Empirical Software Engineering (2022) 27:135
https://doi.org/10.1007/s10664-022-10166-x

Development effort estimation in free/open source software from activity in version control systems

Gregorio Robles[1] · Andrea Capiluppi[2] · Jesus M. Gonzalez-Barahona[1] · Björn Lundell[3] · Jonas Gamalielsson[3]

Accepted: 18 March 2022

Abstract
Effort estimation models are a fundamental tool in software management, and used as a forecast for resources, constraints and costs associated to software development. For Free/Open Source Software (FOSS) projects, effort estimation is especially complex: professional developers work alongside occasional, volunteer developers, so the overall effort (in person-months) becomes non-trivial to determine. The objective of this work it to develop a simple effort estimation model for FOSS projects, based on the historic data of developers' effort. The model is fed with direct developer feedback to ensure its accuracy. After extracting the personal development profiles of several thousands of developers from 6 large FOSS projects, we asked them to fill in a questionnaire to determine if they should be considered as full-time developers in the project that they work in. Their feedback was used to fine-tune the value of an effort threshold, above which developers can be considered as full-time. With the help of the over 1,000 questionnaires received, we were able to determine, for every project in our sample, the threshold of commits that separates full-time from non-full-time developers. We finally offer guidelines and a tool to apply our model to FOSS projects that use a version control system.

Keywords Effort estimation · Open source · Free software · Mining software repositories · Versioning system · Commits

1 Introduction

Effort estimation models are invaluable tools in software management: they help gaining insights on past resources and associated costs; and they serve as models to forecast resource demand and allocation, as well as dealing with predicted constraints. Traditionally, effort estimation is generally used by companies in the earlier stages of a software project to

Communicated by: Kelly Blincoe

✉ Andrea Capiluppi
a.capiluppi@rug.nl

Extended author information available on the last page of the article.

Published online: 20 July 2022

estimate the number of developers, and the amount of time that it will require to develop a software.

In the context of Free/Open Source Software (FOSS) development, there has hardly been a formal way of tracking the effort of developers, especially in its original volunteering form. It was argued that, in some circumstances, small-team development could eventually evolve (Capiluppi and Michlmayr 2007) into a distributed development (e.g., ‘bazaar’). In such cases, effort is even more complex to track, since the level of contributions varies along an onion-type model (Crowston and Howison 2005), where the outer the layer the smaller the code contribution, but the larger the user base, and the bug reporting facility. The inner the layer, the more pronounced the development effort.

This volunteer-based model for FOSS development was later complicated by the participation of companies, alongside volunteers (Riehle et al. 2014). Companies devoted staff and effort into a FOSS project to enter a saturated market, or to gain a foothold in a specialised development, without deploying vast amounts of resources and recreate a competitor product from scratch (Fitzgerald 2006). This hybrid model introduces further complexity in the effort estimation modeling: if the volunteers can still be modeled alongside the basic onion-model, the involvement of external companies and their focused effort become quite challenging to describe and model.

The effort estimation model presented depends on the threshold value of a FOSS project, and on the analysis of its publicly available data on version control systems (VCSs), such as git. Other projects can easily create their tailored model, and survey their developers to obtain their threshold value. As a further contribution of this work, we propose the threshold values of six large FOSS projects. These values can be re-used, compared, or validated with other projects.

Given the fluid nature of FOSS development, all the existing approaches to FOSS effort estimation are currently missing two pieces of crucial information. The first is a precise measure of past effort of the developers, even in the more sophisticated models. In traditional effort estimation models, software projects are able to estimate future effort, because they know their past effort. Given the increasing involvement of companies in FOSS development, this aspect is fundamental to (i) allow potential new participant companies to evaluate the current developers effort, and (ii) fine tune their participatory effort in the FOSS projects.

The second piece of crucial information that is missing is some sort of validation by the developers themselves. Assigning developers to the wrong tiers, or using a too wide (or too small) time window to evaluate the commits activity, produces different values of estimated effort.

For this purpose, in this paper we surveyed over 1,000 developers working on 6 large FOSS projects, with the aim to count how many developers consider themselves as ‘*full-timers*’ in each project. The number of commits (in a six months’ timespan) of these self-identified full-time developers was later used as the *threshold* θ that minimises the error to differentiate full-timers from other contributors. We used that threshold θ to estimate the effort produced by developers in each of the analysed FOSS projects. The contribution of this work is, to the knowledge of the authors, the first developer-validated model to FOSS effort estimation available in literature.

The present work is an extension of a previously published paper (Robles et al. 2014). Its *Future Work* section posited that: “*We envisage to expand this study by 1) studying other FOSS projects to ascertain if our method is applicable in general, and if it is to what extent; 2) performing a scientific experiment to obtain margins of error for the estimation of error for full-time developers (...)*”.

The current paper should therefore be considered as the enactment of the future work proposed in Robles et al. (2014), and provides the following specific contributions:

- We provide analysis and results for five more industrial FOSS projects.
- We offer an extension of a previous SLR on effort estimation including papers from 2016-2020.
- We clarify the model, offering an example of how it can be used in practice.
- We further develop on the concepts of goodness and compensation
- We discuss about the estimation error of the model.
- We discuss the repercussions for practitioners.

This paper is articulated as follows: Section 2 introduces the vision of this research. Section 3 deals with related work, especially focusing on the effort estimation models built for the various generations of FOSS (e.g., fully volunteered projects, hybrid FOSS projects, company-led FOSS projects). Section 4 introduces the model for effort estimation that we propose, alongside the terms and definitions we used throughout this study; Section 5 introduces the characteristics of the analysed projects, while Section 6 describes the survey that was circulated to the developers of the FOSS projects, and how the threshold θ was evaluated for each project. Section 7 shows how the model is deployed for the OpenStack project, while Section 8 proposes further replication, when considering five other projects. Section 9 discusses the findings, especially considering what sort of benefits we envisage for FOSS communities and companies. Section 10 finally concludes.

2 Vision of the Research

The model that we propose below is based on an overarching vision, that is set to maximise the applicability and reproducibility of our model in other FOSS projects. As depicted in the flowchart of Fig. 1, the model requires the parsing of a project's VCS, and that is a relatively simple resource to have an access to, with modern software engineering tooling. The model then uses the responses of developers to a survey to establish a value of commits (θ) above which developers should be considered as full-time.

The second data source of our effort estimation model (i.e., creating, distributing, collecting and analysing the developer surveys) can be time consuming and requires commitment from a FOSS project's management board. To combat this, our model has been built also for those projects that cannot (or do not want) to directly and regularly survey their developers: the vision of this research is to build enough evidence from several FOSS projects, and to obtain threshold values for *classes* of similar projects. Those shared thresholds can then be reused for other projects, thus removing the need to regularly survey developers.

3 Related Research

Traditional effort estimation models are generally used by organizations in the early stages of a project to estimate total effort, number of developers and time (e.g., duration) needed to develop the software. Effort estimation in traditional commercial software has been approached by means of models, either top-down (e.g., by analogy or by expert judgement) or bottom up (e.g., via regression models using past data). In the area of effort estimation, a comprehensive and systematic literature review on effort estimation was presented in 2007

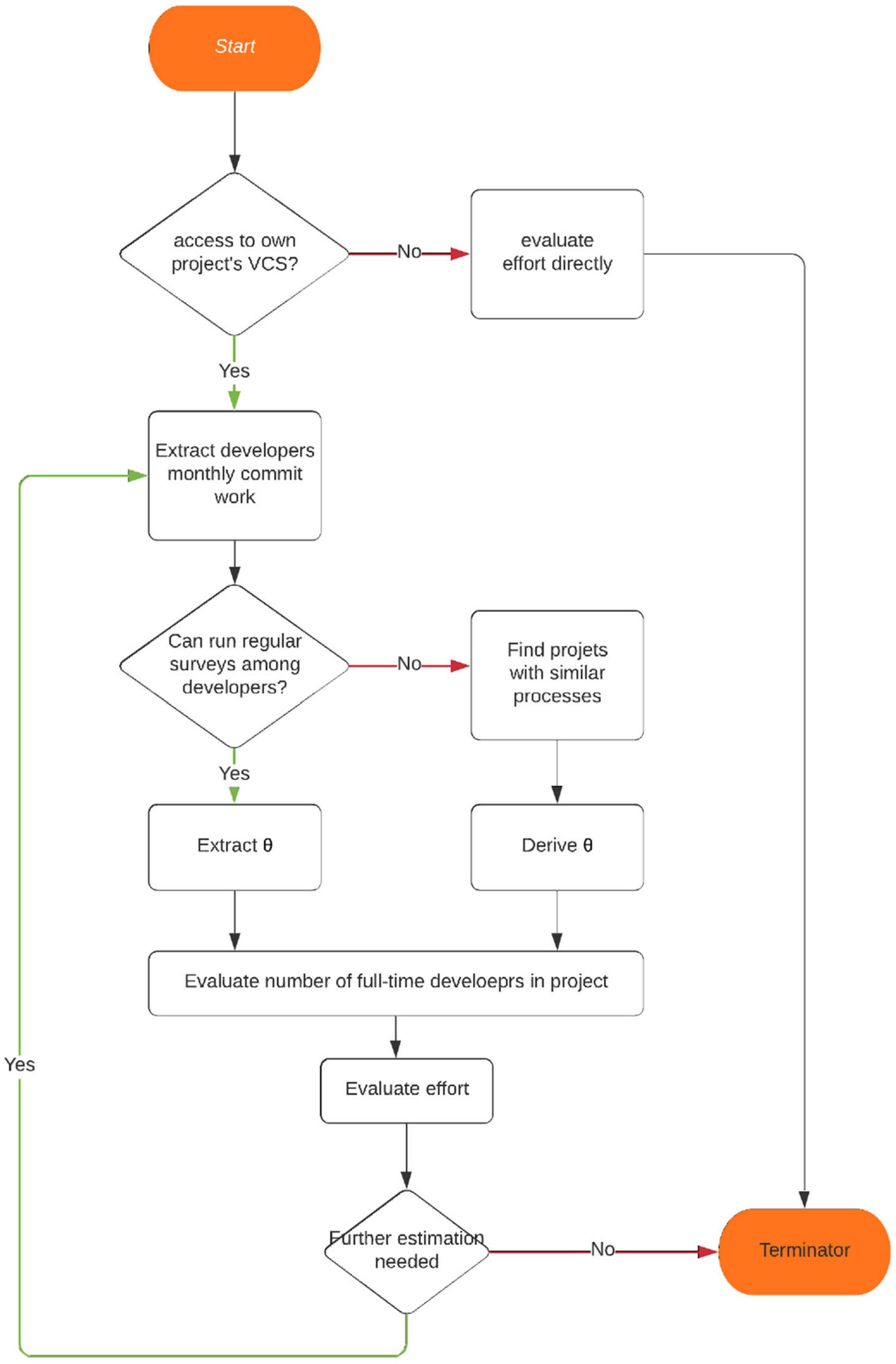

Start
access to own project's VCS?
No
evaluate effort directly
Yes
Extract developers monthly commit work
Can run regular surveys among developers?
No
Find projets with similar processes
Yes
Extract θ
Derive θ
Evaluate number of full-time develoeprs in project
Evaluate effort
Further estimation needed
No
Terminator
Yes

Fig. 1 Flowchart to run our effort estimation model in other FOSS projects

in Jorgensen and Shepperd (2007). The most evident result was to observe that over half of the papers surveyed are based on *history-based* evaluations to build models to estimate future effort.

Similarly to the systematic literature review (SLR) of 2007 (Jorgensen and Shepperd 2007), the research works around effort estimation (specifically for FOSS systems) were extensively analysed in 2016 by Wu et al. (2016). The type of study proposed by the authors, alongside the object of the predictions, were analysed and clustered, as visible in the top part of Fig. 2. Different types of studies were identified, based on the *goal* of the relative research ('Guidelines and Measurement', 'Measure individual contribution', 'Predict maintenance activity resource', 'Predict direct effort' and 'Predict indirect effort') and the *approach* used in the research ('Development of estimation method', 'Case study', 'Experiment', 'History-based evaluation', 'Theory' and 'Comparison study'). The largest set of research studies was found to be under the 'Development of estimation method' to 'predict maintenance activity resources' (comparably to Jorgensen and Shepperd 2007).

In order to include the more recent literature, we repeated the process presented in the 2016 SLR. We analysed the papers that were written between 2016 and 2020, using the same search string,[1] and the same search engines.[2] We obtained 2,279 candidate studies from the database search (without duplicates). After reading abstract and title, we obtained 41 articles; after reading each, we gathered 16 studies that comply with the search criteria from the original SLR. The labels of the SLR were applied to these papers, as visible in the bottom part of Fig. 2.

Similarly to the original study, the largest category was found again to be the studies that predict maintenance activity resources: although most of the papers design and/or deploy a new estimation method (Hönel et al. 2018; Mi and Keung 2016; Thung 2016; Zhao et al. 2016, mostly focused on bug-fixing efforts), we also found three studies that used history-based evaluations to directly predict effort (Malhotra and Lata 2020; Porru et al. 2016; Yang et al. 2016).

Applicability of Traditional Effort Estimation techniques to FOSS The FOSS paradigm, at least in its original form, was deemed as difficult, if not impossible, to fit under the most well-known estimation models (for instance the COCOMO I or COCOMO II models (Boehm 1981; Boehm et al. 2000)). Various research attempts have showed that fundamental assumptions and constructs could not be applied to the FOSS domain; whereas other aspects were simply absent from commercial organizations (Amor et al. 2006; Capiluppi and Izquierdo-Cortázar 2013; Kalliamvakou et al. 2009; Robles et al. 2014).

More specifically, Amor et al. (2006) proposed to measure the *total* effort invested in a project by characterizing *all* the activities of developers that can be traced: commits, e-mails, bug activities, among others. Kalliamvakou et al. (2009) replicated the study and included the contribution of developer effort.

Capiluppi and Izquierdo-Cortázar (2013) determined the average hours worked per day by the Linux kernel developers. They characterized the Linux kernel by the time of day when the commits were observed in the repository and when the author worked most frequently. They divided a working day into traditional office hours (from 9am to 5pm), after

[1]As per the original paper: ("Open source" OR OSS OR FOSS OR FLOSS OR opensource OR libre OR free) AND (maintain OR maintenance OR evolve OR evolution OR "bug fix*" OR "bug-fixing" OR "defect fixing" OR "defect correction" OR "defect resolution" OR effort OR cost OR estimat*, predict*) AND (empirical OR validation OR experiment OR evaluation OR "case study").

[2]Inspec, Compendex, IEEE Xplore, ACM Digital Library, Science Direct and Google Scholar.

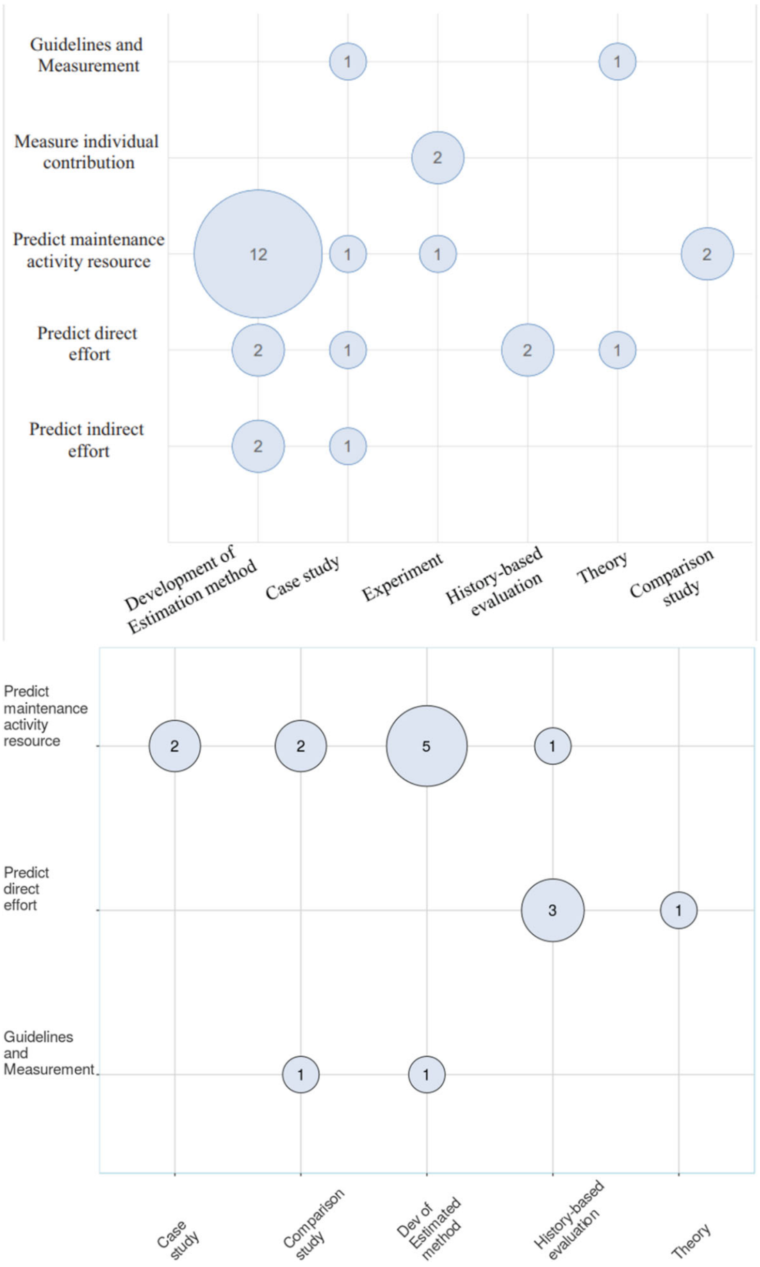


Fig. 2 Topic of the SLR conducted in 2016 by Wu et al. (2016) (top); additional SLR performed for this study (bottom)

office hours (5pm to 1am) and late hours (from 1am to 8am). The authors found that within the Linux kernel community the effort was constant throughout the week, which suggested the need for different estimation models from the ones traditionally used in industrial settings, where the work schedule is presumed to be 9am-5pm, Monday to Friday.

A more general work to address the commit frequency was presented in Kolassa et al. (2013a). Using the http://Ohloh.net repository, the authors evaluated the overall commit frequency, and the average number of commits that developers featured on. The same repository was also analysed to evaluate the size of a typical commit (Kolassa et al. 2013b), where the authors found power laws underlying the patterns of commit size. Later on, the frequency of commits was analysed in Hou et al. (2014) for two Apache projects, POI and Tomcat. The authors also found that commits follow power laws, with large bursts and long tails. Power laws were again found in 4 Apache projects, when analysing the time interval between commits and files committed (Ma et al. 2014).

Mockus and Votta (2000) showed that, out of nearly 400 programmers in the Apache project, the 15 most productive ones contributed 88 percent of the total lines of code (LOC). They compared those 15 Apache developers with programmers in five other commercial projects. They defined *code productivity* as the mean number of lines of code per developer per year (KLOC/developer/year). Koch and Schneider (2002, 2008) reported that in FOSS projects, the distribution of effort between participants (programmers) is skewed as in Mockus et al. (2002). Indeed, the majority of code was written and most of the effort spent by just a few contributors. This paper presents and illustrates the use of a similar approach, when identifying who these ‘major contributors’ are, but instead of imposing an a-priori threshold above which we consider a developer to be a ‘super’ or a ‘major’ contributor (or even a *hero* (Agrawal et al. 2018)), we validate that threshold with the help of the developer surveys. To the best of our knowledge, the identification of developer types in this way is novel.

Moulla et al. (2013, 2014) applied the COCOMO model to the TRIADE (version 7a) FOSS project, using LOCs and regression models with COSMIC Function Points as the independent variable (Dumke and Abran 2016; Abran et al. 2016). They reported that in terms of effort, development of software based on FOSS has advantages over development from scratch. Fernandez-Ramil et al. (2009) used linear regression models to estimate effort and duration for FOSS projects with LOC as the independent variable. Yu (2006) also used linear regression models with LOC as the independent variable to indirectly predict maintenance effort in free and FOSS projects. Effort was measured by examining the LOC added, deleted and modified. Anbalagan and Vouk (2009) investigated predicting how much time developers spend in corrective maintenance of FOSS projects. Their study focused on 72,482 bug reports from over nine releases of Ubuntu, a Linux distribution system. They used a linear regression model with ‘bug reports corrected by developer’ as the independent variable to predict the time taken to correct bugs. They observed that for FOSS estimation maintenance effort is lower than for proprietary development due to higher code quality.

According to Capra et al. (2007), Capra et al. (2008), and Capra et al. (2010), FOSS shows a slower growth of maintenance effort over time. Studies by Koch and Schneider (2002), Koch (2004), and Koch (2008) on effort modelling and developer participation in FOSS projects show that the number of participants, other than programmers, was about one order of magnitude larger than the number of programmers. In summary, a number of studies are available on software effort estimation but few discuss effort and duration in FOSS projects using linear regression models. The categorization of contributors and the size and choice of the datasets also present a challenge for research on FOSS projects estimation.

Effort Estimation in Hybrid FOSS Systems The terminology around FOSS started to change as long as volunteer communities began to collaborate with industrial entities. One of the first research studies that identified the possibility of a mixed development approach was the seminal work by Lerner and Tirole (2002). Various strategies were formulated, where commercial enterprises could be profitable using FOSS as a strategic asset.

A more formal description of the grades of involvement of commercial companies into FOSS projects was formulated in two research works (Shah 2006; Capra et al. 2008). The first work was focused on the motivations of developers and how that was affected by the type of FOSS development. In the second work, and using 'governance' as the term of reference, Capra et al. produced various levels of involvement, from volunteer-based, to company-driven FOSS projects.

Summary of Past Work Previous work on FOSS effort estimation has been typically run by detecting effort using *direct* measurements (e.g., number of developers) and applying proxies to those in order to derive a measurement of effort. In other cases, effort was derived *indirectly*, by using various other measurements, and inferring a relationship to the effort devoted to a software project. Table 1 summarises the past works in terms of how direct and indirect measurements (and which ones) were used when deriving effort estimation models. As visible, the approach that we propose in our paper (and that has been trialled in its earlier

Table 1 Summary of past works based on the use of direct and indirect measurements

	Ref	Type	Attributes	Validated w developers
Source code	Yu (2006)	INDIRECT	lag time, source code changes	NO
	Alomari (2015)	INDIRECT	lag time, delta LOCs, delta files	NO
Process based	Abdelmoez et al. (2012)	INDIRECT	Bugs data	NO
	Ahsan et al. (2009)	INDIRECT	Bugs data	NO
People-based estimation	Koch and Schneider (2002)	DIRECT	Proxy of {lag time, LOCs}	NO
	Capra et al. (2007)	DIRECT	Proxy of {lag time, LOCS, methods}	NO
	Koch (2008)	DIRECT	Proxy of {active developers, lag time, LOCS, methods}	
Activity-based estimation	Amor et al. (2006)	DIRECT	Proxy of {LOCs, commits, emails, bug activity}	NO
	Capiluppi and Izquierdo-Cortázar (2013)	DIRECT	Proxy of {active developers, lag time, LOCs}	NO
People- & activity-based	Robles et al. (2014)	DIRECT + INDIRECT	Proxy of {commits}, surveys	YES
People- & activity-based	Ours	DIRECT + INDIRECT	Proxy of {commits}, surveys	YES

Highlighted are the characteristics of the present paper inception at Robles et al. (2014)) is grounded in both direct and indirect measurements. Furthermore, its results have been validated with developers' responses: this alone is the one aspect that sets our model apart from any other attempt at estimating the effort produced by FOSS developers.

In general, effort estimation models for FOSS projects are based on the data that are collected in the various development activities; these data (especially when they have large variance, or are heterogeneous) need transformation parameters (e.g., logarithmic) that might be needed to normalise a distribution, or to take into account different types of data for the same model. One example of the latter would be devising a transformation parameter to consider all the different activities (emails, blogs, online discussions, etc., as well as the actual code commits) that a developer is engaged into (Amor et al. 2006).

This type of models suffers from a couple of issues: first, it is well known that contributions to FOSS projects are not uniform. A few developers are responsible for a major amount of the work, while a large number of developers contribute with a comparatively small one (Koch and Schneider 2002; Mockus et al. 2002). So, one of the problems when measuring effort of the overall FOSS development process consists in translating the uneven nature of contributions (Mockus et al. 2002) into a consistent estimation model. Inferring the effort devoted by regular and occasional contributors is not easy, and in addition is a source of inaccuracy.

The second issue is that the conversion of that data to effort is far from simple, since it is not possible to exactly determine how much time or effort the developer actually spent on those activities. In general, the information we can gather consists of points in time (i.e., timestamps) where specific actions have occurred (e.g., when the commit was performed, when the e-mail was sent, when a comment to a bug notification was submitted), but not how long these actions actually took (e.g., performing the changes in the commit, reading and writing the e-mail, debugging the software to add information on the bug).

4 Empirical Approach

In this section we introduce the concepts and terminology at its base (4.1), and the two types of input (4.2) that the model uses to evaluate the developers effort. Using an example scenario, we show how to evaluate the threshold θ to identify full-time developers (4.3).

In addition, we discuss how to estimate the error in our model (4.4), and contribute the novel *goodness* performance measure (4.5) that helps in finding the optimal θ. We conclude the section by evaluating the total estimated effort of the example scenario, as well as the relative estimation error (4.6).

4.1 Concepts and Terminology

Our model is build on top of following concepts that are well known in the Software Engineering research literature:

– **Commit:** action by which a developer synchronizes a set of changes to the versioning system repository. A commit is given by a point in time (the timestamp can be obtained from its metadata).

- **Author (or developer, or contributor):** individual who contributes the changes, but may not be the one who performs the commit (which is done by the committer).
- **Committer:** developer who actually performs the commit to the repository, but may have not been the real author of the changes. In our model, we do not consider *committers*, since it would decrease its accuracy.
- **Full-time developer:** Developer who devotes the relative amount of time of a 40 hour-week to the project.[3] In some research literature on FOSS, full-time developers are called "professional" developers as compared to volunteers (Sowe et al. 2008; Steinmacher et al. 2015; Von Krogh et al. 2003). In this paper, we prefer full-time and non-full-time developer, as the latter can be professionals as well, just with a minor participation in the project.
- **Person-Month:** measure of effort. A person-month is observed when a *full-time developer* devotes 1 month of effort to the project.

In addition, for our model we use the following terminology:

- **Period of study n**: timespan (in months) during which the activity (i.e., commits) by developers is aggregated.
- **Threshold** θ: minimum amount of activity (i.e., commits) during the *period of study* by a developer to be considered as a *full-time developer*.
- **Effort** E_d **in a period of study (by developer *d*)**: any developer who performs more commits than θ in a *period of study* of n months will be considered as full-time with n person-months. A developer who committed a commits (with $a < \theta$), will be assigned an effort of $n * \frac{a}{\theta}$ person-months.
- **Effort** E_n **in a period of study n (for the whole project)**: The overall effort E_n is evaluated by summing up the effort in a period of study of all developers (full-time and non-full-time). The formula to obtain the overall effort is as follows:

$$E_n = \sum_{d=1}^{d_{ft}} n + \sum_{d=1}^{d_{nft}} n * \frac{a_d}{\theta}$$

  where d_{ft} is the number of full-time developers, d_{nft} the number of non-full-time developers, and a_d the number of commits that a non-full-time developer has contributed in the period n, smaller than θ.
- **Maximum possible effort M in a period of study (for the whole project)**: This is the effort when all developers are considered full-time, e.g., $\theta = 1$. It can be considered as an upper bound of the total effort, and is calculated as follows:

$$Max(E_n) = \sum_{d=1}^{d_{ft}+d_{nft}} n$$

4.2 Surveying Developers

The first basic input to our model is the self-identification of developers as full- or non-full-time. This is done by means of a short survey that tries to obtain this information.

[3] In some countries there is a different work week, e.g., in the UK it is 35 hours. The model presented can be adjusted if needed.

Table 2 Example scenario to determine θ: developers identify themselves as either full-time (F) or non-full-time (NF). We evaluate their effort by counting the number of their commits

Dev #	Full-time or non-full-time? (*From questionnaires*)	Effort (# commits) (*From data analysis*)
D1	F	12
D2	F	10
D3	F	13
D4	NF	3
D5	NF	11
D6	NF	8
D7	F	10
D8	NF	5

One of the questions of the survey explicitly asks respondents to classify themselves in one or the other category (‘*What do you consider yourself in the project? (full-time, part-time, occasional contributor*’). But as this question can be easily misunderstood, we recommend to use additional, alternative questions that allow to check the input given by developers for consistency. Thus, we asked them how many hours per week they have usually devoted in the last months to the project (‘*On average, how many hours in a week have you spent in the project in the last six months?*; developers could choose among following options: >40 h, 40 h, 30 h, 20 h, 10 h, <5 h).

In addition, on a separate web page of the survey, we also showed developers a chart containing their own personal activity (in number of commits per month). We added a line to the chart with the estimated value of θ, and asked them if we had identified them correctly. Developers could answer in an open text box. Although the aim of this question was to see if developers corroborate our findings, we did not achieve it. This was because (1) our estimated value of θ was far away from the value finally obtained; and (2) developers were not aware of the details of the model and commented on other issues of lesser interest.[4] However, the feedback received clearly indicated that using a period of study of 1 month was not a good choice. For the analysis presented here, we have set 6 months as the most convenient “Period of study n” , as this is the usual time between releases for many projects (Michlmayr et al. 2015). By doing this, all cycles of six months will be affected by the same process (development, feature freeze, release, etc.).

4.3 Determining the Threshold Value θ

Our effort estimation model depends on how accurately we can identify full-time and non-full-time developers. To illustrate its evaluation, we use an exemplary scenario: let’s assume that the development team of a software project is made up of 8 developers, and that they responded to the questionnaire (“*Are you a full-time or a non-full-time developer?*”) as in the second column of Table 2. In the third column, we report the amounts of commits that each worked on during the period **n**, as extracted by parsing the versioning system of their project (Robles et al. 2004).

The first three developers (D1, D2 and D3) self-identified as full-time (F). From the data analysis of their commits, we detected that these developers actually worked on 12, 10 and 13 commits respectively during the period under study (for example, a month). Using the

[4] Details can be found in the reproduction package in an anonymized way

data analysis, and if only these three developers were active, the minimum θ to identify a full-time developer would be 10 commits in the period.

On the other hand, when we consider all the other developers, especially D5, the threshold $\theta = 10$ becomes less effective: D5 classifies his/her work as non-full-time, but he/she is responsible for 11 commits in the period **n**. If we use $\theta = 10$ to detect full-time developers also for D5, we wrongly classify him/her as full-time; if, on the contrary, if we use $\theta = 11$ to identify full-time developers (and to correctly classify D5), we wrongly classify D2 and D7, who both contributed 10 commits, but self-identified themselves as full-time. Determining the value of θ becomes therefore an exercise to minimise the classification errors.

4.4 Determining the Estimation Error

In the example above, we have the typical information retrieval problem with true/false positives and true/false negatives. In particular, and depending on how the threshold θ has been chosen:

- A **true positive** (t_p) means that, using a specific θ, the developer identified as full-time is indeed full-time. Using $\theta = 10$, the example above comprises 4 true positives (e.g., D1, D2, D3 and D7).
- A **false positive** (f_p) means that the developer is flagged as full-time, but s/he is not. Using $\theta = 10$, the example above comprises 1 false positive (e.g., D5).
- A **false negative** (f_n) means that the developer has not been identified as full-time, albeit s/he is. Using $\theta = 10$, the example above does not comprise any false negatives. If we used $\theta = 11$, instead of $\theta = 10$, the example above would contain 2 false negatives (e.g., D2 and D7), but no false positives.
- A **true negative** (t_n) means that a non-full-time contributors has been correctly flagged. Using $\theta = 10$, the example above contains 3 true negatives (e.g, D4, D6 and D8).

The visual representation of these terms is displayed in Fig. 3, using the full-time and non-full-time subsets of developers. Within the realm of information retrieval, we can also evaluate the values of well-known performance measures:

- The **precision** of the model is the ratio of true positives (i.e., the correctly identified full-time developers) and the number of all the identified (correctly or not) full-time developers, or $P_d = \frac{t_p}{t_p+f_p}$.
- The **recall** of the model (or its sensitivity) is the ratio of correctly identified full-time developers and the total number of full-time developers (whether they have been correctly identified or not), as in $R_d = \frac{t_p}{t_p+f_n}$.
- The **accuracy** is the ratio of all the correctly predicted developers, and the overall set of observations: $A_d = \frac{t_p+t_n}{t_p+t_n+f_p+f_n}$.
- The **F-measure** is a combination of both precision and recall, $F_d = 2 * \frac{P_d * R_d}{P_d+R_d}$.

However, these ‘classic’ performance measurements do not take into account the fact that classification errors can compensate each other: depending on the project, the same θ could classify a non-full-time developer as full-time (e.g., a false positive) and at the same time a full-time developer as non-full-time (e.g., a false negative). In this case, we state that the two errors *compensate* each other.

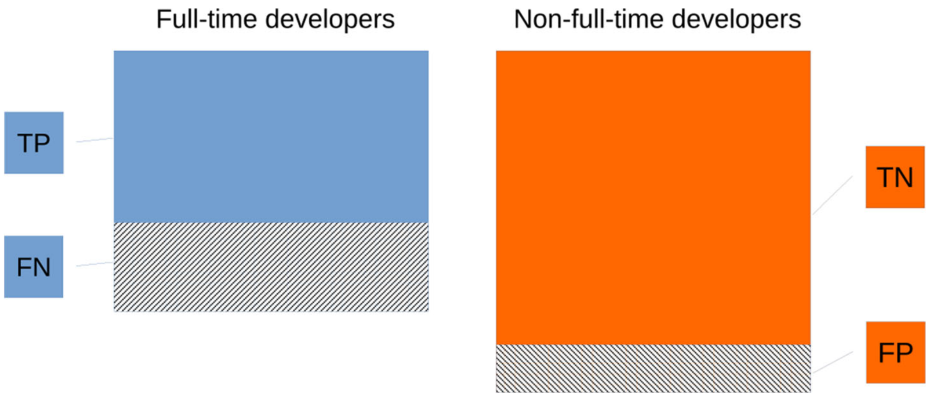


Fig. 3 Visual representation of the tp, fn, fp and tn values for the full-time and non-full-time developer sets

4.5 Compensating the Classification Errors: the *goodness* Measure

The formulation of the *goodness* measurement is designed to explicitly consider the compensation of classification errors: this fact alone sets our model apart from traditional information retrieval techniques. Below we illustrate (i) how the *goodness* performance measure can accommodate the compensation of classification errors, and (ii) how it is pivotal in selecting the most probable threshold θ to detect the full-time developers for our model.

The evaluation of the *goodness* measurement stems from the intrinsic distribution of the developers' work patterns. Depending on their contributions and what type of contributors they consider themselves to be, a larger or smaller number of wrongly classified developers could seriously skew the evaluation of the overall effort. It is also worth noticing that there is currently no mention in the literature of the possibility of classification errors, and the need of error compensation. As it stands, this is the first research work that addresses this open issue. The formula of the *goodness* measure is as follows:

– $\textbf{Goodness} = 1 - \frac{|(t_p+f_p)-(t_p+f_n)|}{t_p+f_n+f_p} = 1 - \frac{|f_p-f_n|}{t_p+f_n+f_p}$

The *goodness* measure depends on the number of *non-compensated* classifications (the numerator, an absolute value, since compensation works both ways), divided by the total number of positives and false negatives. Subtracting this fraction to 1, the *goodness* becomes larger as long as false positives and negatives have been compensated (or if no compensation was necessary). The measure of goodness could also consider the *true negatives* (t_n) instead of the *true positives* (t_p). The reason of not including (t_n) is due to the typical distribution of FOSS developers: a large number of FOSS developers have an extremely low level of contribution. As a result, including a large number of t_n into the definition of goodness would inevitably lower its discriminative power, as compared to using t_p.

Using the same example scenario depicted in Table 2, we report in Table 3 the values of t_p, f_p, f_n and *goodness*, using different values of θ (that is, number of commits). As visible, the relatively high number of false positives (f_p) has a negative impact on the goodness measure, especially when θ is small. As long as more false positives get compensated by false negatives, the goodness measure finds its maximum: we posit that the best values of θ lie where the goodness is maximised (highlighted in Table 3).

Table 3 True positives, false positives and false negatives (as well as the resulting goodness measure) at different values of θ. The data is based on the example from Table 2

θ	t_p	f_p	f_n	*goodness*
1	4	4	0	0.50
2	4	4	0	0.50
3	4	4	0	0.50
4	4	3	0	0.57
5	4	3	0	0.57
6	4	2	0	0.67
7	4	2	0	0.67
8	4	2	0	0.67
9	4	1	0	0.80
10	4	1	0	0.80
11	2	1	2	0.80
12	2	0	2	0.50
13	1	0	3	0.25

4.6 Total Estimated Effort and Estimation Error

The identification of θ allows to evaluate the effort by developers for a certain period, or for the entire project. In the latter case, an adjustment of the threshold might become necessary. In the example scenario that was discussed above (see Tables 2 and 3), the threshold that maximises the *goodness* measure lies between 9 and 11. If we choose $\theta = 10$, the overall effort by developers is 6.6 PM, as in the sum of:

- 5 PM (1 PM each for the D1, D2, D3, D5 and D7 developers)[5]
- $\frac{3}{10}$ PM (for the D4 developer)
- $\frac{8}{10}$ PM (for the D6 developer)
- $\frac{5}{10}$ PM (for the D8 developer)

Choosing a different threshold, but still within the values that maximise the *goodness* function (i.e., the shaded rows in Table 3), has a small effect on the overall effort, that sums up to 6.78 PM in case of $\theta = 9$, and 6.27 PM in case of $\theta = 11$. The estimation errors grow larger as long as one moves away from the ‘safe’ range of θ. The sum to evaluate the overall effort, as presented above for the example scenario, is the same that was used to evaluate the effort of the projects under study.

Table 4 lists the estimation errors (in percentage) if a different θ is selected, as compared to the $\theta = 10$ that maximises the *goodness* of our example scenario.

5 Deployment of the Model

The model presented above is deployed using 6 large FOSS projects that have a different degree of participation of commercial companies. The selection of these projects is based

[5]Full-time developers are assumed to work 1 PM every month, even if they work more than the chosen θ. We call this aspect the *saturation* of a developer’s effort.

Table 4 Values of global effort (in PM) by developers, and the percentages of error when choosing a different threshold. The data is based on the example from Table 2

θ	Effort (PM)	Estimation error
1	8.00	21.21%
2	8.00	21.21%
3	8.00	21.21%
4	7.75	17.42%
5	7.60	15.15%
6	7.33	11.11%
7	7.14	8.23%
8	7.00	6.06%
9	6.78	2.69%
10	6.60	–
11	6.27	−4.96%
12	5.92	−10.35%
13	5.54	−16.08%

on convenience sampling: two of the authors of this paper have a direct contact with the communities under study.

All selected projects follow common practices found nowadays in large FOSS software development projects: they have code review in process (using an external tool like Gerrit,[6] except Ceph that uses GitHub's pull-request driven process) and recommend to do commit squashing (Kalliamvakou et al. 2014). Squashing a commit refers to move changes introduced in a commit into its parent, to end up with one commit instead of two. If you repeat this process multiple times, n commits can be reduced (i.e., 'squashed') to a single one. It is common practice in many FOSS projects to squash all commits that belong to a change, after it is reviewed and accepted (Kononenko et al. 2018).

Summary statistics are provided in Table 5, with an additional column stating whether it is a consortium with commercial participation ('consortium'), or an organization (commercial or not) driving the project ('organization').

OpenStack OpenStack is a software project to build a SaaS (software as a service) platform. Over 200 companies have become involved in the project. AMD, Brocade Communications Systems, Canonical, Cisco, Dell, Ericsson, Groupe Bull, HP, IBM, InkTank, Intel, NEC, Rackspace Hosting, Red Hat, SUSE Linux, VMware Yahoo! and many others have been participating in OpenStack.

Moodle Moodle is an FOSS project that implements a PHP-based learning management system: it started as a volunteer project in 2002, but it has since evolved into a company-driven FOSS project. Moodle HQ is the company that keeps developing it, and there is an estimated 80 international companies that help in its development.

[6]https://www.gerritcodereview.com/

Table 5 Main parameters of the studied projects, January 2014

Project	Type	First release	# Authors (distinct)	# Commits (w/o bots)	SLOC
Ceph	Consortium	07/2008 (v0.2)	158	37,254	$418K$
Linux	Consortium	08/1991	3,665	540,263	$>10M$
Moodle	Organization	08/2002 (v1.0)	174	52,523	$1.28M$
OpenStack	Consortium	10/2010	1,410	68,587	$1.65M$
WebKit	Consortium	06/2005	690	107,471	$5.0M$
MediaWiki	Organization	01/2001	605	357,786	$340K$

Ceph Ceph is a freely available storage platform,[7] and it also started as a volunteer based FOSS project. After being merged into the Linux kernel in 2010, it was incorporated by RedHat that is now responsible for its in-house development. Various large organisations contribute to its development, from Canonical to Cisco, Fujitsu and Intel.

Linux Linux represents the most well-known success story for FOSS projects. Started as a volunteer project back in 1991, Linux grew exponentially, although it is still a community-driven project. The amount of contributions from commercial companies have slowly outgrown those of the individual developers: the latter account now for only 10% of the developers of the operating system.

WebKit WebKit is an FOSS project started by Apple, although it started as a fork of an existing KDE project. It implements the engine that sits behind the Safari web browser. KDE, as well as other FOSS communities, still participates in the development of WebKit, but large commercial organisations (Google, Nokia) have also contributed to it. In April 2013, Google forked the WebKit creating to Blink. Since then it is mainly a project backed by Apple.

MediaWiki Wikimedia is the non-profit organisation that supports the growth of the Wikipedia online encyclopedia. Wikimedia is directly in charge of the MediaWiki FOSS project,[8] that implements the wiki engine that allows pages to be added, modified and contributed to by anyone.

6 Data Gathering

As mentioned above, the estimation model that we propose in this paper obtains data from two sources: (i) the VCS of the project under study, and (ii) the questionnaire data that was sent to the developers identified in the VCS of each project. Below we describe the steps that were performed to gather the data from each source.

[7] https://github.com/ceph/ceph

[8] https://phabricator.wikimedia.org/source/mediawiki/

Table 6 Summary of emails sent to the developers of the sampled projects, and ratio of completed questionnaires

	Sent	Received	Ratio
Ceph	158	24	15.19%
Linux	3,665	652	17.79%
Moodle	174	42	24.14%
OpenStack	1,407	131	9.31%
WebKit	690	85	12.32%
MediaWiki	605	94	15.54%
Total	6,699	1,028	15.35%

6.1 Versioning System Data

The information from the VCS is obtained by means of Dueñas et al. (2018), which is part of the MetricsGrimoire toolset.[9] Perceval retrieves development information and metadata from repositories[10] of the project, and provides it as JSON files, ready for analysis.

The data obtained from the VCS repositories was subject to two cleaning operations: one related to the extraction and reconciliation of *duplicated* developer identities (Robles and Gonzalez-Barahona 2005; Wiese et al. 2016); and one related to the exclusion of commits generated by *automated bots*. We performed those operations for all the projects in the sample.

Using OpenStack as an example, from the analysis of its commits we observed that there are 1,840 distinct author identifiers in the VCS of OpenStack. After applying a merging algorithm (Kouters et al. 2012), manual inspection and feedback from the community, 1,410 distinct real authors have been identified, once duplicate IDs were merged. Additionally, the total number of commits is almost 90,000, although many of these have been done automatically by bots. The committers identified as bots, as well as their commits, were excluded from the analysis. Bots are responsible for 21,284 commits, almost 25% of all commits in the project: those bots and their commits were excluded from the data used to evaluate θ.

OpenStack, as many other FOSS projects, has an uneven contribution pattern where a small group of developers have authored a major part of the project. In summary, 80% of the commits have been authored by slightly less than 8% of the authors, while 90% of the commits correspond to about 17% of all the authors. In addition, as the corporate involvement in OpenStack is significant, this should allow us to identify full-time developers from companies: the OpenStack Foundation estimates that around 250 developers work professionally in the project.

At the end of this data analysis, for each project in our sample we obtained the number of monthly commits per developer.

6.2 Online Survey of Developers Effort

To obtain data of the time commitment of the FOSS developers in the sampled projects, we designed an on-line survey (as described in Section 4.2). We obtained the e-mail addresses

[9]http://metricsgrimoire.github.io

[10]The Ceph, Linux, Moodle, MediaWiki and OpenStack datasets were retrieved from git repositories. The WebKit dataset was retrieved from a Subversion repository.

of the developers from their authorship information in the VCS, and sent them an e-mail with an invitation to participate.

Gathering all the distinct developers from the 6 sampled FOSS projects, an overall 6,700 personalized mails were sent, and the survey was answered by a grand total of 1,028 respondents. We removed 55 survey responses because they were empty or we could see from the answer in the free text box that the respondent had misunderstood/misinterpreted the questions (e.g., there was a developer referring to his contributions to the LibreOffice project, which is not part of the analysed sample of projects). We also amended 32 surveys (e.g., some respondents stated to be professional developers hired by a company and devoted 40 or more hours a week to the project, but had left empty their status, which we set to "full time").

Table 6 reports the number of responses (and the ratio) obtained from the developers of the sampled projects.

7 Model Deployment—OpenStack

This section reports the results of our analysis: we first present an in-depth analysis of the OpenStack project, and later an overall view of all the projects in the sample.

7.1 Identification of Full-Time Developers

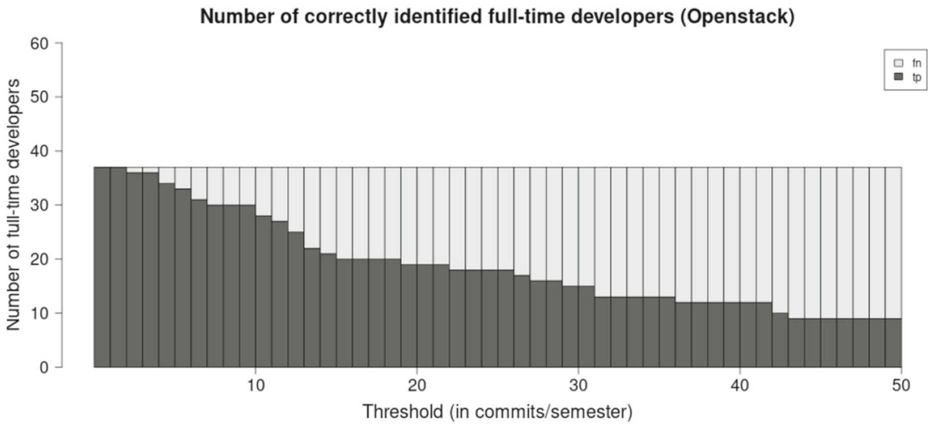
In this section we describe how the full-time developers were identified within the OpenStack contributors, by using the threshold θ and calibrating the results of the questionnaire. The same approach was followed for the other projects in the sample.

As a first step we analysed the questionnaires: we gathered that 37 developers identify themselves as full-time, while 54 developers consider themselves as non-full-time. As a second step, we extracted the number of commits of each developer, in the six months prior the questionnaire. Given those two sources of information, we need to answer the question: what is the *minimum* number of commits that more precisely separates full-time from non-full-time developers?

To answer that question, the two plots in Fig. 4 classify the developers as following (at a certain level of θ):

– *true positives*: number of full-time OpenStack developers that are classified as full-timers;
– *false negatives*: number of full-time OpenStack developers wrongly classified as non-full-timers;
– *false positives*: number of non-full-time OpenStack developers that were classified as full-timers;
– *true negatives*: number of non-full-time OpenStack developers correctly classified as non-full-timers.

Full-Time Developers When θ is set to 1, developers who contribute at least one commit in a six-month time-span will be considered as full-time by our algorithm. From Fig. 4, the precision to detect a full-time developer with $\theta = 1$ is maximum: this means that each of the 37 self-identified full-time developer has contributed at least one commit during the allocated time-span (i.e., six months).



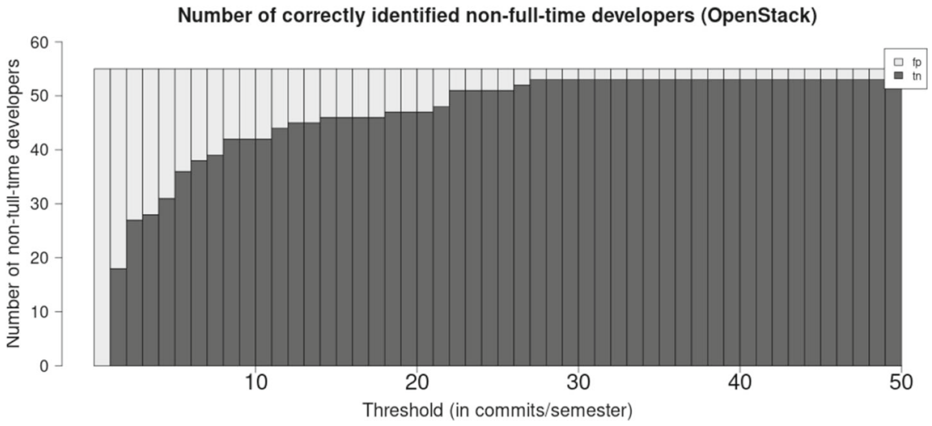


Fig. 4 (Top) True positives (tp) and false negatives (fn) for full-time developer identification for several values of θ (OpenStack project). (Bottom) False positives (fp) and true negatives (tn) for non-full-time developer identification for several values of θ. The dimension of the vertical axis has been maintained to ease the comparison of the size of both populations

The number of true positives (tp) decreases once θ gets larger: for example, out of the 37 self-identified full-timers, only 30 developers contribute at least 8 commits every six months (i.e., $\theta = 8$). The remaining 7 developers are classified as false negatives: even if they are full-time developers, our approach considers them as non-full-timers. In this case, the precision of using a threshold $\theta = 8$ to detect full-time developers is $\frac{30}{37}$.

Non-full-time Developers The same approach applies to non-full-time developers, but based on false positives and true negatives (see bottom plot of Fig. 4): setting $\theta = 1$, all the self-identified non-full-time developers are false positives, and our algorithm considers them as full-timers.

As seen in the bottom plot of the same Figure, also the number of true negatives (tn) increases, as long as θ gets larger. At $\theta = 8$, for example, 38 out of 54 non-full-time developers are correctly identified as non-full-timers (i.e., true negatives); while 6 out of 54 non-full-time developers are still considered as full-timers, at that level of θ (i.e., false positives).

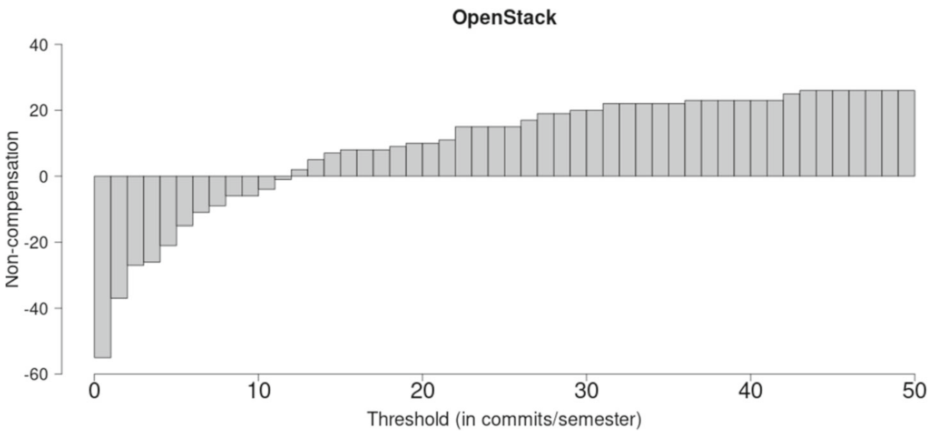


Fig. 5 Compensation between false positives and false negatives for values of t. Negatives values of the y axis indicate false positives that do not get compensated by false negatives, while positive values of the y axis indicate false negatives not compensated by false positives

7.2 Compensation of Error and the Goodness Metric

Figure 5 displays graphically the compensation pattern in the OpenStack project, and for different values of θ. The number of wrongly classified non-full-time developers are subtracted from the number of wrongly classified full-time developers.

As it can be seen, for $\theta = 1$ no full-time developer has been incorrectly classified, but all non-full-time developers have. No compensation occurs then. However, as θ increases, the error of incorrectly classifying full-time developers is compensated by the error of incorrectly classifying non-full-time developers. The value for which the number of non-compensated errors is minimum corresponds to t = 12, where the difference is 1 (i.e., there is one false negative more than false positives).

Figure 6 shows that, for the OpenStack project, the value of goodness peaks at $\theta = 12$ (0.979). Thus, the error for our effort estimation should be smaller for $\theta = 12$, due to the effect of compensation that the goodness factor recognizes.

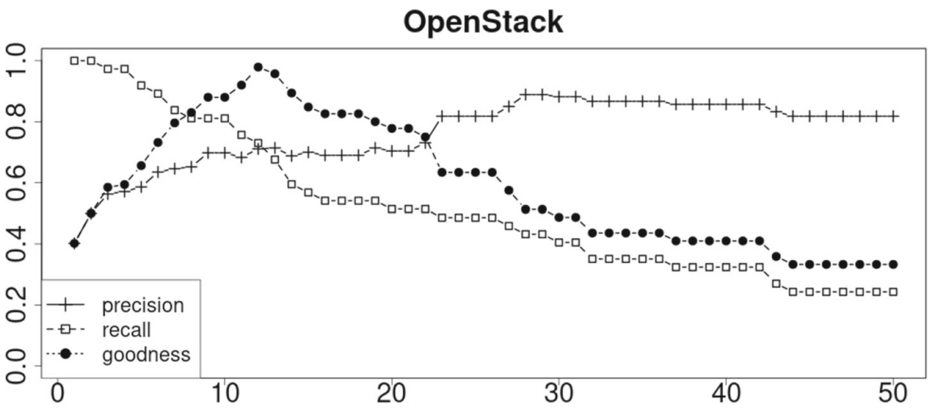


Fig. 6 Relevance of results (for many threshold values) for the OpenStack project

7.3 Effort Estimation

Having obtained feedback from the OpenStack developers through the survey, the identification of full-time developers via θ becomes more precise. Depending on the θ chosen, the effort estimation will provide more or less accurate values.

Table 7 shows the results of the estimated effort in OpenStack, and using several values of θ. Considering the global effort through the whole lifespan of the project, we can see that -as expected- the number of person-months decreases while the threshold increases: less and less developers are considered as full-time, thus pushing the effort down. The upper bound of effort for OpenStack, where a single commit in a semester would have been coded as 6 person-months, results in 17,400 person-months.

According to the analysis, the best values for θ are between 9 and 12. The highlighted value of $\theta = 12$ in Table 7 is what minimizes the error of considering developers as full-time using the feedback from the OpenStack developers. For the $\theta = 12$ scenario, the estimation

Table 7 Estimated total effort (in PM) for the OpenStack project, given per threshold value (column θ). Estimated effort values for all semesters. The darker the background, the higher the value of goodness ($white < 0.8 \leq lightblue < 0.9 \leq greenblue \leq 1.0$)

θ	Effort (total, PM)	Effort (semester, PM)						
		10s2	11s1	11s2	12s1	12s2	13s1	13s2
1	17,400	426	816	1,242	1,890	2,742	4,368	5,916
2	15,255	420	813	1,170	1,647	2,385	3,783	5,037
3	13,804	402	798	1,110	1,486	2,140	3,384	4,484
4	12,747	389	785	1,056	1,383	1,964	3,074	4,098
5	11,891	378	768	1,008	1,295	1,830	2,821	3,791
6	11,213	370	757	964	1,224	1,726	2,618	3,554
7	10,629	362	743	927	1,164	1,635	2,451	3,346
8	10,128	355	732	896	1,112	1,560	2,310	3,164
9	9,683	345	721	866	1,065	1,493	2,189	3,003
10	9,298	337	711	841	1,025	1,434	2,086	2,864
11	8,957	330	700	819	987	1,381	1,997	2,743
12	8,655	324	690	800	955	1,334	1,919	2,634
13	8,370	318	680	782	924	1,289	1,847	2,531
14	8,112	313	672	767	896	1,247	1,781	2,437
15	7,876	308	663	753	872	1,208	1,721	2,351
16	7,662	303	656	740	850	1,173	1,666	2,275
17	7,466	298	648	729	830	1,140	1,615	2,206
18	7,284	294	641	719	812	1,107	1,568	2,142
19	7,109	291	634	708	794	1,077	1,522	2,083
20	6,945	288	628	698	777	1,048	1,481	2,025

of the amount of effort that was invested in the OpenStack project lies around 9,000 person-months (750 person-years). This can be considered as the most accurate, and it reaches only half of the estimation value produced in the upper bound estimation (with $\theta = 1$).

The table also provides information for every semester. As can be observed, the effort devoted to the OpenStack project is increasing with time, with a maximum for all the thresholds in the second semester of 2013. If we take the value of 12 as the threshold, 2,634 person-months have been worked during the last six months. This implies that we estimate that the actual effort on OpenStack has been around 440 person-months during the last six months. When asked about these figures, OpenStack members confirmed that they sound reasonable as the estimation of the OpenStack foundation is that currently over 250 professional developers work on the project hired by companies.[11]

On a side note, it should be noted that the ratio between the effort estimated as the upper bound ($\theta = 1$) and the one with $\theta = 12$ is steadily increasing. In the second semester of 2010 the ratio was 1.31, while for the second semester of 2013 it has grown to 2.25. This is because the number of non-full-time contributors has grown as well. Thus we can see how much the "minor contributors" in the project affect the estimations given by our model. While the error (noise) introduced by these small contributions is included using the upper bound, higher values of θ filter it out. The result is not only a more realistic estimation, but also an estimation where the more error-prone assumptions (i.e., non full-time developers) have a smaller weight in the total effort.

8 Further Model Deployment

The estimation model deployed in the OpenStack project was also evaluated for the other systems composing our sample. In this section, we perform the model deployment for 5 other systems to

(i) identify their full-time and non-full-time developers (8.1),
(ii) extract the values of θ where compensation happens more frequently for each system (8.2), and
(iii) find the goodness value that maximises the accuracy of the (project-specific) model (8.3).

Table 8 shows the summary of the calculations of both the θ value, and the corresponding effort for the other projects in the sample. In the next sections we describe how the estimation model was deployed in each of the additional systems in the sample.

8.1 Identification of Full-Time Developers

The approach that was presented for OpenStack was repeated for all the systems. In all the cases, when θ increases, the more precise it becomes to identify the non-full time developers, based to their responses to the questionnaires. Similarly, when θ grows, the accuracy at identifying full-time developers decreases for all but one project. Ceph (second from bottom plots in Fig. 7) is an expected variation to this trend: all its self-identified full-time devel-

[11]This type of informal feedback was gathered only from OpenStack developers, but not repeated for the other projects in the sample.

Table 8 Estimated total effort (in PM) for all projects from 2010s2-2013s2 (3.5 years), given per threshold value (column θ). The darker the background, the higher the value of goodness ($white < 0.8 \leq lightblue < 0.9 \leq greenblue \leq 1.0$)

θ	OStack	t	Moodle	t	WebKit	t	Linux	t	Ceph	t	MWiki
1	17,400	1	2,688	1	11,976	1	57,390	1	1,056	1	5,508
...	...	...	...	...	...	...	...	...	...	...	...
6		8		11		11	32,156	18	574	30	3,348
7		9		12		12	31,157	19	567	31	3,324
8	10,128	10		13	8,022	13	30,240	20	560	32	3,302
9	9,683	11	1,846	14	7,887	14	29,393	21	554	33	3,279
10	9,298	12	1,803	15	7,753	15	28,602	22	548	34	3,258
11	8,957	13	1,765	16	7,619	16	27,865	23	542	35	3,237
12	8,655	14	1,731	17	7,493	17	27,173	24	536	36	3,217
13	8,370	15	1,698	18	7,372	18	26,524	25	530	37	3,198
14	8,112	16	1,666	19	7,257	19	25,914	26	525	38	3,179
15	7,876	17	1,636	20	7,146	20	25,334	27	520	39	3,161
16	7,662	18	1,609	21	7,037	21	24,781	28	516	40	3,142
17	7,466	19	1,584	22	6,933	22	24,256	29	511	41	3,124
18	7,284	20	1,561	23	6,833	23	23,761	30	506	42	3,106
19		21		24	6,737	24	23,290	31	501	43	3,089
20		22		25	6,646	25	22,839	32	496	44	3,072
21		23		26	6,558	26	22,411	33	492	45	3,056
22		24		27	6,472	27	22,002	34	488	46	3,040
23		25		28	6,390	28	21,609	35	484	47	3,025

opers have an extremely high number of commits in the six-month time-frame, higher than the 50 commits depicted in the figure.

8.2 Compensation

Similarly to what achieved in Section 7.2, the compensation levels were evaluated to identify the minimum between false negatives and false positives. The plots of the compensation values, per project, are available in Fig. 8. Given a compensation plot, one can obtain the value of θ for each project, and more precisely where the compensation is closest to zero. In the extreme case of the Ceph project, there are no false negatives, and only false positives, therefore the compensation is negative for any threshold value.

8.3 Goodness (and Threshold Values Obtained)

In Fig. 9 we have plotted similar metrics to Fig. 6 (i.e., precision, recall and goodness) for the rest of the analysed systems. The θ value for a project is selected in correspondence of the maximum value of the goodness: as demonstrated above, that threshold value minimizes the estimation error.

We collected the values of θ, per project, in Table 9. In the Discussion (in particular in Section 9.5), we elaborate about why projects might have different values of θ, and how

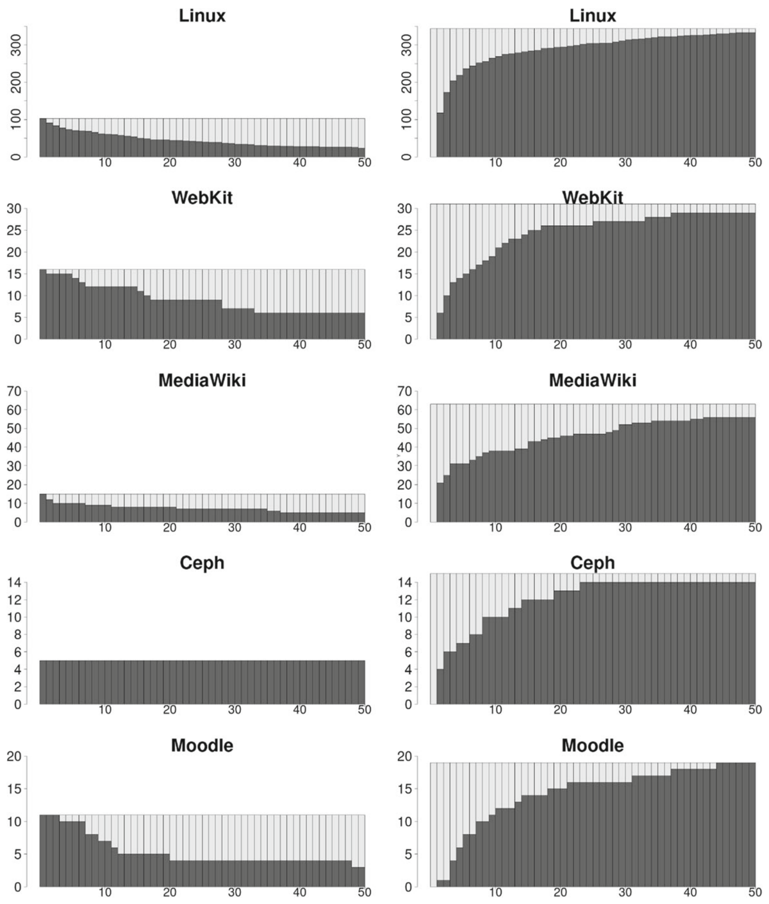


Fig. 7 Correct identification of full-time (l) and non-full-time (r) developers depending on the threshold value (x-axis). The darker shade gives those developers that have been correctly identified, while the lighter one is for those that were not. The vertical axis has been kept the same for each project in order to ease visual comparison of the population of each group

this knowledge can be leveraged to extend our model to other projects, and without having to directly survey their developers.

9 Discussion

In this section we discuss the limitations, implications and further possibilities of our research. We will start presenting the threats to validity (9.1), and then elaborate further on the sources of error of our model (9.2). Finally, possible impact for practitioners and managers (9.3), the representativeness of our survey (9.4) and the possibilities of generalizing to other FOSS projects (9.5) are presented and discussed.

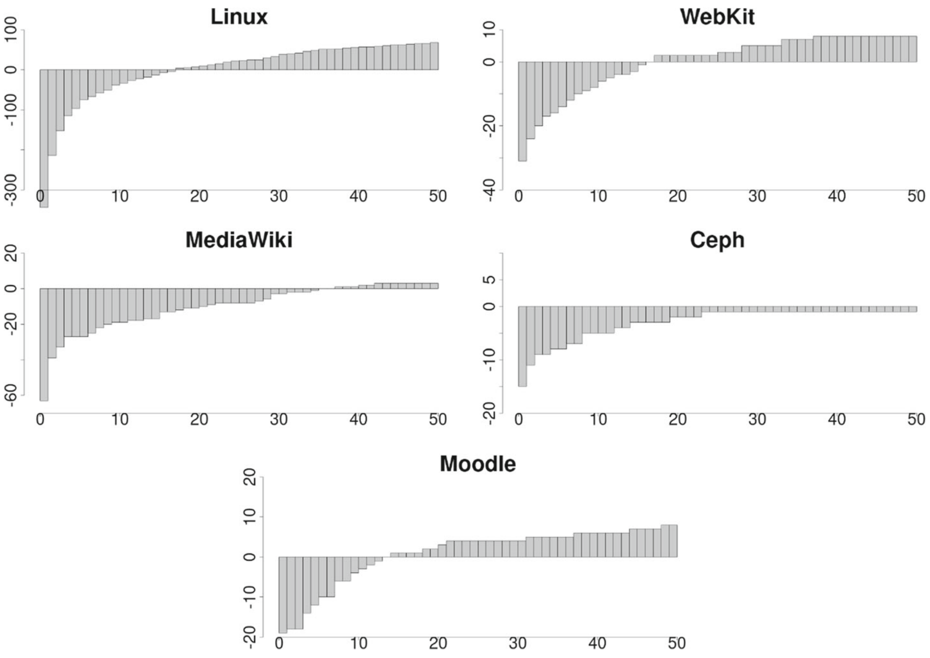


Fig. 8 Compensation between false positives and false negatives for values of θ. Y-axis is given in number of developers; the x-axis corresponds to the θ value. Negatives values of the y axis indicate false positives that do not get compensated by false negatives, while positive values of the y axis indicate false negatives not compensated by false positives. Note that the magnitude of the vertical axis is different for each plot

9.1 Threats to Validity

All empirical studies, such as this one, are subject to threats to validity. Here, we point the most relevant ones and discuss how to mitigate or control these threats if possible.

Conceptual Assumptions and assertions have been done. We have tried to make them explicit, as for instance the assumptions that a person-month equates to 40h/week. Some of them may require further support, but this does not invalidate the plausibility of the argument.

Internal Having only considered one of the many sources of activity (and data) may result in our estimation model being not accurate. Further work is expected to be of broad interest and supportive of the claims presented.

External Originally, OpenStack was chosen as the illustrative case study. We consider that this project is representative of many FOSS projects, especially those where many industry players cooperate. There may be certain practices that could only be found for OpenStack: for instance, their review process produces large (and few) commits, so values of θ should be selected differently for other FOSS projects. We have applied the model to other five FOSS projects, and the results are similar.

The model has been conceived to work well when participation follows a power-law and the number of full-time developers that can be easily identified because their activity is much higher then by the rest of developers—else we could have many false negatives (i.e.,

Fig. 9 Values of precision, recall and goodness relative to θ. Legend: Precision (✢), Recall (❏), Goodness (●). Note that in Ceph the values of goodness coincide with the ones of precision

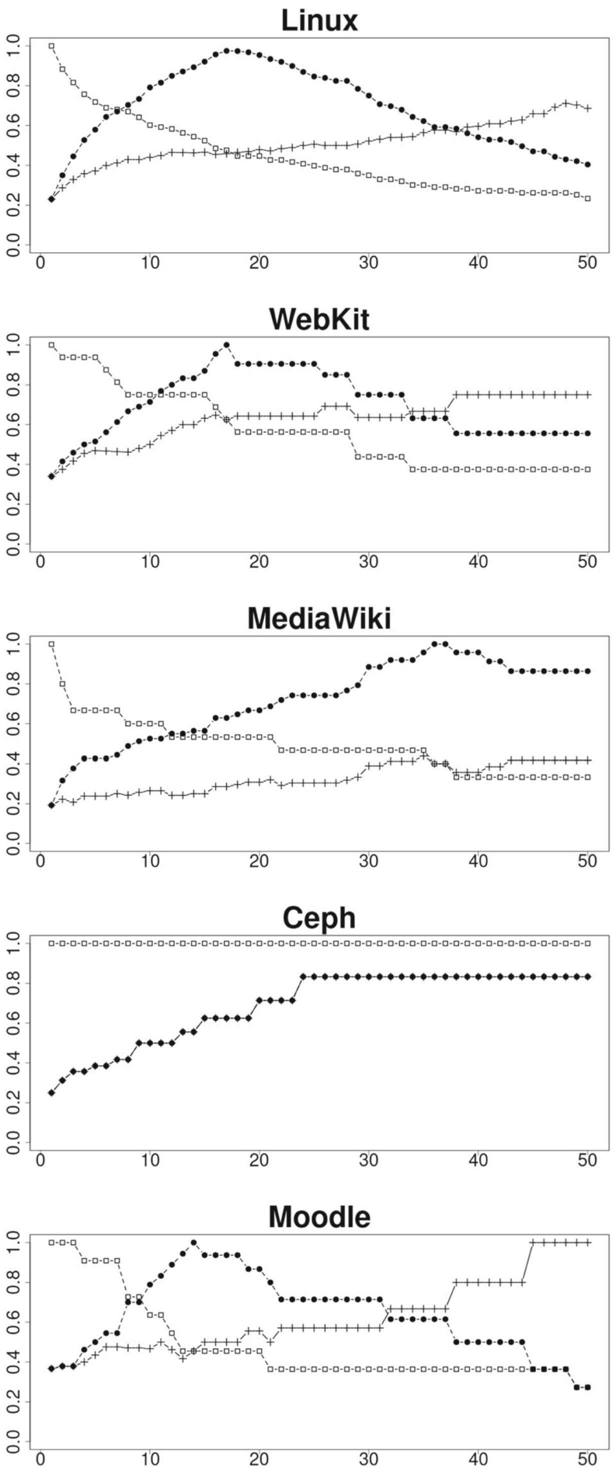


Table 9 Values of θ for all the projects analysed

Project	θ
OpenStack	12
Moodle	14
Linux	17
WebKit	[16–19]
Ceph	24
MediaWiki	[36–37]

many full-time developers not identified as such). What we have assumed is usually the case in FOSS development, but has not to be true for all FOSS projects. We have applied our model to large project (with hundreds of committers), but cannot ensure it will offer meaningful results for other FOSS projects, especially small ones.

The selection of the further 5 systems was based on convenience sampling: as said before, the communities of the projects selected were more easily reached, due to two of the authors of the being paper in direct contact with all of them. Although the sample might be unrepresentative, the systems were only used to deploy the model, not to show its absolute value for any other system. In any case, the generalization of the model to other FOSS projects remains to be investigated.

Construct A replication package,[12] following the guidelines in González-Barahona and Robles (2012), is offered to others to reproduce our findings. The replication package contains all scripts and public information, but not all the results of the surveys, as personal information was collected. Aggregated and anonymized information from the surveys is available.

It is important to note that it is not possible to establish a unique *ground truth* for the total effort made by developers in FOSS projects. This is because FOSS projects do not typically make use of timesheets to track the actual effort of developers. Therefore, a 'traditional' measure of effort is not clearly computable.

The measures that we propose do not help in establishing what is the ground truth, but rather to minimise the error in the evaluation of the estimated effort based on actual input by the developers of the project.

A larger threat to construct validity was identified, based on the representativeness of the survey sample. It should be, however, noted that for determining the best value of θ, we do not need a representative sample of the *whole* project population, but just a representative sample that allows us to effectively discriminate between full-time and non-full-time developers. We thus require basically data that is representative of (very) active developers to perform this analysis. With a very high probability, all the developers with a low activity (i.e., few commits) are non-full-time developers—and our model will accurately label them as such. We check if this affects our results in more detail in Section 9.4.

9.2 On the Sources of Estimation Error

Our effort estimation model is based on the identification of full-time developers. Once identified, full-timers are assigned the maximum possible effort (1PM every month):

[12]http://gsyc.urjc.es/~grex/repro/2022-emse-effort-estimation

non-full-timers are only assigned a share of it, depending on the number of their commits and the θ. In this situation, the sources of error are the following:

1. Wrong identification of full-time developers (false negatives) or non-full-time developers (false positives).
2. Wrong assignment of effort to non-full-timers.

Note that we assume that the assignment of effort to full-timers is error-free, as we argue that by definition the effort of a full-time developer is 1PM. We argue below (Sections 9.2.1 and 9.2.2) why both sources of error do not pose a major threat to the validity of our model and its results.

9.2.1 Wrong Identification

In our model false negatives (FN) refer to full-time developers that we have not identified as such. We have included a short comment in the threats to validity (Section 9.1), stating that results may not hold if there are many false negatives in a certain project.

However, we consider it safe to assume that the number of false negatives will be small in the general case of FOSS development. The reasons for this are as follows:

1. The number of full-time developers should be (very) low compared to the amount of total participants in a FOSS project. This is because the distribution of participation is very skewed, as in a power-law or Rayleigh-type curve (as reported in the research literature (Koch 2008; Sowe et al. 2008)), in FOSS. So, in general, full-time developers will have profiles with high commit activity, well above the threshold value, that is, the lowest bound of activity for full-timers.
2. A false negative might be compensated by a false positive. Some full-time developers might have a low activity, below θ. When they are below θ, compensation (i.e., non-full-time developers that are incorrectly identified as full-time (false positives)) softens this situation. In other words, because we have chosen θ where goodness is highest (i.e., the one that maximizes compensation), the number of FNs that are not compensated should be the lowest possible.
3. If not compensated, the impact of the error is small, as an aggregate. Having a FN that is not compensated means that a developer is assigned a fraction of the effort depending on the number of commits and the threshold. So, if for OpenStack θ is 12 commits in a 6-month period, and we have a full-time developer with 6 commits in that period, we would have a false negative and this developer would be accounted with an effort of 6/12 * 6PM = 3 PM instead of with 6PM. This would imply an error of 100% at the *individual* level. But our estimation model is based on aggregation of all developers, so even if the error for one individual developer might be high, the fact that it is low for many other developers will pay off for the whole project.

A similar reasoning applies to false positives (FP) in our model, i.e., to non-full-time developers that we have not identified as such. Even if the population of non-full-time developers was large, using the highest goodness value ensures highest compensation and, thus, the lowest aggregated number of FPs. In this case, the individual error introduced would be because a non-full-time developer is assigned 6PMs in a period of six months, when the *real* effort is less (e.g., 3PMs). Again, the error at the *individual* level might be large, but its impact on the whole project will be small.

For the six projects under study in this paper, the number of non-compensated FNs or FPs are: 1 developer for OpenStack, Linux and Ceph and 0 developers for WebKit, MediaWiki and Moodle. This can be seen from the Figs. 5 and 8, This strengthens our argument: the impact of an incorrect identification of developers in our model is very low, even negligible in large FOSS projects.

9.2.2 Wrong Assignment of Effort to Non-full-timers

Another source of error may come with the effort estimation assigned to non-full-time developers. In our model we assign non-full-time developers the fraction of commits/threshold (i.e., if they did 2 commits in 6 months and θ is 12, as in OpenStack, then they will be assigned an effort of 1/6 * 6 = 1 PM for the 6 months).

The rationale for this estimation is easy to understand: these developers are not directly estimated for their effort, but as its fraction to the minimum activity that is required to be considered as a full-time developer. In other words, if a non-full-time developer has authored 2 commits and we expect full-time developers to submit at least 12 commits ($\theta = 12$), our model will assign this developer 1/6 of the minimum effort needed to be considered a full-time developer.

9.3 Impact

Based on the models of FOSS development, the analysis that was presented, and its findings, can have a direct impact on the decisions taken by practitioners and managers. The estimation of past efforts, with the addition of an understanding of what is already implemented, what is missing, and what needs to be maintained, can help to plan for the future.

If managers can estimate past effort, they can also evaluate the effort that is likely to be needed in the near future. This is particularly important for those projects where many developers, from different companies, contribute with very different levels of dedication, and where estimating past effort is very difficult. In those contexts, the impact of a company leaving the project can be estimated; in a similar way, the impact of a new company allocating its developers at different levels of effort (e.g, full-time, part-time, occasional, etc.). All these levels of developer engagement can be re-estimated periodically, so that managers can account from divergences from the plan.

In particular, from our experience the focus on the estimation of past effort in FOSS is important usually in two situations:

1. When a company is evaluating the overall effort put in the development of a FOSS project, because they have a direct interest in assuming all or part of that cost. This is the case, for example, when the software is strategic to a company, and they want to evaluate how much effort was put in it in the past. This evaluation will be then used as an estimation of future effort, in the short and medium term, and considering to hire a part of the developing team: knowing the past effort gives also an evaluation of which fraction of the development effort they are hiring.
2. When some FOSS is developed mainly by developers hired by companies, usually in the context of some Foundation or managing board, where companies involved decide about the resources they allocate to the project. In those case, having estimations about past efforts, and allocations of those efforts to companies (and to developers hired by those companies) is fundamental for the negotiations where new companies want to gain influence in the project, a seat in the board, or just negotiate how to move forward features that are important for them. We have specifically observed this kind of

negotiations in projects such as Xen and OpenStack, and in some projects under the Linux Foundation umbrella.

In addition to this, when companies participate to FOSS endeavours, they are fully aware of the effort produced by their own developers, and how they engage with the FOSS projects that they have an interest in. Having a better estimate of how much effort (overall) has been produced into a FOSS project would make a clearer case for a company's Return-on-Investment (RoI). As companies are usually aware of the effort they put into the project, they can compare this with the total effort obtaining a ratio of how much of this effort comes from the community. The returned effort is what a company gains for its stake over a FOSS, as contributed by the community around it. That gain would necessarily influence a company's strategic decision of investing its own developer efforts (or procure and pay for development efforts from other specialist companies) into the FOSS project, and at what level. Even more importantly, in case of other commercial enterprises participating to the same development, a company would be able to tailor and adapt their own input to the project, and based on other companies' behaviour.

9.4 Representativeness of the Survey Sample

The rate of responses obtained in the questionnaires to the developers is around 25% at best (in the case of the Moodle project). This means that most developers have not responded to the questionnaires, and the representativeness of the sample could be put in question. Limited to the OpenStack project, Fig. 10 shows two box-plots with the analysis of the developers who responded to the survey (left) and all the active developers (right). The measured activity in number of commits considers the six months preceding the survey.

In order to check whether the two distributions of commits come from the same population, we applied the Kolmogorov-Smirnov's test, a non-parametric test that considering two samples, evaluates the null hypothesis H_0: *are the two samples extracted from the same population?*. We considered various activity levels (e.g., developers committing 0 or more commits; developers committing 1 or more commits and so on) and tested their distribution against the overall active population of developers, at the same activity level. The significance level for each of those statistical tests was set to a standard $\alpha = 0.5$: Table 10 summarises the p-values (last column) along with other attributes.

As visible from Table 10, we reject the base H_0 at all activity levels up to 4 commits per period: hence, between 1 and 4 commits, we *can* reject the hypothesis that the surveyed developers represent the overall population. With higher commit activities (5 commits or more), the H_0 cannot be rejected at the $\alpha = 0.05$ level: from 5 commits up, we *cannot* reject the hypothesis that the surveyed developers represent the overall population. As the values of activity increase, the surveyed population becomes more representative of the project: since our model is based on activity and the classification is performed only on active developers, these results give a stronger support to our model.

The same analysis was extended to the other sampled systems, and summarised in the boxplots of Fig. 11. From top left, the activity of the responding developers (in the 6 months preceding the questionnaire) for the Linux, WebKit, MediaWiki, Ceph and Moodle systems was compared with the activity of all developers in each project. Similarly to OpenStack, we *cannot* reject the hypothesis that the surveyed developers of those projects represent the overall population, for activities of more than 3 commits.

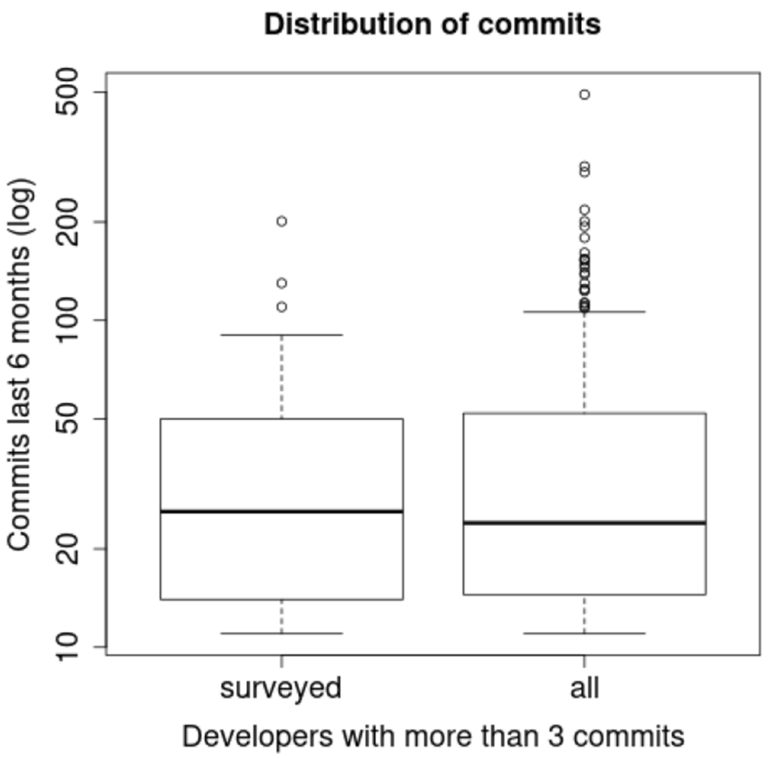


Fig. 10 Boxplot with the activity (in number of commits during the last 6 months before the survey) for the active developers surveyed in the OpenStack project (left) and for all the active developers (right) in the last 6 months before the survey

Table 10 Summary of population measures. Several populations have been selected, depending on a minimum number of commits. *D* and *p-value* as given by the Two-sample Kolmogorov-Smirnov test

Commits	Population	Min.	1st Q	Median	Mean	3rd Q	Max.	D	p-value
≥0	All (1,626)	0.00	0.00	1.00	9.62	6.00	491.00	0.203	0.0001
	Survey (125)	0.00	0.00	4.00	14.12	14.00	201.00		
≥1	All (986)	1.00	1.00	3.00	13.76	11.00	491.00	0.197	0.0028
	Survey (92)	1.00	1.00	5.00	16.35	19.00	201.00		
≥2	All (693)	2.00	3.00	7.00	19.15	18.00	491.00	0.194	0.0128
	Survey (74)	2.00	5.00	12.00	23.61	27.00	201.00		
≥3	All (563)	3.00	5.00	9.00	23.11	22.00	491.00	0.196	0.0243
	Survey (64)	3.00	6.75	13.50	26.98	31.00	201.00		
≥4	All (490)	4.00	6.00	11.00	26.11	26.00	491.00	0.136	0.2562
	Survey (63)	4.00	7.00	14.00	27.37	31.00	201.00		
≥5	All (427)	5.00	7.00	13.00	29.37	30.00	491.00	0.121	0.4406
	Survey (58)	5.00	8.50	16.50	29.38	34.75	201.00		
≥8	All (314)	8.00	12.00	19.00	37.77	41.00	491.00	0.119	0.6229
	Survey (46)	8.00	13.00	22.00	35.57	42.75	201.00		
≥11	All (256)	11.00	14.75	24.00	44.34	52.00	491.00	0.084	0.9631
	Survey (41)	11.00	14.00	26.00	38.83	50.00	201.00		

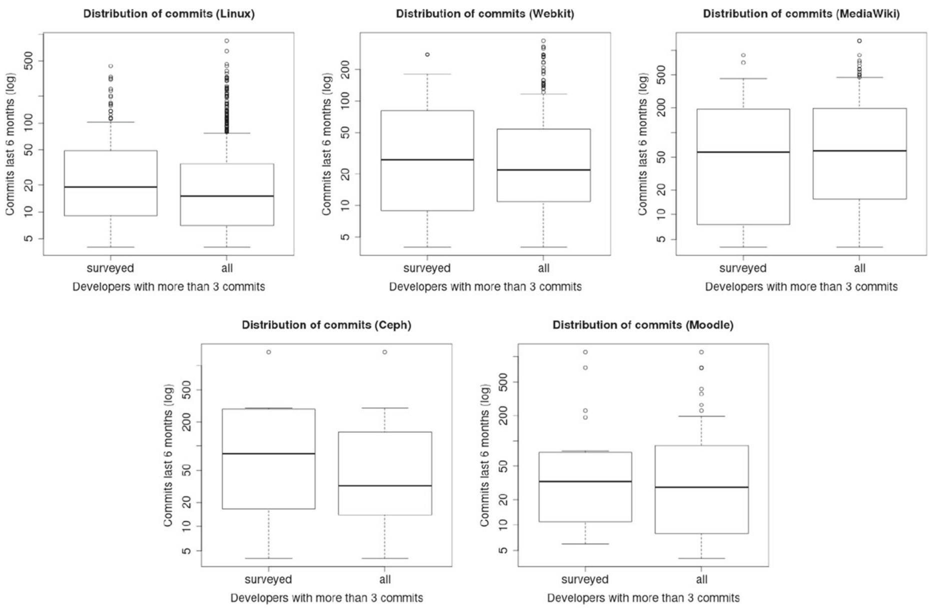


Fig. 11 Boxplot with the activity (in number of commits during the last 6 months before the survey) for active surveyed developers and for all the active developers in the last 6 months before the survey

9.5 Generalization to Other Projects

If a project wants to know its own θ with accuracy, the amount of data that they need is limited and is easy to gather. By polling their developers for their full-time/part-time status, they could use our model to find their particular θ. Some FOSS projects regularly survey their developers for knowing their community better, including personal, academic, working, community and other matters. So, for instance, since 2011 OpenStack performs a yearly developer survey.[13] Gathering information for our model would just require two questions to be added to those surveys (e.g., '*do you consider yourself to be a full-time developer*' and '*how many commits have you performed in the last 6 months*') .

We think, however, that the value of θ is dependent on intrinsic properties of the project, and that a FOSS project can use the model without having to perform a developers survey, but to use a θ from a similar project. The results of applying our model to the 6 FOSS projects in this study make us think that θ depends on the process that the project follows. Thus, depending on the project's practices, commits may require more or less effort to be approved before being merged, depending on whether a review process is in practice. If that is the case, code is not formally committed to the repository until it has been through extensive review, usually including several revisions. In addition, and particularly in those projects that have more strict code review practices, projects require to *squash* all commits into a single one once a change is accepted; then, this only commit can be pushed to the repository. This has as a consequence that commits are larger and more costly (in time and effort) for those projects, than for others that do not follow this practice.

[13] https://insights.stackoverflow.com/survey/

This can be confirmed from the projects that we have studied in this paper. The values of θ shown in Table 9 are similar for those projects that have strict code reviewing practices, such as OpenStack ($\theta = 12$), Moodle (14), Linux (17) and WebKit (16–19). All of them use Gerrit to support code review. Ceph (24) and MediaWiki (36–37) have higher numbers of θ, because their code review practices are *lighter* (i.e., less demanding) and commit squashing is not that strictly followed as in the former projects. For instance, Ceph uses GitHub's pull-request mechanism, which requires just one approval from a maintainer, while for OpenStack changes require to be approved by at least two maintainers.

Projects that do not use commit squashing, could use 'active days' instead of 'commits' in the model. That way, all commits during the same day would be considered as a contribution, mitigating the impact of different commit behaviours of committers (Kolassa et al. 2013a).

10 Conclusion and Future Work

This paper has tackled two challenges. The first is how to design a simple, but sound estimation model to measure the effort provided in a sparse, distributed and uneven development scenario, like the FOSS one. The second challenge is how to design the model so that it offers not only a reasonable prediction, but also credible.

In order to maximise the *simplicity* of our estimation model for FOSS development, we only discriminate between two types of developers: *full-timers* and the rest. For harnessing the *credibility* of our model, we have obtained feedback data from over a thousand developers, who work at different levels across six large FOSS projects.

The model establishes a threshold θ that separates the level of activity of *full-timers* from the rest of developers. The value of θ has been optimised with the developers' responses: thanks to their feedback, we have achieved a much more realistic separation of developer types, minimizing the estimation error. Using θ in our model, the estimation of the overall effort results in a simple calculation, and just using two developer types.

We conjecture that the model being dependent on this value of θ is what allows the model to be useful for different projects: the relationship of commits to effort (which is at the core of the model) may be very different from project to project, but we have shown that it can be captured with this single parameter.

Using 6 large FOSS projects as case studies, we have shown how the model can be applied and fine-tuned. Although further research on this is needed, the results obtained make us hypothesize that the value of θ depends on the development practices—basically, how strict code reviewing practices are and if commit squashing is frequently used when merging changes into the source code. If so, the future applicability of our model would not require to survey developers, as it would suffice to use a value of θ obtained from projects that follow similar practices.

We envisage to expand this study by 1) studying other FOSS projects to ascertain if our method is applicable in general, and if so, to what extent; 2) performing a scientific experiment to obtain margins of error for the estimation of error for non-full-time developers; 3) comparing our results with the ones provided by traditional software estimation models used in industry, such as COCOMO; 4) after quantifying the effort required in a FOSS project, establishing whether it is more profitable for a prospective adopting company to redo ("make") their own system, or to invest ("buy") in the existing FOSS system as discussed in Asundi (2005); and 5) comparing our approach with previous effort estimation techniques for FOSS projects, as the one proposed in Capra et al. (2007) based on the measure of entropy to calculate maintenance costs.

Acknowledgements We want to express our gratitude to Bitergia[14] for the support they have provided when questions have arisen. We acknowledge the support of the Government of Spain through the "BugBirth" project (RTI2018-101963-B-100). We also acknowledge the work by Carlos Cervigón on an earlier version of the manuscript.

Declarations

Conflict of Interest The authors whose names are listed immediately below certify that they have NO affiliations with or involvement in any organization or entity with any financial interest (such as honoraria; educational grants; participation in speakers' bureaus; membership, employment, consultancies, stock ownership, or other equity interest; and expert testimony or patent-licensing arrangements), or non- financial interest (such as personal or professional relationships, affiliations, knowledge or beliefs) in the subject matter or materials discussed in this manuscript.

References

Abdelmoez W, Kholief M, Elsalmy FM (2012) Bug fix-time prediction model using naïve bayes classifier. In: 2012 22nd International conference on computer theory and applications (ICCTA). IEEE, pp 167–172

Abran A, Desharnais J-M, Aziz F (2016) 3.5 measurement convertibility—from function points to cosmic ffp. Cosmic Function Points: Theory and Advanced Practices 214

Agrawal A, Rahman A, Krishna R, Sobran A, Menzies T (2018) We don't need another hero? The impact of "heroes" on software development. In: Proceedings of the 40th international conference on software engineering: software engineering in practice, pp 245–253

Ahsan SN, Ferzund J, Wotawa F (2009) Program file bug fix effort estimation using machine learning methods for oss. In: SEKE, pp 129–134

Alomari H (2015) A slicing-based effort estimation approach for open-source software projects. Int J Adv Comput Eng Netw (IJACEN) 3(8):1–7

Amor JJ, Robles G, Gonzalez-Barahona JM (2006) Effort estimation by characterizing developer activity. In: Proceedings of the 2006 international workshop on economics driven software engineering research. ACM, pp 3–6

Anbalagan P, Vouk M (2009) On predicting the time taken to correct bug reports in Open Source projects. In: IEEE international conference on software maintenance. ICSM 2009. IEEE, pp 523–526

Asundi J (2005) The need for effort estimation models for open source software projects. ACM SIGSOFT Softw Eng Notes 30(4):1–3

Boehm B (1981) Software engineering economics

Boehm BW, Madachy R, Steece B et al (2000) Software cost estimation with COCOMO II with CDROM. Prentice Hall PTR

Capiluppi A, Izquierdo-Cortázar D (2013) Effort estimation of FLOSS projects: a study of the Linux kernel. Empir Softw Eng 18(1):60–88

[14] http://bitergia.com/

Capiluppi A, Michlmayr M (2007) From the cathedral to the bazaar: an empirical study of the lifecycle of volunteer community projects. In: IFIP International conference on open source systems. Springer, pp 31–44

Capra E, Francalanci C, Merlo F (2007) The economics of open source software: an empirical analysis of maintenance costs. In: IEEE international conference on software maintenance. ICSM 2007. IEEE, pp 395–404

Capra E, Francalanci C, Merlo F (2008) An empirical study on the relationship between software design quality, development effort and governance in Open Source Projects. IEEE Trans Softw Eng 34(6):765–782

Capra E, Francalanci C, Merlo F (2010) The economics of community open source software projects: an empirical analysis of maintenance effort. Advances in Software Engineering

Crowston K, Howison J (2005) The social structure of free and open source software development. First Monday 10(2)

Dueñas S, Cosentino V, Robles G, Gonzalez-Barahona JM (2018) Perceval: software project data at your will. In: Proceedings of the 40th international conference on software engineering: companion proceedings, pp 1–4

Dumke R, Abran A (2016) COSMIC function points: theory and advanced practices. Auerbach Publications

Fernandez-Ramil J, Izquierdo-Cortazar D, Mens T (2009) What does it take to develop a million lines of Open Source code? In: Open source ecosystems: diverse communities interacting. Springer, pp 170–184

Fitzgerald B (2006) The transformation of open source software. Mis Quarterly 587–598

González-Barahona JM, Robles G (2012) On the reproducibility of empirical software engineering studies based on data retrieved from development repositories. Empir Softw Eng 17(1–2):75–89

Hönel S, Ericsson M, Löwe W, Wingkvist A (2018) A changeset-based approach to assess source code density and developer efficacy. In: Proceedings of the 40th international conference on software engineering: companion proceedings, pp 220–221

Hou Q, Ma Y, Chen J, Xu Y (2014) An empirical study on inter-commit times in svn. In: SEKE, pp 132–137

Jorgensen M, Shepperd M (2007) A systematic review of software development cost estimation studies. IEEE Trans Softw Eng 33(1):33–53

Kalliamvakou E, Gousios G, Spinellis D, Pouloudi N (2009) Measuring developer contribution from software repository data. MCIS 2009:4th

Kalliamvakou E, Gousios G, Blincoe K, Singer L, German DM, Damian D (2014) The promises and perils of mining github. In: Proceedings of the 11th working conference on mining software repositories, pp 92–101

Koch S (2004) Profiling an open source project ecology and its programmers. Electron Mark 14(2):77–88

Koch S (2008) Effort modeling and programmer participation in open source software projects. Inf Econ Policy 20(4):345–355

Koch S, Schneider G (2002) Effort, co-operation and co-ordination in an open source software project: GNOME. Inf Syst J 12(1):27–42

Kolassa C, Riehle D, Salim MA (2013a) The empirical commit frequency distribution of open source projects. In: Proceedings of the 9th international symposium on open collaboration, pp 1–8

Kolassa C, Riehle D, Salim MA (2013b) A model of the commit size distribution of open source. In: International conference on current trends in theory and practice of computer science. Springer, pp 52–66

Kononenko O, Rose T, Baysal O, Godfrey M, Theisen D, De Water B (2018) Studying pull request merges: a case study of shopify's active merchant. In: Proceedings of the 40th international conference on software engineering: software engineering in practice, pp 124–133

Kouters E, Vasilescu B, Serebrenik A, van den Brand MG (2012) Who's who in GNOME: using LSA to merge software repository identities. In: 2012 28th IEEE international conference on software maintenance (ICSM). IEEE, pp 592–595

Lerner J, Tirole J (2002) Some simple economics of open source. J Ind Econ 50(2):197–234

Ma Y, Wu Y, Xu Y (2014) Dynamics of open-source software developer's commit behavior: an empirical investigation of subversion. In: Proceedings of the 29th annual ACM symposium on applied computing, pp 1171–1173

Malhotra R, Lata K (2020) Using ensembles for class-imbalance problem to predict maintainability of open source software. Int J Reliab Qual Safety Eng 2040011

Mi Q, Keung J (2016) An empirical analysis of reopened bugs based on open source projects. In: Proceedings of the 20th international conference on evaluation and assessment in software engineering, pp 1–10

Michlmayr M, Fitzgerald B, Stol K-J (2015) Why and how should open source projects adopt time-based releases? IEEE Softw 32(2):55–63

Mockus A, Votta LG (2000) Identifying reasons for software changes using historic databases. In: International conference on software maintenance. Proceedings. IEEE, pp 120–130

Mockus A, Fielding RT, Herbsleb JD (2002) Two case studies of open source software development: Apache and mozilla. ACM Transa Softw Eng Methodol (TOSEM) 11(3):309–346
Moulla D, Kolyang (2013) COCOMO model for software based on open source: application to the adaptation of triade to the university system. Int J Comput Sci Eng (IJCSE) 5(6):522–527
Moulla DK, Damakoa I, Kolyang DT (2014) Application of function points to software based on open source: a case study. In: 2014 Joint conference of the international workshop on software measurement and the international conference on software process and product measurement. IEEE, pp 191–195
Porru S, Murgia A, Demeyer S, Marchesi M, Tonelli R (2016) Estimating story points from issue reports. In: Proceedings of the the 12th international conference on predictive models and data analytics in software engineering, pp 1–10
Riehle D, Riemer P, Kolassa C, Schmidt M (2014) Paid vs. volunteer work in open source. In: 2014 47th Hawaii international conference on system sciences. IEEE, pp 3286–3295
Robles G, Gonzalez-Barahona JM (2005) Developer identification methods for integrated data from various sources. ACM SIGSOFT Softw Eng Notes 30(4):1–5
Robles G, Koch S, González-Barahona JM, Carlos J (2004) Remote analysis and measurement of libre software systems by means of the cvsanaly tool. In: Proceedings of the 2nd ICSE workshop on remote analysis and measurement of software systems (RAMSS). IET, pp 51–56
Robles G, González-Barahona JM, Cervigón C, Capiluppi A, Izquierdo-Cortázar D (2014) Estimating development effort in free/open source software projects by mining software repositories: a case study of openstack. In: Proceedings of the 11th working conference on mining software repositories. ACM, pp 222–231
Shah SK (2006) Motivation, governance, and the viability of hybrid forms in open source software development. Manag Sci 52(7):1000–1014
Sowe SK, Stamelos I, Angelis L (2008) Understanding knowledge sharing activities in free/open source software projects: an empirical study. J Syst Softw 81(3):431–446
Steinmacher I, Conte T, Gerosa MA, Redmiles D (2015) Social barriers faced by newcomers placing their first contribution in open source software projects. In: Proceedings of the 18th ACM conference on computer supported cooperative work & social computing, pp 1379–1392
Thung F (2016) Automatic prediction of bug fixing effort measured by code churn size. In: Proceedings of the 5th international workshop on software mining, pp 18–23
Von Krogh G, Spaeth S, Lakhani KR (2003) Community, joining, and specialization in open source software innovation: a case study. Res Policy 32(7):1217–1241
Wiese IS, da Silva JT, Steinmacher I, Treude C, Gerosa MA (2016) Who is who in the mailing list? comparing six disambiguation heuristics to identify multiple addresses of a participant. In: 2016 IEEE International conference on software maintenance and evolution (ICSME). IEEE, pp 345–355
Wu H, Shi L, Chen C, Wang Q, Boehm B (2016) Maintenance effort estimation for open source software: a systematic literature review. In: 2016 IEEE international conference on software maintenance and evolution (ICSME). IEEE, pp 32–43
Yang Y, Harman M, Krinke J, Islam S, Binkley D, Zhou Y, Xu B (2016) An empirical study on dependence clusters for effort-aware fault-proneness prediction. In: 2016 31st IEEE/ACM international conference on automated software engineering (ASE). IEEE, pp 296–307
Yu L (2006) Indirectly predicting the maintenance effort of open-source software. J Softw Maint Evol: Res Pract 18(5):311–332
Zhao Y, Zhang F, Shihab E, Zou Y, Hassan AE (2016) How are discussions associated with bug reworking? an empirical study on open source projects. In: Proceedings of the 10th ACM/IEEE international symposium on empirical software engineering and measurement, pp 1–10

Gregorio Robles is a Full Professor at the Universidad Rey Juan Carlos, Madrid, Spain. Gregorio is specialized in free/open source software research, basically by mining software repositories. Lately, he investigates as well how to assess the development of Computational Thinking skills. He is one of the founders of Bitergia, a software development analytics company. His homepage can be found at http://gsyc.urjc.es/~grex.

Andrea Capiluppi works as an Associate Professor at University of Groningen, The Netherlands. Andrea's work focuses on empirical software engineering and open source and he's currently directing the NLPforSE (Natural Language Processing for Software Engineering) research group with a focus on machine learning techniques for software engineering.

Björn Lundell received a PhD from the University of Exeter in 2001. He is a professor at the University of Skörvde where he leads the Software Systems Research Group, and has been a staff member and researcher since 1984. Professor Lundell's research contributes to theory and practice in the software systems domain and centres on different aspects of openness (in particular open source and open standards) related to development, use, and procurement of software systems. His research addresses fundamental socio-technical challenges concerning software systems, and focuses on different aspects of lock-in, interoperability, and longevity of systems. His research is reported in over 100 papers in a variety of international journals and conferences. Professor Lundell has been active in a number of international and national research projects which have lead to significant scientific and societal impact, and has been an expert advisor and contributed to guidelines and policies in the field.

Jonas Gamalielsson is a researcher at the University of Skövde, Sweden. has worked with software development in industry from 1990 to 2001, and has thereafter (since 2001) been involved in teaching and research at the University of Skövde. He received his Ph.D. in computer science from Heriot-Watt University (Edinburgh) in 2009. He is a member of the Software Systems Research Group at University of Skövde and has conducted research related to open source software in a number of national and international projects since 2008. Focus has been on challenges related to lock-in, interoperability and long-term maintenance of software systems in various contexts. His research is reported in publications in a variety of international journals and conferences.

Affiliations

Gregorio Robles[1] · **Andrea Capiluppi**[2] · **Jesus M. Gonzalez-Barahona**[1] · **Björn Lundell**[3] · **Jonas Gamalielsson**[3]

Gregorio Robles
gregorio.robles@urjc.es

Jesus M. Gonzalez-Barahona
jesus.gonzalez.barahona@urjc.es

Björn Lundell
bjorn.lundell@his.se

Jonas Gamalielsson
jonas.gamalielsson@his.se

1 Department of Telematic and Computational Systems Engineering, Universidad Rey Juan Carlos, Madrid, Spain

2 Department of Computer Science, University of Groningen, Groningen, The Netherlands

3 School of Informatics, University of Skövde, Skövde, Sweden