure 4.8 shows that *Low-Politeness* issues indeed have the lowest number of sentences with Medium- and High-Politeness containing most of the sentences. In other words, negative discussions seem to conclude with less discussion.

[19]A hexagon bin plot is a kind of scatterplot where instead of individual dots for each data point, all data points in a hexagonal area are collapsed and the color of the hexagon shows how many data points are in that area. Hexbin plots are very informative in cases where many data points would overlap and one would not know how many points are overlapping.

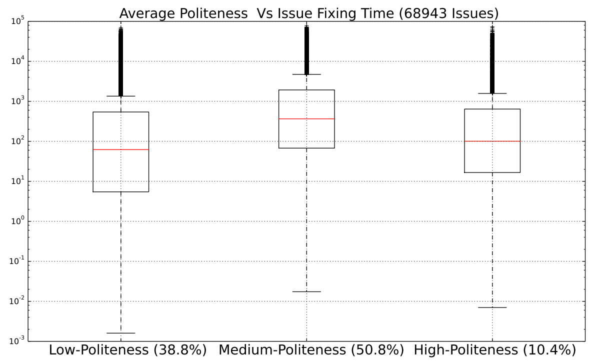


Figure 4.6: Average Issue Politeness versus Issue Fixing Time Boxplot

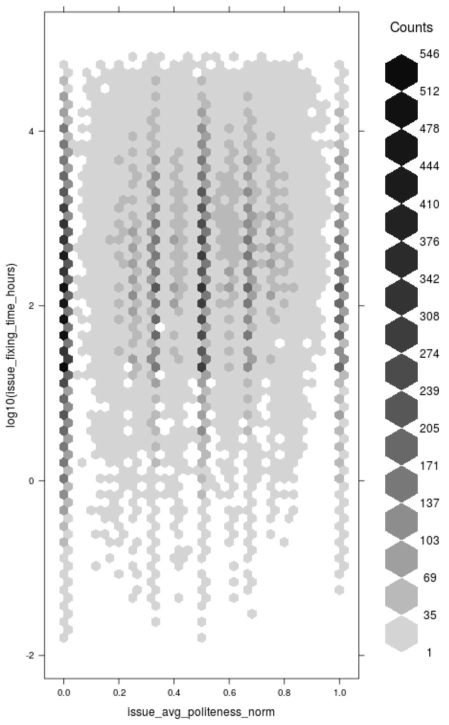


Figure 4.7: Average Issue Politeness versus Issue Fixing Time Hexbin Plot

Furthermore Figure 4.9 shows the box plot of *issue # developers* for the three categories of average politeness. Here, the extreme politeness cases both have the lowest number of participants, with a median value of 2 developers. *Medium-Politeness* issues have a median value of the *issue # developers* of 4. Taken together, issues with extreme politeness involve less developers and (at least for negative politeness) have shorter comments, both of which could provide part of the reason why their issue fixing time is shorter. More research is needed to fully understand these observations.

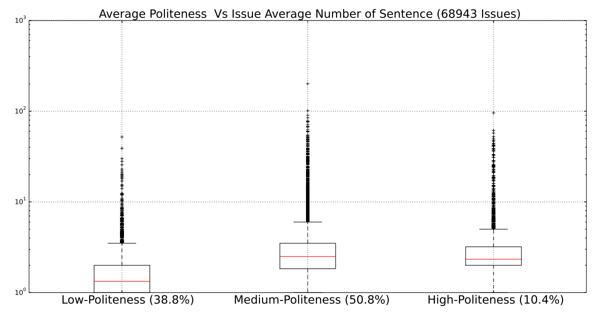


Figure 4.8: Distribution of Average Politeness versus Average Number of Sentences for the three groups of issues.

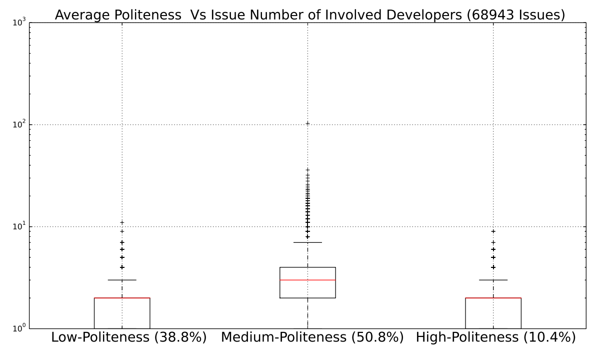


Figure 4.9: Distribution of number of developers versus Politeness for the three groups of issues.

4.3.8 Threats To Validity

Threats to internal validity concern confounding factors that can influence the obtained results. We assume a causal relationship between a developer's emotional state and what he or she writes in issue report comments, based on empirical evidence (in another domain) [80]. Moreover, since developer communication has as first goal information sharing, removing or disguising emotions may make comments less meaningful and cause misunderstanding. Since the comments used in this study were collected over an extended period from developers not aware of being monitored, we are confident that the emotions we mined are genuine. This is also why we could not involve the authors of the comments in our study. That said, we do not claim any causality between any of our metrics and the issue fixing time. We mainly built an explanatory model to understand the characteristics of issues with short and long fixing time.

Threats to construct validity focus on how accurately the observations describe the phenomena of interest. Mining of emotions from textual issue report comments presents difficulties due to ambiguity and subjectivity. To reduce these threats, the authors adopted Parrott's framework as a reference for emotions. Finally, to avoid bias due to personal interpretation, during the annotation of 4000 sentences for the training corpus of the emotion classifier, each sentence was analyzed by at least two raters. Furthermore the affectiveness measures are approximations and cannot 100% correctly identify the correct affective context, given the challenges of natural language and subtle phenomena like sarcasm. To deal with these threats, we used state-of-the-art tools like SentiStrength, the tool of Desmedt et al. [31] and Danescu et al.'s politeness tool, in addition to our own emotion classifier.

Threats to external validity correspond to the generalizability of our experimental results [20]. In this study, we manually analyze a sample of 4000 sentences of comments from issue reports belonging to 14 open source projects. We consider the projects as a representative sample of the universe of open source software projects, with different development teams and satisfying different customers' needs. Replications of this work on other open source systems and on commercial projects are needed to confirm our findings.

Threats to reliability validity correspond to the degree to which the same data would lead to the same results when repeated. This research is the first attempt to manually investigate different measures of affectiveness from issue reports, and their impact on the issue fixing time, hence no ground truth exists to compare our findings. We defined the ground truth through agreement or disagreement of the raters for measuring emotions and existing tools provided for measuring sentiment and politeness.

This study is focused on text written by developers *for* developers. To correctly depict the affectiveness embedded in such comments, it is necessary to understand the developers' dictionary and slang. This assumption is supported by Murgia et al. [73] for measuring emotions. We are confident that the tools used for measuring sentiment and politeness are equally reliable in the software engineering domain as in other domains.

Chapter 5

Related Works

5.1 Effectiveness

5.1.1 Software Maintenance and Fixing Time

Software maintenance types and issue resolution time are topics commonly analyzed in software engineering. However, with rare exceptions [71, 115], these topics are never analyzed together. This section presents how the maintenance activities have been considered in context of issue resolution time. The bug-oriented repository Bugzilla played a key-role for the analysis of the time spent on the maintenance activity. Demeyer et al. showed that this repository is the most common used in mining software conferences [32]. Bugzilla has been widely studied because it stores bugs of Eclipse and Mozilla, two of the most common case studies [81, 11, 45, 124]. Other bug-oriented repositories employed for these type of study have been the FreeBSD's bug repository [16] and the Google Code's bug tracker [11].

Panjer predicts the bug fixing time analyzing the bug reports of the project Eclipse [81]. He points out that the resolution time of bugs with severity blocker, critical and trivial is lower than the resolution time of enhancements. Similar analysis was performed by Zhang et al. to explore developer's delays during bug fixing for three projects of Eclipse [124]. They use Bugzilla's severity to distinguish between bug and enhancement. Investigating the factors that are relevant for the delay of the triaging, they discover that developers who fix a bug are faster in updating the bug status on the repository.

Bhattachary and Neamtiu investigate which factors influence the bug fixing time using the bug reports of Chrome, Eclipse and three products from the Mozilla project [11]. They build a bug-fix time prediction model and demonstrated that several models proposed in literature cannot be replicated when adopted on large projects used in bug studies. Their results indicate that the predictive power of such models is between 30% and 49% and they conclude that there is a need for more independent variables to construct a prediction model. Similar analysis is performed by Giger et al. [45] and Bougie [16]. In Giger et al. [45] they use the projects Eclipse, Mozilla, and Gnome, in Bougie et al. [16] they use the FreeBSD's bug repository. Unfortunately, in these cases the authors do not make any distinction related to the type of issue involved (e.g.; enhancement, bug), so it is not possible to make any conjecture related to the type of maintenance they analyzed.

Due to the bug-oriented nature of the repository, the previous works are mainly focused on the issue resolution time related to the corrective maintenance.

One step toward the analysis of the relationship between maintenance type and issue resolution time is done by Weiss et al. and Mockus and Votta [115, 71].

Weiss et al. take into account issues labeled as bug and feature, namely issues that can be assumed to be related to corrective and adaptive maintenance [115]. This study does not have any analysis on perfective and preventive maintenance. Moreover, the number of issues ascribable as corrective and maintenance activity are only 273 and all of them belong to just one project. Finally, they only use mean and standard deviation to characterize the difference between the issue resolution time of bugs and features. These statistics are not reliable to characterize right-skewed or fat-tail distributions such as the issue resolution time distributions. From the comparison of the mean values, they show that corrective maintenance is faster than adaptive maintenance.

Mockus and Votta analyze the issue resolution time associated to the corrective, adaptive, and perfective maintenance activities [71]. In the study, the type of maintenance activity is only inferred by the authors reading the developer commit message. Indeed, the issue tracking system used in the study did not have any field to specify the type of issue. Moreover, the analysis is limited to only one commercial switching software. The study uses only plots to show how the issue resolution time changes according to the type of maintenance. Comparing the plots, the authors highlights that corrective changes have the shortest resolution time, followed by perfective changes. Unfortunately, the analysis does not provide many details related to the distribution of the issue resolution time.

The last two studies, even if they consider more than one type of maintenance, are not focused on how to model the distribution of the issue resolution time.

There are two main differences between our study and the previous ones.

- We take into account all categories of maintenance reported in ISO/IEC

14764, namely corrective, preventive, perfective and adaptive maintenance [1]. Our investigation, which involves 34 projects, uses the GitHub's repository where developers keep track of any maintenance activity performed in the system.

- We model the issue resolution time using statistical distribution. The benefits of this analysis is not only the possibility to distinguish among the different maintenance activities, but also to provide reliable estimates of the issue resolution times.

5.1.2 Developers Working Team and Fixing Time

The *bazaar* is one of the most used figurative expression to explain the Linux's development style, and several studies have used the "Cathedral-Bazaar" metaphor [92] to describe the properties of a (differently) organized approach of development [10] [15] [12] [21] [65].

The structure of such decentralized development has been deeply analyzed by many researchers. Small world phenomenon and scale free behaviors are found in the SourceForge development network by Xu et al [119], considereding two developers socially related if they participate in the same project. Wagstrom et al. [113] congregated empirical social network data from blogs, email lists and web sites, to build a models of development used to simulate how users joined and left projects. Ehrlich et al [36] used social network analysis to study how individuals in global software development teams detect and gain expertise.

Crowston et al. [27] examined 120 project teams from SourceForge, detecting that open source development teams vary in their communications centralization, from projects centered on one developer to projects highly decentralized and exhibit a distributed pattern of conversation between developers and active users. Larger teams tend to have more decentralized communication patterns. Other researchers[69] examined the structure of developer collaboration with the developer network in order to predict failures at the file level. Failure prediction models were developed using test and post-release failure data from two releases of a mature Nortel networking product, then validated against a subsequent release.

Sharif et al. [99] showed that open source developers are "implementation centric" and "team focused" in their use of mailing lists.

Other studies have analyzed the structure of the open source software communities to understand social aspects in development Steinmacher et al. [105], identified 20 studies providing empirical evidence of barriers faced by newcomers to OSS projects while making a contribution. They identified 15 different barriers, which we grouped into five categories: social interaction, newcomers' previous knowledge, finding a way to start, documentation, and technical hurdles. The authors also classified the problems with regard to their origin: newcomers, community, or product. Zhou et al. [126] found, using issue tracking data of Mozilla and Gnome, that the probability for a new joiner to become a Long Term Contributor is associated with her willingness and environment. Shah [98] explored the motivations of participants from two software development communities and finds that most participants are motivated by either a need to use the software or an enjoyment of programming. The latter group, hobbyists or enthusiasts, are critical to the long-term viability and sustainability of open source software code: they take on tasks that might otherwise go undone, are largely "need-neutral" as they make decisions, and express a desire to maintain the simplicity, elegance, and modularity of the code. The motives of hobbyist evolve over time; most join the community because they have a need for the software and stay because they enjoy programming in the context of a particular community. G Ortu et al. [77] studied 14 open source software projects developed using the Agile board of the JIRA repository. They analysed all the comments committed by the developers involved in the projects and we studied whether the politeness of the comments affected the number of developers involved over the years and the time required to fix any given issue. Results indicated that the level of politeness in the communication process among developers does have an effect on both the time required to fix issues and the attractiveness of the project to both active and potential developers. The more polite developers were, the less time it took to fix an issue, and, in the majority of the analysed cases, the more the developers wanted to be part of project, the more they were willing to continue working on the project over time.

5.2 Affectiveness

5.2.1 Emotions

Software development is a collaborative activity dependent on human interaction between developers towards the timely completion of a high quality software system [70]. Existing software development studies are not focused on the analysis of individual feelings, even though the morale and emotions of individuals can impact on the collaboration process and subsequently the success of a product [18, 2]. In our study, emotions are the first-class object, we study how developers feel towards software artifacts and/or colleagues. A project that is not appealing and unable to motivate developers to join, can be destined to fail [43]. For this reason, there has been a great deal of research on the reasons of developers joining and leaving a software project. Sim et. al have investigated dynamic strategies and conditions that characterize the joining process of software immigrants in a software development team [103]. Ducheneaut analysed the socialization process of new developers by visualizing the dynamic networks of both human and material resources incorporated in the email and code databases of open source software [34]. Herraiz et al. studied the duration and basic characteristics of the joining process for the developers and found two groups with clearly different joining patterns: new professionals and volunteer developers [59]. They could relate those patterns to the different behavior of volunteers and hired developers.

Rigby et al. analyzed the five big personality traits [78] of software developers in the Apache httpd server mailing list [93]. They analyzed the personality of four top developers, assessing the personality of two top developers who have left the project. Bazelli et al. replicated Rigby's work on Stack Overflow to analyze the personality traits of different authors [6]. Although both studies deal with the behaviour of developers, they do not consider the emotions of project members and how these impact the motivation of project members. Typically, emotion and sentiment mining studies focus on emotions that people convey through twitter, question and answer sites and product reviews [79, 66, 68]. Choudhury et al. [23] applied sentiment analysis on microblogging data of a large global Fortune 500 software corporation used to "disseminate product/service updates to the larger enterprise community, such as details of new upcoming features, news about future team meetings, releases or trade shows". They found that there is a potential for building systems that assess feelings of employees in an enterprise. To improve emotional awareness in software development teams, Guzman et al. proposed a sentiment analysis approach for discussions in mailing lists and web-based software collaboration tools like Confluence [54]. They used Latent Dirichlet Allocation to find the topics discussed in email and web discussions of students in the context of a class project, then use lexical sentimental analysis to obtain an average emotion score for each of the topics. Dullemond et al. [35] extended a microblogging tool with a happiness indicator, then deployed the tool across distributed teams in a company. Employees used the tool to share their message and emotions with their colleagues in order to stay more connected and be aware of each other while collaborating. Gómez, using a grep-like approach, mined GitHub for extracting developers commit messages [47]. In his blog he collected several expressions ascribable as anger, joy and surprise, used during coding activity by developers. Differently from sentiment analysis on high-level feature and meeting announcements, our study performs emotion mining on technical artifacts, i.e., the comments of issue reports. Although emails and web discussions contain more technical details than the microblogging data sources, issue reports contain even more technical detail, since they are used by team members to ask for advice, express opinion and share opinions related to software maintenance and evolution. In contrast to sentiment analysis on a given emotion, we tried to identify and mine the different types of emotions.

5.2.2 Politeness and Agile

Several researchers have analysed [76] [73] [108] [118] [111] the effect of politeness. Gupta et al. [51] presented POLLy (Politeness for Language Learning), a system which combines a spoken language generator with an artificial intelligence planner to model Brown and Levinson's theory of politeness in collaborative task-oriented dialogue, with the ultimate goal of providing a fun and stimulating environment for learning English as a second language. An evaluation of politeness perceptions of POLLy's output shows that: perceptions are generally consistent with Brown and Levinson's predictions for choice of form and for discourse situation, i.e. utterances to strangers need to be much more polite than those to friends; (2) our indirect strategies which should be the politest forms, are seen as the rudest; and (3) English and Indian native speakers of English have different perceptions of politeness. Pikkarainen et al. [86] showed that agile practices improve both informal and formal communication. The studies indicates that, in larger development situations involving multiple external stakeholders, a mismatch of adequate communication mechanisms can sometimes even hinder communication. The study highlights the fact that hurdles and improvements in the communication process can both affect the feature requirements and task subtask dependencies as described in coordination theory. While the use of SCRUM and some XP practices facilitate team and organizational communication of the dependencies between product features and working tasks, the use of agile practices requires that the team and organization use also additional plan-driven practices to ensure the efficiency of external communication between all the actors of software development. Korkala et al. [64] showed that effective communication and feedback are crucial in agile development. Extreme programming (XP) embraces both communication and feedback as interdependent process values which are essential for projects to achieve successful results. The research presents the empirical results from four different case studies. Three case studies had partial onsite customers and one had an onsite customer. The case studies used face-to-face communication to different extents along with email and telephone to manage customer-developer communication inside the development iterations. The results indicate that an increased reliance on less informative communication channels results in higher defect rates. These results suggest that the selection of communication methods, to be used inside development iterations, should be a factor of considerable importance to agile organizations working with partially available customers.

5.2.3 Affectiveness

The Manifesto for Agile Development [8] indicates that individuals and interactions are more important than processes and tools. David Parnas defined software engineering as multi-person development of multi-version programs [82] [83].

As such, the study of social aspects and psychological states [62] in software engineering is gaining, lately, more and more importance. Roberts and al. [95] conducted a study that reveals how the different motivations of open source developers are interrelated, how these motivations influence participation, and how past performance influences subsequent motivations. Researchers are focusing their effort on understanding how the human aspects of a technical discipline can affect the final results [17] [38][61]. Feldt et al. [39] focused on personality as one important psychometric factor and presented initial results from an empirical study investigating correlations between personality and attitudes to software engineering processes and tools. To enhance emotional awareness in software development teams, Guzman et al. proposed a sentiment analysis approach for discussions in mailing lists and web-based software collaboration tools like Confluence [53]. They used lexical sentiment analysis to analyze the relationship between emotions expressed in commit comments, with different factors such as programming language, time and day of the week in which the commit was made. Results showed that projects developed in Java have more negative commit comments, and that commit comments written on Mondays tend to contain more negative emotion. Steinmacher et al. [104] analyzed social barriers that hampered newcomers' first contributions. These barriers were identified considering a systematic literature review, students contributing to open source projects, and responses collected from OSS projects' contributors. The authors indicated how impolite answers are considered as a barrier by newcomers.

Rigby et al. [94] analyzed the five big personality traits of software developers in the Apache httpd server mailing. Bazelli et al. [7] studied the personality traits of authors of questions on StackOverFlow.com. 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.

Tourani et al. [110] evaluated the usage of automatic sentiment analysis to identify distress or happiness in a development team. They extracted sentiment values from the mailing lists of two of the most successful and mature projects of the Apache software foundation considering both users and developers. They found that user and developer mailing lists bring both positive and negative sentiment and that an automatic sentiment analysis tool obtains only a modest precision on email messages due to their relatively long size compared to tweets or issue comments, and Murgia et al. [73] analyzed whether development artifacts like issue reports carry any emotional information about software development. The significant result of the study is that issue reports express emotions towards design choices, maintenance activity or colleagues.

Gomez et al. [46] analyzed 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 is correlated with team members' personality traits like extroversion and team climate factors such as participation. They derived guidelines for software project managers with respect to team formation.

Ortu et al. [77] studied 14 open source software projects developed using the Agile board of the JIRA repository. They analysed all the issue comments written by the developers involved in the projects to study whether the politeness of the comments affected the number of developers involved over the years and the time required to fix any given issue. Results indicated that the level of politeness in the communication process among developers has an effect on both the time required to fix issues and the attractiveness of the project to both active and potential developers. The more polite developers were, the less time it took to fix an issue, and, in the majority of the analysed cases, the more the developers wanted to be part of a project, the more they were willing to continue working on the project over time.

Compared to Ortu et al. [77], our study analyzes two additional affectiveness metrics (emotions and sentiment), as well as uses logistic regression to compare the impact of all affectiveness metrics and common issue report metrics together, instead of using a univariate model using only politeness.

Chapter 6

Conclusion

The work conducted during the three years of the present PhD is aimed to analyze the development process of software artifacts from two point of view: the *Effectiveness* and *Affectiveness*. The first is meant to analyze the productivity of Open Source Communities by measuring the time required to resolve an issue. The latter provided a novel approach for studying the development process by analyzing the affectiveness expressed by developers in their comments posted during an issue resolution. Affectivenes is measured by measuring *Sentiment*, *Politeness* and *Emotions*. All the study present in this work are based on two real cases of software repositories: Jira and GitHub as described in 2.1.1.

6.1 Effectiveness

This study first analyzed to which extends the time required to fix an issue is influenced by the issue's maintenance type. Software maintenance is a process difficult to understand and manage and yet crucial to organize the company's resource. In literature there is little information on how the maintenance activity influences the issue resolution time. Moreover, in the few cases where these two topics are both considered, the analysis is oversimplified since it adopts few statistic measures that cannot properly describe the specificities (e.g; right-skewness) of the issue resolution time distribution. Our empirical study analyzes the data stored in the GitHub's issue tracking system. From this repository we analyze 34 open source projects and more than 14000 issues that were ascribable as requiring corrective, adaptive or perfective maintenance. Empirical results shows that the issue resolution time depends on the type of maintenance performed. We discovered that corrective and perfective maintenance are generally shorter than the other maintenance activities, whereas adaptive and perfective maintenance requires generally the highest resolution-time. Moreover, it points out that models for effort estimation of maintenance activities based mainly on data extracted from bug oriented repositories (e.g.; Bugzilla) may provide biased estimation (towards corrective maintenance). By the use of lognormal and Weibull distribution models this study enables to quantify the contribute of the maintenance type on the issue resolution time. For both models, the study demonstrates their suitability to analyze samples with different size and further demonstrates that statistical distribution models can be exploited for project's scheduling.

The second factor that the present study considered for studying effectiveness is the working team. We analyzed the developer networks of 7 open source projects hosted in JIRA by building a network in which developers who posted issues or commented issues are represented by nodes and edges represent a developer posting a comment on an issue posted by another developer. We then applied a clustering algorithm in order to detect communities. We found the presence of developers communities for all the projects analyzed. This result agrees with other studies about the social structure of open source projects. We further investigated how the productivity is distributed across the communities. To measure the productivity we considered factors such as the community size, the number of fixed issues, the distribution of fixed issues' maintenance type and priority, and the average issue fixing time. We found the presence of Pareto's law (20% of developers doing 80% of the work), there are a few developers that post and comment the majority of issues. The presence of Pareto's law need further investigation, one may expect this is due to the nature of JIRA issue tracking system and the structure of the open source community and how we built the developer's working network. For example there may be a group of core developers devoted to report issues. We analyzed the average community issue fixing time and we found it varies across the communities. We showed the independence of the average issue resolution time from the other factor considered, such as the community size and the kind of issues maintenance and priority. There are many other factors that may impact the average community issue fixing time, for example software component involved in the issue resolution or the portion of code involved.

This study is a starting point to better understand how groups of developers perform when working together. The issue resolution time is a useful metric that represents the productivity of a certain community.

We conclude this first part highlighting the main contributions.

- We built two statistical distribution models for the issue resolution time that takes into account the issue maintenance type. These models are based on real data and we showed, by mean of examples, that they can be adopted for issue scheduling.

- We provided a preliminary study for analyzing the productivity of developers' working teams by analyzing communities in developers' network built from Issue Tracking System.

- Results confirmed that productivity is distributed across working teams according to Pareto's law.

- The average issue resolution time greatly varies across communities and preliminary results suggest that it is independent from the community size (measured by the number of developers of a community).

6.2 Affectiveness

The work presented in this thesis continues analyzing the development process of software artifacts from the point of view of the affectiveness expressed by developers on their comments posted on issue reports. The study was performed on a real case study based on several projects of the Apache Software Foundation hosted in Jira. Human *Affectiveness* such as the emotional state of a person influences human behaviour and interaction. Software development is a collaborative activity and thus it is not exempt from such influence. Affective analysis, e.g., measuring emotions, sentiment and politeness, applied to developer issue reports, can be useful to identify and monitor the mood of the development team, allowing project leaders to anticipate and resolve potential threats to productivity (especially in remote team settings), as well as to discover and promote factors that bring serenity and productivity in the community.

In order to measure affectiveness we considered three metrics: *Sentiment*, *Politeness* and *Emotion*. Sentiment and politeness are measured using two state-of-art tool, respectively: SentiStrenght and the politeness tool provided by Danescu et. al. [30]. By the time of this thesis, and to the best of our knowledge, there is no free available tool for detecting emotions in written text. Since the lack of such tools for measuring emotions and since emotions have never been studied before in the software engineering field we first analyzed the feasibility of studying emotions in software artifacts. As a first step towards evaluating the feasibility of a tool for automatic emotion mining, we performed an exploratory study of developer emotions in almost 800 issue comments posted during software development, maintenance and evolution. Our study confirms that issue reports do express emotions towards design choices, maintenance activity or colleagues. Regarding agreement among human raters, results shows that some emotions like `LOVE`, `JOY` and `SADNESS` are easier to agree on, but that additional context can cause doubt for raters, unless more raters are used. Findings suggest that for `LOVE`, `JOY` and `SADNESS` it makes sense and eventually might be feasible to automate emotion mining.

Proved the presence of emotions in comments posted by developers during development, this work considered the role played by politeness during software development for system developed using the Agile board of the JIRA repository. This study presents the results about politeness and attractiveness, as defined by [120], on 14 open source software projects developed using the Agile board of the JIRA repository. Results show that the level of politeness in the communication process among developers does have an effect on both the time required to fix issues and the attractiveness of the project to both active and potential developers. The more polite developers were, the less time it took to fix an issue and, in the majority of the analyzed cases, the more the developers wanted to be part of project, the more they were willing to continue working on the project over time. This work provided a a starting point and further research on a larger number of projects is needed to prove and validate these findings.

Previous studies showed that emotions are present in developers comments and for some emotions such as *JOY*, *LOVE* or *SADNESS* it is feasible to build an automatic tool for emotion mining and thus we built a machine learning tool for emotion mining, as described in 4.3.2.

We can now consider affectiveness by measuring sentiment, politeness and emotions.

This work is a first attempt to highlight the impact of developer affectiveness on productivity in the form of issue fixing time. First, we showed that the three affective metrics, i.e., emotions, sentiment and politeness, are independent, showing a weak correlation of at most 0.36. Then, we showed how affectiveness metrics statistically improve an explanation model of issue fixing time compared to a model based on control metrics. The 4th, 5th and 6th most important metrics in the model correspond to *% of love comments* (-50.19%), *issue average politeness* (+49.76%) and *% of sadness comments* (+38.39%). In other words, comments containing *JOY* and *LOVE* emotions have shorter issue fixing time, while comments containing *SADNESS* emotion have a longer fixing time. Although we found that the politeness of the last comment has a shorter issue fixing time, it is unexpected that less polite comments are linked with shorter fixing time. After investigation we found that for about the 50% issue reports with extreme politeness (polite and impolite) have shorter issue fixing time. Those reports tend to only have a median number of 2 developers discussing the issue, and the negative issues have the lowest number of sentences in the comments.

We can conclude that affectiveness plays a major role in software development being able to explain the time required to fix an issue. The present work represent a first attempt to analyze the software development from this prospective, providing the following contributions.

- A systematic study that highlight the presence of emotions in developers' comments and the ability of human raters to detect and agree on some basic emotions such as *JOY*, *LOVE* and *SADNESS*.
- A manually annotated, emotion-based, corpus of 4000 sentences extracted form the Jira repository of the Apache Software Foundation.
- A novel approach for emotions mining based on machine learning classifier with good performance.
- An explanatory model for issue resolution time showing that affective metrics are indeed significant for explaining the issue resolution time.

Bibliography

[1] International standard - iso/iec 14764 ieee std 14764-2006. ISO/IEC 14764:2006 (E) IEEE Std 14764-2006 Revision of IEEE Std 1219-1998), 2006. [cited at p. 18, 28, 91]

[2] S. Ambler. "Agile modeling: effective practices for extreme programming and the unified process". John Wiley & Sons, Inc. New York, 2002. [cited at p. 42, 92]

[3] A. Bacchelli, M. Lanza, and R. Robbes. Linking e-mails and source code artifacts. In ICSE (1), pages 375–384, 2010. [cited at p. 50]

[4] A. Bacchelli, T. D. Sasso, M. D'Ambros, and M. Lanza. Content classification of development emails. In Proc. of the 34th Intl. Conf. on Software Engineering (ICSE), pages 375–385, 2012. [cited at p. 50]

[5] M. Bastian, S. Heymann, M. Jacomy, et al. Gephi: an open source software for exploring and manipulating networks. ICWSM, 8:361–362, 2009. [cited at p. 31]

[6] B. Bazelli, A. Hindle, and E. Stroulia. On the personality traits of stackoverflow users. In Proceedings of the 2013 IEEE International Conference on Software Maintenance, ICSM '13, pages 460–463, Washington, DC, USA, 2013. IEEE Computer Society. [cited at p. 93]

[7] B. Bazelli, A. Hindle, and E. Stroulia. On the personality traits of stackoverflow users. In Software Maintenance (ICSM), 2013 29th IEEE International Conference on, pages 460–463. IEEE, 2013. [cited at p. 95]

[8] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, et al. Manifesto for agile software development. 2001. [cited at p. 95]

[9] A. Begel, N. Nagappan, C. Poile, and L. Layman. Coordination in large-scale software teams. In Proceedings of the 2009 ICSE Workshop on

Cooperative and Human Aspects on Software Engineering, pages 1–7. IEEE Computer Society, 2009. [cited at p. 69]

[10] N. Bezroukov. A second look at the cathedral and the bazaar. First Monday, 4(12), 1999. [cited at p. 91]

[11] P. Bhattacharya and I. Neamtiu. Bug-fix time prediction models: can we do better? In MSR, pages 207–210, 2011. [cited at p. 89]

[12] C. Bird, D. Pattison, R. DâSouza, V. Filkov, and P. Devanbu. Chapels in the bazaar? latent social structure in oss. In 16th ACM SigSoft International Symposium on the Foundations of Software Engineering, Atlanta, GA. Citeseer, 2008. [cited at p. 91]

[13] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment, 2008(10):P10008, 2008. [cited at p. 31]

[14] J. Bollen, H. Mao, and X. Zeng. Twitter mood predicts the stock market. Journal of Computational Science, 2(1):1–8, 2011. [cited at p. 42]

[15] A. Bonaccorsi and C. Rossi. Why open source software can succeed. Research policy, 32(7):1243–1258, 2003. [cited at p. 91]

[16] G. Bougie, C. Treude, D. M. Germán, and M. D. Storey. A comparative exploration of freebsd bug lifetimes. In MSR, pages 106–109, 2010. [cited at p. 19, 28, 89, 90]

[17] A. P. Brief and H. M. Weiss. Organizational behavior: Affect in the workplace. Annual review of psychology, 53(1):279–307, 2002. [cited at p. 95]

[18] J. Brodkin. Linus Torvalds defends his right to shame Linux kernel developers. http://www.webcitation.org/6O2zErgzE, July 2013. [cited at p. 41, 92]

[19] F. P. Brooks, Jr. No Silver Bullet Essence and Accidents of Software Engineering. Computer, 20(4):10–19, Apr. 1987. [cited at p. 42]

[20] D. T. Campbell and J. C. Stanley. Experimental and quasi-experimental designs for generalized causal inference. Houghton Mifflin, 1963. [cited at p. 58, 87]

[21] A. Capiluppi and M. Michlmayr. From the cathedral to the bazaar: An empirical study of the lifecycle of volunteer community projects. In Open Source Development, Adoption and Innovation, pages 31–44. Springer, 2007. [cited at p. 91]

[22] L. F. Capretz and F. Ahmed. Making sense of software development and personality types. IT professional, 12(1):6–13, 2010. [cited at p. 69]

[23] M. D. Choudhury and S. Counts. Understanding affect in the workplace via social media. In Proc. of the intl. conf. on Computer Supported Cooperative Work (CSCW), pages 303–316, 2013. [cited at p. 93]

[24] J. Cohen. A coefficient of agreement for nominal scales. Educational and psychological measurement, 20(1):37–46, 1960. [cited at p. 45]

[25] G. Concas, M. Marchesi, A. Murgia, R. Tonelli, and I. Turnu. On the distribution of bugs in the eclipse system. IEEE Trans. Software Eng., 37(6):872–877, 2011. [cited at p. 7, 22]

[26] G. Concas, M. Marchesi, S. Pinna, and N. Serra. Power-laws in a large object-oriented software system. Software Engineering, IEEE Transactions on, 33(10):687–708, 2007. [cited at p. 7, 22]

[27] K. Crowston and J. Howison. The social structure of free and open source software development (originally published in volume 10, number 2, february 2005). First Monday, 2005. [cited at p. 91]

[28] K. Crowston, Q. Li, K. Wei, U. Y. Eseryel, and J. Howison. Self-organization of teams for free/libre open source software development. Information and software technology, 49(6):564–575, 2007. [cited at p. 30]

[29] B. Curtis, H. Krasner, and N. Iscoe. A field study of the software design process for large systems. Communications of the ACM, 31(11):1268–1287, 1988. [cited at p. 69]

[30] C. Danescu-Niculescu-Mizil, M. Sudhof, D. Jurafsky, J. Leskovec, and C. Potts. A computational approach to politeness with application to social factors. In Proceedings of ACL, 2013. [cited at p. 10, 11, 73, 99]

[31] T. De Smedt and W. Daelemans. Pattern for python. The Journal of Machine Learning Research, 98888:2063–2067, 2012. [cited at p. 73, 87]

[32] S. Demeyer, A. Murgia, K. Wyckmans, and A. Lamkanfi. Happy birthday! a trend analysis on past msr papers. MSR '13, pages 353–362, 2013. [cited at p. 89]

[33] G. Destefanis, S. Counsell, G. Concas, and R. Tonelli. Software metrics in agile software: An empirical study. In Agile Processes in Software Engineering and Extreme Programming, pages 157–170. Springer, 2014. [cited at p. 29]

[34] N. Ducheneaut. Socialization in an open source software community: A socio-technical analysis. Computer Supported Cooperative Work (CSCW), 14(4):323–368, 2005. [cited at p. 93]

[35] 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 Proc. of the 10th Working Conf. on Mining Software Repositories (MSR), pages 267–276, 2013. [cited at p. 93]

[36] K. Ehrlich and K. Chang. Leveraging expertise in global software teams: Going outside boundaries. In Global Software Engineering, 2006. ICGSE'06. International Conference on, pages 149–158. IEEE, 2006. [cited at p. 91]

[37] H. A. Elfenbein and N. Ambady. On the universality and cultural specificity of emotion recognition: a meta-analysis. Psychological bulletin, 128(2):203, 2002. [cited at p. 58, 73]

[38] A. Erez and A. M. Isen. The influence of positive affect on the components of expectancy motivation. Journal of Applied Psychology, 87(6):1055, 2002. [cited at p. 95]

[39] R. Feldt, R. Torkar, L. Angelis, and M. Samuelsson. Towards individualized software engineering: empirical studies should collect psychometrics. In Proceedings of the 2008 international workshop on Cooperative and human aspects of software engineering, pages 49–52. ACM, 2008. [cited at p. 95]

[40] J. L. Fleiss. Measuring nominal scale agreement among many raters. Psychological bulletin, 76(5):378, 1971. [cited at p. 45]

[41] J. H. Fowler, N. A. Christakis, et al. Dynamic spread of happiness in a large social network: longitudinal analysis over 20 years in the framingham heart study. Bmj, 337:a2338, 2008. [cited at p. 69]

[42] B. L. Fredrickson. The role of positive emotions in positive psychology: The broaden-and-build theory of positive emotions. American psychologist, 56(3):218, 2001. [cited at p. 42]

[43] R. A. Ghosh and V. V. Prakash. The orbiten free software survey. First Monday, 5(7), 2000. [cited at p. 93]

[44] 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, pages 52–56. ACM, 2010. [cited at p. 75, 79, 81]

[45] E. Giger, M. Pinzger, and H. Gall. Predicting the fix time of bugs. RSSE '10, pages 52–56. ACM, 2010. [cited at p. 89, 90]

[46] M. N. Gómez, S. T. Acuña, M. Genero, and J. A. Cruz-Lemus. How does the extraversion of software development teams influence team satisfaction and software quality?: A controlled experiment. International Journal of Human Capital and Information Technology Professionals (IJHCITP), 3(4):11–24, 2012. [cited at p. 96]

[47] R. Gómez. Exploring expressions of emotions in github commit messages. http://www.webcitation.org/6N9nD4IQN, May 2012. [cited at p. 93]

[48] G. Gousios. The ghtorrent dataset and tool suite. In Proceedings of the 10th Working Conference on Mining Software Repositories, MSR'13, pages 233–236, 2013. [cited at p. 7, 28]

[49] T. U. Grund. Network structure and team performance: The case of english premier league soccer teams. Social Networks, 34(4):682–690, 2012. [cited at p. 69]

[50] J. Guillory, J. Spiegel, M. Drislane, B. Weiss, W. Donner, and J. Hancock. Upset now?: emotion contagion in distributed groups. In Proc. of the SIGCHI Conf. on Human Factors in Computing Systems (CHI), pages 745–748, 2011. [cited at p. 12]

[51] S. Gupta, M. A. Walker, and D. M. Romano. How rude are you?: Evaluating politeness and affect in interaction. pages 203–217, 2007. [cited at p. 94]

[52] 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, pages 352–355. ACM, 2014. [cited at p. 10]

[53] 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, pages 671–674. ACM, 2013. [cited at p. 69, 95]

[54] E. Guzman and B. Bruegge. Towards emotional awareness in software development teams. In Proc. of the 2013 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE), pages 671–674, 2013. [cited at p. 93]

[55] A. Guzzi, A. Bacchelli, M. Lanza, M. Pinzger, and A. van Deursen. Communication in open source software development mailing lists. In The 10th Working Conference on Mining Software Repositories, pages 277–286, 2013. [cited at p. 44]

[56] J. T. Hancock, K. Gee, K. Ciaccio, and J. M.-H. Lin. I'm sad you're sad: emotional contagion in CMC. In Proc. of the 2008 ACM conf. on Computer Supported Cooperative Work (CSCW), pages 295–298, 2008. [cited at p. 12]

[57] A. Hars and S. Ou. Working for free? motivations of participating in open source projects. In System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on, pages 9–pp. IEEE, 2001. [cited at p. 30]

[58] A. Heritage Dictionary. The American Heritage science dictionary. http://dictionary.reference.com/browse/, 2005. [cited at p. 13]

[59] I. Herraiz, G. Robles, J. J. Amor, T. Romera, and J. M. González Barahona. The processes of joining in global distributed software projects. In Proc. of the 2006 intl. workshop on Global software development for the practitioner (GSD), pages 27–33, 2006. [cited at p. 93]

[60] A. Hindle, D. M. German, and R. Holt. What do large commits tell us?: A taxonomical study of large commits. In Proceedings of the 2008 International Working Conference on Mining Software Repositories, MSR '08, pages 99–108, New York, NY, USA, 2008. ACM. [cited at p. 20, 28]

[61] E. Kaluzniacky. Managing psychological factors in information systems work: An orientation to emotional intelligence. IGI Global, 2004. [cited at p. 95]

[62] W. Ke and P. Zhang. The effects of extrinsic motivations and satisfaction in open source software development. Journal of the Association for Information Systems, 11(12):784–808, 2010. [cited at p. 95]

[63] R. Koch. The 80/20 principle: the secret to achieving more with less. Crown Business, 2011. [cited at p. 34]

[64] M. Korkala, P. Abrahamsson, and P. Kyllonen. A case study on the impact of customer communication on defects in agile software development. In Agile Conference, 2006, pages 11–pp. IEEE, 2006. [cited at p. 94]

[65] S. Krishnamurthy. Cave or community?: An empirical examination of 100 mature open source projects. First Monday, 2002. [cited at p. 91]

[66] 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, pages 633–642. ACM, 2012. [cited at p. 93]

[67] R. Lambiotte, J.-C. Delvenne, and M. Barahona. Laplacian dynamics and multiscale modular structure in networks. arXiv preprint arXiv:0812.1770, 2008. [cited at p. 31, 33]

[68] Y. Lu, P. Tsaparas, A. Ntoulas, and L. Polanyi. Exploiting social context for review quality prediction. In Proc. of the 19th intl. conf. on World Wide Web (WWW), pages 691–700, 2010. [cited at p. 93]

[69] A. Meneely, L. Williams, W. Snipes, and J. Osborne. Predicting failures with developer networks and social network analysis. In Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, pages 13–23. ACM, 2008. [cited at p. 91]

[70] I. Mistrìk, J. Grundy, A. Hoek, and J. Whitehead, editors. Collaborative Software Engineering. Springer, 2010. [cited at p. 92]

[71] A. Mockus and L. G. Votta. Identifying reasons for software changes using historic databases. In Proceedings of the International Conference on Software Maintenance (ICSM'00), ICSM '00, pages 120–, Washington, DC, USA, 2000. IEEE Computer Society. [cited at p. 18, 28, 89, 90]

[72] A. Murgia, G. Concas, R. Tonelli, M. Ortu, S. Demeyer, and M. Marchesi. On the influence of maintenance activity types on the issue resolution time. In Proceedings of the 10th International Conference on Predictive Models in Software Engineering, pages 12–21. ACM, 2014. [cited at p. 63]

[73] 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 Working Conference on Mining Software Repositories, pages 262–271. ACM, 2014. [cited at p. 10, 70, 72, 87, 94, 96]

[74] J. A. Nevin. Signal detection theory and operant behavior: A review of david m. green and john a. swets' signal detection theory and psychophysics. 1. Journal of the Experimental Analysis of Behavior, 12(3):475–480, 1969. [cited at p. 81]

[75] M. E. Newman. Power laws, pareto distributions and zipf's law. Contemporary physics, 46(5):323–351, 2005. [cited at p. 10]

[76] N. Novielli, F. Calefato, and F. Lanubile. Towards discovering the role of emotions in stack overflow. In Proceedings of the 6th International Workshop on Social Software Engineering, pages 33–36. ACM, 2014. [cited at p. 94]

[77] 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, page in press. Springer, 2015. [cited at p. 10, 84, 92, 96]

[78] D. J. Ozer and S. P. Reise. Personality assessment. Annual review of psychology, 45(1):357–388, 1994. [cited at p. 93]

[79] A. Pak and P. Paroubek. Twitter as a Corpus for Sentiment Analysis and Opinion Mining. In LREC, 2010. [cited at p. 93]

[80] B. Pang and L. Lee. Opinion Mining and Sentiment Analysis. Foundations and Trends in Information Retrieval, 2(1-2):1–135, Jan. 2008. [cited at p. 11, 58, 87]

[81] L. D. Panjer. Predicting eclipse bug lifetimes. In MSR, page 29, 2007. [cited at p. 19, 28, 89]

[82] D. L. Parnas. Software engineering or methods for the multi-person construction of multi-version programs. pages 225–235, 1975. [cited at p. 95]

[83] D. L. Parnas. Software engineering: multi-person development of multi-version programs. 2011. [cited at p. 95]

[84] W. Parrott. Emotions in Social Psychology. Psychology Press, 2001. [cited at p. 13, 43]

[85] T. Perry. Drifting toward invisibility: The transition to the electronic task board. In Agile, 2008. AGILE'08. Conference, pages 496–500. IEEE, 2008. [cited at p. 60]

[86] M. Pikkarainen, J. Haikara, O. Salo, P. Abrahamsson, and J. Still. The impact of agile practices on communication in software development. Empirical Software Engineering, 13(3):303–337, 2008. [cited at p. 94]

[87] C. Piller. Everyone is a critic in cyberspace. Los Angeles Times, 3(12):A1, 1999. [cited at p. 42]

[88] D. Pletea, B. Vasilescu, and A. Serebrenik. Security and emotion: sentiment analysis of security discussions on github. In Proceedings of the 11th Working Conference on Mining Software Repositories, pages 348–351. ACM, 2014. [cited at p. 10]

[89] R. Plutchik. The Nature of Emotions Human emotions have deep evolutionary roots, a fact that may explain their complexity and provide tools for clinical practice. American Scientist, 89(4):344–350, 2001. [cited at p. 13]

[90] R. Plutchik and H. Van Praag. The measurement of suicidality, aggressivity and impulsivity. Progress in Neuro-Psychopharmacology and Biological Psychiatry, 13:S23–S34, 1989. [cited at p. 42]

[91] R. Purushothaman and D. E. Perry. Toward understanding the rhetoric of small source code changes. IEEE Trans. Softw. Eng., 31(6):511–526, June 2005. [cited at p. 18, 20, 28]

[92] E. S. Raymond. The cathedral and the bazaar, 2000. Available from World Wide Web: http://www. catb. org/˜ esr/writings/cathedral-bazaar, 2004. [cited at p. 91]

[93] 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 MSR, page 23, 2007. [cited at p. 93]

[94] 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, page 23. IEEE Computer Society, 2007. [cited at p. 95]

[95] J. A. Roberts, I.-H. Hann, and S. A. Slaughter. Understanding the motivations, participation, and performance of open source software developers: A longitudinal study of the apache projects. Management science, 52(7):984–999, 2006. [cited at p. 95]

[96] M. D. Robinson. Personality as Performance Categorization Tendencies and Their Correlates. Current Directions in Psychological Science, 13(3):127–129, 2004. [cited at p. 13]

[97] I. Rowe. Civility 2.0: a comparative analysis of incivility in online political discussion. Information, Communication & Society, 18(2):121–138, 2015. [cited at p. 69]

[98] S. K. SHAH. Understanding the nature of participation & coordination in open and gated source software development communities. In Academy of Management Proceedings, volume 2004, pages B1–B5. Academy of Management, 2004. [cited at p. 92]

[99] K. Y. Sharif, M. English, N. Ali, C. Exton, J. Collins, and J. Buckley. An empirically-based characterization and quantification of information seeking through mailing lists during open source developersâ software evolution. Information and Software Technology, 57:77–94, 2015. [cited at p. 91]

[100] E. Shihab, Z. M. Jiang, W. M. Ibrahim, B. Adams, and A. E. Hassan. Understanding the impact of code and process metrics on post-release defects: a case study on the eclipse project. In Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, page 4. ACM, 2010. [cited at p. 74, 75]

[101] S. N. Shivhare and S. Khethawat. Emotion detection from text. Computer Science, Engineering and Applications, 2012. [cited at p. 13]

[102] S. Siegel. Nonparametric statistics for the behavioral sciences. 1956. [cited at p. 20, 64]

[103] S. Sim and R. Holt. The ramp-up problem in software projects: a case study of how software immigrants naturalize. In Proc. of the Intl. Conf. on Software Engineering (ICSE), pages 361–370, 1998. [cited at p. 93]

[104] I. Steinmacher, T. U. Conte, M. Gerosa, and D. Redmiles. Social barriers faced by newcomers placing their first contribution in open source software projects. In Proceedings of the 18th ACM conference on Computer supported cooperative work & social computing, pages 1–13, 2015. [cited at p. 70, 95]

[105] I. Steinmacher, M. A. G. Silva, M. A. Gerosa, and D. F. Redmiles. A systematic literature review on the barriers faced by newcomers to open source software projects. Information and Software Technology, 59:67–85, 2015. [cited at p. 91]

[106] C. Strapparava, A. Valitutti, et al. Wordnet affect: an affective extension of wordnet. In LREC, volume 4, pages 1083–1086, 2004. [cited at p. 73]

[107] E. B. Swanson. The dimensions of maintenance. In Proceedings of the 2nd international conference on Software engineering, ICSE '76, pages 492–497. IEEE Computer Society Press, 1976. [cited at p. 15, 18, 28]

[108] S. Tan and P. Howard-Jones. Rude or polite: Do personality and emotion in an artificial pedagogical agent affect task performance? In 2014 GLOBAL CONFERENCE ON TEACHING AND LEARNING WITH TECHNOLOGY (CTLT 2014) CONFERENCE PROCEEDINGS, page 41, 2014. [cited at p. 94]

[109] J. Tepperman, D. R. Traum, and S. Narayanan. " yeah right": sarcasm recognition for spoken dialogue systems. In INTERSPEECH. Citeseer, 2006. [cited at p. 53]

[110] P. Tourani, Y. Jiang, and B. Adams. Monitoring sentiment in open source mailing lists -â 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. [cited at p. 74, 95]

[111] J. Tsay, L. Dabbish, and J. Herbsleb. Letâs talk about it: Evaluating contributions through discussion in github. FSE. ACM, 2014. [cited at p. 94]

[112] R. van Solingen, V. Basili, G. Caldiera, and H. D. Rombach. Goal Question Metric (GQM) Approach. John Wiley Sons, Inc., 2002. [cited at p. 16]

[113] P. Wagstrom, J. Herbsleb, and K. Carley. A social network approach to free/open source software simulation. In Proceedings First International Conference on Open Source Systems, pages 16–23, 2005. [cited at p. 91]

[114] D. J. Watts and S. H. Strogatz. Collective dynamics of âsmall-worldânetworks. nature, 393(6684):440–442, 1998. [cited at p. 33]

[115] C. Weiß, R. Premraj, T. Zimmermann, and A. Zeller. How long will it take to fix this bug? In MSR, page 1, 2007. [cited at p. 19, 89, 90]

[116] 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, page 1. IEEE Computer Society, 2007. [cited at p. 64]

[117] F. Wilcoxon and R. A. Wilcox. Some rapid approximate statistical procedures. Lederle Laboratories, 1964. [cited at p. 20, 64]

[118] H. Winschiers and B. Paterson. Sustainable software development. In Proceedings of the 2004 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries, pages 274–278. South African Institute for Computer Scientists and Information Technologists, 2004. [cited at p. 94]

[119] J. Xu, Y. Gao, S. Christley, and G. Madey. A topological analysis of the open souce software development community. In System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on, pages 198a–198a. IEEE, 2005. [cited at p. 91]

[120] K. Yamashita, S. McIntosh, Y. Kamei, and N. Ubayashi. Magnet or sticky? an oss project-by-project typology. In Proceedings of the 11th Working Conference on Mining Software Repositories, pages 344–347. ACM, 2014. [cited at p. 62, 100]

[121] R. K. Yin. Case study research design and methods third edition. Applied social research methods series, 5, 2003. [cited at p. 35]

[122] R. K. Yin. Case study research: Design and methods, volume 5. sage, 2009. [cited at p. 28]

[123] J. Zaino. Analysis goes from sentiment to emotion. http://www.webcitation.org/6N9n9NJEF, May 2013. [cited at p. 11]

[124] F. Zhang, F. Khomh, Y. Zou, and A. E. Hassan. An empirical study on factors impacting bug fixing time. In Proceedings of the 2012 19th Working Conference on Reverse Engineering, WCRE '12, pages 225–234, Washington, DC, USA, 2012. IEEE Computer Society. [cited at p. 9, 89]

[125] H. Zhang, L. Gong, and S. Versteeg. Predicting bug-fixing time: an empirical study of commercial software projects. In Proceedings of the 2013 International Conference on Software Engineering, pages 1042–1051. IEEE Press, 2013. [cited at p. 63]

[126] M. Zhou and A. Mockus. What make long term contributors: Willingness and opportunity in oss community. In Proceedings of the 2012 International Conference on Software Engineering, pages 518–528. IEEE Press, 2012. [cited at p. 92]

List of Publications Related to the Thesis

Published papers

Conference papers

1. Concas, G., Destefanis, G., Marchesi, M., Ortu, M., & Tonelli, R. (2013). Micro Patterns in Agile Software (pp. 210-222). Springer Berlin Heidelberg.

2. Destefanis, Giuseppe, and Marco Ortu. "A Preliminary Analysis on Induced Micro Patterns Changes and Refactoring.", RefTest Workshop, XP 2013, Vienn

3. Concas, Giulio, et al. "Refactoring Clustering in Java Software Networks." Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation. Springer International Publishing, 2014. 121-135.

4. Murgia, A., Ortu, M., Tonelli, R., Concas, G., Marchesi, M., & Counsell, S. Measurements to assess the effort related to different kinds of software maintenance.

5. Murgia, A., Tourani, P., Adams, B., & Ortu, M. (2014, May). Do developers feel emotions? an exploratory analysis of emotions in software artifacts. In Proceedings of the 11th Working Conference on Mining Software Repositories (pp. 262-271). ACM.

6. Murgia, A., Concas, G., Tonelli, R., Ortu, M., Demeyer, S., & Marchesi, M. (2014, September). On the influence of maintenance activity types on the issue resolution time. In Proceedings of the 10th International Conference on Predictive Models in Software Engineering (pp. 12-21). ACM.

7. Ortu, M., Destefanis, G., Kassab, M., Counsell, S., Marchesi, M. & Tonelli, R. “Would you mind fixing this issue? An Empirical Analysis of Politeness and Attractiveness in Software Developed Using Agile Boards”, 16th International Conference on Agile Software Development, May 25-29 2015, Helsinki, Finland

8. Ortu, M., Adams, B., Destefanis, G., Tourani, P., Marchesi, M. & Tonelli, R. “Are Bullies more Productive? Empirical Study of Affectiveness vs. Issue Fixing Time”, The 12th Working Conference on Mining Software Repositories, May 16-17. Florence, Italy

9. Ortu, M., Destefanis, G., Kassab, M., & Marchesi, M. “Measuring and Understanding the Effectiveness of JIRA Developers Communities”, 6th International Workshop on Emerging Trends in Software Metrics,17 May 2015 - Florence, Italy

10. Ortu, M., Destefanis, G., Orrù, M., Tonelli, R., & Marchesi, M., “Could Micro Patterns be Used as Software Stability Indicators?” Patterns Promotion and Anti-patterns Prevention (PPAP) colocated with SANER 2015, Montréal