


Università degli Studi di Cagliari

DOTTORATO DI RICERCA IN

INGEGNERIA ELETTRONICA E INFORMATICA

Ciclo XXIX

TITOLO TESI

ACHIEVING QUALITY THROUGH SOFTWARE MAINTENANCE AND EVOLUTION: ON THE ROLE OF AGILE METHODOLOGIES AND OPEN SOURCE SOFTWARE

Settore scientifico disciplinare di afferenza

ING-INF/05 Informatica

Presentata da:	Simone Porru
Coordinatore Dottorato:	Fabio Roli
Tutor:	Michele Marchesi
Co-Tutor:	Roberto Tonelli

Esame finale anno accademico 2015 – 2016
Tesi discussa nella sessione d'esame marzo – aprile 2017

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

Achieving Quality through Software Maintenance and Evolution: on the role of Agile Methodologies and Open Source Software

Simone Porru

Adviser: Prof. Michele Marchesi
PhD Coordinator: Prof. Fabio Roli
Curriculum: ING-INF/05 Informatica

XXIX Cycle
April 2017

In memory of Giulio

Acknowledgements

My gratitude goes first to my family and friends, and especially to Antonio, Anna, and Filippo, for all their support over the last three years, and to both my adviser Michele Marchesi and assistant adviser Roberto Tonelli.

I am greatly thankful to Professor Serge Demeyer and PhD Alessandro Murgia for the assistance and trust given to me during my research period in Antwerp.

Special thanks go to Scrum Master Tars Joris, from the company Inventive Designers, for all the precious interviews and his friendly attitude.

Last, but not least, I thank Professor Giulio Concas.

In addition, I gratefully acknowledge the Sardinia Regional Government for the financial support of my PhD scholarship (P.O.R. Sardegna F.S.E. Operational Programme of the Autonomous Region of Sardinia, European Social Fund 2007-2013 Axis IV Human Resources, Objective l.3, Line of Activity l.3.1).

Abstract

Agile methodologies, open source software development, and emerging new technologies are at the base of disruptive changes in software engineering. Being effort estimation pivotal for effective project management in the agile context, in the first part of the thesis we contribute to improve effort estimation by devising a real-time story point classifier, designed with the collaboration of an industrial partner and by exploiting publicly available data on open source projects. We demonstrate that, after an initial training on at least 300 issue reports, the classifier estimates a new issue in less than 15 seconds with a mean magnitude of relative error between 0.16 and 0.61. In addition, issue type, summary, description, and related components prove to be project-dependent features pivotal for story point estimation. Since story points are the most popular effort estimation metric in the agile context, in the second study presented in the thesis we investigate the role of agile methodologies in software maintenance and evolution, and prove its undoubted influence on the refactoring research field over the last 15 years. In the later part of the thesis, we focus on recent technologies to understand their impact on software engineering. We start by proposing a specialized blockchain-oriented software engineering, on the basis of the peculiar challenges the blockchain sector must confront with and statistical data retrieved from a corpus of open source blockchain-oriented software repositories, identified relying upon the 2016 Moody's Blockchain Report. We advocate the need for new professional roles, enhanced security and reliability, novel modeling languages, and specialized metrics, along with new research directions focusing on collaboration among large teams, testing, and specialized tools for the creation of smart contracts. Along with the blockchain, in the later part of this work we also study the growing mobile sector. More specifically, we focus on the relationships between software defects and the use of the underlying system API, proving that our findings are aligned with those in the literature, namely, that the applications which are more connected to API classes are also more defect-prone. Finally, in the last work presented in the dissertation, we conducted a statistical analysis of 20 open source object-oriented systems, 10 written in the highly popular language Java and 10 in the rising language Python. We leveraged two statistical distribution functions–the log-normal and the double Pareto distributions–to provide good fits, both in Java and Python, for three metrics, namely, the NOLM, NOM, and NOS metrics. The study, among other findings, revealed that the variability of the number of methods used in Python classes is lower than in Java classes, and that Java classes, on average, feature fewer lines of code than Python classes.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent times, open source software development has grown to play a pivotal role in software engineering. Easily accessible tools and services, ranging from issue tracking systems (e.g. JIRA) to free source code hosting websites (e.g. GitHub), support and foster open source development, and provide extensive and easily accessible data for research purposes [1]. Such tools heavily leverage social networking, thus easily reaching more potential contributors. Indeed, Nakakoji et al. [2] reported that a large base of voluntary contributing members is one of the key success factors for OSS development.

Over the last fifteen years, agile software methodologies have also gained increasing popularity. Since the publication of the agile manifesto [3], a growing number of organizations have chosen to adopt agile methods to drive software development. Their strength resides in the ability to handle unstable requirements and deliver products in shorter timeframes and under strict budget constraints [4].

Agile software development proved to have many similarities with open source software development. As Warsta and Abrahamsson showed in [5], the OSS approach can be seen as one variant of the multifaceted agile methods, the most notable differences being the location and size of the developer team, the primary objective of the project, and the customers' representation.

The aim of the present thesis is to gain a deeper insight on how agile methodologies and open source software development are shaping software engineering, and especially on how their combined use supports and can be leveraged to improve software maintenance and evolution.

The main contribution of this thesis is the implementation of an automated, real-time classifier for story point estimation. The estimation tool has been developed by leveraging both publicly available issue reports on open source software projects and the lessons learned from the interaction with an agile developer team working in Inventive Designers[1], an IT company based in Belgium, which also provided all the issue reports of their core product, Scriptura Engage[2]. We show that, after an initial training on at least 300 issue reports, the classifier estimates a new issue in less than 15 seconds with a mean magnitude of relative error between 0.16 and 0.61. In addition, issue type, summary, description, and related components prove to be project-dependent features pivotal for story point estimation.

[1] https://www.inventivedesigners.com/. Accessed: 2017-03-19.
[2] https://www.scripturaengage.com/. Accessed: 2017-03-19.

The leading role agile methodologies cover in software effort estimation bring us to the second part of the work, in which we explore the research field of refactoring through a bibliometric study, proving that agile methodologies have been fundamental to the overall evolution of the refactoring research field.

The last part of the thesis is focused on investigating how rapidly-evolving open source software and increasingly popular languages are shaping OSS development. The opening work–the most recent one in the thesis–focuses on the need for software engineers to devise specialized tools and techniques for blockchain-oriented software development. Through the analysis of a curated corpus of blockchain-oriented software repositories, detected by exploiting the information enclosed in the 2016 Moody's Blockchain Report and the market capitalization of cryptocurrencies, we discuss the current challenges and new research directions for blockchain-oriented software engineering.

We then move from the most disruptive blockchain technology to the mobile sector. We use a complex network approach to gain deeper insight on internal and external dependencies of mobile applications, aiming at studying their influence on software quality. The results we obtained proved to be aligned with those of Syer et al. in [6], in that more external dependencies lead to higher defect-proneness.

Finally, we propose a statistical comparison between 20 open source object-oriented systems with the purpose of detecting differences in metrics distribution between Java and Python projects. We show that a metrics analysis can be successfully leveraged to identify differences in programming practices related to the use of methods and statements in the two languages.

1.1 Thesis Overview

The thesis is structured as follows.

Chapter 2 and 3 focus on the influence of Agile methodologies on software maintenance and evolution. Chapter 2 addresses effort estimation by presenting an automated real-time estimator, developed by exploiting a dataset constituted by open source projects which use story points, and an industrial project which relies on a most popular agile estimation practice (i.e. the Planning Poker). Chapter 3 takes a broader approach on software maintenance and evolution by investigating the evolution of the refactoring research field in the last fifteen years, ultimately detecting the pivotal role of Agile Methodologies across the research on Refactoring.

In Chapter 5 and 6, we leverage popular open source systems to detect pressing challenges for software maintenance and evolution. The most recent work in the thesis is discussed in Chapter 4, where we focus on the blockchain technology by acknowledging the need for software engineers to devise specialized tools and techniques for blockchain-oriented software development. We present the current challenges and propose new research directions on the basis of a curated corpus of blockchain-oriented software repositories, detected by exploiting the information enclosed in the 2016 Moody's Blockchain Report and the market capitalization of cryptocurrencies. In Chapter 5, we investigate the communication between mobile applications and the underlying open source operating system, and how they can impact software evolution, whereas in Chapter 6 we present a statistical comparison between open source software projects built with the highly popular programming language Java and the rising Python language.

Chapter 2

Estimating Story Points from Issue Reports

2.1 Introduction

Since the publication of the agile manifesto many organizations have chosen to adopt agile methods to drive software development [3]. In the agile context, effort estimation is pivotal for effective project management. Indeed, good estimates are essential to avoid poor resource allocation [7, 8].

A common means to estimate task effort are the story points [9, 10]. Within the context of agile development, story points are typically assigned via structured group meetings named *Planning Poker* sessions [11]. These meetings heavily rely upon human judgement: the better the developers understand the task, the better they can provide a sound estimate. However, human judgement may be hindered by several factors. Humans are generally optimistic and this bias is even more evident in group interactions [12, 13, 14]. In addition, developer estimation is known to be influenced by the presence of a project manager, other senior developers, or dominant personalities in the meeting [15].

To address these problems, we explore the use of a machine to support story point estimation. More specifically, we propose a machine learning classifier which leverages information within an issue report to file it in the most appropriate category. We target issue reports since they are commonly used by developers to describe development tasks, and are also heavily used in Planning Poker sessions [16].

The advantage of employing an automated classifier is threefold. First, the classifier has detailed knowledge about the project since its inception and produces its estimates on the basis of all the previous issues in the issue tracking system. Second, the classifier does not feel any pressure from or is influenced by other individuals, since its estimates can be traced back to the features used for classification. Third, the estimation is repeatable and predictable: the machine will never get tired and will always provide a consistent output.

We envision this classifier working *within* the team, providing its estimates along with other developers. In this manner, the classifier plays the role of an extra developer that produces an estimate in real-time on the basis of an objective analysis. As a first step towards building this classifier, we investigate whether it satisfies the preconditions to be successfully integrated during project planning. For this reason, we pursue the following research

questions:

RQ1: What performance can be expected from a machine learning classifier when estimating story points? We aim to provide real-time support to a development team in story point estimation. Therefore, we need to state upfront how fast the machine can provide an answer, and how trustworthy such answer is.

RQ2: Which attributes of an issue report are relevant for estimating story points? To gain credibility within a development team, the classifier should be able to provide the rationale behind the estimates it produces. As part of the answer, we investigate the dominant features employed by the classifier.

RQ3: How much training data is needed to obtain a satisfactory estimation? Agile teams work in small sprints, hence the production of a few hundred issues over the course of a project. Since the correctness of the classifier estimation depends on the amount of training data, we investigate how the behavior of the classifier changes when we increase the amount of training data and from which point onwards the estimates become satisfactory for the team.

To address these research questions, we rely on a data set containing issue reports stored in the JIRA repositories obtained from one industrial and eight open source projects. To evaluate the quality of the estimation we use the Mean Magnitude of Relative Error (MMRE), a cost estimation metric often used while investigating effort estimation in agile contexts [17]. This way, we demonstrate that it is feasible to exploit the information available in issue reports to estimate story points in real-time. Indeed, for all projects but one, the classifier achieves a MMRE spanning from 0.16 to 0.61 after an initial training on at least 300 issues. Moreover, once the system is set up, the time required for estimating a new issue ranges from 8 to 15 seconds.

The rest of the chapter is organized as follows: 2.2 gives background notions on story points and issue tracking systems; 2.3 presents the issue selection criteria, the preprocessing, the feature extraction and selection; 2.4 discusses the validation process; 2.5 addresses the research questions; 2.6 discusses threats to validity; 2.7 presents related work and, finally, 2.8 draws the conclusions and presents future work.

2.2 Background

This section describes what story points are and how they are estimated in agile development. In addition, as the issues were retrieved from JIRA, this issue tracking system is also introduced, along with the concept of issue.

2.2.1 Story Points

The story point is a metric used by agile teams to estimate the effort required to complete a development task [9][10]. Rather than quantifying the amount of work that needs to be done to complete a given task, the number of story points is an estimate of how hard a given task is for the development team. As a first step, the team usually agrees on the number of story points that a baseline task deserves. From that point on, effort estimation is based on the comparison with that baseline. Story points are commonly assigned following a sequence which slightly deviates from the Fibonacci series (i.e. 0, 0.5, 1, 2, 3, 5, 8, 13, 20, 40, 100, ∞) [18]. This sequence reflects the uncertainty inherent to the estimation of complex

tasks in real software projects [10]. Note that the story points are meant to represent consensus within a team and not an objective estimate of the task effort. In fact, different teams may, for the same task, provide different estimates.

2.2.2 Effort Estimation – Planning Poker

Most software projects rely solely on human judgment for effort estimation [19]. Among effort estimation techniques based on human judgment, those based on group estimation are the most popular. More precisely, when it comes to story points estimation, the most common approach is the Planning Poker [11]. Planning Poker requires the customer to discuss an issue they wish to be addressed. Then, developers meet for a poker session where, for each issue to be estimated, participants individually select a card with the chosen story points, and finally reveal all the cards at the same time. The developer that provide the lowest and highest estimate must justify their choice, thus eventually triggering further discussion which is followed by another group estimation. The process continues until the team agrees upon a consensus estimate. Note that the story points in our industrial dataset stem from Planning Poker sessions; for the open source projects we do not know whether Planning Poker was adopted.

2.2.3 JIRA

JIRA is a widely known issue tracking system. Here, an issue may represent a software bug, a project task, an enhancement, or anything that requires maintenance activity from the developers [20, 21]. In JIRA an issue is characterized by several standard fields and, quite often, also by fields introduced by JIRA administrators, known as custom fields. Among standard fields, we note: the *Summary* and the *Description*, which report textual information about the issue; the *Assignee*, which refers to the developer appointed to work on the issue; the *Component/s*, that specifies in which module of the project the task has to be implemented; the *Issue Type* (e.g. `Bug`, `New Feature`), which clarifies the required maintenance activity; the *Status* (e.g. `Open`, `Resolved`, `Closed`), which keeps track of the issue life cycle. Agile teams also use the *Story Points* custom field to record the estimated issue effort.

In addition to creating custom fields, JIRA administrators are also allowed to customize field values to satisfy their organization guidelines. This often happens with the *Status* field. In particular, an issue can be characterized by a *Status* marked with values such as `Closed`, `Fixed`, `Completed` or the like, which ultimately have the same meaning. Also the *Resolution* field can be set to different values that share the same meaning, such as `Completed` and `Fixed`. In fact, different organizations generally use a different *Status*/*Resolution* combination to refer to the same stage of the issue life cycle.

2.3 Experimental Setup

This section presents the cases under investigation and the construction of a data set for training and testing the classifier. Then, it describes the approach used for issue selection, feature extraction, and feature selection. All the previous activities are reported in Figure 2.1, for the case when the classifier is used in Planning Poker sessions.

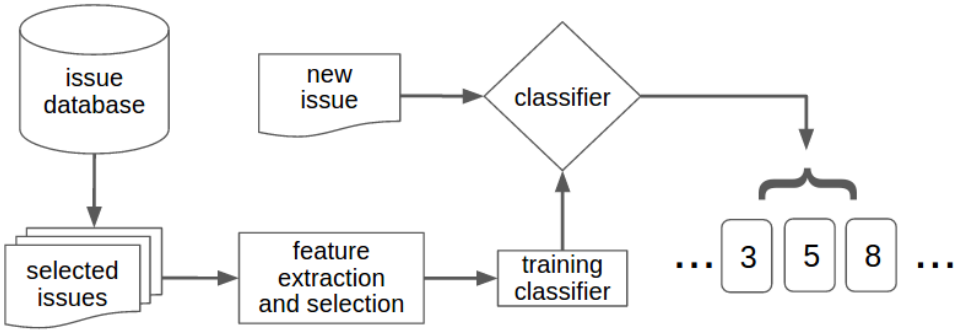

Figure 2.1: Estimation flow chart

Table 2.1: Issue types per project (number of issues (percentage))

Project	Total	Bug	Documentation	Epic	Improvement	New Feature	Story	Sub-task	Task	Technical Debt	Technical task	Wish
APSTUD	228	129 (56.58%)	0 (0.0%)	0 (0.0%)	39 (17.11%)	0 (0.0%)	46 (20.18%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	14 (6.14%)	0 (0.0%)
DNN	858	551 (64.22%)	0 (0.0%)	0 (0.0%)	157 (18.3%)	4 (0.47%)	73 (8.51%)	48 (5.59%)	25 (2.91%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
MESOS	387	169 (43.67%)	26 (6.72%)	0 (0.0%)	101 (26.1%)	0 (0.0%)	6 (1.55%)	0 (0.0%)	84 (21.71%)	0 (0.0%)	0 (0.0%)	1 (0.26%)
MULE	805	440 (54.66%)	0 (0.0%)	0 (0.0%)	153 (19.01%)	67 (8.32%)	9 (1.12%)	1 (0.12%)	135 (16.77%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
NEXUS	421	332 (78.86%)	0 (0.0%)	0 (0.0%)	78 (18.53%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	10 (2.38%)	1 (0.24%)	0 (0.0%)	0 (0.0%)
SCR	**699**	**469 (67.1%)**	**0 (0.0%)**	**0 (0.0%)**	**0 (0.0%)**	**0 (0.0%)**	**36 (5.15%)**	**0 (0.0%)**	**194 (27.75%)**	**0 (0.0%)**	**0 (0.0%)**	**0 (0.0%)**
TIMOB	634	524 (82.65%)	0 (0.0%)	3 (0.47%)	52 (8.2%)	36 (5.68%)	17 (2.68%)	1 (0.16%)	0 (0.0%)	0 (0.0%)	1 (0.16%)	0 (0.0%)
TISTUD	1215	751 (61.81%)	0 (0.0%)	0 (0.0%)	200 (16.46%)	7 (0.58%)	196 (16.13%)	1 (0.08%)	0 (0.0%)	0 (0.0%)	60 (4.94%)	0 (0.0%)
XD	360	87 (24.17%)	0 (0.0%)	0 (0.0%)	34 (9.44%)	0 (0.0%)	231 (64.17%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	8 (2.22%)	0 (0.0%)
all	*5607*	*3452 (61.57%)*	*26 (0.46%)*	*3 (0.05%)*	*814 (14.52%)*	*114 (2.03%)*	*614 (10.95%)*	*51 (0.91%)*	*448 (7.99%)*	*1 (0.02%)*	*83 (1.48%)*	*1 (0.02%)*

Table 2.2: Story points developers estimates per project (number of issues (percentage))

Project	Total	<1	1	2	3	5	8	13	20	40
APSTUD	228	1 (0.44%)	21 (9.21%)	7 (3.07%)	20 (8.77%)	62 (27.19%)	70 (30.7%)	40 (17.54%)	5 (2.19%)	2 (0.88%)
DNN	858	3 (0.35%)	375 (43.71%)	295 (34.38%)	130 (15.15%)	41 (4.78%)	13 (1.52%)	1 (0.12%)	0 (0.0%)	0 (0.0%)
MESOS	387	3 (0.78%)	120 (31.01%)	98 (25.32%)	106 (27.39%)	45 (11.63%)	14 (3.62%)	1 (0.26%)	0 (0.0%)	0 (0.0%)
MULE	805	171 (21.24%)	152 (18.88%)	45 (5.59%)	113 (14.04%)	154 (19.13%)	138 (17.14%)	32 (3.98%)	0 (0.0%)	0 (0.0%)
NEXUS	421	161 (38.24%)	199 (47.27%)	44 (10.45%)	17 (4.04%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
SCR	**699**	**89 (12.73%)**	**174 (24.89%)**	**187 (26.75%)**	**157 (22.46%)**	**72 (10.3%)**	**18 (2.58%)**	**2 (0.29%)**	**0 (0.0%)**	**0 (0.0%)**
TIMOB	634	13 (2.05%)	47 (7.41%)	63 (9.94%)	182 (28.71%)	192 (30.28%)	102 (16.09%)	35 (5.52%)	0 (0.0%)	0 (0.0%)
TISTUD	1215	1 (0.08%)	50 (4.12%)	64 (5.27%)	303 (24.94%)	489 (40.25%)	262 (21.56%)	45 (3.7%)	1 (0.08%)	0 (0.0%)
XD	360	1 (0.28%)	117 (32.5%)	76 (21.11%)	88 (24.44%)	54 (15.0%)	24 (6.67%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
all	*5607*	*443 (7.9%)*	*1255 (22.38%)*	*879 (15.68%)*	*1116 (19.9%)*	*1109 (19.78%)*	*641 (11.43%)*	*156 (2.78%)*	*6 (0.11%)*	*2 (0.04%)*

2.3.1 Cases

The dataset is composed of both industrial and open source issue reports mined from JIRA repositories. The dataset includes different issue types, all with story points, as shown in Table 2.1. Even if the terms used for issue types depends on the project organization, the type Bug is present in every case study and is the most represented type, accounting for more than 60% of the issues in the dataset. The distribution of story points per class is reported in Table 2.2. Here we can see that 89% (5000/5607) of the issues fall within classes ranging from 1 to 8 story points.

Industrial Project

We use 699 issues belonging to the issue tracking system of Inventive Designers, a Belgian software company. The main product-suite, Scriptura Engage, offers solutions to create and manage customer communications for print, email, online, social, mobile, or interactive tasks. For the experiment, one of the development teams was involved. This team is composed of 5 developers and adopts a customized version of the Extreme Programming (XP) process. The team works on issue reports, which describe the smallest units of work. In the last two years they adopted Planning Poker. On these data, we perform a retrospective analysis, namely the data recorded is not influenced by our investigation.

We perform a retrospective analysis on these data, namely, the recorded data is not influenced by our investigation.

Open Source Projects

To allow for comparison against the industrial project, we selected a range of open source projects that use JIRA, provide public access to the repository, and record estimated story points. These projects provide a wide range of possible scenarios in terms of (i) project domain, (ii) number of issues, and (iii) developer experience. As such, these projects limit the threats to external validity of our findings.

The projects we use for the analysis are: Aptana Studio (APSTUD)[1], a web development IDE; Dnn Platform (DNN)[2], a web content management system; Apache Mesos (MESOS)[3], a cluster manager; Mule (MULE)[4], a lightweight Java-based enterprise service bus and integration platform; Sonatype's Nexus (NEXUS)[5], a repository manager for software “artifacts" required for development; Titanium SDK/ CLI (TIMOB)[6], an SDK and a Node.js-based command-line tool for managing, building, and deploying Titanium projects; Appcelerator Studio (TISTUD)[7], an Integrated Development Environment (IDE); Spring XD (XD)[8], a unified, distributed, and extensible system for data ingestion, real time analytics, batch processing, and data export.

The number of selected issues ranges from 228 (APSTUD) to 1215 (TISTUD), amounting to 5607 issues in total.

[1] http://www.aptana.com/. Accessed: 2017-03-19.
[2] http://www.dnnsoftware.com/products. Accessed: 2017-03-19.
[3] http://mesos.apache.org/. Accessed: 2017-03-19.
[4] https://www.mulesoft.com/resources/esb/what-mule-esb. Accessed: 2017-03-19.
[5] http://www.sonatype.org/nexus/. Accessed: 2017-03-19.
[6] https://jira.appcelerator.org/browse/TIMOB. Accessed: 2017-03-19.
[7] http://goo.gl/vGu8k1. Accessed: 2017-03-19.
[8] http://projects.spring.io/spring-xd/. Accessed: 2017-03-19.

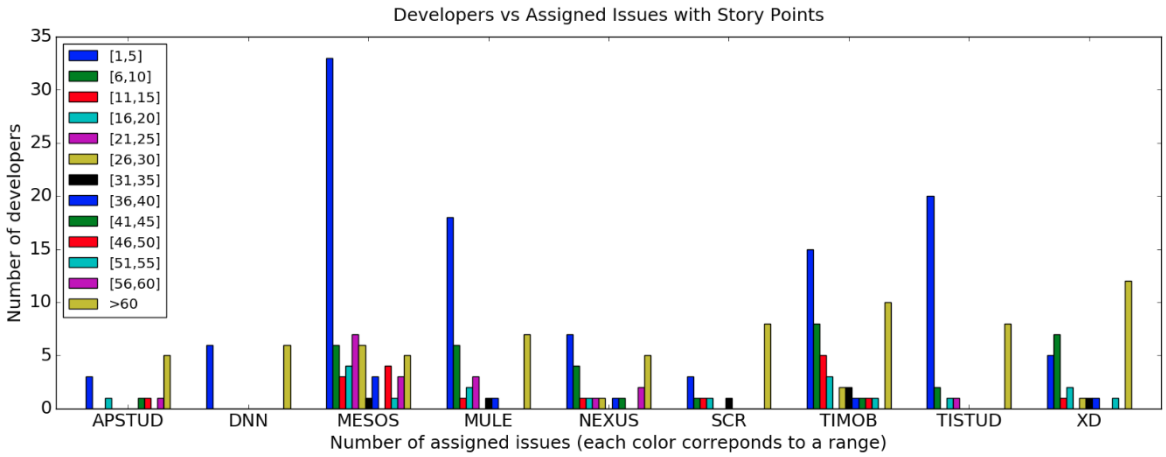

Figure 2.2: Number of developers vs the total number of issues assigned to them across the projects under study. Only estimated issues (i.e., issues with story points) are considered.

The projects involve developers with a different level of experience in story point estimation. Figure 2.2 shows, for each project, how many developers have been working with issues that required an estimation. As we can see, there are a lot of seemingly unexperienced estimators, i.e. developers that have been working only on 1-5 issues with story points.

2.3.2 Issue Selection

To train and test the classifier, issues which satisfy the following conditions are automatically selected:

1. Story points must have been assigned once and never updated afterward. In fact, if the story points estimate gets updated, it may mean that the initial version of the issue report had misleading information, which would confuse the classifier. This explains why we filter out such issue reports.
2. The issue must be *addressed*. We consider an issue addressed when its *Status* is set to `Closed` (or similar, e.g. `Fixed`, `Completed`) and its resolution field is set to `Fixed` (or similar, e.g. `Done`, `Completed`). Note that fields such as *Story Points* and *Description* may be adjusted or updated at any given time. However, once the issue is addressed updates rarely happen. For instance, in the industrial project this event happens for less than 4% (49/1368) of the issues. Here as for the other projects, we filter out issue reports not addressed, because they are likely to be unstable, hence they might confuse the classifier.
3. Once the story points are assigned, the *informative* fields of the issue (i) must be already set and (ii) their value must not have been changed afterward. We define informative fields: *Issue Type*, *Description*, *Summary*, and *Component/s*. We filter out issues whose informative fields are updated after story points initialization because they, again, are likely to represent unstable issues.
4. The values in the story points field must correspond to one of those included in the Planning Poker cards set —0, 0.5, 1, 2, 3, 5, 8, 13, 20, 40, 100, ∞. We consider values out of the series as erroneous entries and filter them out of the dataset.

Prior to issue filtering, the dataset counts 16,523 issues. After applying the first three filtering criteria, the issue number drops to 5,730. Finally, after filtering out issues not in the Planning Poker card set, 5607 issues are left. We only provide these intermediate values since (i) the first three conditions can be applied in any order, and (ii) different filters may filter out overlapping sets of issues. The total number of issues for each project and type is shown in Table 2.1.

2.3.3 Feature Extraction

Certain pre-processing steps are necessary before using a machine learning classifier. In this section, we describe how features were automatically extracted from the issue report fields along with the type of performed pre-processing.

Summary and Description

We use the fields *Summary* and *Description* since (i) they are used by developers for effort estimation and (ii) they have been shown to be effective for story point estimation by Abramhamsson et al. [22].

The textual information from summary and description are joined together to obtain a single text that we will refer to, from now on, as the *context*. Developers can include code snippets in the issue description; as a consequence, the context has always one part hosting natural language text, and sometimes another hosting code. The code snippets are analyzed separately from the rest of the context, since coding words may have a different meaning from those found in natural language text. For instance, "Kill bill" in natural language has a completely different meaning than the statement "bill.kill()" for stopping a process.

To extract features from the context, we employ what is known as a *term frequency – inverse document frequency transformation* (abbreviated as TF-IDF from here on), first on the natural language text and then on the code portion. As TF-IDF needs to be applied, the mono-grams and bi-grams vocabularies corresponding to each of the two text corpus (natural language text and code snippets) are separately built. Stop-words are then filtered out. Stemming is not applied since it revealed to be detrimental to the quality of the estimation.

After building the n-grams vocabulary, term occurrences are computed to transform the contexts with TF-IDF. Namely, a numerical feature is associated to the occurrences of each term in a context. Since we use binary detection, each feature is initially set to either 1 (if found in the context) or 0 (if not found). Afterward, each feature is weighted by dividing it by the n-gram frequency across the corpus.

At this stage, we have a matrix where each row corresponds to an issue, and each element of the row corresponds to the numerical feature associated to a specific term in the issue context. As we obtain one matrix from applying TF-IDF to the natural language portion and another from the code portion, the two matrix are joined to obtain a single one, which we refer to as the feature matrix (rows are issues, columns are features). The feature matrix is subsequently extended with the features extracted from the *Component/s* and the *Issue Type* fields, and the document size.

Component/s and Issue Type

An issue related to a specific component can have, among its JIRA fields, a tag containing related component(s) name(s). After interviewing the developers of the industrial project, we learned that the fields *Component/s* and *Issue Type* were prime candidates for the story point estimation. This was confirmed by our tests, that eventually lead us to exploit the two fields to extract features. We assigned a numerical feature to each component and issue type, corresponding to 1 or 0 if it is used or not in the issue report, respectively.

2.3.4 Feature Selection

To reduce the computational time required for estimation we use two approaches. First a univariate feature selection on the feature matrix is performed, which selects features on the basis of their statistical distribution. Then, a selection based on the classifier is applied. The latter uses the coefficients assigned to the features by the classifier to rank them according to their importance.

2.4 Evaluation

Once the issues are extracted from the issue tracking system and transformed in a suitable format for training, we evaluate the classifier performance.

2.4.1 Classifier Validation

To validate our classifier, we used a 10-fold cross-validation since (i) cross-validation reduces overfitting, and (ii) using 10 folds requires less computational time than a higher number of folds, with negligible differences in the results.

2.4.2 Performance Metrics

To evaluate the performance of the classifier (in terms of estimates) we compute the Mean Magnitude of Relative Error (MMRE) and the accuracy. The former is frequently adopted for evaluating cost estimations in agile software projects [17]. The latter is computed to simplify comparisons with the related works.

The MMRE is defined as the mean value of the Magnitude of Relative Error (MRE) across all the estimates. The MRE definition is:

$$MRE = \frac{|y - \hat{y}|}{y} \tag{1}$$

In our study, y are the story points recorded in the dataset and $\hat{y}$ is the estimate computed by our model.

Accuracy is simply defined as the ratio between the number of correct estimates and the total number of estimates. Total and correct estimates are also reported in the results.

In addition, the confusion matrix was also used to get a more detailed insight into the industrial project [23]. The confusion matrix reports the correct estimates along the main diagonal, whereas the other elements along each row pinpoint which class the incorrect estimates were confused with.

Table 2.3: Performance metrics for the industrial project (699 issues) with SVM, NB, KNN, and DT estimators

Estimator	Correct estimates	Accuracy	MMRE
SVM	413	0.59	0.50
NB	309	0.44	0.85
KNN	249	0.36	0.70
DT	158	0.23	0.98

2.5 Results and Discussion

This section presents the results of the experiment on the industrial and the open source projects. For each research question we first discuss its motivation, followed by the approach used and, finally, the results.

RQ1. What performance can be expected from a machine learning classifier when estimating story points?

Motivation. Since we aim at providing real-time support to the development team in story point estimation, we need to identify a well-suited classifier for the task, also taking into account the estimation time. Hence, we need to evaluate the classifier performance in terms of both correctness of the estimation (e.g. MMRE) and computational time.
Approach. We investigate which typical machine learning classifier is the best one for estimating story points. For this purpose we compare the following algorithms: Support Vector Machine (SVM), K-Nearest Neighbor (KNN), Decision Tree (DT), and Naive Bayes (NB). The algorithms SVM, KNN, and DT were chosen since they are generally well suited for text classification applications. Moreover, the KNN and DT produce easily interpretable classifiers. Finally, the algorithm NB was chosen due to its simplicity and yet high accuracy [24].

Once identified the best algorithm, we use it on the industrial and open source projects. It is beyond the scope of this study to present all of the analysis results; rather, we present the results achieved with the best algorithm. The other results are available in the replication package[9].

To evaluate the performance of the classifier we compute both MMRE and the accuracy to simplify comparisons with related works. In addition, we compare our results with a ZeroR classifier, a classifier that always selects as the outcome the most represented class. We use the ZeroR to verify how much better is our classifier compared to arbitrary approximation. We complete the discussion by providing the computational time required by the estimation.
Findings. We present the results achieved on the industrial project and on the open source projects in two separate sections.

Industrial Project. Table 2.3 reports the performance obtained on the industrial project using different algorithms. The SVM achieves the best performance having: MMRE 0.5, 413 correct estimates, and 59% accuracy. This result is aligned with the state of the art since the SVM has been proven to be well suited for text categorization [25].

To better understand the results achieved by the SVM classifier, we report in Table 2.4 the confusion matrix for the industrial project. Here, we can observe that, for any given

[9] http://goo.gl/mLe26i. Accessed: 2017-03-19.

Table 2.4: Confusion Matrix for the industrial project (SCR)

	<1	**1**	**2**	**3**	**5**	**8**	**13**
<1	**46**	13	16	11	3	0	0
1	11	**116**	26	11	10	0	0
2	12	27	**119**	24	4	1	0
3	7	16	25	**100**	6	3	0
5	3	11	8	16	**30**	4	0
8	1	5	5	2	3	**2**	0
13	0	0	1	1	0	0	**0**

class, most issues are correctly classified (main diagonal). In addition, when the classifier misclassified an issue, estimated story points fell in adjacent classes. However, classes 8 and 13 do not follow this positive trend, most likely because there were not enough training data; these categories only account for 2.87% of the total amount of issues. The result is encouraging since it shows that estimates tend to converge toward the correct estimation.

Open Source Projects. Table 2.5 presents the performance achieved by the SVM classifier on the eight open source projects. For comparison, the results achieved on the industrial project are reported in bold. Here, we can notice that the worst performance of the classifier is a MMRE of 1.07 on the MULE project. Leaving out this exception, APSTUD and DNN respectively represent the (second) worst and the best case study for the SVM classifier. For APSTUD, accuracy and MMRE are, respectively, 0.63 and 0.61, whereas for DNN, accuracy and MMRE are, respectively, 0.78 and 0.16. In light of the foregoing, the performance on the industrial project falls in between.

In order to compare our classifier with the state of the art, we can refer to the work by Abrahamsson et al., where a classifier for story point estimation is presented [22]. Inferring the development time via story points, they compute the MMRE as the difference between the actual (provided by developers after implementing the task) and estimated development time (provided by the classifier) [22]. Their classifier, tested on two industrial projects, led to a MMRE higher (worst) than that achieved by our classifier.

Another comparison is made with the work of Augen [26], who ran an experiment on 19 developers using Planning Poker, computing the MMRE by considering the story points assigned by developers before and after task implementation. He achieved an MMRE of 0.48 [26], which is aligned with the average in our analysis (0.47). From this point of view, our classifier performs as good as a human, hence is sufficiently reliable to serve in an actual poker planning session.

We evaluate the reliability of the SVM classifier by comparing its performance with the ZeroR classifier (Table 2.6). The last row of Table 2.6 shows that the SVM classifier is better than the ZeroR, since it has, on average: (i) a two times higher (0.64/0.34) accuracy and (ii) a MMRE roughly 25% (0.47/0.64) lower. On the industrial project, the performance is even better: the accuracy is 2.18 higher, and MMRE is around 63% of the MMRE achieved by the ZeroR. The results achieved by the SVM and the ZeroR classifiers can be partially explained looking at the distribution of story points in both projects. MULE has almost the same number of issues in each of the classes <1, 1, 3, 5, and 8; on the contrary, DNN has more than 0.40 of the issues just in class 1, and more than 0.33 in class 2. In view of these differences,

Table 2.5: SVM performance metrics, with total and correct estimates.

Project	**Total est.**	**Correct est.**	**Accuracy**	**MMRE**
APSTUD	228	143	0.63	0.61
XD	360	223	0.62	0.42
MESOS	387	223	0.58	0.39
NEXUS	421	341	0.81	0.16
TIMOB	634	380	0.6	0.6
SCR	**699**	**413**	**0.59**	**0.5**
MULE	805	416	0.52	1.07
DNN	858	669	0.78	0.16
TISTUD	1215	810	0.67	0.35
mean	*623*	*402*	*0.64*	*0.47*

Table 2.6: ZeroR performance metrics, with total and correct estimates.

Project	**Total est.**	**Correct est.**	**Accuracy**	**MMRE**
APSTUD	228	70	0.31	1.2
XD	360	117	0.32	0.45
MESOS	387	120	0.31	0.44
NEXUS	421	199	0.47	0.46
TIMOB	634	192	0.3	0.92
SCR	**699**	**187**	**0.27**	**0.79**
MULE	805	171	0.21	0.62
DNN	858	375	0.44	0.33
TISTUD	1215	489	0.4	0.52
mean	*623*	*213*	*0.34*	*0.64*

it is understandable why both the ZeroR classifier and, to a lesser extent, the SVM classifier scores a better performance on the DNN project.

All of the analysis were performed on a machine equipped with an Intel® Core™ i7-6500U quad-core CPU at 2.50GHz, with 8 GB DDR3 SDRAM. Running a 10-fold cross-validation on TISTUD, the project with more than 1200 issues, requires only 6 seconds. In addition, once the system is set up, the total required time for retrieving a new issue from the issue tracking system repository and estimating its story points ranges from 8 to 15 seconds. Thus, the proposed classifier is fast enough to be integrated in project planning sessions (e.g. Planning Poker).

A SVM classifier for story point estimation can achieve a MMRE spanning between 0.16 and 0.61, namely comparable with developer estimates. Moreover, the estimation process takes less than 6 seconds.

Table 2.7: Top 10 features (positions 1-5)

Project	1	2	3	4	5
APSTUD	(typestory, 6.62)	(change, 5.84)	(invoke, 5.81)	(componenthtml, 5.5)	(against, 5.43)
DNN	(typestory, 10.1)	(typebug, 9.38)	(dnn search, 6.6)	(for user, 6.56)	(from page, 6.53)
MESOS	(what, 5.38)	(on, 5.23)	(necessary, 5.17)	(statistics, 5.14)	(in, 5.1)
MULE	(execution, 10.63)	(componentextensionsapi, 10.49)	(async, 10.3)	(consistent, 10.17)	(consistency, 9.95)
NEXUS	(minor, 5.24)	(repository metadata, 4.93)	(seeing, 4.85)	(detected, 4.83)	(clicking, 4.83)
SCR	**(tests for, 7.8)**	**(scenario, 7.48)**	**(services, 7.33)**	**(included, 7.29)**	**(started, 7.25)**
TIMOB	(txt, 6.72)	(classic, 6.41)	(create an, 6.28)	(we have, 6.02)	(great, 5.73)
TISTUD	(npe, 9.11)	(each, 7.7)	(update process, 7.56)	(switching, 7.31)	(typebug, 7.23)
XD	(module, 8.11)	(consider, 7.77)	(changed, 7.54)	(containers, 7.5)	(typebug, 7.47)

Table 2.8: Top 10 features (positions 6-10)

Project	6	7	8	9	10
APSTUD	(need to, 5.37)	(down, 5.34)	(at, 5.28)	(now, 5.26)	(top, 5.25)
DNN	(get, 6.4)	(usage, 6.39)	(platform build, 6.37)	(executed, 6.33)	(description, 6.31)
MESOS	(fetchercachetest, 5.05)	(make, 4.93)	(verify, 4.9)	(attempts, 4.9)	(run, 4.81)
MULE	(java, 9.83)	(concurrent, 9.69)	(to read, 9.67)	(including, 9.59)	(payload value, 9.56)
NEXUS	(properly, 4.72)	(in case, 4.59)	(hide, 4.55)	(not, 4.55)	(testsuite, 4.46)
SCR	**(cache, 7.1)**	**(vm, 7.06)**	**(fine, 6.97)**	**(do not, 6.87)**	**(next to, 6.77)**
TIMOB	(code, 5.7)	(get, 5.67)	(should use, 5.55)	(machine, 5.54)	(root, 5.52)
TISTUD	(to remove, 7.05)	(detection, 6.98)	(defined, 6.98)	(case where, 6.78)	(read, 6.77)
XD	(because, 7.2)	(so we, 7.18)	(to get, 7.11)	(supports, 7.09)	(out, 7.07)

RQ2. Which attributes of an issue report are relevant for estimating story points?

Motivation. We want to analyze the top 10 features used by the classifier. We are interested in contextualizing these features within the project from where they come from in order to show their meaningfulness and validate the consistency of the results obtained in RQ1. For this reason, we only refer to the SVM classifier, since it achieves the best performance.
Approach. We sort the features used by the classifier in RQ1 by their weight, in descending order. With regard to the weight assigned for feature ranking, the absolute sum of the features coefficients over all the classes was computed as a proxy of the feature importance. Finally, we provide a qualitative analysis of the keywords used across the projects.
Findings. Tables 2.7 and 2.8 report the top 10 features for each project in the dataset. Each feature is reported along with its weight. The Tables present text in the form of single keywords (e.g. `UTC`), bigrams (e.g. `ci bug`), components (e.g. `component html`), and issue type (e.g. `type bug`). Being in the top 10, all the previous informative fields are extremely relevant for the classification. To a lesser extent, also the length of the description is relevant for classification purposes. With TIMOB, the feature length is ranked 322, whereas with MESOS its rank is 472. We confirm the results found by Abhramsson et al. [22], namely, that textual information and its length are effective predictors for story point estimation.

For each project, a different set of features appears in the top 10. This result is consistent with the fact that projects have different goals and a specific vocabulary. For instance,

the DNN project is a content management system. Here views and user searches are core aspects, and keywords such as `from page` and `dnn search` reflect this relevance. Similarly the MULE project, a runtime engine for combining data and application integration across legacy systems, uses keywords such as `consistency, async, concurrent`. Finally, we can also notice that there are common features across projects, such as the maintenance type (`type bug`, `type story`). This result is consistent with Murgia et. al, who found that the issue fixing time is affected by the type of maintenance performed [27]. Except for the maintenance type, projects tend to have specific keywords. As a consequence, a machine for story point estimation requires a specific training on the project issues.

> *In an issue report, the fields containing summary, description, names of related components, and issue type provide relevant features for story point estimation. Most frequently, these features are project dependent.*

RQ3. How much training data is needed to obtain a satisfactory estimation?

Motivation. We are interested in introducing the classifier during the developers' estimation process. However, in the first stage of the project development, the number of issues per class may not be enough to ensure proper training. As a result, the provided estimates would be poor and unsatisfactory. Therefore, we investigate the amount of training data needed to obtain a sufficiently precise classifier.

Approach. We use the SVM classifier since it achieves the best performance in RQ1. For this classifier, we investigate how the size of the training set affects its "learning curve". The learning curve describes how the performance of the classifier (y-axis) evolves along with the number of issues considered (x-axis). To reproduce a real scenario, we first sort all issues by their estimation age, namely by the time of story points initialization. Then we divide the issues in blocks[10] of 20. Using the first block, we evaluate the MMRE by means of a 10-fold cross-validation. We repeat this procedure progressively considering 40, 60, and so on issues. By using the 10-fold cross-validation, we can evaluate the evolution of the classifier performance against the number of blocks considered, and compare it with the results reported in RQ1.

Findings. Figure 2.3 shows the evolution of the MMRE for all 9 projects. Here we can observe that below 200 issues, many projects have an erratic evolution of the MMRE. For instance, the projects SCR, MESOS, APSTUD, TIMOB show MMRE variations higher than 0.2. Over 300 issues, with the exception of MULE, all projects achieve a MMRE lower than 0.62. In particular, the projects DNN and NEXUS reach a MMRE lower than 0.2.

In any project, estimates quality changes along with the number of issues used for training. Except for TISTUD, with more than 300 issues we cannot observe a clear decreasing (nor increasing) trend of the MMRE. Yet, we can only speculate that the value of MMRE would stay "limited". Indeed, with SCR, TISTUD, TIMOB, NEXUS, DNN, XD, and MESOS, the MMRE

[10] We consider a block size of 20 as a trade-off. In fact, we observed that smaller blocks (e.g. 10 issues) generate too many points and thus a clotted graph. On the other hand, for small projects, bigger blocks (e.g. 50 issues) generate too few points for any trend to be observable.

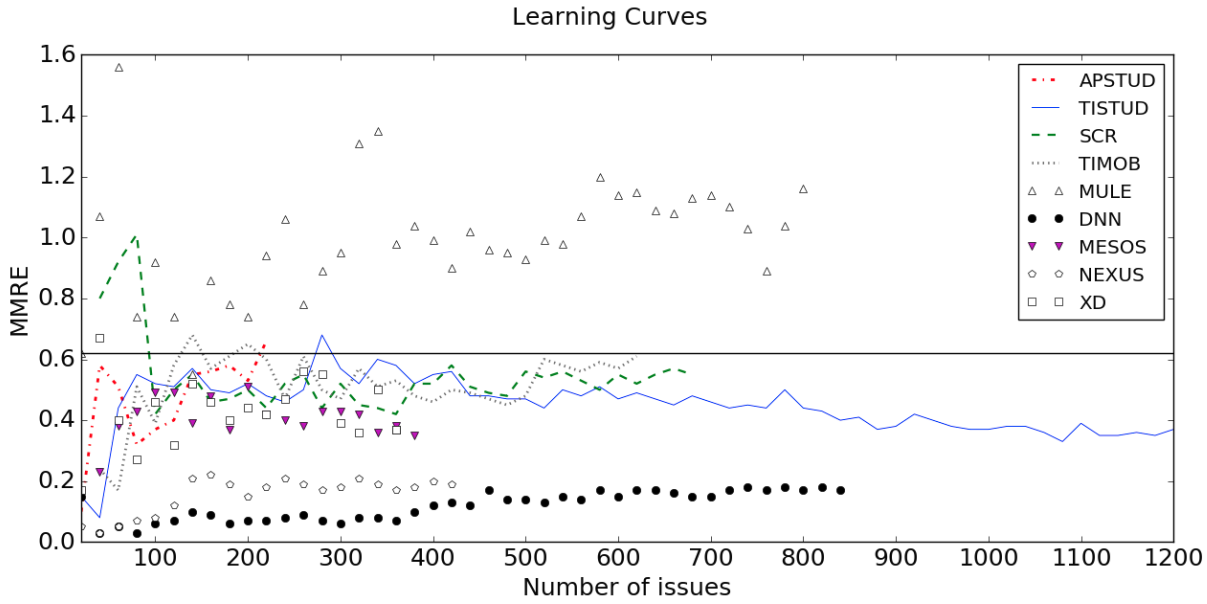

Figure 2.3: Learning curves with line at MMRE=0.62

shows a variation[11] which is less than or equal to 0.135. It is then pivotal to accompany the classifier evaluation with the number of issues used for its training.

In the graph, we may observe several peaks. We analyzed a few cases and, as a representative one, we took the peak value of TISTUD when the number of issues is 280. We found out that such variations are mainly caused by misclassifications involving the inclusion of issues with a higher number of story points[12].

> *It is better to train the classifier with more than 200 issues before employing it for story point estimation. If 0.61 is a satisfactory value for the MMRE, the team can generally start using the classifier with just 300 issues. However, the performance achieved depends on the project.*

2.6 Threats to Validity

In this section we discuss factors which can hinder the validity of our study, and the measures taken to counteract them.

Construct validity. MMRE is not necessarily the most suitable metric to evaluate a story point estimator. We chose the MMRE since its wide adoption allows for an easy comparison with the state of the art. Nevertheless, as story point classes are generally unbalanced,

[11] Using TISTUD, the project with the highest variation of MMRE after 300 issues, as a reference, we observe that 0.6 and 0.33 are the highest and lowest MMRE values, namely, at a distance of 0.27 (or ±0.135) MMRE with respect to the value between 0.6 and 0.33.

[12] This result was somehow expected since the classifier cannot correctly estimate story points for class of issue reports barely represented. Even a slight change in their population at the beginning of the learning curve may lead to a considerable change in MMRE.

and misclassifications are supposed to have a different weight according to the true and predicted classes, the use of a specifically tailored metric for story point estimation would have been advisable.

Internal validity. The experiments were performed under the assumption of a causal relationship between the textual information on an issue report and the story point estimation. However, there are confounding factors which can influence the estimation process, such as company political pressures, the organization estimation process, or the expertise of the issue reporter. Developer's expertise, for instance, can lead to poorly written issue reports. Nevertheless, as pointed out in the experimental setup, we applied a whole range of selection criteria specifically aimed at minimizing the effects of misleading issue reports.

External validity. Given the limited number of projects in our study, namely 8 open source and 1 industrial, we can not assume that the observed estimation performance holds true for other projects. More specifically, estimation processes and company politics might lead to different results. However, since we selected heterogeneous software projects with highly varied story point distributions, we are confident that these threats to external validity are limited.

In addition, we have to remark that we only relied on JIRA repositories. However, the proposed estimator is based on textual information which can be extracted from any issue tracking system, hence basically requiring to only adapt the software to a different API.

Another limitation might be the lack of issue reports at project start. Under this condition, the estimator can be set to only provide estimates when its confidence is higher than a specified threshold.

2.7 Related Work

A realistic and reliable effort estimation process is crucial for a correct project planning. In the agile domain, researchers found that effort is commonly estimated in story points, and that the estimation techniques based on human judgment, such as the Planning Poker, are the most frequently applied [17]. Several studies discussed the problems associated to these estimation strategies.

Hughes showed that judgmental bias of either social or cognitive nature can affect estimations. Social judgmental bias, which may for instance be caused by managerial pressure, can lead estimators to reduce the information used with the aim of simplifying a development task to one of manageable proportions [28]. Abdel-Hamid et al. focused on the effects of schedule compression and pointed out that schedule pressure via underestimation turns into a higher development cost and bug rate [29]. Finally, DeMarco reported that human estimators may tend to underestimate the time required to do a task when they are appointed as assignees for that task [30].

Previous results give room to the adoption, during developer estimation meetings, of a machine-mediated estimation capable of grounding its outcome to objective data. Regarding this matter, several studies have investigated the adoption of a machine for effort estimation. Generally, these approaches involve machine learning models which, via different algorithms, estimate the time required to fix a bug [31][32][33]. However, there are also cases where Neural Networks [34] and association rules have been adopted [35]. Despite the interest on issue effort estimation, none of these studies estimate the effort in terms of story points. As far as the authors know, only Abrahamsson et al. built a classifier for es-

timating story points [22]. They used the classifier on the user stories provided by an agile company with more than 1325 issues[13] [22]. The best result they achieved was a MMRE of 0.9, obtained by using the SVM algorithm. Similarly to Abrahamsson et al., we tried different machine learning classifiers to estimate story points. We confirm that the SVM algorithm is the best one for processing the issue reports; however, through leveraging different features, we achieved better results. Using more than 300 issues, we obtained a MMRE lower than 0.61 in 8 out of 9 analyzed projects. In addition, we also discussed how the number of issues affects the MMRE.

On the human side, Augen [26] analyzed developer-mediated estimation of user stories. He focused on 101 estimates reported by an XP development team during release planning. He compared the story points estimated by developers at the initial release planning meeting with the story points effectively assigned once the task was implemented. The study reports that developer estimations achieve a MMRE of 0.48. From this point of view, our classifier —reaching (on average) a MMRE of 0.46— produces estimates aligned with those provided by developers.

2.8 Conclusion and Future Work

Effort estimation is a cornerstone of every software project. In the agile context, effort estimation is frequently done via *story points* (an estimate which is a proxy for the effort needed for the completion of a task) and group discussions (to achieve consensus on a given estimate). All these estimation techniques are based on human judgment, that has clear advantages but some drawbacks as well. To counteract these drawbacks, we explore the feasibility of a tool for supporting human judgment during story point estimation. More specifically, we propose a machine learning classifier capable of processing an issue report and providing an effort estimate in terms of story points in real-time. Based on an analysis involving one industrial project and eight open source projects, we demonstrate that such tool support is feasible. Indeed, we implement a classifier which achieves —for all projects but one— an MMRE spanning from 0.16 to 0.61, after an initial training of at least 300 issues. Moreover, once the system is set up, the time required for estimating a new issue ranges from 8 to 15 seconds.

We also found that issue type, summary, description, and related components carry project dependent information that contain relevant features for story point estimation. Finally, an MMRE of 0.61 is generally obtainable by training the estimator on more than 300 issues.

As a future development, we plan to improve the estimator usability. For this reason, the estimator will (i) be implemented as a JIRA Cloud add-on [36], and (ii) be able to highlight, in the issue report, the most important keywords adopted for the estimation. The latter functionality would reduce the chances of having relevant keywords passing unnoticed. Further down the line, we plan to investigate (i) the role of emotions in the estimation process [37] and (ii) developer-estimator interaction. For the latter, it is indeed known from literature that humans may trust more their own peers than a machine [38].

[13] In the paper Abrahamsson et al. use the classifier also on a commercial project. However, this project has only 13 issues.

Chapter 3

The Evolution of Knowledge in the Refactoring Research Field

3.1 Introduction

It has been 15 years since the first edition of Fowler's work [39] and even more since the time of Opdyke's PhD dissertation [40]. It is time for researchers involved in software engineering to ask what has been done so far and which are the future perspectives in the research field of refactoring.

Several authors have recently focused on the research on Refactoring [41][42][43][44]. This suggests that researchers are showing interest about the current state and future directions of this research area. However, with the exception of [41], most of the previous works addressed specific aspects (e.g. bad smells) and did not focus on the entire scientific production on Refactoring.

This chapter tackles the problem of representing the up-to-date body of knowledge using an approach based on science mapping [45]. Science mapping gives researchers a means for representing the existing relations among research themes within a research field, allowing to discover research sub-areas, and revealing hidden information on the cognitive, intellectual and social structure, as well as the evolution of a scientific discipline.

Our objective is to reveal existing relationships among the main themes investigated by researchers in the Refactoring research field and their evolution. The best model to represent the properties we are interested in is the so called *co-occurrence network*. In a co-occurence network, nodes are the keywords found in publications corpus, and edges represent co-occurrence of two keywords within the same publication.

To this purpose, we followed the process illustrated by Cobo et al. [45]: first, we retrieved bibliographic data by the ISI Web of Science (ISIWoS) database[1], one of the most important bibliographic database; second, we discharged useless or redundant data through a pre-processing stage; then, we built the co-occurrence network; and finally, we performed a complete science mapping analysis on the retrieved network.

The obtained results allowed us to identify main and side research themes on software refactoring, to reveal research trends, and to recognize abandoned topics. Researchers could

[1] https://webofknowledge.com. Accessed: 2017-03-06.

exploit our findings to understand how their specific interests fit in the wider context of the research on Refactoring, or how the topic they are most interested in evolved.

It must be noted that a similar approach was already presented in [46] and [47], but it was applied on a different context. In fact, no similar studies had been conducted on the Refactoring research field. The most comprehensive related works on Refactoring are that of Mens et al. [48] and the more recent work of Abebe et al. [41]. However, in the mentioned studies the authors used a totally different approach, and did not consider bibliographic networks.

This chapter is structured as follows. In 3.2 we report useful background knowledge for the understanding of the following paragraphs. In 3.3 we thoroughly describe the adopted methodology, whereas in 3.4 we give a detailed description of how we set our experiment. 3.5 we discuss the results of our study. In 3.7 we report some threats to validity. Finally, in 3.8 we draw some conclusions, and also propose interesting future extensions of the work presented in this chapter.

3.2 Background

Science Mapping provides a spatial representation of a discipline, highlighting the reciprocal relationships among its elements [49]. This way it is possible to study the structure and evolution of a discipline. In order to achieve this goal it is possible to use several techniques (co-citation analysis, co-word analysis, etc). We applied this mapping to the research field of Refactoring in software engineering, performing a co-word analysis on the keywords used by authors to characterize their papers. Introduced by Callon [50], co-word analysis focuses on terms contained in documents, analysing how these terms are shared among different elements of a body of knowledge. Co-occurrence analysis has been used in several research fields, specifically of technical areas, including software engineering [51]. A work dealing with a similar problem using the same approach, but applied to a different research field (chemical engineering), is that of Peters et. al [47]. In the field of software engineering there is only the study of Coulter [51], that analyzed the research in software engineering using a keyword co-occurrence analysis. In the present work we focus on a specific research niche, software Refactoring, and we use a wider set of methods and tools. In the Refactoring field there has been recently some attempts to summarize the results collected during the last years. By and large the authors of these works focus on specific issues (refactoring opportunities [43], code smells [44], etc.) and not on the entire literature on Refactoring. For example Al Dallal et al. [43] carried out a research focused on methods to detect refactoring opportunities, whereas in Vale et al [44] the authors studied the literature on Refactoring with specific regards to the code smell they fix. On the other hand, to the best of the authors' knowledge, systematic surveys about Refactoring where carried out by Mens et al. [48] and Ababa et al. [41]. In the first work the authors studied the available literature on techniques and formalism, types of software artifacts, effect of Refactoring on software process and Refactoring activities whereas in the second one the authors focused their attention on trends and opportunities. None of these works used the approach proposed in the present paper.

In this work we attempt to study the cognitive structure of the software refactoring research field, and how this structure evolved during the a period from 1999 to 2015. To perform this analysis we rely on the SciMat software application [52].

3.3 Methodology

For the science mapping analysis we adopted the workflow presented in [45], that we briefly outline in the following paragraph.

3.3.1 Science Mapping Process

A science mapping process is basically structured in the following steps:

Data Retrieval The main source for our research is the ISIWoS bibliographic database [2]. We queried the ISIWoS database to retrieve works dealing with software Refactoring.

Preprocessing In order to avoid misleading results, a preliminary preprocessing phase is required. This implies to look for duplicated records, to remove plurals, and so on. To obtain relevant information we discarded redundant or useless information. For example, we decided to discard all-comprehensive keywords, like "refactoring" (all papers deal with it), "software", and similar keywords.

Network Extraction In this step we built a co-occurrence network, namely a network where nodes are keywords and edges represent a co-occurrence of two keywords in the same paper. Edges can also be weighted by the number of papers where the two keywords co-occurred. In this preliminary analysis we set all weights to one.

Normalization As a normalization metric we used the Jaccard Index.

Mapping In this phase firstly a clustering algorithm is applied and than then a mapping process to associate documents to their proper cluster is performed.

Analysis Several analysis could be performed at this step. In our case we are interested in a co-word analysis.

Visualization In this work we represent our data through a strategic diagram.

Interpretation Based on the knowledge of the field, it is possible to draw some conclusions about the structure and evolution of the research field on Refactoring.

3.3.2 Bibliographic databases

There are several bibliographic databases: ISIWos[3], Scopus[4], PubMed[5], GoogleScholar[6], just to name a few of the most popular, having each of them its own peculiarity. In this work we decided to use data retrieved from ISIWoS because it presents highly standardized data, and this allows an easy replication of the results. It is highly structured, and this renders queries quick and reliable. Finally it is recognized world wide and it is accessible by all main research institutions or societies, like universities or research centres.

[2] See footnote 1.
[3] See footnote 1.
[4] http://www.scopus.com. Accessed: 2017-03-06.
[5] http://www.ncbi.nlm.nih.gov/pubmed. Accessed: 2017-03-06.
[6] http://scholar.google.com. Accessed: 2017-03-06.

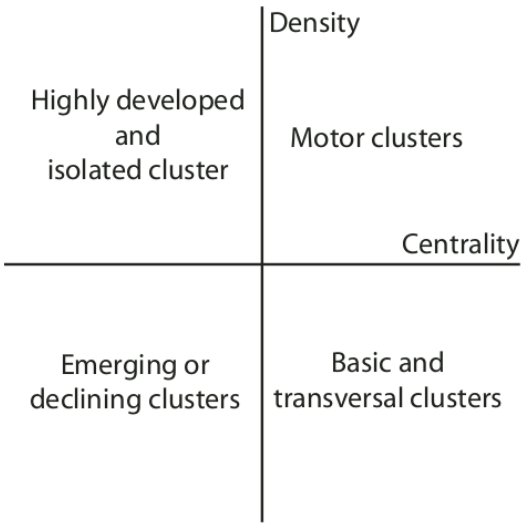

Figure 3.1: Cluster typologies in a strategic diagram

3.3.3 Co-occurrence Network

This work is specifically aimed at studying the conceptual structure and evolution along the time of the research field on Refactoring. This kind of studies might be performed by means of the co-occurrence network. A co-occurrence network (and the relative matrix) is composed by nodes that represent keywords associated to a paper by their authors, and edges connecting nodes represent the co-occurrence relationships among them. When two keywords i and j appear together in the same document, then a co-occurrence relationship between them is detected. It is also possible to associate the number of document where the keywords appear to the nodes and the number of document where they co-occur to the edges. In this work we retrieved the co-occurrence adjacency matrix that can be associated to a corresponding co-occurrence network. After having retrieved the co-occurrence matrix, we computed the similarity among keywords by means of an index, the *equivalence index* [53]: $e_{i,j} = \frac{c_{ij}^2}{c_i c_j}$, where c_{ij} is the number of documents in which the keywords i and j appear together, whereas c_i and c_j represent the number of documents in which the keywords i and j appear, respectively. If the two keywords occur together in each document, then the equivalence index is 1, whereas if they are never found together in the same document the equivalence index is 0. After computing the equivalence index it is possible to perform a clustering analysis, for example by means of the Simple Centre algorithm [51], in order to group keywords into thematic areas and establish associations between words. In this way it is possible to obtain the so-called "thematic networks". The name assigned to each thematic network corresponds to the most relevant keyword for the associated theme (usually, the most central keyword for the theme, or cluster).

3.3.4 Strategic Diagram

Once thematic networks between keywords are created, we can report the results on a *strategic diagram* like that portrayed in Figure 3.1. A strategic diagram is a two dimensional diagram where thematic areas are reported based on their associated values of *centrality* (on

the horizontal axes) and *density* (on the vertical axes). Centrality and density are two metrics that measure the interaction of bibliographic unit of some kind, not only keywords. Specifically, centrality measures how strong are the ties of a given theme with the other themes and is defined as: $c = 10\sum e_{kh}$, where k ranges over words belonging to the network of the given theme, and h are words belonging to the other different thematic networks. It returns an information about how much a theme is relevant with respect to the others of the same research field and could be seen as a measure of external cohesion with other themes. On the other hand density represents a measure of internal cohesion, since it measures how strong are the links among keywords belonging to the same theme. It is defined as follows: $d = 100\sum \frac{e_{ij}}{n}$, where i and j are two words belonging to the same theme and n the total number of the keywords for that specific theme. In some sense, the density measures how much the theme is a well developed topic. Depending on their reciprocal position in a strategic diagram, research themes could be classified as:

Motor Themes : characterized by both high centrality and density, they are well developed and relevant themes.

Peripheral themes : although well developed themes, they are more isolated if compared to the others. They usually correspond to highly specialized themes.

Emerging or declining themes : Having both low centrality and density, these themes are disappearing or emerging themes.

Basic themes : Not well developed but relevant themes. They are usually general themes.

Their respective position is shown in Figure 3.1. Strategic themes are usually represented as spheres of different size, this way adding a third dimension to the diagram. Spheres' size is generally associated to a bibliometric measure of performance (e.g. average number of documents).

3.4 Experimental Setup

The analysis we performed is structured in several steps. First and foremost we queried the selected bibliographic database, *ISIWoS*. We searched for the word "refactoring" either if it appeared in the title, in the abstract, among the keywords or in the text. We considered only works written in English, selected among books, articles and reviews . At the end we retrieved 432 works. In order to avoid as much as possible the occurrence of false positive results, namely works that were retrieved by the search engine but that were not actually related to software Refactoring, we manually checked out each result. We found and discarded 16 out of the 432 works that are not actually related to the Refactoring topic (at least not in the sense intended in the software engineering field) and that ended up in the ISIWoS search results because the term "refactoring" was in the abstract, though it has a different meaning.

Periods We consider 3 periods, each of them of 5 years, starting from 2001 to 2014. Our query to ISIWoS did not return works for 1999, 2000 and 2015.

Network We retrieved the co-occurrence network, specifically considering only author's keywords.

Table 3.1: Number of documents per year from ISIWoS

	Years	N. Documents
P1	from 2001 to 2004	66
P2	from 2005 to 2009	148
P3	from 2010 to 2014	217

Normalization As a measure of normalization we decided to use the Jaccard Index [47].

Clustering Algorithm In order to retrieve information about how the nodes of the co-occurrence network are connected we need to compute which and how many clusters are structured in. We chose Simple Centre Algorithm as a clustering algorithm [51].

Mapper To map documents to each cluster we consider a Core Mapper. It returns the documents that are present in at least two nodes, and can be generalized to k-nodes.

Performance As a measure of performance we consider the average number of citation and the H index [54].

3.5 Results

In this Section we report the results of our analysis. In Table 3.1 is reported the distribution of works for each of the analyzed periods: 3 periods of 5 years from 1999 to 2015, discarding years 1999, 2000 and 2015, for which *ISIWoS* do not report works. The number of works dealing with Refactoring (or related issues) has an increasing trend during the analyzed periods. For each diagram the volume of the spheres is proportional to the number of documents.

As we can see from the diagrams the number of thematic networks increased along the years being 3 themes present in P1, 5 in P2 and 7 in P3. It is worth to point out that, by and large, core themes does not appear in the different diagrams, with the exception of *Architecture* in P1 and P2, and *Object-Oriented systems* in P2 and P3. According to the definitions presented in Section 3.3 we can distinguish motor themes, that for each period are respectively: *Object-Oriented Design* for P1, *Agile Methodologies* and *Languages* for P2 and *Experimentation* for P3. It is worth to note that we do not have themes that belongs to the second and fourth quadrant (considered counterclockwise from the top-righ quadrant) in P1, whereas in P2 and P3 all the quadrants host at least a thematic network, even if in P2 both the two themes in those quadrants, *Reverse Engineering* and *Object-Oriented Design*, are very close to the origin.

In P2 the theme *Architecture* has both lower density and centrality if compared to the previous period. The same occurs to *Object-Oriented Systems* between P2 and P3. While *Architecture* belong to the same quadrant both in P1 and P2, an interesting case in that of the *Object-Oriented Systems* theme between P2 and P3, being in the fourth quadrant (basic theme) in P2 and in the third quadrant (emerging or declining term) in P3. Moreover, whereas the number of papers in *Architecture* increased along the time, in *Object-Oriented Systems* it decreases from 8 to 3 between P2 and P3. Additionally, despite with different positions, we can distinguish elements in the high developed themes areas for P2 and P3: respectively *Reverse Engineering* for the former and *Refactoring Classification* for the latter. For the

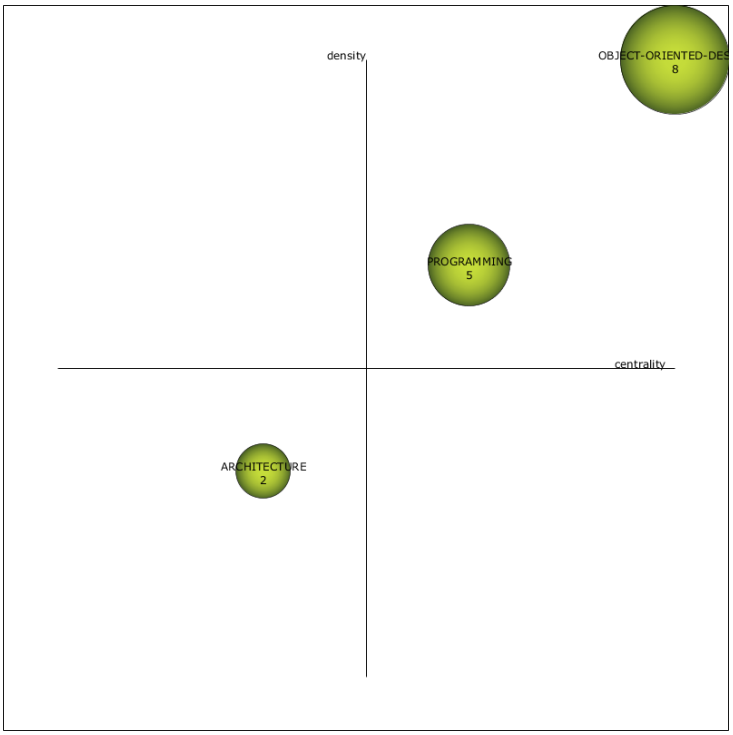

Figure 3.2: Strategic diagram for P1 (2001-2004).

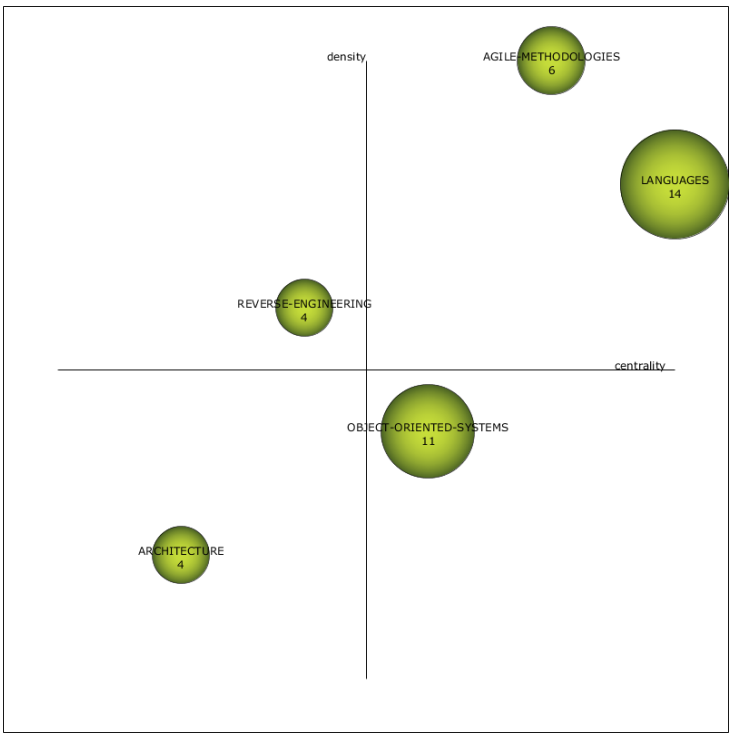

Figure 3.3: Strategic diagram for P2 (2005-2009).

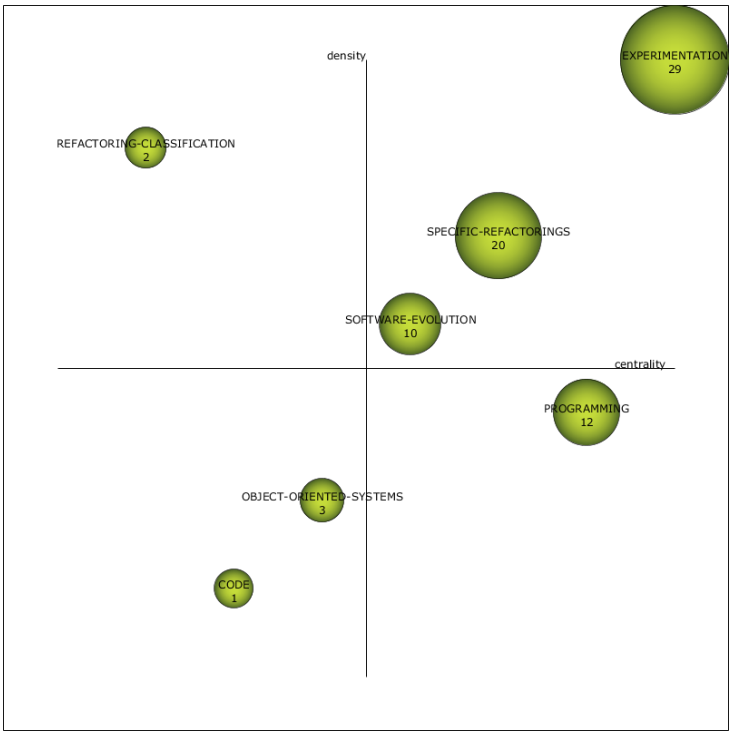

Figure 3.4: Strategic diagram for P3 (2010-2014).

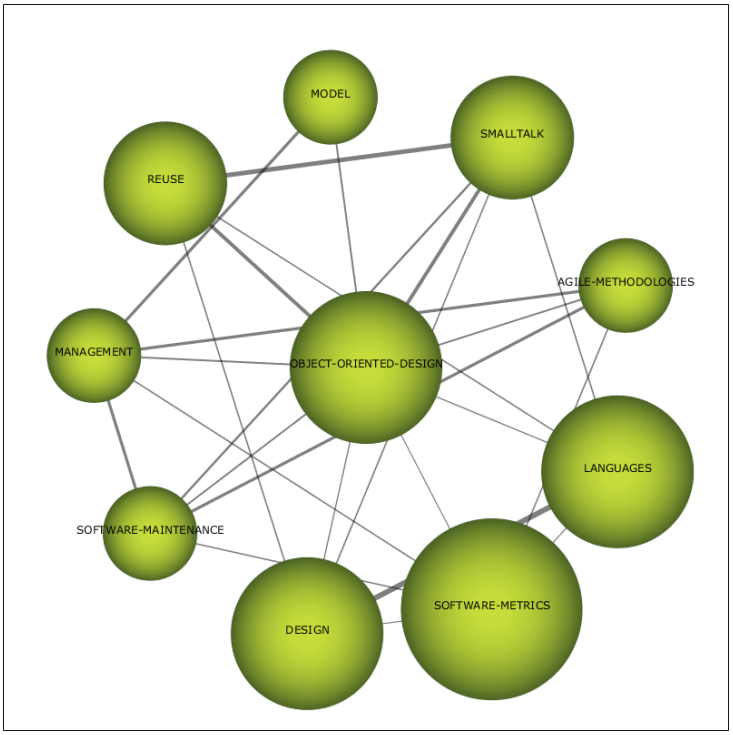

Figure 3.5: Cluster network *Object-Oriend Design* in P1

Table 3.2: Main concepts for motor themes

	Keywords	N. Documents
P1	Object-Oriented Design	8
P2	Languages	14
P3	Experimentation	29

last two periods, we find that *Object-Oriented Systems* and *Programming*, in chronological order, are basic themes.

With regards to the themes' size, namely to the number of papers involved in each area, for the P1 the maximum number is on the *Object-Oriented Design* (motor theme) area, *Languages* for P2 and *Experimentation* for P3. Each of them is a motor theme. The second area per size is *Programming*, *Object-Oriented Systems* and *Specific Refactorings* respectively for P1, P2 and P3. Just in P2 it is not the case of a motor theme: as reported above, *Object-Oriented Systems* in P2 is a basic theme.

If we focus on the inner structure of the larger clusters we found the following results. In Figure 3.5 the cluster network of *Object-Oriented Design* for P1 is shown. As it can be seen the central theme is related to theme that are recurrent in the Object-Oriented literature: *Smalltalk* is one of the first Object-Oriented language, *Reuse*, *Model*, *Software Metrics* are certainly relevant concepts in the Object-Oriented field, though not only in this field. *Design* is a most general concept, that regards also different programming paradigm. It is worth to note that in this cluster we found *Agile Methodologies* and *Languages*. This cluster also include the *Programming* concept that was the second motor theme of P1.

In Figure 3.6 the cluster network of *Languages* is shown whereas in Figure 3.7 is reported that relative to *Experimentation*. For both the clusters it is worth to note the presence of keywords that belong to different clusters the previous periods. For example *Object-Oriented Design* for the first one and *Experimentation* for the last one. The significant relative size of sub-clusters like *Analysis* and *Design*, when compared with the central concept, it also worth to be noted.

3.6 Discussion

In this Section we discuss the results reported in the previous section. As previously stated, there is a significant increment on the number of paper along time: the number of publications almost double every five years (95,6%) and is followed by an increase in the number of clusters. This number depends on the increase of density among keywords, when this increment is localized on subsets of keywords among which there are denser connection if compared to the rest of the network. It suggests that there is a trend for an higher number of papers to share fewer keywords or there is an increment of keywords in the network along with the number of papers. In this second case, the relationships among them should be arranged in such a way to form more clusters than in the past. In both cases this might indicate that researchers are deepening their studies by focusing on specific aspects of the research field.

For example, it is not surprising that in P1 the most central keywords are quite generic. Since we are at the research early times, there are plenty of topics to address for researchers,

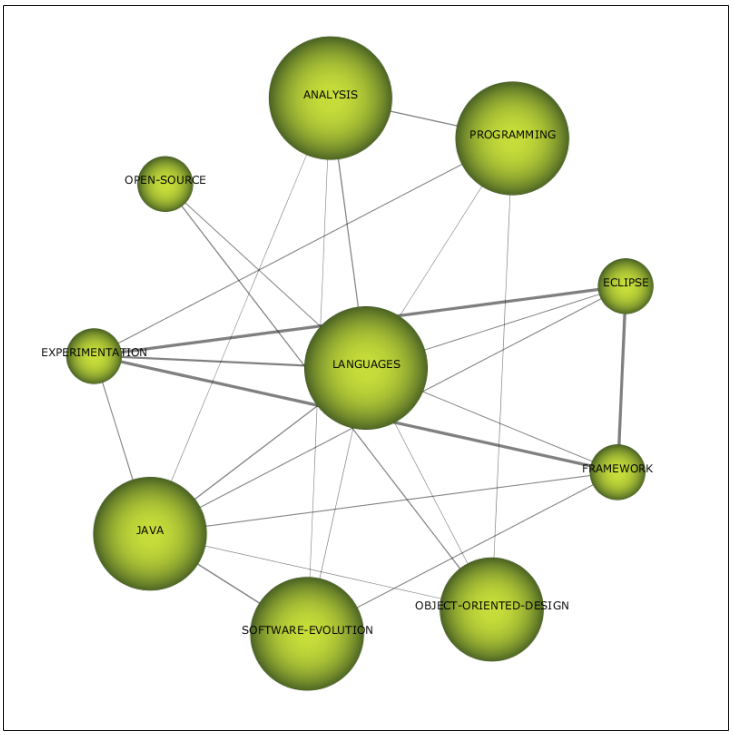

Figure 3.6: Cluster network *Languages* in P2.

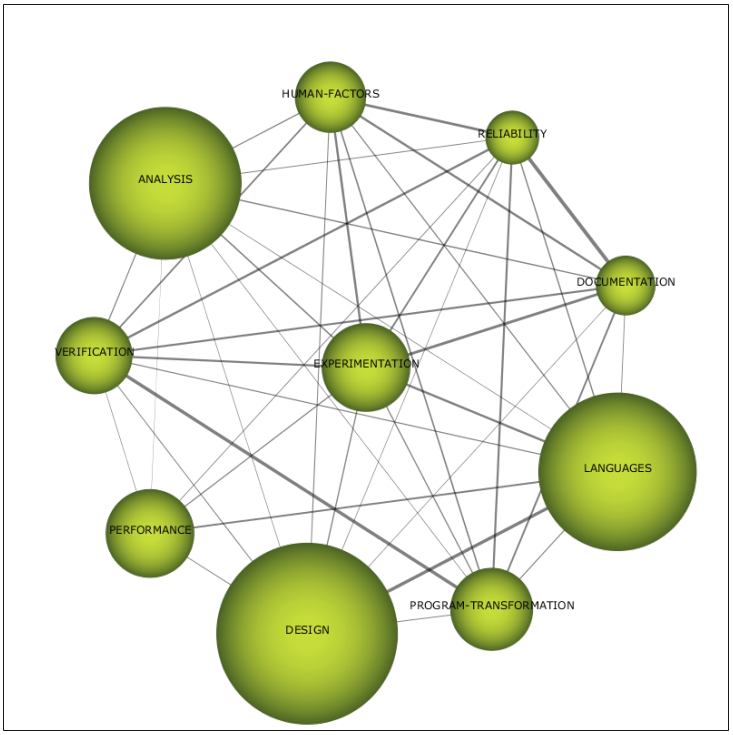

Figure 3.7: Cluster network *Experimentation* in P3

starting from their expertise. It is worth to note that Refactoring practices seem to be studied specifically in relationship with Object-Oriented technology. Instead, looking at the P3 strategic diagram, we found more specific research theme like *Specific Refactorings*: according to our cathegorization this group of words encompasses all the keywords which are related to specific refactorings (e.g. Extract Class, Move Method, etc.)

Let's focus our attention on the first cluster of Figure 3.5 relative to P1. Among the keywords, *Agile Methodologies* and *Languages* stand out as those which, over the next period, P2, both cover central roles but are located into two separate clusters. It seems that there is a tendency to split the motor theme of P1 in two motor themes in P2. Additionally, if we consider Figure 3.6, inside the cluster we can find *Object-Oriented Design*. According to the results of our analysis the last concept loose centrality if compared to *Languages*. Conversely, the same circumstance occurs with *Languages* and *Experimentation* between period P2 and P3.

Concerning the meaning conveyed by the different keywords, we make an attempt to draw some conclusion. In we consider the *Object-Oriented Design* cluster on P1, it seems to suggest that researchers are interestend in structural aspects (*Design*) and might suggest that there is some interest in code-level analysis (*Reuse*, *Languages*). On the other hand the main motor theme on P2 is *Languages* where the linked keywords are, besides *Object-Oriented Design*, *Analysis* and *Software Evolution*. There are also other words but they seem less significant because intuitively related to the main one (e.g. *Programming*). From the size of the corresponding cluster we can also appreciate the relevance of the Java programming language among the others. The way this cluster is structured seem to point to the fact that languages are put in relationship not only with the programming activity, but also with other stages (like the Analysis one). Moreover they seem to be a key element while studying software evolution. Main motor them for P3 is *Experimentation*: its cluster is formed by some of the keyword in the previously analyzed cluster: *Analysis*, *Languages* are the most important keyword along with the word *Design* - no reference at all to *Object-Oriented Design*. The most straightforward interpretation for this cluster is that the keyword represent an object of the experimentaion, either if it is *Performance* or *Software Evolution*. By and large, if we consider the strategic diagrams, we can hypotesize that along the time researchers not only found most specific research topics, as it is natural as the knowledge of a discipline evolve, but they address their efforts to study languages. Additionally we can conjecture, considering the relevance of *Experimentation* in the more recent periods, that there is an interest in making experiment that can provide evidence and in the same experimental process.

3.7 Threats to Validity

Our study suffers from some threats to validity that are reported in the following according to the distinction among internal, external and construction validity.

Internal validity - We consider that the evolution of a discipline could be detected by analyzing the keywords chosen by the research for their papers. Despite the fact that the co-word analysis is a known methodology [50] and has been already used for the same kind of study, we can not be sure that is sufficient to thoroughly represent the cognitive structure of the discipline. Perhaps it could be useful to address this issue also with other approaches that interest also the entire body of text.

External validity - For most of the computation we relied on SciMat [52]. Even if we consider this software reliable, we can not exclude that computation performed using different systems could lead to different results.

Construction validity - In order to retrieve our dataset we queried the bibliographic database *ISIWoS*. There exists other bibliographic databases that we did not yet addressed (Scopus, Google Scholar, etc). Although we retrieve a significant number of data from a reliable dataset we can not be sure that the analysis could not lead to different results with a different dataset retrieved from a different database. Results could also be affected by a relative bias depending on some of our experimental settings (e.g. the kind of algorithm chosen for clustering).

3.8 Conclusion

It is pivotal for researchers to be aware of the state of the art of their research field, understand its past and be able to foresee its future evolution. In this work we investigated the cognitive structure of the research on Refactoring and how it evolved along time. First, we retrieved bibliographic data from ISIWoS, one of the most important bibliographic databases. Then, after having performed a pre-processing stage in order to discard redundant or duplicated content, we built a co-occurrence network. Finally, we analyzed the obtained network by focusing on the reciprocal relationships between research themes. To this purpose, we applied a clustering analysis by means of a clustering algorithm. We analyzed 3 not-overlapping periods of 5 years each, spanning from 2001 to 2014. This analysis allow us to represent the most important thematic areas researchers were devoted to for each period.

Chapter 4

Blockchain-oriented Software Engineering: Challenges and New Directions

4.1 Introduction

In the past years, a lot of attention has been paid to the emerging concepts of Blockchain and Smart Contract. Organizations such as banking and financial institutions, and public and regulatory bodies, started to explicitly talk of the importance of these new technologies. Some observers are even talking of the dawn of a new era, declaring that *"we should think about the blockchain as another class of thing like the Internet [...]"* [55] and that the *"wide adoption of blockchain technology has the potential of reshaping the current financial services technical infrastructure."* [56].

From an economic and financial perspective, the overall capitalization of digital currencies is about 12 billions USD, as of October 2016, 80% of which is the capitalization of Bitcoins alone[1]. Venture capital investments in blockchain startups has been steadily increasing, from $93.8 million in 2013, to $315 million in 2014, to $490 million in 2015. Some people are even saying that Bitcoin and blockchain remind them of the Internet circa 1993, when huge amounts of venture capital started to flow into Internet startups, eventually leading to the emergence of companies such as Cisco, Yahoo, Google, Amazon and others.

Alongside these good news, there are also bad news, however. Ever since digital currencies started to represent a real monetary value, also hacks and attacks started. The "exchanges", Web sites allowing to store digital currencies and to trade them against other currencies, legal or digital, experimented big attacks. The biggest was the MtGox attack that occurred at the beginning of 2014, leading to a declared loss of $600 million. The latest was the Bitfinex attack of August 2016, with a loss of $65 million. Another remarkable exploit was that sustained by the DAO organization in June 2016, leading to a withdrawal of Ether digital currency funds worth $50-60 million. The Ethereum community defended themselves from this attack by performing a hard fork of the Ethereum software that forcibly recovered the stolen Ethers and gave them back to the original owners. Almost all of these attacks can be

[1] https://coinmarketcap.com/. Accessed: 2016-09-23.

attributed to poor software development practices.

The feeling many software engineers have about such increased interest in blockchain technologies and, in particular, on the numerous software projects rapidly born and quickly developed around the various blockchain implementations is that of unruled and hurried software development. The scenario is that of a sort of competition on a first-come-first-served base which does not assure neither software quality, nor that all the basic concepts of software engineering are taken into account.

This chapter aims at revealing the current issues and new directions for blockchain-oriented software engineering, and investigating the need for novel specialized software engineering practices for the blockchain sector. To this purpose, we:

1. identified the most relevant challenges for the state-of-practice blockchain-oriented software engineering;
2. highlighted peculiarities of some of the most popular blockchain-oriented software projects going on in the world;
3. proposed new research directions for blockchain-oriented software engineering, based on the results obtained from the previous steps.

The chapter is organized as follows. In 4.2 we report the most relevant related work; in 4.3 we present the current challenges for blockchain-oriented software engineering; in 4.4 we provide statistical data on open source blockchain-oriented GitHub repositories, whose selection has been guided by the 2016 Moody's report on the blockchain sector [57]; in 4.5 we propose new research directions on the basis of the dataset analysis; and finally, in 4.6 we report the conclusion.

4.2 Related Work

In [55], Swan discusses the applications of the Blockchain starting from the concept of cryptocurrency to that of the more advanced Smart Contract, in an attempt to demonstrate that the Blockchain can prove to be as disruptive a computing paradigm as the Internet and the mobile. When compared to the Blockchain, however, mobile development has raised less pressing issues from the software engineering perspective. In fact, the need for mobile-specific software engineering techniques has grown only recently, precisely as mobile applications started to be employed for business-critical purposes and, therefore, to require enhanced security and higher quality [58]. Conversely, business is the main reason behind the creation of blockchain-oriented software. The demand for security in blockchain applications is, therefore, even more pressing, and thus that for specialized software engineering processes.

4.3 Blockchain-oriented Software Engineering: Challenges

We define as Blockchain-oriented Software (BOS) all software working with an implementation of a Blockchain. A Blockchain is a data structure characterized by the following key elements:

- data redundancy (each node has a copy of the Blockchain);
- check of transaction requirements before validation;
- recording of transactions in sequentially ordered blocks, whose creation is ruled by a consensus algorithm;
- transactions based on public-key cryptography;
- possibly, a transaction scripting language.

Considering the distinctive marks of a Blockchain, software engineers could benefit from the application of BOS-specific software engineering practices. Such practices would constitute the base of a Blockchain-oriented Software Engineering (BOSE).

In this section, we identify the most relevant BOSE challenges, and the issues which originate from them. To this purpose, for most challenges, we also provide excerpts from the SWEBOK[2] [59] to properly frame the related issues.

New professional roles. *"A recognized profession entails specialized skill development and continuing professional education"*[2]. Due to the business-critical nature of the Blockchain, finance and legal subjects have shown increasing interest toward BOS. At the same time, bootcamps for Blockchain developers are flourishing. The Blockchain sector will need professional figures with a well-defined skills portfolio comprising finance, law, and technology expertise. An example of a new role could be that of an intermediary between business-focused contractors with low technology expertise and IT professionals.

Security and reliability. *"Software Security Guidelines span every phase of the software development lifecycle"* and *"Software Reliability Engineered Testing is a testing method encompassing the whole development process"*[2]. A Blockchain must guarantee data integrity and uniqueness to ensure Blockchain-based systems are trustworthy. Ensuring security and reliability in BOS development might require specific methodologies such as Cleanroom Software Engineering [60] or thorough software reviews. Furthermore, mathematically sound analysis techniques could help enforcing reliability and security-related properties in blockchain-oriented applications.

Testing techniques can also enhance system security and reliability. IBM recently expressed the need for continuous testing techniques to ensure blockchain software quality[3].

Testing techniques should be based on the nature of the application[2] which, in the case of BOS, is that of security-critical systems. In particular, there is a need for testing suites for BOS. These suites should include:

- Smart Contract Testing (SCT), namely specific tests for checking that smart contracts i) satisfy the contractors' specifications, ii) comply with the laws of the legal systems involved, and iii) do not include unfair contract terms.
- Blockchain Transaction Testing (BTT), such as tests against double spending and to ensure status integrity (e.g. UTXO[4]).

[2]SWEBOK 2004 version

[3]IBM. https://twitter.com/ibmsoftware/status/776605297037172736. Accessed: 2017-03-12. Published: 2016-09-15.

[4]Unspent Transaction Output

Software architecture. Specific design notations, macroarchitecture patterns, or meta-models may be defined for BOS development. To this purpose, software engineers should define criteria for selecting the most appropriate blockchain implementation, evaluating the adoption of sidechain technology, or the implementation of an ad-hoc blockchain. For example, Ethereum[5] has adopted a key-value store, which is a very simplistic database. By adopting a higher level data representation such as an Object Graph, it would be possible to speed up many operations which would otherwise be expensive using a key-value store [61].
Modeling languages. Blockchain-oriented systems may require specialized graphic models for representation. More specifically, existing models might also be adapted to BOS. UML diagrams might be modified or even created anew to account for the BOS specificities. For example, diagrams such as the Use Case Diagram, Activity Diagram, and State Diagram could not effectively represent the BOS environment.
Metrics. BOSE may benefit from the introduction of specific metrics. To this purpose, it could be useful to refer to the Goal/Question/Metric (GQM) method, that was originally intended for establishing measurement activities, but it can also be used to guide analysis and improvement of software processes [59].

Due to the distributed nature of the Blockchain, specific metrics are required to measure complexity, communication capability, resource consumption (e.g. the so-called gas in the Ethereum system), and overall performance of BOS systems.

4.4 Blockchain-oriented Software Repositories

In order to define new research directions for the BOSE on the basis of the state-of-practice of blockchain-oriented software, we conducted an exploratory study on a corpus comprising 1184 GitHub software repositories, which were identified with the use of the Moody's Blockchain Report [57] and CoinMarketCap[6]. We first identified the most relevant projects and players in the blockchain sector; then, we collected metadata on the corresponding BOS repositories. Information from the corresponding issue tracking systems were also considered.[7]. In the remainder of this section, we provide details about the methodology used to build the BOS corpus, and the preliminary results we obtained.

4.4.1 Building a Dataset of Blockchain-oriented Software

In this paper, we define a BOS project as a software project which contributes to the realization of a blockchain project. This definition includes both blockchain platforms, such as Bitcoin and Ethereum, and general blockchain software [62].

To identify BOS repositories we start from the corresponding blockchain projects. Moody's Investor Services recently identified more than 120 publicly announced blockchain projects in an in-depth report of the blockchain sector [57]. The projects list covers rated issuers across financial institutions, nonfinancial corporates and the official sector, and can be considered as a comprehensive list of blockchain projects going on in the world. The Moody's

[5] https://www.ethereum.org/. Accessed: 2017-03-12.
[6] https://coinmarketcap.com/. Accessed: 2016-09-23. We refer to the market capitalization of September 23, 2016.
[7] We focused on Github-hosted projects (see section 4.4.1), which come with the integrated GitHub issue tracking system

list is not a list of software projects; nevertheless, BOS projects stem from the blockchain projects on the list.

In addition to the Moody's list, we searched for the software associated to the currencies and assets with the highest capitalization, as reported by CoinMarketCap. Since we took the Moody's list as a baseline, we did not include currencies and assets with a lower capitalization than Stellar, the least capitalized cryptocurrency in the Moody's list for which we found a related software repository. Being Stellar on the 17th position at CoinMarketCap, we focused on the first 17 most capitalized currencies and assets.

Finding the software corresponding to a project in the Moody's list is not straightforward. When the project name is within a list entry (e.g., The Hyperledger Project), the software can be easily found by searching for the specified project name. When not specified, we searched for the involved blockchain startups, which often choose to publish their software on code-hosting platforms (e.g. GitHub, Bitbucket). As for the currencies and assets found on CoinMarketCap, this process was easier since each list entry is linked to the official website.

We focused on freely accessible, open source software hosted on GitHub, a platform hosting the vast majority of the detected blockchain-oriented software projects. We decided to only consider software hosted on GitHub repositories because GitHub provides homogeneous metadata, which, in turn, allow us to compare projects on the basis of standard features. For instance, it is possible to evaluate project popularity by relying on the amount of stars given to a project by GitHub users, or on the number of forks stemming from it.

4.4.2 Dataset Analysis

At the end of the selection process, we identified 52 GitHub accounts, which comprise 1184 repositories. We extracted information on popularity (*Stargazers*), programming languages, community involvement (*Contributors*, *Open Issues*, *Watchers*, *Forks*), and age (time elapsed since creation).

To focus on the most relevant repositories, we only considered those that i) are base repositories (not a fork from a previously existing repository), ii) had been updated in the previous 30 days, and iii) were created more than 30 days before (i.e. have been modified at least once since creation)[8]. By using these criteria, we retained 193 repositories out of the initial 1184.

4.4.3 Preliminary Results

Figure 4.1 reports the most used programming languages among the 193 retained repositories. JavaScript, Python, Go, C++, and Ruby are the top 5 languages, with BOS JavaScript repositories accounting for more than 30% of the total. It is interesting to note the presence of Python and Go in the podium, especially in comparison with the number of Java repositories–not reaching 4% of the total.

Table 4.1 shows that among the top 10 most popular repositories (i.e. those with the highest number of *Stargazers*) `ethereum/mist` and `coinbase/toshi` were created less than 1 year and roughly 2 years ago respectively. As expected, Bitcoin is the most popular project, neatly distinguished as for stargazers (9966) and contributors (396). Ethereum is also very popular, with three associated repositories in the top 10; in particular, the one written in Go

[8]All data were retrieved on September 23, 2016.

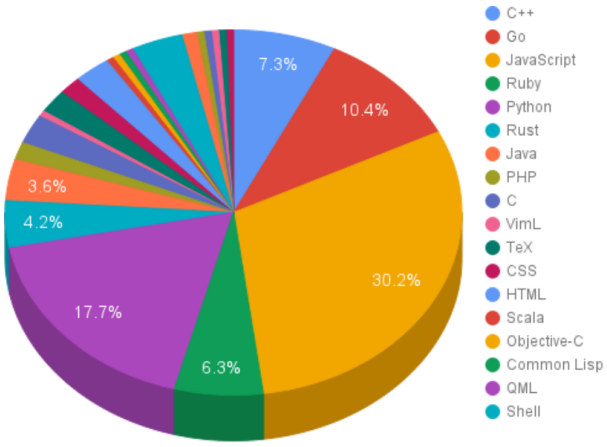

Figure 4.1: Languages across 193 BOS repositories

Table 4.1: Extracted statistics across the top 10 BOS Repositories

GitHub Repository	stargazers	contributors	open issues	age (days)	watchers	forks	first language
bitcoin/bitcoin	9966	396	547	2105	1211	4266	C++
ethereum/go-ethereum	2160	78	285	1002	367	695	Go
ledger/ledger	1813	108	14	3055	103	255	C++
digitalbazaar/forge	1584	41	137	2260	103	241	JavaScript
ripple/ripple-client	1244	51	21	1437	968	486	JavaScript
ethereum/mist	1168	35	198	471	210	299	JavaScript
dogecoin/dogecoin	1153	300	52	1022	149	505	C++
ripple/rippled	1144	53	118	1782	246	338	C++
coinbase/toshi	839	18	97	749	98	187	Ruby
ethereum/cpp-ethereum	723	89	212	1001	196	270	C++

is just behind the main Bitcoin repository. The top 10 BOS repositories were created around 4 years ago on average, and most of them have a considerable number of open issues. The statistics about forks are staggering, topping at 4266 for the main bitcoin repository, followed by the Go repository from Ethereum with 695 forks.

4.5 Blockchain-oriented Software Engineering: New Research Directions

Based on the results from section 4.4, we hereby suggest some new research directions for BOSE.

Testing. A recent study on over 50000 GitHub projects [63] has proved that a bigger team size leads to a higher number of test cases, whereas the number of test cases per developer decreases with an increase in the team size. It would be interesting to investigate whether the same can be said about BOS, considering that the most popular repositories have an un-

usually high number of contributors, even for open source projects. For instance, almost 400 GitHub users are contributing to the `bitcoin/bitcoin` repository, as reported in Table 4.1.

Collaboration. The high number of voluntary contributors testifies to the attractiveness of BOS in the open source landscape. A large base of voluntary contributing members has been shown to be a pivotal success factor in OSS evolution [2]. To achieve sustainable development and improve software quality, specific practices to enhance the synergy between the system and the community would be highly beneficial to BOSE [64].

Enhancement of testing and debugging for specific programming languages. Figure 4.1 shows that a number of programming languages such as Go, Python, and Ruby are gaining increasing popularity among BOS projects. This arises the need for enhanced testing and debugging suites, tailored upon the most popular BOS languages. Indeed, Java testing suites have undergone much more testing than Go.

In addition, as BOS projects work with the Blockchain, which is distributed by definition, testing in isolation would require properly mocking objects capable of effectively simulate the Blockchain.

Creation of software tools for smart contract languages. The implementation of Smart Contract Development Environments (SCDEs)–the blockchain-oriented declination of IDEs–might be pivotal for the building and diffusion of BOS expertise. Such environments could streamline smart contract creation through specialized languages (e.g. Solidity, a language designed for writing contracts in Ethereum).

4.6 Conclusion

Blockchain-oriented software engineering must respond to many challenges and define new directions to allow effective software development. In the present work, we highlighted the most evident issues of state-of-art Blockchain-oriented software development, by advocating the need for new professional roles, enhanced security and reliability, novel modeling languages, and specialized metrics. In addition, we used the 2016 Moody's Blockchain Report and the market capitalization of cryptocurrencies to build a dataset of blockchain-oriented software repositories. After retaining 193 repositories out of 1184, we extracted statistical information on popularity, collaboration, repository age, and programming languages. On the basis of the results of the analysis, we proposed new directions for blockchain-oriented software engineering, focusing on collaboration among large teams, testing activities, and specialized tools for the creation of smart contracts.

Chapter 5

A Preliminary Study on Mobile Apps Call Graphs through a Complex Network Approach

5.1 Introduction

Over the last years, mobile apps have become increasingly more popular. They represent a huge source of income for the software industry, with the global app economy worth $ 53Bn in 2012, and expected to rise to $ 143Bn in 2016 [65]. Most apps are made available to the public by such relevant companies as Google, Apple, and Microsoft through dedicated app stores. Considering the apps present in the Google Play only, their number increased from 650k+ in 2013 to 1400k+ in 2014 [66]. As a decrease is not expected in the near future, the recent past can be considered the best predictor for the mobile software development over the next years.

The increasing spread of mobile devices and, consequently, the increasing demand for mobile apps poses software engineering new challenges to confront with. As a matter of fact, as opposed to other kinds of applications (e.g., desktop applications) mobile apps must run smoothly on devices with limited computational capabilities and limited display size, have limited functionalities, and consistently rely on touch interfaces. Those limitations are actually the strength of mobile apps, since they allow small teams with limited experience to successfully develop popular apps. In this aspect, apps are similar to web-apps, but, again, have yet another advantage: they offer the best user experience on handheld, small devices, always located at an arm's length from the user.

Recently, several distinguished scholars devoted their efforts to investigate the multi-faceted aspects of mobile app development [67, 6, 68]. Starting from their own domain of expertise, some of them focused on security issues, while others on other specific software engineering areas. In particular, mobile apps have the peculiarity of being more intertwined with the hosting operating system than other applications, because Android apps are not stand-alone applications; in fact, they can be regarded as plugins into the Android framework [69]. The existence of an extensive practice of code reuse [68] lead to a significant inter-dependency between apps and the underlying operating system API. This could have a number of consequences, e.g., on software portability [70] and software quality [67, 6].

Specifically, in Syer et al. [6] the authors studied the interdependency between mobile apps and the Android API, finding an interesting relationship with software quality. In this preliminary work we propose a different approach, based on the concept of complex networks [71, 72, 73, 74]. This approach consists in representing a mobile app as a graph, whose nodes are associated to software modules (files, classes, methods, etc.) and graph edges represent the different relationships existent in software systems (dependency, inheritance, call, etc.).

The main advantage of this approach is that it enables researchers to investigate topological aspects of a software system, thus obtaining a deeper insight into the nature of the interconnections between software modules. By using *network metrics* it is possible to extract information that are different from those collected with traditional metrics (e.g. Chidamber-Kemerer [75]). Several authors already used this approach to investigate the nature of the interconnections of the elements in a software network [76, 77, 78, 79, 80] and found out that software networks actually share some properties with networks of different kind (being scale-free [81] or Small-world networks [82] , having a community structure [80]). We expect mobile apps will be increasingly complex in the near future, at least the most popular apps, because of the increasing computational capabilities of the handheld devices and the need for popular applications to offer more innovative functionalities than their competitors. This happened for desktop applications in the past. Thus, using software networks surely would be an innovative and promising approach for studying mobile applications. Today, the most popular apps satisfy new users' needs by adding different layers of complexity. Thus, there is a need for more effective tools to better understand how apps can be improved more effectively. In this scenario, complex networks undoubtedly have an advantage over other methodologies. Witnessing this period of everlasting transition, we decided to employ them to see which results we could obtain so far.

Other authors addressed their efforts towards understanding whether and to which extent network metrics could shed light on the hidden underlying software structure [83, 84, 74]. Recently, scientific interest has been increasing towards on a relatively less studied property, i.e. the community structure, and its strictly related property, the modularity function [85]. Community structure [86] (i.e., the way a network is structured in groups of tightly connected nodes), could be indicative of a more or less pronounced modular organization inherently present in a software system.

In this chapter we study mobile app networks and the related network metrics with the objective to investigate whether, and to which extent, they differ from the networks extracted from other software systems. Due to their peculiarity, we can not exclude *ex ante* that mobile app networks meaningfully differ from other software networks. In particular, we focus on a specific class of networks, known as call-graph networks. Call-graph networks represent the interactions among classes, which originate from a method in a class calling a method in another class. As a practical application of this approach we deal with the problem already studied in [70, 6] where the authors analysed apps with respect to dependencies with the hosting operating system API. Our work aims at contributing to the study of software networks through using a complex network approach. In fact, to the best of our knowledge, there are no other works which have relied on the analysis of complex networks to investigate mobile app development.

This chapter is structured as follows. In 5.2 we reported the relevant works in the Android apps and complex networks research fields. In 5.3 we describe the adopted methodology and the experimental settings. In 5.4 the results of our work are presented, followed by threats to validity in 5.5 and, finally, in 5.6, we draw some conclusions, and also outline the future

perspectives of the present work.

5.2 Related work

During recent years, the scientific interest towards operating systems and mobile applications has been steadily increasing. This interest is propelled by the seemingly ever growing app market. The market value growth cause also security concerns to grow; this is also reflected by the number of papers which have been published recently. Android system and apps are more studied than their Apple counterparts; the reason for this preference from the scientific community is to be found in the underlying principles on which the Android Open Source Project is based, that increase software's availability, and boost independent developers involvement. For these reasons, our interest is mainly focused on the study of Android API and third-party libraries dependencies through a complex networks approach. Our main goal is to provide useful information about how the tools available to Android developers are used by popular applications.

The frequent apps releases, combined with the extensive API use (apps can be actually considered as Android plugins [69]) makes of Android API a timely topic. Vasquez [67] focused on Android platform and third-party libraries "update" bugs (i.e., API-related issues). He proposed an approach whose aim is to help Android developers to effectively deal with breaking changes and bugs caused to their Android apps by API dependencies. The author addressed four types of API changes as the ones to be specifically addressed in the development of an Android App (i.e., breaking changes, buggy changes, smelly code, and new features). The contribution of this work is presented as the attempt to define templates for automatic code adaptation state-of-the-art, without specifically aiming at automated apps adaptation to the API changes. In addition, Vasquez, in the attempt to visualize examples of API usages, considered open source apps available on the free market F-Droid [1], as we also did for some of the applications we considered. Thus, the study of Android apps demonstrates to be a valuable opportunity for scientific research, thanks to open source code availability.

The work Vasquez et al. presented at ESEC 2013 [87] also focused on the use of API in Android applications. The authors analysed the relationship between the fault- and change-proneness of APIs. The main conclusion of their study is that there is no significant difference between both the average fault-proneness and change-proneness of APIs used by successful and unsuccessful apps. The work they proposed did not involve third-party libraries dependencies, whereas we also focus on external API dependencies.

In this paper we propose a different approach to the study of the mobile apps, based on the concept of complex networks. The research conducted so far in the field of complex networks [71] provides a valuable body of knowledge that can be applied to the study of software systems. The nature of software systems is complex and increasingly large [88, 79]; in particular, it is fair to expect mobile apps to grow in complexity in the next years, making complex networks optimal candidates to rigorously represent them. To effectively apply this approach to apps, we therefore focused our attention to sufficiently large applications, as those studied in the work of Syer [6]. Several scholar adopted this approach and found that software networks share several properties with networks of different nature [81, 82, 80, 83]. One of the most interesting properties of complex network, even in a software engineering

[1] https://f-droid.org/. Accessed: 2017-03-12.

Table 5.1: Metrics (mainly size related) for the analyzed software systems

	KeePassDroid	XBMCRemote	ConnectBot	FBReaderJ	SipDroid
Blank Lines	3,590	763	1,799	33,627	3,323
Classes	495	151	317	5,119	435
Code Lines	30,734	7,796	25,747	306,199	35,805
Comment Lines	6,477	1,758	4,891	17,653	3,317
Comment to Code Ratio	0.21	0.23	0.19	0.06	0.09
Declarative Statements	9,179	2,674	6,581	83,459	8,578
Executable Statements	11,731	2,857	11,463	119,879	17,590
Files	379	87	186	2,675	437
Functions	2,591	533	1,371	26,760	2,330
Lines	40,792	10,313	32,429	357,259	42,437

perspective, is the presence of a community structure. A community [71] is represented by a group of nodes tightly connected. The problem of determining the best community structure, along with that of computing the modularity [89, 90], has been faced by scholars of diverse background, starting from different perspectives (algorithmically [91], statistically, etc). Finding a community structure for a software network could be seen as the task of finding software modules which show the highest interaction, meaning that they likely belong to the same functional group.

5.3 Experimental Setup

Due to their nature mobile apps are usually of relatively small size. In Table I are reported several size related metrics where we can find that one of the software system (FBReaderJ) has a large number of classes (around five thousands). Depending on the software elements we associate to a node we can have **class call-graphs** (CCG) and **method call-graphs** (MCG). In a CCG nodes are associated to classes and edges represent the calls that methods of one class perform on methods of another and viceversa. On the other hand, MCG are those where nodes represent methods, and edges represent method calls. Not surprisingly, the first one is generally smaller than the second one. In order to retrieve both call-graphs we relied on Java Call-Graph[2]. It is possible to retrieve a call graph even with Understand[3] but, as far as we are concerned, not automatically for the entire project. Java Call-Graph perform an automatic parsing of the entire source code providing information also about external dependencies.

Our analysis was performed over the set of 5 mobile apps studied in Syer et al. [6]. They are:

KeePassDroid[4] is a port of the KeePass Password safe for the Android platform. KeePass is a free and open source (OSI certified) password manager, which helps users manage their passwords in a secure way. One master key or a key file is used to access a

[2] https://github.com/gousiosg/java-callgraph. Accessed: 2017-03-05.
[3] https://scitools.com/. Accessed: 2017-03-05.

database, that can be store locally and shared with multiple devices through a cloud hosting service, thanks to the existence of both a desktop and Android version. The databases are encrypted using the most popular and secure encryption algorithms currently known (AES and Twofish).

XBMC[5] allows to control XBMC Media Center with an Android device. It is officially brought to the public by Team XBMC.

ConnectBot[6] ConnectBot is an open source Secure Shell (SSH) client. It can manage simultaneous SSH sessions, create secure tunnels, and copy/paste between other applications. This client allows the user to connect to Secure Shell servers that typically run on UNIX-based servers.

FBReaderJ[7] FBReader is a free, customisable e-book reader for various platforms, including Android devices. FBReader includes a browser/downloader for network ebook catalogs/stores. It supports popular ebook formats (e.g. ePub, fb2, mobi, rtf, html, plain text).

SipDroid[8] adds SIP/VoIP support to Android contacts. Sipdroid allows the mobile user to choose where VoIP will be used, on WLANs only, on 3G, or EDGE networks.

The main difference between our dataset and the original is that we considered only the most recent code base. Because of this difference, some results may differ, but our aims are not the same as those of Syer; the choice has been made in order to allow for a future extension of our work, with a focus on the evolution of an extended dataset. This will be interesting for providing evidence of the growing complexity of popular apps in the mobile stores, and then provide sound support to the use of complex networks for studying mobile apps.

Data about maintenance activities carried out during software development are stored in systems called Issues Tracking Systems (ITS). For instance, popular ITS are BugZilla and Jira. On the other hand information about the intervention on the code are reported in Source Control Managers (SCM) like Git and SVN. By and large, there is not a straight correspondence between ITS and SCM like Git, so it is not straightforward to associate, for example, a bug fixing intervention to a specific commit on a SCM log. Since presently there is not a simple way to integrate information belonging to these two different systems, we follow the same approach as in [6].

We consider as a marker of a corrective maintenance intervention the presence in the corresponding commit of words like *Bug(s), Defect(s), Fix(es/ed), Patch(e/s)* and their variations (for example, word capitalized or not) [6]. After having retrieved the log file from the corresponding repository, we parsed this log, looking for commits that contains an appropriate regular expression. Finally, we found the commits associated to the files (i.e., commits in which these words appeared and a specific file was modified or created) and we counted the number of those commits for each file. Data are reported in Table II where it is shown that we did find occurrences for the first two words (Bug(s) and Fix(ed/s)) and no occurrences for the last two (Defect(s) and Patch(es/s)). In addition, it is worth to note that, for each project, we are considering a log file that encompasses the entire history of the software. Consequently, it is possible that some comments are referring to files that are not present any more, because removed or renamed. We then obtained a rough estimate of the degree of defectiveness of a file.

Table 5.2: Number of commit occurrences for the selected words

	Bug(s)	Fix(ed/es)	Defect(s)	Patch(es/s)	Defect. Files
ConnectBot	114	455	0	0	0.444
FBReaderJ	179	3087	0	0	0.309
KeePassDroid	3	290	0	0	0.034
XBMCRemote	698	3396	0	0	0.641
SipDroid	139	842	0	0	0.166

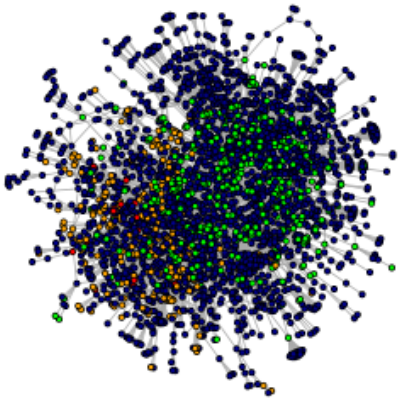

Figure 5.1: Method call-graph for KeePassDroid software

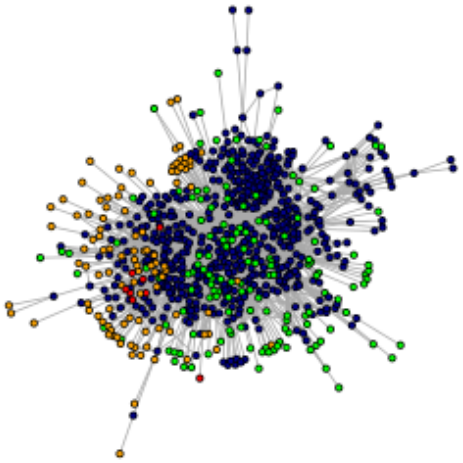

Figure 5.2: Class call-graph for KeePassDroid software

Our analysis is based on the approach proposed in [6]. Using the same approach could be useful for studying differences between our results and those of the original paper.

5.4 Results

To analyze the software networks, we extracted diverse data, some of them, related to method call graph networks, reported in Table 5.3. As we can see, analysed software networks present a significant community structure, since modularity is around 0.5 [71]. Surprisingly enough, clustering coefficient is very low. This suggests that the call graph does not belong to the small-world networks. We reported a method call-graph for the KeePassDroid application in Figure 5.1. It is worth to mention that it includes also the external dependencies (the coloured ones).

We also focused on the CCG, a graphical example of which can be seen in Figure 5.2. With respect to this, our aim is to find out whether classes which are more connected to those belonging to external dependencies are also more defect-prone.

To do so we associated a value to each class. This value was obtained from the analysis performed over the commits in the git repository of a mobile app. As illustrated in Section 5.3, we retrieved the number of occurrences of a regular expression, specifically conceived

Table 5.3: Several network metrics for the method call graph (MCG) of the analyzed software systems

	no. nodes	no. edges	mean degree	clustering coeff	modularity	no. communities
ConnectBot	618	2533	8.197	0.074	0.473	12
FBReaderJ	6287	36530	11.621	0.009	0.529	23
KeePassDroid	784	3642	9.291	0.084	0.533	8
XBMCRemote	306	1283	8.386	0.100	0.462	7
SipDroid	743	3247	8.740	0.093	0.475	10

Table 5.4: Several network metrics for the class call graph (CCG) of the analyzed software systems

	Defect. Classes/Tot. Classes	Mean Degree	Mean Degree Defect.	Mean Degree NOT Defect.
ConnectBot	0.432	8.197	9.400	7.282
FBReaderJ	0.040	11.620	7.474	11.794
KeePassDroid	0.149	9.290	9.111	9.322
XBMCRemote	0.078	8.385	8.708	8.358
SipDroid	0.088	8.7402	10.727	8.546

for detecting bug-related commits, among all the commits where the analysed class was modified. This is possible because commits are associated to files, each of them containing one or more classes, thus allowing us to associate each class to the appropriate files, and each file to the commits where it was modified. In case of multiple classes we used a majority law and we associated bugs to the class with the highest LOC (lines of code) value.

We found the file-class correspondences using Understand[9], a widely used software analysis tool. Understand allowed us to retrieve several software metrics, such as those reported in Table 5.1. After associating all the defect-related information with the corresponding classes, we performed other analyses to know whether classes that seemed to be more connected with external libraries were also addressed by bug fixing operations. First, we extracted, for each network node, the network constituted by its first neighbours (i.e., all the nodes that are directly connected to the first one). Second, we computed the ratio between the number of neighbours that belong to external libraries (external neighbours) and the total number of neighbours. We performed these computations for both the defective and not defective classes. Table 5.4 summarizes the most relevant data we obtained. As we can see, defective classes are not a significant part of the network. We regard this as dependent on a consistent part of the information retrieved from the repositories which refers to files not available when the analysis was performed, as they were either removed or renamed.

Concerning the two systems that show a higher ratio between defective and not defective classes (KeePassDroid and ConnectBot), we performed a basic statistical analysis which is illustrated by the box-plots in Figure 5.4 and 5.4. While for KeePassDroid the number of

[9]See footnote 3.

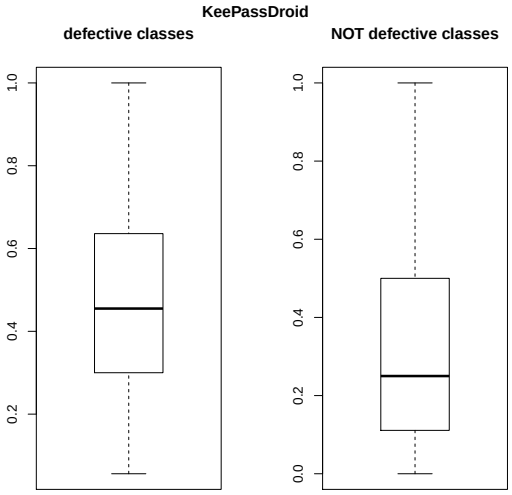

Figure 5.3: Boxplot representing a comparison for the distribution of external dependencies for KeePassDroid

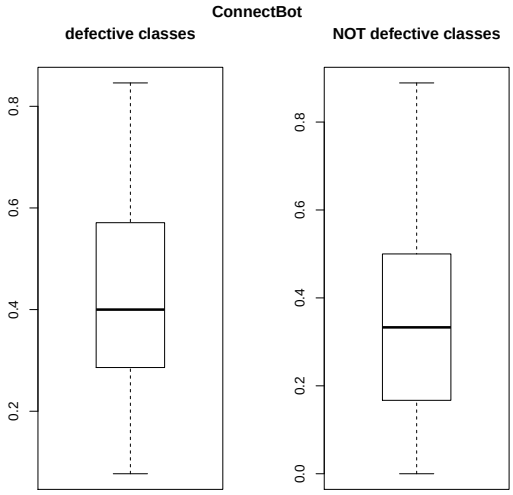

Figure 5.4: Boxplot representing a comparison for the distribution of external dependencies for ConnectBot

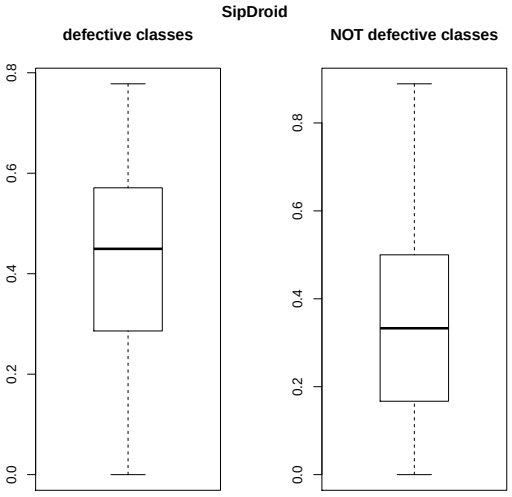

Figure 5.5: Boxplot representing a comparison for the distribution of external dependencies for SipDroid

connections between defective classes and external dependencies is fairly high on average, we can not come to the same conclusion for ConnectBot. On the other hand, if we consider SipDroid (Figure 5.5), there is not a high percentage of defective classes, as the associated boxplot shows that external classes tend to be first neighbours of defective classes.

5.5 Threats to Validity

This work suffers from some threats to validity, reported in this section. They are mainly construction threats to validity. In order to retrieve reciprocal relationships between classes we relied on Scitools software Understand, a well-known application for software analysis. Although we checked whether the relationships were correctly extracted on a representative sample, we can not be sure that no errors occurred. Moreover, we used two different systems to compute system networks (Understand and Java-Call graph). Some differences in data returned by these two software might exist. As previously stated, using complex networks at this time of the evolution of the mobile app market means also that we must satisfy our need for large code bases with the available popular software, that is nowadays just enough to allow us to effectively employ this tools. Nevertheless, as our research leverages an approach never used in this field, we are confident that the increasing complexity of the mobile software (at least, the most popular software) in the nearest future will provide us with larger code bases to analyse with complex networks.

5.6 Conclusion

Mobile apps are certainly an important part of the software industry and for this reason it is crucial to better understand their structure and evolution, with the ultimate purpose of providing useful information to practitioners and software engineers. The increasing mobile app market value could be a trigger for an increasing complexity of the most used apps in the stores. This could eventually lead to the need for more effective tools for studying ever growing mobile software complexity. This need could be answered by the use of complex networks, that proved to be valuable tools for the analysis of software systems. Although several scholars have already devoted their efforts to the study of mobile apps, this is the first time that complex networks are used in this field. We computed the software networks associated to 5 mobile apps and studied their metrics, thus showing that they share the same features as software networks of other kinds. Doing so, we applied this approach to a practical problem, already faced by other scholars [6]: the investigation of the existing relationships between mobile apps external dependencies and their quality. We show that, to a certain extent, our results agree with those reported in those previous works. Future work will involve the extension of the dataset, in order to assess the validity of this approach. In addition, we are considering to better distinguish the different kinds of dependencies, both from a programming language perspective, and from a functional perspective (which kind of external dependency is involved: with user interface, with network device, etc). We are confident that the increasing complexity of mobile apps will help us in obtaining even more meaningful results in the near future through the use of the complex networks approach.

Chapter 6

A Statistical Comparison of Java and Python Software Metric Properties

6.1 Introduction

With the advent of the object-oriented approach, specialized metrics have been introduced for the evaluation of software systems. The first attempt in this direction is the introduction of Chidamber and Kemerer's metrics, which have become the most popular object-oriented metrics suite.

The definition of new metrics still constitutes a vibrant research area, but theoretical reasons alone may not justify the adoption of specialized metrics. More specifically, software engineers need empirical evidence to prove that such metrics are actually related to software quality. In general, the process of ensuring high software quality is associated with practices leading to software products which are accurate, effective, delivered on time and within budget.

To the best of our knowledge, little has been written about the distribution of traditional metrics for Python software, mainly because this programming language has been only recently growing in popularity [92, 93]. Conversely, several studies concern Java systems.

In this chapter, we present the results obtained from the analysis of the distributions of traditional software metrics for 20 popular open source software systems. Half of the 20 software systems are written in Java, the other half are written in Python. The main purpose of the study is to investigate possible differences in the use of the two popular languages.

The remainder of this chapter is structured as follows. In the next section, we provide background and related work. In 6.3 we present the methodology used for this study. In 6.4 we discuss the results. In 6.5 we present some threats to validity. Finally, in 6.6, we summarise the findings.

6.2 Background and related work

Several studies have analysed software metric distributions with regard to software quality and to define a methodology for guiding software process development. Many empirical studies have been performed to validate empirically different software metrics for those purposes. The Chidamber and Kemerer (CK) [94] suite tackles these two aspects, showing an acceptable correlation between CK metrics values and software fault-proneness and difficulty of maintenance [95, 96, 97, 98, 99, 100]. Other metrics like micro-patterns have also been proposed to help managers and developers in monitoring software quality [101, 102].

Louridas et al. [103] found that distributions with long, fat tails in software are much more pervasive than previously established, appearing at various levels of abstraction, in diverse systems and languages.

Alshayeb et al. [104] found that Object Oriented metrics are effective in predicting design efforts and source lines of code added, changed, and deleted in the short-cycled agile process and ineffective in predicting the same aspects in the long-cycled framework process.

Potanin et al. [105] examined the graphs formed by object-oriented programs written in a variety of languages, and found that these turn out to be scale-free networks.

Shatnawy et al. [106] validated the effect of power laws on the interpretation of software metrics. The authors have studied five open source systems to investigate the distribution and their effect on fault prediction models. The results show that power law behaviour has an effect on the interpretation and usage of software metrics and in particular the CK metrics. Many metrics have shown a power law behaviour. Threshold values are derived from the properties of the power law distribution when applied to open source systems. The properties of a power law distribution can be effective in improving the fault-proneness models by setting reasonable threshold values.

Concas et al. [107] presented an extensive analysis of software metrics for 111 Object Oriented systems written in Java. For each system, authors considered 18 traditional metrics such as LOC and CK metrics, as well as metrics derived from complex network theory and social network analysis; they also considered two metrics at system level, namely the total number of classes and interfaces and the fractal dimension. They discuss the distribution of these metrics and their correlation both at class and system level. They found that most metrics followed a leptokurtotic distribution. Only a couple of metrics have patent normal behaviour while three others are very irregular and even bimodal.

Different software metrics may follow different statistical distributions. In particular, there are metrics whose values may differ largely from those of other metrics, like the DIT (depth of inheritance tree), which typically assumes values lower than ten. In our study, we did not consider such metrics because the related statistics can hardly be useful to distinguish one language from the other. We focused on metrics whose distributions have wider ranges, since this choice increases chances of finding differences between the two languages. Typically these software metrics follow an approximate power law in the tail of the distribution and can be fitted by many statistical distributions.

Among all the candidate distribution functions, we chose only two of them. We searched for distributions capable of providing not only the best fit for all the projects and the analysed metrics, but which could also yield significant results suitable for revealing differences among the metrics of choice. The statistical distributions we employed are the log-normal and the double Pareto distribution.

The double Pareto distribution is an extension of the standard power-law, in which two different power-law regimes exist, provided by the two parameters α and β in eq. 6.1. There are different forms for the double Pareto distribution in literature [108, 109, 110, 111]. We use the form described in [111] for the Complementary Cumulative Distribution Function (CCDF), because it provides a smoother transition across the two different power-law regimes:

$$P(x) = 1 - \left[\frac{1 + (m/t)^{-\alpha}}{1 + (x/t)^{-\alpha}} \right]^{\beta/\alpha} \tag{6.1}$$

The double Pareto distribution is able to fit a power-law tail in the distribution, with the advantage of being more flexible than a single power-law in fitting also the head of the distribution, where it is very similar to a log-normal.

6.3 Methodology

In the first step we have selected active projects from the Java Qualitas Corpus [112], subsequently trying to identify similar Python projects, considering system dimension in terms of number of classes, number of recent commits and application domain [113]. We decided to select open source projects and, to ensure uniformity among the systems, we have considered only projects with similar features.

The number of classes for the projects considered in our corpus ranges from the minimum value of 3357 (FreeCol, Python) to the maximum value of 31604 (Vuze, Java). Considering the selection criteria, during the systems selection process we have considered two features: popularity and high activity. The former concept is related to the total number of users of the specific software, the latter to the number of commits. We have considered the statistics provided by Open HUB[1]. On this portal it is possible to compare a huge number of open source projects in terms of popularity and in terms number of commits related to a specific project. We have considered systems belonging to the same area; to do that we have analysed the metadata provided by the Qualitas Corpus, openhub.net, and other detailed information available on the official websites of each project present in our corpus.

The systems we investigated are reported in Table 6.1.

Table 6.1: Python and Java Systems

Python	Java
Biopython	Apache Ant
Calibre	Apache Commons Collections
Matplotlib	Apache JMeter
Open Edx - platform	Castor
OpenERP	Cayenne
Py-Pandas	Freecol
Sage	Jena
SCons	JFreeChart
Tribler	Vuze
Unknown Horizons	Weka

[1] https://www.openhub.net/. Accessed: 2017-03-12.

We used Understand[2] software to calculate a set of nine metrics (Number of base classes, Number Of Declared Instance Methods, Number of Declared Instance Variables, Number Of Local Methods, Total number of Methods, Number of Lines of code, Number of Lines of comments, Number of statements, Depth of inheritance tree) from each of the twenty systems, computed the empirical cumulative distribution function, and employed the Log-normal and the double Pareto distributions to identify differences between the metrics. The statistical analysis on the results were performed using R[3] and Matlab[4].

After computing the previously listed metrics, we only focused on three of them, namely Number Of Local Methods (NOLM), Number of Methods (NOM) and Number Of Statements (NOS), as they proved to yield the most meaningful results. We have proceeded the following way. We have performed best fitting procedure on the metrics empirical distributions by employing the two chosen statistical distributions functions, the log-normal and the double Pareto distributions, for both the ten Java projects and for the ten Python projects. We have systematically verified the goodness of fit for all of them, provided by R^2, the goodness of fit coefficient, which ranges from 0 to 1. When R^2 is close to 1, the fit is very good. When R^2 is larger than 0.9, the fit is still considered fairly statistically good, while smaller values provide a worse fit. We have obtained the values of the best fitting parameters across the twenty analysed projects, which statistically represent the metrics values for each system. These parameters are μ and σ for the log-normal distribution, and the α and β parameters in the case of the double Pareto distribution. We then statistically compared the two sets of measures, those related to the Java projects and those related to the Python projects. We did not only measured mean values and standard deviations for all the parameters using the ten measures for the two sets of projects, but also performed statistical tests for measuring whether the two sets belong to two different distributions. These are the rank sum test, and the Kruskal-Wallis test, both non parametric, in order to avoid any arbitrary assumption on the distribution function of the parameters across the different projects.

6.4 Results

Most of the empirical distributions resulting from the analysis, given a specific metric, are at a first glance, quite similar to each other. Nevertheless, some of them show that differences between Java and Python projects do exist. In particular, four metrics appeared to be suitable candidates:

- the number of methods that can be called on an instance of the class in which they are declared (Number Of Declared Instance Methods, NODIM);
- the number of methods that can be called on a class and on an instance of the class in which they are declared (Number Of Local Methods, NOLM);
- the total number of methods (NOM), inherited methods included;
- the number of statements (NOS);

[2] https://scitools.com. Accessed: 2017-03-19.
[3] https://www.r-project.org. Accessed: 2017-03-19.
[4] www.mathworks.com. Accessed: 2017-03-19.

The first two metrics are capable of distinguishing between instance methods and static methods, since NODIM accounts only for the methods that can be called on an instance of the class, whereas NOLM accounts both for those methods and those that can be called directly in the class. Python projects present the same values for NODIM and NOLM in all classes, since instance methods, static methods and class methods can all be called on an instance of a Python class. Thus, NODIM and NOLM analysis yield the same results, as far as Python projects are concerned. We then considered only the last three metrics, i.e. NOLM, NOM and NOS.

We first analysed the results obtained with NOLM metric. The NOLM Empirical Cumulative Distribution Function (ECDF) shows that the Log-normal distribution fits for Python and Java systems, as the coefficient of determination is close to one for all the projects, with an average of 0.983 and of 0.985 for the ten Java projects and the ten Python projects respectively. The fitting procedure provides different averages for the location parameter, thus showing a difference between projects written in one language with respect to the other. In particular, if we look at the mean value of the location parameter for each set of projects, Python projects show a lower value. Table 6.2 shows the comparison between average values for the Log-normal distribution parameters we extracted.

Table 6.2: NOLM ECDF Averaged Log-normal fit parameters with standard deviation

	μ	σ	R^2
Java	1.369 ± 0.297	1.048 ± 0.110	0.983 ± 0.015
Python	1.271 ± 0.231	0.990 ± 0.093	0.985 ± 0.006

Even if the average values for parameter μ are apparently dissimilar, their large standard deviation suggests a possible overlap among the values of μ for projects belonging to the two languages. On the contrary, the parameter σ displays two fairly close average values, but their standard deviations are quite small and the overlap among the two sets appears reduced.

We performed the rank sum test and the Kruskal-Wallis statistical test on the twenty previously computed values of μ and of σ, ten for each programming language, in order to establish if the two sets show statistical differences between their metrics.

For the log-normal distribution both tests provide a negative answer. Namely, the parameters μ and σ, obtained from the best fitting with a log-normal distribution for the metric NOLM do not allow to distinguish between Java and Python projects, as Table 6.3 shows. This result confirms that the position parameters present a good overlap between the two languages, and that the dispersion parameter sigma, even if it presents a smaller overlap, does not allow to statistically distinguishing between Java and Python using the metric NOLM.

Table 6.3: Rank sum and Kruskal-Wallis tests results for NOLM Log-normal fits

	μ	σ
Rank sum (p-value)	0.3075	0.0890
Kruskal-Wallis (p-value)	0.2899	0.0821

The situation changes in the case of the double Pareto distribution (see Figures 6.1 and 6.2).

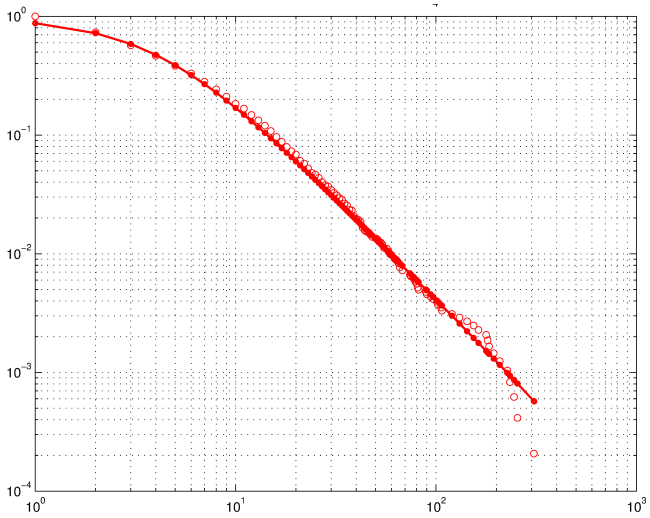

Figure 6.1: NOLM ECDF with double Pareto fit for Jena 2.11.1 (Java)

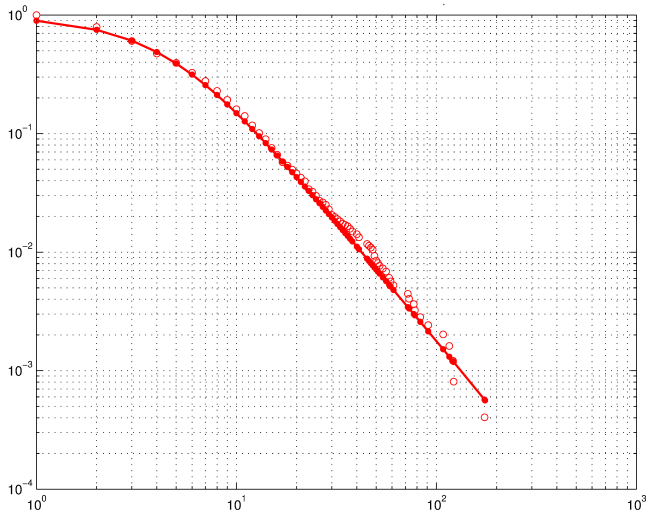

Figure 6.2: NOLM ECDF with double Pareto fit for Calibre 1.18.0 (Python)

Table 6.4 reports the average values and the standard deviations for α and β best fitting parameters obtained for the metric NOLM. Also in this case the coefficient of determination R^2 is very close to one, meaning that the fitting with a double Pareto distribution is good. In this case the advantage is that the parameter β averages obtained from Java projects are different enough from Python ones, as reported in Table 6.4.

This consideration is confirmed by the rank sum and Kruskal-Wallis tests (see Table 6.5), which reveal that differences in the parameter β are statistically meaningful for distinguishing Java projects from Python ones, even if the parameter α does not show differences between the two languages.

Parameter α provides the power-law exponent in the first power-law regime, whereas parameter β provides the power-law exponent for the second regime. Therefore, the obtained results show that, even though the statistical distributions may appear the same in the bulk

Table 6.4: NOLM ECDF Averaged double Pareto fit parameters with standard deviation

	α	β	R^2
Java	1.8993 ± 0.3031	1.2583 ± 0.1797	0.9868 ± 0.0141
Python	1.8917 ± 0.2151	1.4383 ± 0.0845	0.9894 ± 0.0052

Table 6.5: Rank sum and Kruskal-Wallis tests results for NOLM double Pareto fits

	α	β
Rank sum (p-value)	0.7337	0.0257
Kruskal-Wallis (p-value)	0.7055	0.0233

of the analysed data, that is, for classes with NOLM metric below the first regime threshold and ranging from 5 to 6 both for Java and for Python, major differences are evident along the queue of the distributions, namely for higher values of the NOLM metric. Thus, the use of local methods in Java and Python classes is different for classes having a large number of local methods. A difference between metric distributions for Python and Java projects has been found, as far as NOLM metric is concerned.

Table 6.6 reports the same results for the best fitting parameters of the log-normal distribution with respect to the metric NOM.

Table 6.6: NOM ECDF Averaged Log-normal fit parameters with standard deviation

	μ	σ	R^2
Java	2.122 ± 0.430	1.451 ± 0.180	0.948 ± 0.044
Python	2.42 ± 0.387	1.322 ± 0.125	0.945 ± 0.061

The log-normal distribution provides a worse fitting for the NOM metric ECDF than the previous case, being the coefficient of determination below 0.95 on average for projects in both languages. Nevertheless, the analysis suggests that the related dispersion parameter can be used for distinguishing Java projects from Python projects, since the relatively small standard deviations for σ for both Java and Python projects do not show a consistent overlap between the two averages. In Figure 6.3 and 6.4 we plot the NOM ECDF and related log-normal fit for Apache Commons Collection and Pandas, a Java and a Python project respectively.

We have verified this result performing rank sum and Kruskal-Wallis tests for the distributions of μ and σ relative to the ten Java projects and the ten Python projects, as we previously did with the NOLM metric. Test results are displayed in Table 6.7 and show that the dispersion parameter σ can be used for distinguishing between Java and Python projects.

We have applied the best fitting procedure for the NOM ECDF also using the double Pareto statistical distribution, obtaining results similar to those obtained through the log-normal, as far as the coefficient of determination is concerned; nevertheless, they are useless with regard to the capability of distinguishing between Java and Python projects. Table 6.8

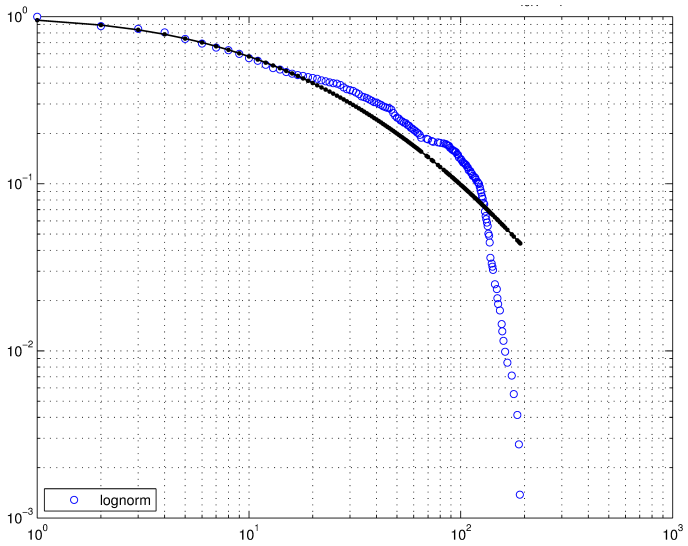

Figure 6.3: NOM ECDF with log-normal fit for Apache Commons Collections 4-0-RC5 (Java)

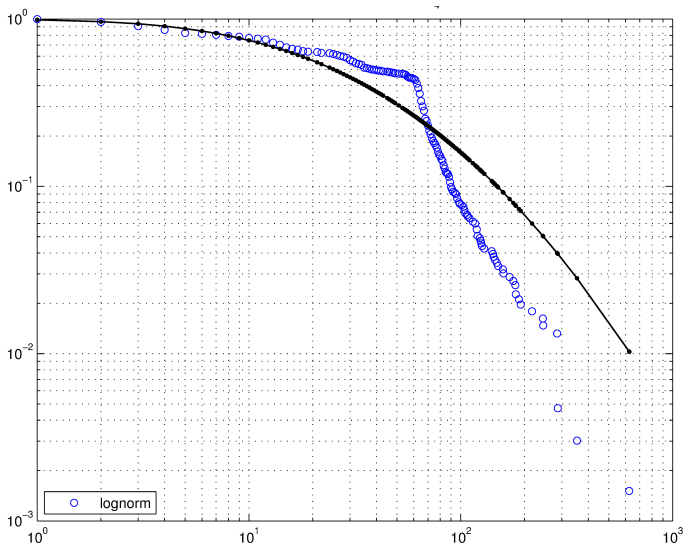

Figure 6.4: NOM ECDF with log-normal fit for Pandas 0.13.1 (Python)

reports the results for the best fitting parameters averages and the coefficient of determination averages for the double Pareto case for the NOM metric, as well as the relative standard deviations.

In this case it is clear that the high variability of parameter β for both languages, related to the shape of the NOM ECDF, renders it useless for distinguishing between the two languages, even if the two averages are quite different. Rank sum and Kruskal-Wallis tests confirm this result, according to Table 6.9.

On the other hand, examining the possible outliers, the project "JFreeChart" in Java results in a β value of 15.8542, which is clearly out of scale, considering that the maximum of the remaining values is around 1.7. Repeating the analysis after removing such outlier reduces the β variance in Java, producing more interesting results. In fact, in such case the rank

Table 6.7: Rank sum and Kruskal-Wallis tests results for NOM Log-normal fits

	μ	σ
Rank sum (p-value)	0.1405	0.0452
Kruskal-Wallis (p-value)	0.1306	0.0413

Table 6.8: NOM ECDF Averaged double Pareto fit parameters with standard deviation

	α	β	R^2
Java	1.0728 ± 0.2107	2.6849 ± 4.6332	0.9525 ± 0.0439
Python	1.1227 ± 0.1844	1.8015 ± 0.7450	0.9441 ± 0.0630

Table 6.9: Rank sum and Kruskal-Wallis tests results for NOM double Pareto fits

	α	β
Rank sum (p-value)	0.3847	0.0640
Kruskal-Wallis (p-value)	0.3643	0.0588

sum test provides a p-value of 0.0133 for the comparison between the two sets of β values, and the Kruskal-Wallis test applied using nine Java and Python projects provides a p-value of 0.0243, indicating that parameter β holds the capability of distinguishing between the two languages.

Finally, NOS ECDF show more significant differences between Java and Python languages than NOM ECDF (see Figure 6.5).

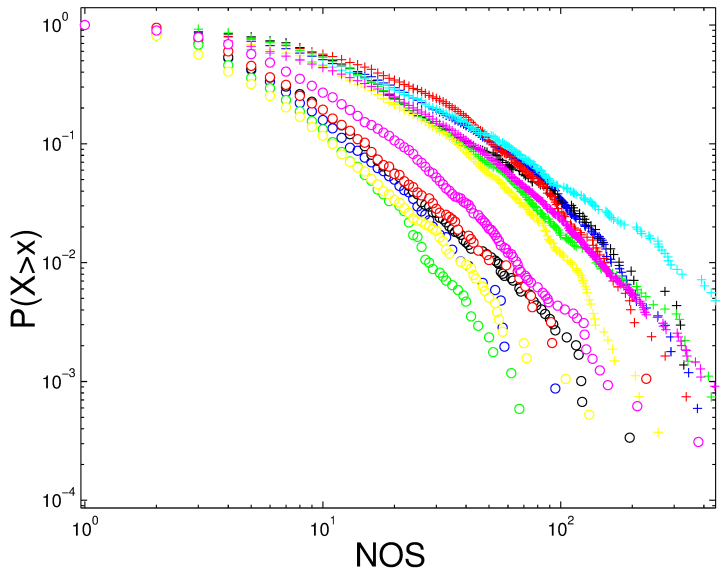

Figure 6.5: NOS ECDF for eight Python projects (O) and eight Java projects (+)

The average value for the coefficient of determination is very high for both Python and

Java, and for log-normal as well as double Pareto best fits. The log-normal location parameters are dissimilar: the average value for Java systems is around 2.23, whereas for Python is around 1.44. Mean and standard deviation for Java and Python projects are presented in Table 6.10.

Table 6.10: NOS ECDF Averaged Log-normal fit parameters with standard deviation

	μ	σ	R^2
Java	2.225 ± 0.360	1.278 ± 0.186	0.991 ± 0.007
Python	1.444 ± 0.215	0.940 ± 0.114	0.989 ± 0.004

The rank sum and Kruskal-Wallis tests provide meaningful results for both μ and σ, according to Table 6.11, which demonstrates Java projects are distinguishable from Python projects through the Log-normal distribution.

Table 6.11: Rank sum and Kruskal-Wallis tests results for NOS Log-normal fits

	μ	σ
Rank sum (p-value)	< 0.00044	< 0.00044
Kruskal-Wallis (p-value)	< 0.0004	< 0.0004

In Figure 6.5 the ECDF for eight Python projects and eight Java projects are illustrated.

Conversely, the results obtained with the double Pareto distribution show that only parameter α allows to distinguish between the two languages, while the parameter β is useless (see Table 6.12).

Table 6.12: NOS ECDF Averaged double Pareto fit parameters with standard deviation

	α	β	R^2
Java	1.2857 ± 0.1634	1.6221 ± 0.6256	0.9898 ± 0.0063
Python	1.9978 ± 0.3180	1.7224 ± 0.3161	0.9920 ± 0.0032

This means that, unlike the NOLM metric, solely the initial portion of the double Pareto distribution makes it possible to differentiate between the two languages. Namely, the statistical distributions appear different in the bulk of the analysed data, that is, for classes having NOS values below the first regime threshold, which is around 8 for Java and Python, while they are similar along the queue of the distributions, namely for larger value of NOS metric. Thus the use of statements in Java and Python classes appears to be different for classes having a small number of local methods. Table 6.13 shows the results for rank sum and Kruskal-Wallis tests for the double Pareto parameters for metric NOS.

These results show that the metric Number of Statements appears the most different in Java and Python projects, since they can be easily distinguished using both best fitting distributions, and in both cases the fit is fairly good.

Considering the obtained results, we succeeded in identifying 10 Java projects and 10 Python projects having comparable dimensions, belonging to similar application domains, and presenting similar statistical distributions for the corresponding metrics. We were also

Table 6.13: Rank sum and Kruskal-Wallis tests results for NOS double Pareto fits

	α	β
Rank sum (p-value)	0.0010	0.1859
Kruskal-Wallis (p-value)	8.8074e-04	0.1736

able to identify statistical distributions providing fairly good fits for the empirical distributions of their metrics, for all Java and Python projects. Finally, the best fitting parameters have been successfully used for a comparison of the metric distributions.
We found that systems developed in Java and Python do present different metrics distributions.
The double Pareto distribution fit for NOM metric reveals that, even though the statistical distributions for projects written in Java and Python may appear the same for lower values of the metric, major differences can be noticed along the queue of the distributions. The same analysis, performed with NOS metric, reveals that only the initial portion of the distribution shows differences between the two languages. Finally, the dispersion parameter associated to the log-normal distribution fit for NOM metric could also be used for distinguishing Java projects from Python ones.

The number of classes across the ten Java projects and the ten Python projects varies consistently from one system to the other, but the average number of classes for Java projects is twice the average number of classes for Python projects. Nevertheless, the average number of lines of code is roughly the same in both cases. This indicates that on average the number of lines of code in Python classes is roughly twice the corresponding number in Java classes. Thus Python projects use fewer classes than Java projects, but the same number of lines of code, on average. On the other hand, our data for the NOS metric show that the average number of statements declarations per class is larger in Java, despite the classes contain less lines of code on average. Since each statement normally requires one line of code, these results seem contradictory, and indicate that there must be a specific reason which drives Java software developers to write code richer in statement declarations than Python code.

6.5 Threats to Validity

Some projects were selected even if they are at a stage of their life cycle in which it cannot inherently be possible to notice an appreciable number of recent commits. This happens because they are less prone to show major bug issues, as they are mature systems. So, they can be considered as fully representative for the language used in their development.

In addition to this, it is worth to mention that not every Python system has a correspondent Java system, and vice versa. This is true as far as software scope is concerned. It was possible to identify only a limited subset of projects whose scope could be considered similar to that of each other. So, we are not only considering software pairs that serve almost the same purposes, but also other projects that do not show this relationship.

One of the projects, SCons, might not be qualified as Python project, because of the lower percentage of Python code with respect to that of XSLT. This could be considered as a minor issue, because there is a most relevant difference between Python and XSLT, more relevant than that between Java and Python. Although, it must be added that analyzing the code implies specifically looking for Python files for Python projects and Java files for Java ones.

6.6 Conclusion

We presented a statistical analysis of 20 open source object-oriented systems (10 written in Java, 10 written in Python), with the purpose of detecting differences between Java and Python projects through the study of the related metrics distributions.

We identified two statistical distribution functions, the log-normal and the double Pareto distributions, capable of providing good fits for three metrics for all Java and Python projects in the corpus; this made it possible to compare the distributions of the corresponding metrics for the two programming languages.

Our analysis showed that the best fitting parameters of these distributions allow for distinguishing between the two programming languages when using NOLM, NOM, and NOS metrics.

In particular, the metric NOLM revealed differences between Java and Python, detected through the parameter β of the double Pareto distribution fitting, meaning that classes with a higher number of local methods are differently used by developers in Java and in Python. The metric NOM showed differences in the dispersion parameter of the log-normal distribution, and also in the parameter β of the double Pareto distribution, meaning that the variability of the number of methods used in Python classes is lower than in Java classes, and also that classes with more methods (not only considering local methods), are differently used in the two languages.

Finally, the metric NOS revealed that statements are differently used in the two languages, and also that the average number of statements declarations in Java is much larger, even if Java classes feature fewer lines of code on average.

In conclusion, our study showed that a metrics analysis can be successfully leveraged to identify differences in programming practices related to the use of methods and statements in Java and Python, thus revealing meaningful differences between the two programming languages.

Chapter 7

Concluding Remarks

During the three years of the PhD course, the author conducted a series of studies focused on the influence of agile methodologies and new technologies on software maintenance and evolution. In the first part of the dissertation, we have also shown how information provided by private companies can be leveraged to properly interpret and exploit publicly available data on open source projects, and eventually implement software tools that can effectively support software maintenance and evolution.

The major contribution of this thesis is a machine learning classifier for effort estimation, capable of processing an issue report and of providing an effort estimate in terms of story points in real-time. As reported in Chapter 2, its adoption as a support tool was proven feasible on the basis of the feedback gathered from a Belgian IT company. It was possible to collect feedback from one of the company developer teams thanks to the support of the industrial partner, which allowed us to participate to their Planning Poker sessions. In addition, they granted us access to their issue tracking system and gave us valuable hints on which information would have been more relevant for the classification task. Leveraging the partner support and a dataset comprising the issue reports of their core product[1] and other eight open source projects, we developed a classifier which achieves–on every project but one–an MMRE spanning from 0.16 to 0.61, after an initial training on at least 300 issues. Once the system is set up, the time required for estimating a new issue is remarkably short, namely, between 8 to 15 seconds. In addition, the issue type, summary, description, and the components related to an issue proved to include project-dependent features relevant to story point estimation.

Continuing on the subject of the synergies between software maintenance and evolution and agile methodologies, in Chapter 3 we investigated the cognitive structure of the research on Refactoring from 2001 to 2015. The comparison of the strategic diagrams for the three selected periods revealed that agile methodologies assumed a leading, aggregating role in the period 2005-2009. During the same period, Java and Eclipse gained increasing popularity among researchers. Over the last period under study, spanning from 2009 and 2015, agile methodologies are still to be found among the motor themes in the research of Refactoring.

In the second part of the thesis, the focus shifted to the challenges for software maintenance and evolution in emerging software systems.

[1] Scriptura Engage, https://www.scripturaengage.com/. Accessed: 2017-03-12.

More specifically, in Chapter 4, the staggering development rate of the Blockchain sector led us to study current challenges and propose new research directions for blockchain-oriented software engineering. We advocated the need for new professional roles, enhanced security and reliability, novel modeling languages, and specialized metrics. Moreover, we relied upon the 2016 Moody's Blockchain Report and the market capitalization of cryptocurrencies to retrieve and extract preliminary data on the most popular open source blockchain-oriented software repositories. On the basis of the results obtained, we found that new research directions should focus on collaboration among large teams, testing, and specialized tools for the creation of smart contracts.

In Chapter 5, we leveraged complex networks to analyze a set of open source android applications with the purpose of investigating the relationship between mobile apps external dependencies and software quality. We found that our results are aligned with previous findings, namely, that the applications which are more connected to API classes are also more defect-prone.

Finally, in Chapter 6, we presented a statistical analysis of 20 open source object-oriented systems, 10 written in the highly popular language Java and 10 in the rising language Python. We leveraged two statistical distribution functions–the log-normal and the double Pareto distributions–to provide good fits, both in Java and Python, for three metrics, namely, the NOLM, NOM, and NOS metrics. The NOLM metric showed that classes with a higher number of local methods are differently used by Java and Python developers; the NOM metric revealed that the variability of the number of methods in Python classes is lower than in Java classes, and also that classes with more methods are differently used in the two languages; finally, the NOS metric showed that, on average, more statement declarations are to be found in Java classes, even if Java classes are generally smaller, featuring, on average, fewer lines of code than Python classes.

7.1 An Overall Perspective

As stated in the Introduction, this research work aims to gain a deeper insight on how agile methodologies and open source software development are shaping software engineering, and especially on how their combined use supports and can be leveraged to improve software maintenance and evolution. In this section, we present the studies reported in Chapter 2-6 from the perspective of the thesis objectives.

The research study on effort estimation (reported in Chapter 2) heavily relied upon open source projects' issues repositories, where more than 5.5k issues were found to be qualified as suitable for training the real-time story point classifier. Effort estimation, which is a staple for effective project management in the Agile context, is performed across all the issues repositories in the form of story points estimation, story points being a metric whose use is most common among agile teams. It must be noted that, in order to understand which data could have been deemed relevant to effort estimation, we had to gather information on how story points are actually used on the field by discussing with developers from a private IT company. However, an agile practitioner could have devised and trained the classifier by relying solely on the publicly available data.

As story points help speed up software maintenance and evolution, building an automated tool to improve story point estimation proved how open source software development and agile methodologies can be leveraged to support maintenance and evolution. This also

testifies that is possible to build upon the combined use of open source software development and agile methodologies to implement tools which further enhance their synergy.

In Chapter 3, we presented a study on the evolution of the Refactoring research field from 2001 to 2014. Refactoring is a key factor for increasing internal software quality throughout all the software lifecycle. More specifically, we showed that Agile Methodologies had a significant role in shaping the research on Refactoring, which is a critical process used to achieve sustainable maintenance and evolution by reducing their cost.

In Chapters 4 and 5, we reported two studies that focus on software maintenance and evolution in popular and emerging open source software systems, with the objective of obtaining deeper insight on the peculiar challenges, strengths, and weaknesses which software engineering practitioners must confront with.

In Chapter 4, we investigated the current challenges for the Blockchain software development by focusing on open source software repositories detected through the 2016 Moody's Blockchain Report, eventually proposing new research directions for blockchain-oriented software engineering. Moreover, we advocated the need for new professional roles, enhanced security and reliability, collaboration among large teams, and specialized metrics.

In Chapter 5, we once again relied upon open source software repositories, to investigate the communication between mobile applications and the underlying open source operating system, and thus to gain deeper insight on their impact on mobile software maintenance and evolution. More specifically, we found that the greater is the use of the underlying software system API, the higher is the defect-proneness of the application, confirming the results found by Syer et al. [6].

Finally, in Chapter 6, we also employed open source software, this time to perform a statistical analysis of 10 software projects written in Java, and 10 projects written in Python, to investigate whether the source code reflects software engineering practices that are peculiar to Java and Python. The study, among other findings, revealed that the variability of the number of methods used in Python classes is lower than in Java classes, and that Java classes, on average, feature fewer lines of code than Python classes. From the perspective of software engineering, these results can be built upon to devise specialized practices or tools which might be applied to software developed with a specific language.

7.2 Future Perspectives

As a future work, we first plan to improve the automated story point estimator presented in Chapter 5. The improvements will focus on detecting additional predictors and improving usability. On the one hand, the features set may be expanded through the inclusion of previously unnoticed, or simply not considered, information (e.g. comments typed and inserted into the issue report during Planning Poker sessions). On the other hand, the classifier usability can also be improved. More specifically, the estimator could be modified to (i) be used as a JIRA Cloud add-on [36], and (ii) to detect and reveal the most informative keywords for the estimation within the issue report. The latter feature would reduce the chances of having relevant keywords passing unnoticed. Further down the line, we plan to investigate (i) the role of emotions in the estimation process [37] and (ii) the human-machine interaction. For the latter, it is indeed known from literature that humans may trust more their own peers than a machine [38].

Due to its very nature, the exploratory study on the Blockchain sector reported in Chapter 4 is surely a source of new ideas as to the development of blockchain-oriented software engineering. In particular, the dataset used in the study, which comprises almost 1.2k open source repositories, could be better exploited to shed some more light on the blockchain-oriented software development and evolution. Moreover, the discussed challenges and the new research directions, mainly concerning testing, collaboration, and specialized languages for smart contracts, each can also inspire a more detailed study on the matter.

Future developments can also build upon the research on mobile applications discussed in Chapter 5. In particular, the investigation on mobile applications dependencies can be expanded by differentiating dependencies from both a programming language perspective and a functional perspective (e.g. dependency with Android API and external libraries, user interface, or network device). It would also be interesting to detect possible relations between the evolution pattern of the network associated to a mobile application and other business-relevant metrics (e.g. user votes or number of downloads as a measure of an app success).

Bibliography

[1] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. “A large scale study of programming languages and code quality in github”. In: *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM. 2014, pp. 155–165.

[2] Kumiyo Nakakoji, Yasuhiro Yamamoto, Yoshiyuki Nishinaka, Kouichi Kishida, and Yunwen Ye. “Evolution patterns of open-source software systems and communities”. In: *Proceedings of the international workshop on Principles of software evolution*. ACM. 2002, pp. 76–85.

[3] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. *Manifesto for agile software development*. 2001.

[4] Ming Huo, June Verner, Liming Zhu, and Muhammad Ali Babar. “Software quality and agile methods”. In: *Proceedings of the 28th Annual International Conference on Computer Software and Applications*. IEEE. 2004, pp. 520–525.

[5] Juhani Warsta and Pekka Abrahamsson. “Is open source software development essentially an agile method”. In: *Proceedings of the 3rd Workshop on Open Source Software Engineering*, pp. 143–147.

[6] M.D.a Syer, M.a Nagappan, B.b Adams, and A.E.a Hassan. “Studying the relationship between source code quality and mobile platform dependence”. In: *Software Quality Journal* (2014).

[7] Lionel C Briand. “On the many ways software engineering can benefit from knowledge engineering”. In: *Proceedings of the 2002 international conference on Software engineering and knowledge engineering*. ACM. 2002, pp. 3–6.

[8] James W Paulson, Giancarlo Succi, and Armin Eberlein. “An empirical study of open-source and closed-source software products”. In: *IEEE Transactions on Software Engineering* 30.4 (2004), pp. 246–256.

[9] Vibhu Srinivasan. *"What is a story point?"*. https://goo.gl/Vfb9v1. Accessed: 2016-06-02. Published: 2007.

[10] Mike Cohn. *"It's Effort, Not Complexity"*. https://goo.gl/nNYlL1. Accessed: 2016-06-02. Published: 2010.

[11] James Grenning. “Planning poker or how to avoid analysis paralysis while release planning”. In: *Hawthorn Woods: Renaissance Software Consulting* 3 (2002).

[12] Rupert Brown. *Group Processes: Dynamics Within and Between Groups*. Wiley-Blackwell, 2000.

[13] Elliot Aronson, Timothy D. Wilson, and Robin M. Akert. *Social Psychology (3rd Edition)*. Pearson, 1999.

[14] Rita L. Atkinson, Richard C. Atkinson, Edward E. Smith, Daryl J. Bem, and Susan Nolen-Hoeksema. *Hilgard's Introduction to Psychology (12th Edition)*. Orlando: Harcourt Brace College Publishers, 1996.

[15] Jorge Aranda and Steve Easterbrook. "Anchoring and Adjustment in Software Estimation". In: *Proceedings of the 2005 European Software Engineering Conference Held Jointly with the 2005 International Symposium on Foundations of Software Engineering*. ACM, 2005, pp. 346–355.

[16] *Planning Poker 3.0*. https://goo.gl/Tl5mws. Accessed: 2016-06-16.

[17] Muhammad Usman, Emilia Mendes, Francila Weidt, and Ricardo Britto. "Effort estimation in agile software development: A systematic literature review". In: *Proceedings of the 2014 International Conference on Predictive Models in Software Engineering*. ACM, 2014, pp. 82–91.

[18] Mountain Goat Software. *"Why do the cards deviate slightly from the Fibonacci sequence?"*. http://goo.gl/xgs7PF. In: Mountain Goat Software FAQs. Accessed: 2016-07-13.

[19] Adam Trendowicz, Jürgen Münch, and Ross Jeffery. "State of the practice in software effort estimation: a survey and literature review". In: *Software Engineering Techniques*. Springer, 2008, pp. 232–245.

[20] Atlassian. *"What is an issue"*. https://goo.gl/JduWpL. In: Atlassian Documentation. Accessed: 2016-07-13.

[21] GitHub. *"Mastering issues"*. https://goo.gl/OR1gwr. In: GitHub Guides. Accessed: 2016-07-13.

[22] Pekka Abrahamsson, Ilenia Fronza, Raimund Moser, Jelena Vlasenko, and Witold Pedrycz. "Predicting development effort from user stories". In: *Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement*. IEEE. 2011, pp. 400–403.

[23] Nathalie Japkowicz and Mohak Shah. *Evaluating learning algorithms: a classification perspective*. Cambridge University Press, 2011.

[24] I. Rish. *An empirical study of the naive bayes classifier*. Tech. rep. 2001.

[25] Thorsten Joachims. *Text categorization with support vector machines: Learning with many relevant features*. Springer, 1998.

[26] Nils C Augen. "An empirical study of using planning poker for user story estimation". In: *Agile Conference, 2006*. IEEE. 2006.

[27] Alessandro Murgia, Giulio Concas, Roberto Tonelli, Marco Ortu, Serge Demeyer, and Michele Marchesi. "On the influence of maintenance activity types on the issue resolution time". In: *Proceedings of the 2014 International Conference on Predictive Models in Software Engineering*. ACM. 2014, pp. 12–21.

[28] Robert T Hughes. “Expert judgement as an estimating method”. In: *Information and Software Technology* 38.2 (1996), pp. 67–75.

[29] Tarek K Abdel-Hamid. “Investigating the cost/schedule trade-off in software development”. In: *Software, IEEE* 7.1 (1990), pp. 97–105.

[30] Tom DeMarco. *Controlling software projects: Management, measurement, and estimates*. Prentice Hall PTR, 1986.

[31] W. AbdelMoez, M. Kholief, and F. M. Elsalmy. “Improving bug fix-time prediction model by filtering out outliers”. In: *Proceedings of the 2013 International Conference on Technological Advances in Electrical, Electronics and Computer Engineering*. 2013, pp. 359–364.

[32] Lionel Marks, Ying Zou, and Ahmed E Hassan. “Studying the fix-time for bugs in large open source projects”. In: *Proceedings of the 2011 International Conference on Predictive Models in Software Engineering*. ACM. 2011, p. 11.

[33] Cathrin Weiss, Rahul Premraj, Thomas Zimmermann, and Andreas Zeller. “How long will it take to fix this bug?” In: *Proceedings of the 2007 International Workshop on Mining Software Repositories*. IEEE Computer Society. 2007, p. 1.

[34] Hui Zeng and David Rine. “Estimation of software defects fix effort using neural networks”. In: *Proceedings of the 2004 Annual International Conference on Computer Software and Applications*. Vol. 2. IEEE. 2004, pp. 20–21.

[35] Qinbao Song, Martin Shepperd, Michelle Cartwright, and Carolyn Mair. “Software defect association mining and defect correction effort prediction”. In: *IEEE Transactions on Software Engineering* 32.2 (2006), pp. 69–82.

[36] Atlassian. *"Pros and Cons of Cloud vs. Server"*. https://goo.gl/5KF2rI. In: Atlassian Documentation. Accessed: 2016-07-13.

[37] Alessandro Murgia, Parastou Tourani, Bram Adams, and Marco Ortu. “Do Developers Feel Emotions? An Exploratory Analysis of Emotions in Software Artifacts”. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 262–271.

[38] Alessandro Murgia, Daan Janssens, Serge Demeyer, and Bogdan Vasilescu. “Among the Machines: Human-Bot Interaction on Social Q&A Websites”. In: *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. ACM, 2016, pp. 1272–1279.

[39] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.

[40] William F. Opdyke. “Refactoring Object-oriented Frameworks”. PhD thesis. 1992.

[41] M. Abebe and C.-J. Yoo. “Trends, opportunities and challenges of software refactoring: A systematic literature review”. In: *International Journal of Software Engineering and its Applications* 8.6 (2014), pp. 299–318.

[42] M. Misbhauddin and M. Alshayeb. “UML model refactoring: a systematic literature review”. In: *Empirical Software Engineering* 20.1 (2015), pp. 206–251.

[43] J. Al Dallal. “Identifying refactoring opportunities in object-oriented code: A systematic literature review”. In: *Information and Software Technology* 58 (2015), pp. 231–249.

[44] G. Vale, E. Figueiredo, R. Abilio, and H. Costa. “Bad smells in software product lines: A systematic review”. In: 2014, pp. 84–94.

[45] M. J. Cobo, A. G. Lopez-Herrera, E. Herrera-Viedma, and F. Herrera. “Science Mapping Software Tools: Review, Analysis, and Cooperative Study Among Tools”. In: *J. Am. Soc. Inf. Sci. Technol.* 62.7 (July 2011), pp. 1382–1402.

[46] Manolo J. Cobo, Antonio G. Lopez-Herrera, Francisco Herrera, and Enrique Herrera-Viedma. “A Note on the ITS Topic Evolution in the Period 2000-2009 at T-ITS”. In: *Trans. Intell. Transport. Sys.* 13.1 (Mar. 2012), pp. 413–420.

[47] H.P.F. Peters and A.F.J. van Raan. “Co-word-based science maps of chemical engineering. Part I: Representations by direct multidimensional scaling”. In: *Research Policy* 22.1 (1993), pp. 23 –45.

[48] T. Mens and T. Tourwé. “A survey of software refactoring”. In: *IEEE Transactions on Software Engineering* 30.2 (2004), pp. 126–139.

[49] Henry Small. “Visualizing science by citation mapping”. In: *Journal of the American Society for Information Science* 50.9 (1999), pp. 799–813.

[50] Michel Callon, Jean-Pierre Courtial, William A. Turner, and Serge Bauin. “From translations to problematic networks: An introduction to co-word analysis”. In: *Social Science Information* 22.2 (1983), pp. 191–235.

[51] Neal Coulter, Ira Monarch, and Suresh Konda. “Software Engineering As Seen Through Its Research Literature: A Study in Co-word Analysis”. In: *J. Am. Soc. Inf. Sci.* 49.13 (Nov. 1998), pp. 1206–1223.

[52] M.J. Cobo, A.G. Lopez-Herrera, E. Herrera-Viedma, and F. Herrera. “SciMAT: A New Science Mapping Analysis Software Tool”. In: *J. Am. Soc. Inf. Sci. Technol.* 63.8 (Aug. 2012), pp. 1609–1630.

[53] M. Callon, J. Courtial, and F. Laville. “Co-word analysis as a tool for describing the network of interactions between basic and technological research: The case of polymer chemistry”. In: *Scientometrics* 22.1 (1991), pp. 155–205.

[54] J. E. Hirsch. “An index to quantify an individual’s scientific research output”. In: *Proceedings of the National Academy of Sciences of the United States of America* 102.46 (2005), pp. 16569–16572.

[55] Melanie Swan. *Blockchain: Blueprint for a new economy*. O’Reilly Media, Inc., 2015.

[56] Unicredit. *Blockchain Technology and Applications from a Financial Perspective*. 2016.

[57] Moody’s Investors Service. *Robust, Cost-effective Applications Key to Unlocking Blockchain’s Potential Credit Benefits*. 2016.

[58] Anthony I Wasserman. “Software engineering issues for mobile application development”. In: *Proceedings of the FSE/SDP workshop on Future of software engineering research*. ACM. 2010, pp. 397–400.

[59] Alain Abran, James W Moore, P Bourque, R Dupuis, and LL Tripp. “SWEBOK: Guide to the Software Engineering Body of Knowledge 2004 Version”. In: *IEEE Computer Society, Los Alamitos, California* (2004).

[60] Harlan D Mills, Michael Dyer, and Richard C Linger. “Cleanroom software engineering”. In: *IEEE Software* 4.5 (1987), p. 19.

[61] Daniel Larimer. *"Introducing BitShares Object Graph"*. https://goo.gl/TWWSif. Accessed: 2016-10-20. Published: 2014-12-24.

[62] Garrick Hileman. *"State of Blockchain Q1 2016"*. http://www.coindesk.com/state-of-blockchain-q1-2016/. Accessed: 2017-03-05. Published: 2016-05-11.

[63] Pavneet Singh Kochhar, Tegawendé F Bissyandé, David Lo, and Lingxiao Jiang. "Adoption of Software Testing in Open Source Projects–A Preliminary Study on 50,000 Projects". In: *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*. IEEE. 2013, pp. 353–356.

[64] Mark Aberdour. "Achieving quality in open-source software". In: *IEEE software* 24.1 (2007), pp. 58–64.

[65] Visionmobile Ltd. *App Economy Forecasts 2013-2016*. Vision Mobile, July 2013.

[66] Ariel Michaeli. *"App Stores Growth Accelerates in 2014"*. http://blog.appfigures.com/app-stores-growth-accelerates-in-2014/. Accessed: 2017-03-05. Published: 2015-01-13.

[67] Mario Linares-Vásquez. "Supporting evolution and maintenance of Android apps". In: *Companion Proceedings of the 36th International Conference on Software Engineering - ICSE Companion 2014*. ACM Press, 2014, pp. 714–717.

[68] I.J.a Mojica Ruiz, M.a Nagappan, B.b Adams, and A.E.a Hassan. "Understanding reuse in the Android Market". In: 2012, pp. 113–122.

[69] S.a Arzt, S.a Rasthofer, C.a Fritz, E.a Bodden, A.b Bartel, J.b Klein, Y.b Le Traon, D.c Octeau, and P.c McDaniel. "FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps". English. In: Association for Computing Machinery, 2014, pp. 259–269.

[70] Mark D. Syer, Bram Adams, Ying Zou, and Ahmed E. Hassan. "Exploring the Development of Micro-apps: A Case Study on the BlackBerry and Android Platforms". In: *2011 IEEE 11th International Working Conference on Source Code Analysis and Manipulation*. IEEE, Sept. 2011, pp. 55–64.

[71] M. E. J. Newman. "The structure and function of complex networks". In: *SIAM REVIEW* 45 (2003), pp. 167–256.

[72] A.-L Barabasi and R. Albert. "Emergence of scaling in random networks". In: *Science* 286 (1999), pp. 509–512.

[73] A. Barabasi, R. Albert, and H. Jeong. "Scale-free characteristics of random networks: the topology of the World Wide Web". In: *Phys. A* 281 (2000), pp. 69–77.

[74] Giulio Concas, Michele Marchesi, Alessandro Murgia, and Roberto Tonelli. "An empirical study of social networks metrics in object-oriented software". In: *Adv. Soft. Eng.* 2010 (Jan. 2010), 4:1–4:21.

[75] S. Chidamber and C. Kemerer. "A Metrics Suite for Object-Oriented Design". In: *IEEE Trans. Software Eng.* 20.6 (1994), pp. 476–493.

[76] Lian Wen, Diana Kirk, and R. G. Dromey. "Software Systems as Complex Networks". In: *Proceedings of the 6th IEEE International Conference on Cognitive Informatics*. IEEE Computer Society, 2007, pp. 106–115.

[77] Deyi Li, Yanni Han, and Jun Hu. “Complex Network Thinking in Software Engineering”. In: *Proceedings of the 2008 International Conference on Computer Science and Software Engineering - Volume 01*. IEEE Computer Society, 2008, pp. 264–268.

[78] Jens Dietrich, Vyacheslav Yakovlev, Catherine McCartin, Graham Jenson, and Manfred Duchrow. “Cluster analysis of Java dependency graphs”. In: *Proceedings of the 4th ACM symposium on Software visualization*. ACM, 2008, pp. 91–94.

[79] Christopher R. Myers. “Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs”. In: *Phys. Rev. E* 68.4 (2003), p. 046116.

[80] L. Šubelj and M. Bajec. “Community structure of complex software systems: Analysis and applications”. In: *Physica A Statistical Mechanics and its Applications* 390 (Aug. 2011), pp. 2968–2975.

[81] Valverde S. Cancho R. and V. Sole. “Scale Free Networks From optimal Design”. In: *Europhysics Letters* 60 (2002).

[82] Sergi Valverde and Ricard V. Sole. “Hierarchical Small Worlds in Software Architecture”. In: arXiv:cond-mat/0307278v2 (2003).

[83] Giulio Concas, Cristina Monni, Matteo Orrù, and Roberto Tonelli. “A Study of the Community Structure of a Complex Software Network”. In: *Proceedings of the 2013 ICSE Workshop on Emerging Trends in Software Metrics*. ACM, 2013, pp. 14 –20.

[84] Giulio Concas, Michele Marchesi, Alessandro Murgia, Sandro Pinna, and Roberto Tonelli. “Assessing traditional and new metrics for object-oriented systems”. In: *Proceedings of the 2010 ICSE Workshop on Emerging Trends in Software Metrics*. ACM, 2010, pp. 24–31.

[85] M Girvan and M E J Newman. “Community structure in social and biological networks”. In: *Proc. Natl. Acad. Sci. U. S. A.* 99 (2001), pp. 8271–8276.

[86] S. Fortunato. “Community detection in graphs”. In: *Physics Report* 486 (Feb. 2010), pp. 75–174.

[87] Mario Linares-Vasquez, Gabriele Bavota, Carlos Bernal-Cardenas, Massimiliano Di Penta, Rocco Oliveto, and Denys Poshyvanyk. “API change and fault proneness: a threat to the success of Android apps”. In: *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2013*. ACM Press, 2013, p. 477.

[88] Sergio Focardi, Michele Marchesi, and Giancarlo Succi. “A stochastic model of software maintenance and its implications on extreme programming processes”. In: Addison-Wesley Longman Publishing Co., Inc., 2001, pp. 191–206.

[89] M. E. J. Newman. “From the Cover: Modularity and community structure in networks”. In: *Proceedings of the National Academy of Science* 103 (June 2006), pp. 8577–8582.

[90] Diaz-Guilera A. Duch J. Danon L and A. Arenas. “Community Structure Identification, a modern review”. In: *Large Scale Structure and Dynamics of Complex Networks: From Information Technology to Finance and Natural Science, World Scientific* (2007), pp. 93–113.

[91] A. Clauset, M. E. J. Newman, and C. Moore. “Finding community structure in very large networks”. In: *Phys. Rev. E* 70.6, 066111 (Dec. 2004), p. 066111.

[92] Giuseppe Destefanis. “Which programming language should a company use? A Twitter-based analysis”. In: *CRIM Technical Report*. CRIM. 2014, pp. 47–48.

[93] Giuseppe Destefanis, Steve Counsell, Giulio Concas, and Roberto Tonelli. “Software metrics in agile software: An empirical study”. In: *Agile Processes in Software Engineering and Extreme Programming*. Springer, 2014, pp. 157–170.

[94] Shyam R Chidamber and Chris F Kemerer. “A metrics suite for object oriented design”. In: *Software Engineering, IEEE Transactions on* 20.6 (1994), pp. 476–493.

[95] Victor R Basili, Lionel C Briand, and Walcélio L Melo. “A validation of object-oriented design metrics as quality indicators”. In: *Software Engineering, IEEE Transactions on* 22.10 (1996), pp. 751–761.

[96] Shyam R Chidamber, David P Darcy, and Chris F Kemerer. “Managerial use of metrics for object-oriented software: An exploratory analysis”. In: *Software Engineering, IEEE Transactions on* 24.8 (1998), pp. 629–639.

[97] Ramanath Subramanyam and Mayuram S. Krishnan. “Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects”. In: *Software Engineering, IEEE Transactions on* 29.4 (2003), pp. 297–310.

[98] Thomas Zimmermann and Nachiappan Nagappan. “Predicting defects using network analysis on dependency graphs”. In: *Proceedings of the 30th international conference on Software engineering*. ACM. 2008, pp. 531–540.

[99] Alessandro Murgia, Giulio Concas, Roberto Tonelli, and Ivana Turnu. “Empirical study of software quality evolution in open source projects using agile practices”. In: *Proc. of the 1st International Symposium on Emerging Trends in Software Metrics*. 2009, p. 11.

[100] Alessandro Murgia, Giulio Concas, Michele Marchesi, Roberto Tonelli, and Ivana Turnu. “An analysis of bug distribution in object oriented systems”. In: *arXiv preprint* (2009). arXiv:0905.3296.

[101] Giulio Concas, Giuseppe Destefanis, Michele Marchesi, Marco Ortu, and Roberto Tonelli. “Micro Patterns in Agile Software”. In: *Agile Processes in Software Engineering and Extreme Programming: 14th International Conference, XP 2013, Vienna, Austria, June 3-7, 2013, Proceedings*. Vol. 149. Springer. 2013, p. 210.

[102] Giuseppe Destefanis, Roberto Tonelli, Ewan Tempero, Giulio Concas, and Michele Marchesi. “Micro Pattern Fault-Proneness”. In: *Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on*. IEEE. 2012, pp. 302–306.

[103] Panagiotis Louridas, Diomidis Spinellis, and Vasileios Vlachos. “Power laws in software”. In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 18.1 (2008), p. 2.

[104] Mohammad Alshayeb and Wei Li. “An empirical validation of object-oriented metrics in two different iterative software processes”. In: *Software Engineering, IEEE Transactions on* 29.11 (2003), pp. 1043–1049.

[105] Alex Potanin, James Noble, Marcus Frean, and Robert Biddle. “Scale-free geometry in OO programs”. In: *Communications of the ACM* 48.5 (2005), pp. 99–103.

[106] Raed Shatnawi and Qutaibah Althebyan. “An Empirical Study of the Effect of Power Law Distribution on the Interpretation of OO Metrics”. In: *ISRN Software Engineering* 2013 (2013).

[107] Giulio Concas, Michele Marchesi, Alessandro Murgia, Sandro Pinna, and Roberto Tonelli. “Assessing traditional and new metrics for object-oriented systems”. In: *Proceedings of the 2010 ICSE Workshop on Emerging Trends in Software Metrics*. ACM. 2010, pp. 24–31.

[108] Michael Mitzenmacher. “Dynamic models for file sizes and double pareto distributions”. In: *Internet Mathematics* 1.3 (2004), pp. 305–333.

[109] William J Reed and Barry D Hughes. “From gene families and genera to incomes and internet file sizes: Why power laws are so common in nature”. In: *Physical Review E* 66.6 (2002).

[110] William J Reed and Murray Jorgensen. “The double Pareto-lognormal distribution–a new parametric model for size distributions”. In: *Communications in Statistics-Theory and Methods* 33.8 (2004), pp. 1733–1753.

[111] Colin P Stark and Niels Hovius. “The characterization of landslide size distributions”. In: *Geophysical Research Letters* 28.6 (2001), pp. 1091–1094.

[112] Ewan Tempero, Craig Anslow, Jens Dietrich, Ted Han, Jing Li, Markus Lumpe, Hayden Melton, and James Noble. “The Qualitas Corpus: A curated collection of Java code for empirical studies”. In: *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*. IEEE. 2010, pp. 336–345.

[113] Matteo Orrú, Ewan Tempero, Michele Marchesi, Roberto Tonelli, and Giuseppe Destefanis. “A Curated Benchmark Collection of Python Systems for Empirical Studies on Software Engineering”. In: *Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering*. ACM. 2015.

List of Publications Related to the Thesis

Conference Proceedings

Papers

- Simone Porru, Alessandro Murgia, Serge Demeyer, Michele Marchesi, and Roberto Tonelli, "Estimating Story Points from Issue Reports", in *Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering,* 2016. (Relation to Chapter 2)

- Matteo Orrú, Simone Porru, Michele Marchesi, and Roberto Tonelli, "The evolution of knowledge in the refactoring research field", in *Scientific Workshop Proceedings of the XP2015,* 2015. (Relation to Chapter 3)

- Matteo Orrú, Simone Porru, Roberto Tonelli, and Michele Marchesi, "A Preliminary Study on Mobile Apps Call Graphs through a Complex Network Approach", in *Proceedings of the 11th International Conference on Signal-Image Technology & Internet-Based Systems,* 2015. (Relation to Chapter 5)

- Giuseppe Destefanis, Marco Ortu, Simone Porru, Stephen Swift, and Michele Marchesi, "A statistical comparison of Java and Python software metric properties", in *Proceedings of the 7th International Workshop on Emerging Trends in Software Metrics,* 2016. (Relation to Chapter 6)

Extended Abstracts

- Simone Porru, Andrea Pinna, Michele Marchesi, and Roberto Tonelli, "Blockchain-oriented Software Engineering: Challenges and New Directions", in *Proceedings of the 39th International Conference on Software Engineering,* 2017. (Relation to Chapter 4)