2016 IEEE International Conference on Software Maintenance and Evolution

Maintenance Effort Estimation for Open Source Software: A Systematic Literature Review

Hong Wu[1,2], Lin Shi[1,4], Celia Chen[4], Qing Wang[1,2,3], Barry Boehm[4]
[1]Laboratory for Internet Software Technologies, Institute of Software, Chinese Academy of Sciences, Beijing, China
[2]University of Chinese Academy of Sciences, Beijing, China
[3]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
[4]Center for Systems and Software Engineering, University of Southern California, Los Angeles, U.S.A.
Email: {wuhong, shilin, wq}@itechs.iscas.ac.cn, {qianqiac, boehm}@usc.edu

***Abstract*—Open Source Software (OSS) is distributed and maintained collaboratively by developers all over the world. However, frequent personnel turnover and lack of organizational management makes it difficult to capture the actual development effort. Various OSS maintenance effort estimation approaches have been developed to provide a way to understand and estimate development effort. The goal of this study is to identify the current state of art of the existing maintenance effort estimation approaches for OSS. We performed a systematic literature review on the relevant studies published in the period between 2000-2015 by both automatic and manual searches from different sources. We derived a set of keywords from the research questions and established selection criteria to carefully choose the papers to evaluate. 29 out of 3,312 papers were selected based on a well designed selection process. Our results show that the commonly used OSS maintenance effort estimation methods are actual effort estimation and maintenance activity time prediction; the most commonly used metrics and factors for actual effort estimation are source code measurements and people related metrics; the most commonly mentioned activity for maintenance activity time prediction is bug fixing. Accuracy measures and cross validation is used for validating the estimation models. Based on the above findings, we identified the issues in evaluation methods for actual maintenance effort estimations and the needs for quantitative OSS maintenance effort inference from size-related metrics. Meanwhile, we highlighted individual contribution and performance measurement as a novel and promising research area.**

I. Introduction

Open Source Software (OSS) is contributed and maintained collaboratively and voluntarily by large numbers of developers all over the world. Unlike commercial software, OSS projects might not have specific and detailed project plans, schedules, road maps or a list of deliverable at the beginning of its life cycle. Since developers aren't required to sign any contracts before joining, personnel turnover is extremely high among OSS projects. Consequently, the development effort invested in maintaining an OSS project is usually unknown, even after the software has been released.

Maintenance effort refers to the resource and time spent on satisfying the demands in resolving defects, undesirable behavior, enhancements, and for the general maintenance of the software [22]. Software maintenance support preparation should not happen after the system is delivered, rather its planning should happen during the planning stage of software development, and its preparation accomplished during development. Maintenance of open source projects is quite different compared to software systems in general. For OSS projects, more effort is devoted on maintaining a healthy community, such as participating threads in issue tracking system, and answering questions on chatting channels or mailing lists. Besides, making the projects easy to contribute is very important for OSS maintainers. OSS maintainers spend more effort on submitting good commit messages, conforming with coding styles, writing useful wiki, and so on.

Prediction or estimation model is commonly used for maintenance planning, which helps organizations better allocate resources hence reduce actual maintenance effort [34]. With rapid growth in the number of individuals and organizations that are actively involved in OSS projects, the area of OSS maintenance effort estimation has been highly interested in the past years. Through acquiring estimated maintenance effort, OSS projects can alleviate management issues such as lack of prior planning. Given the benefits that OSS projects can gain from better understanding maintenance effort estimation, it is extremely important to research and understand existing OSS maintenance effort estimation approaches in order to contribute to this area significantly. Therefore in this paper, we present a systematic literature review on OSS maintenance effort estimation approaches in order to understand its current state of the art and to identify opportunities for future research.

The remainder of the paper is organized as follows. Section 2 describes the methodology. Section 3 provides the results of our research questions. Section 4 presents the implications and future directions for the research. Section 5 discusses the threats to validity of the study. Finally, Section 6 concludes our work.

II. Research Methodology

We performed a systematic literature review by following the guidelines proposed by Kitchenham and Charters [19]. In this section, we present a detailed description of steps involved in planning and conducting this study.

A. Planning

We developed a protocol to prescribe a controlled procedure to conduct this study in order to reduce the possibility of

978-1-5090-3806-0/16 $31.00 © 2016 IEEE
DOI 10.1109/ICSME.2016.87

researcher bias. Our protocol mainly included a set of research questions, a search strategy with inclusion/exclusion criteria, a data extraction form, a set of quality assessment criteria, data extraction and data synthesis. The first author developed the protocol. The second and third co-authors performed a pilot search and data extraction to evaluate and improve the protocol. The last two co-authors performed expert reviews on the protocol. With all of the above, we obtained a final version of the protocol to conduct this study. Different components of the protocol are presented in the following subsections.

B. Research Questions

The questions and the associated motivations for this study are presented as follows.

- RQ1: What evidence is there for maintenance effort estimation techniques/methods for OSS projects? We aim to identify the classification of current research focus, and identify the related research type.
- RQ2: What metrics related to OSS development records are extracted for maintenance effort estimation and how can they be classified? We aim to identify software metrics commonly used in OSS maintenance effort estimation.
- RQ3: What are common projects and the size of dataset used as study cases in OSS maintenance effort estimation, and how has the frequency of approaches related to the size of dataset? We aim to identify the dataset used in the studies and investigate the studies external validity.
- RQ4: What methods/approaches are used to estimate actual project maintenance effort (including those from the usual incomplete OSS development records)? We aim to identify trends and possible opportunities for estimation method focus.
- RQ5: What is the overall estimation accuracy of OSS maintenance effort estimation? We aim to identify to what extent the studies provide accurate prediction.

C. Search Strategy

Based on the questions above, a search strategy for primary studies was defined.

1) Search Terms: The search terms were retrieved from the PICO (Population, Intervention, Comparison and Outcome) terms in the research questions. Since the focus of this study was not limited to comparative studies, comparison was not considered in the search strategy. Therefore we focused on Population, Intervention and Outcome.

We identified a set of variants and acronyms from the research questions for each PICO term:

- **Population:** open source. ***Terms:*** "Open source", OSS, FOSS, FLOSS, opensource, libre, free.
- **Intervention:** maintenance effort estimation. ***Terms:*** maintain, maintenance, evolve, evolution, bug fix, bug fixing, defect fixing, defect resolution, effort, cost, estimation, prediction.
- **Outcome:** accuracy and usefulness of the OSS maintenance estimation approaches are the important factors to practitioners, which can be determined by empirical evidence. ***Terms:*** empirical, validation, experiment, evaluation, "case study".

To build up the query string, we first composed the terms inside Population, Intervention and Outcome through an OR connector. Then we composed the terms outside Population, Intervention and Outcome through an AND connector. The resulting query string is as the following: ("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) Literature resources: Since Kitchenham *et al.* [19] pointed out that both automatic search through bibliographic databases and manual search through journals and conference publications have drawbacks, we decided to combine both approaches by first automatically search through some selected databases and then manually search through relevant conference publications and journals.

The following electronic databases were selected for the automated search:

- Inspec
- Compendex
- IEEE Xplore
- ACM Digital Library
- Science Direct
- Google Scholar

These databases were chosen because most of the publication venues that published papers in the maintenance effort estimation are indexed by these databases.

3) Search Process: We adopted a two-stage search process to avoid leaving out any relevant primary studies.

- **The initial search stage.** We used the proposed search terms to search for primary candidate studies in each literature resource separately, based on the title, abstract and keywords. All the searches were limited to articles published from 1 January 2000 to 31 December 2015. Then we merged the search results by removing duplicated articles.
- **The secondary search stage.** For identifying more relevant studies, we performed snowballing [8] technique based on the search studies in the initial search stage. Whenever a highly relevant paper was found, we added it to the set of the primary relevant studies.

D. Study Selection

The study selection process of this work was performed as the following.

First, we evaluated the candidate studies obtained in the step of search literature resources based on the title and abstract to select primary studies. The included and excluded criteria were defined as follows to determine whether an candidate study should be retained or rejected.

Inclusion Criteria:

- Studies that are in the field of Software Engineering ***AND***
- Studies that are of the context of OSS ***AND***
- Studies that are presenting effort estimation ***AND***
- Studies that are presented in English ***AND***
- Studies that are accessible in full-text

Exclusion Criteria:

- Books and gray literature ***OR***
- Studies that are duplicate publications

Gray literature is generally referred to publications that are not commercially published, which produced for the purpose of information dissemination, such as technical reports, dissertations, and ongoing works [19]. Based on the above criteria, each candidate study was reviewed by three researchers according to titles. 169 out of 3,312 candidate studies were retained. Then we performed a fast full-text reading to further identify the most relevant studies. 31 candidate studies satisfying the inclusion and exclusion criteria were selected and then added to the set of the primary studies, which is the results from the initial selection stage. Second we reviewed the references list of the 31 primary studies, identified the relevant studies which are not included in the initial selection stage, and manually searched and added 9 studies to the set of primary studies to ensure the coverage of the relevant studies. After that, we performed the quality assessment to the 40 studies. Finally, we obtained 29 primary studies. Fig. 1 lists the number of selection results from each step of the study selection process.

To ensure the quality of the selection results, the inclusion and exclusion criteria have been evaluated by the last two co-authors, and the co-authors who performed the selection have arrived the same understanding on the criteria. During the selection, each candidate study was reviewed and categorized based on the inclusion and exclusion criteria, and only the ones with an agreement made by all of the three reviewers can be included in the final results. Any candidate studies that didn't match the searched topics would be reviewed by the co-authors to make a final decision of retention of rejection.

E. Study Quality Assessment

We adopted a quality assessment (QA) process to enhance our review study. Four QA questions were devised to assess the rigorousness and credibility of the selected primary studies, as shown in the following:

- QA1: Are the objectives of the study clearly defined?
- QA2: Is there a description of the study context?
- QA3: Is the proposed solution well presented?
- QA4: Are the limitations of the study analyzed explicitly?

For each QA question, there are three optional answers: "Yes", "Partially", and "No". These answers are scored as follows: "Yes"=1, "Partially"=0.5, and "No"=0. For each selected primary study, its quality score is computed by summing up the scores of the answers to all the four QA questions. The quality assessment of the selected primary studies was performed by the first three authors individually. All disagreements on the QA results were discussed until a final consensus was reached among the co-authors. We collected 40 studies after snowball and manual search. We classify the quality level into High ($score = 4$), Medium ($2 < score < 4$), and Low ($score <= 2$). We selected 29 studies that scored in high and medium levels as our final selection[1].

F. Data Extraction and Synthesis

We created a data extraction form to extract data from the identified primary studies. The extraction was performed by the first three authors separately, and then evaluated by cross-review. Any uncertainty or inconsistency of the extracted data was discussed and reached an agreement among the co-authors.

TABLE I
THE MAPPING BETWEEN EXTRACTED PROPERTIES AND RQS

Property	**RQ**
Topic	RQ1, RQ4
Research Type	RQ1
Metrics/Factors	RQ2
Project	RQ3
Size of dataset	
Estimation Approaches	RQ4, RQ5
Accuracy Metrics	
Accuracy Level	

TABLE I presents a brief mapping between each primary study's properties and RQs. We adopted six different research types to classify the primary studies based on the literature [17, 39]. The description of each research type is presented as follows.

- **Development of estimation method:** This research type includes studies developing new effort (or size) estimation models, processes, tools or new hybrid methods that combined various estimation approaches.
- **Case study:** This research type includes case-based studies such as in-depth study of estimation processes on one or a very small number of software projects.
- **Experiment:** This research type includes experiment-based studies, which are launched to control over the estimation process systematically.
- **History-based evaluation:** This research type includes studies evaluating estimation methods or other estimation-relevant relationships on previously completed software projects.
- **Theory:** This research type includes non-empirical research approaches or theoretical evaluation of properties of estimation models. Only studies that rely heavily on non-empirical research methods in their evaluation and development of estimation approach are included in this paper.
- **Comparison study:** This research type includes studies involving a systematized endeavor to compare two or more estimation models or techniques, with an eye toward

[1]More details on QA results:
http://itechs.iscas.ac.cn/cn/membersHomepage/wuhong/SLR_OSS.html

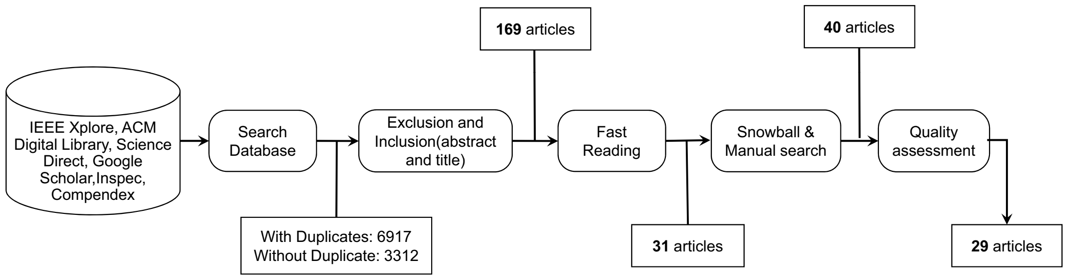

Fig. 1. Number of included primary studies during the study selection process

identifying points that the models hold in common, along with citing areas where the two models differ.

We synthesized and tabulated all the extracted data from the primary studies. Several synthesis methods were adopted in this study. We used the narrative synthesis method with visualization tools (e.g. bar charts, line charts and bubble plots) to summarize the findings for RQ1 and reciprocal translation and voted counting methods for RQ2-RQ5.

III. Results

In this section, we present the overview of the selected studies and discuss the results related to the research questions listed in Section II.B.

A. *Overview of the Selected Studies*

We investigated where the OSS maintenance effort estimation studies were published to identify the common publication platforms that such research can be found. The selected studies were published in the following publication sources:

- Journal: Advances in Software Engineering, Information and Software Technology, Information Economics and Policy, Information Systems Journal, Journal of Advanced Computational Engineering and Networking, Journal of Empirical Software Engineering, Journal of Software Maintenance and Evolution: Research and Practice, Journal of Software: Evolution and Process.
- Conference: MSR, ICSME, ISEC, SEKE, PROMISE, ICCTA, IACC
- Workshop: QuASoQ, MUD, EDSER, RSSE, WOSSE

From the list of publication sources, we observed that studies on OSS maintenance effort estimation are widely accepted by a broad range of well-established venues. This reflects that the current community preserves a diversity of interests in the OSS maintenance effort estimation.

To investigate the general publication platforms for OSS maintenance effort estimation studies, we presented the number of publications in three different publication types: workshop, conference, and journal where OSS maintenance effort estimation studies were present.

Fig. 2 provides an overview of the distribution of primary studies among the three platform types. The results show that conference is the main publication type for the studies, which has 13 articles published. Journal type has 10 articles and workshop type has 6 articles.

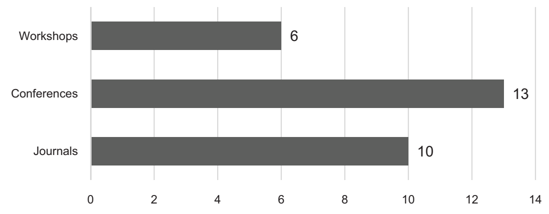

Fig. 2. Number of studies by venues types

We further investigated the specific publication platforms that have published OSS maintenance effort estimation studies the most. TABLE II shows the number of published studies on different venues. 5 papers out of 29 were published in International Workshop on Mining Software Repositories(MSR), 3 papers were published in PROMISE, and 2 papers were published in ICSME, Journal of Information and Software Technology, and Journal of Empirical Software Engineering, while the rest were one paper distributed across a wide range of venues.

TABLE II
Major Publication Platforms

Publication platform	**#Studies**
International Workshop on Mining Software Repositories (MSR)	5
International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE)	3
International Conference on Software Maintenance and Evolution (ICSME)	2
Information and Software Technology	2
Journal of Empirical Software Engineering	2

B. *RQ1: Classification and Research Type*

As shown in TABLE III, we identified five main research topics from the selected studies. Typically, four types of maintenance have been identified by ISO/IEC 14764, which are corrective maintenance, adaptive maintenance, perfective maintenance, and preventive maintenance [1]. The scope of this literature review include the four types of maintenance. For example, studies under the topic of bug fixing effort prediction are related to corrective maintenance. Studies under the topics of entire project maintenance effort prediction include adaptive, perfective, and preventive maintenance tasks.

The first topic referred to studies that predict indirect effort of the OSS projects. Existing studies assumed that the amount of source code changes could indirectly represent the effort spent in the maintenance phase between two consecutive versions. The assumption was confirmed by Yu in 2006 [40] by finding the strong positive correlation between actual effort and source code changes in NASA SEI data. There are three articles identified under this topic and we grouped them into two sub-topics according to their prediction objectives. Studies in source code-based estimation used size metrics to indirectly indicate effort while process-based estimation used state change duration to determine the effort.

The second topic referred to studies that predict direct effort needed for software evolution in a given duration by considering data from source code repository and discussion channels. Four articles used overall characteristics of developers, such as manpower function as the inputs to their estimation methods, thus we classify them into people-based estimation. Two articles estimated the maintenance effort through the time spent on different development activities and we classify them into activity-based estimation.

The third topic referred to studies that predict effort spent on typical maintenance activities. The activities for maintaining OSS projects include defects resolution, software enhancement, and general maintenance of the software[22]. We considered the time and resource required to perform those activities to be the OSS maintenance effort. Based on the types of maintenance activities, we grouped the studies into three sub-topics: peer code review, duplicate issues identification, and bug fixing. Among the three sub-topics, most studies focused on predicting bug fixing time, which includes 14 articles. For peer code review and duplicate issues identifications, one article was identified for each.

The fourth topic referred to studies that provide contribution indicators of individual OSS developers. These indicators include the code each developer modified and their communication with other collaborators in the maintenance phase. We identified two articles in this topic. Both articles assessed individual contribution among OSS developers and used it to indicate the overall effort of OSS developers spent on maintaining OSS projects.

TABLE III
CLASSIFICATION OF TOPICS

Topic	**Sub-topic**	**Paper Reference**
Indirect effort prediction of the entire project	Source code-based estimation	[40],[4]
	Process-based estimation	[23]
Direct effort prediction of the entire project	People-based estimation	[21],[12],[20],[35]
	Activity-based estimation	[5],[11]
Effort prediction of maintenance activity	Peer code review	[25]
	Duplicate issues identification	[32]
	Bug fixing	[24],[6],[3],[38],[30], [14],[10],[16],[36],[29], [13],[2],[26],[27]
Individual contribution measurement		[15],[33]
Guidelines and discussion		[7],[37]

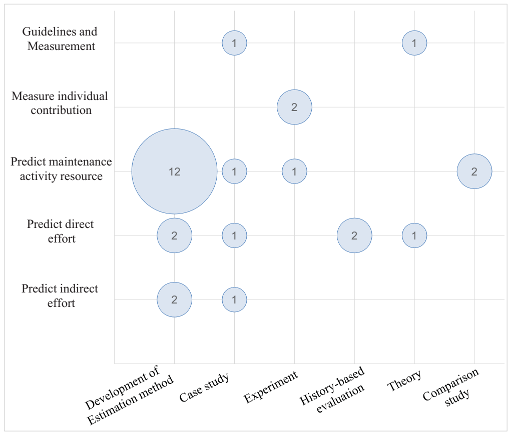

Fig. 3. Number of studies in each research type related to topics

The fifth topic included studies that demonstrated guidelines to some possible solutions to estimating OSS maintenance effort, discussions about the needs for new OSS effort estimation models and the issues threaten the effectiveness of such effort-aware models.

The results from classifying topics showed that studies aim to predict effort of maintenance activity mainly concentrated on bug fixing time prediction, while less efforts contributed to other types of activities such as peer code review and duplicate issues identification.

Based on the guidelines[31] and cost estimation reviews[18], we classified the research types of the selected studies as development of estimation method, case study, experiment, history-based evaluation, theory, and comparison study. Most of the studies that predict bug fixing time developed an estimation method and conducted a case study to verify the proposed method. However, in this paper, we treated such studies as development of estimation method only. We investigated the relationship between topics and research types, as shown in Fig. 3.

The results show that the highest frequency was in the research for developing estimation method. Within this method, the highest frequency was in predicting maintenance activity effort.

C. RQ2: Metrics/Factors

The selected studies used various metrics or factors as inputs of OSS maintenance effort estimation methods. The metrics/factors were divided into five categories as follows.

- Project. This category includes metrics that statistically described the characteristics of the entire project. There are six aspects in terms of content: size, time, task, bugs, commits, and developers.
- Changes. This category includes quantitative measurements that gave insight into characteristics of source code in patches. There are eight aspects: Cyclomatic complexity (CC), function, LOC, file, class, operator, parameter and return.

- Issue report. This category includes metrics derived from issue reports. There are six aspects: basic information (info), stakeholder, time, similarity, process, and quality.
- Participant. This category includes metrics that can quantitatively represent the experience and the capability of the participants involved in the OSS maintenance activity. There are four roles: bug reporter, bug owner, bug trigger, and bug collaborator.
- Community. This category includes metrics about the OSS community characteristics. There are three aspects: contributor, activity, and workload.

TABLE IV shows the types of metrics used by each topic. We used {ALL} to indicate all the metrics under a certain category. Studies on indirect effort prediction mainly used metrics in project category and changes category. Since the actual effort is difficult to measure in OSS projects, studies used size-related metrics to indirectly estimate the effort on an entire project [40][4].

TABLE VI shows all the metrics and their corresponding total number of the frequencies in the selected studies. The metrics list in each category are ranked in descending order of their frequencies. There are 85 metrics[2] in total. For the metrics derived from issue reports, we use attribute name to indicate the related metrics. For example, we use Summary to indicate metrics related to the summary such as its length and textual sequences.

Studies focused on predicting direct effort used metrics from project category, participant category, and community category, which were all related to predict objectives of manpower.

Studies about maintenance activity effort prediction had a wide range of metrics selection in terms of all the five categories. This is mainly due to the diversity of the maintenance activity type such as code review, duplication identification, and bug fixing. Studies on bug fixing time prediction mainly use metrics from issue reports. However, human factors about the fixers are rarely selected, which can be used as powerful predictors to improve the performance of bug fixing time prediction.

Metrics in participant category are mainly used by studies on individual contribution measurement. Rastogi *et al.* [33] contributed most by providing 11 metrics based on roles and performance of participants.

We considered metrics selection in each paper as a transaction and used Apriori Algorithm to obtain the frequent metric sets to figure out the most used metric combination. Since studies on bug fixing time account for most of the selected studies, we conducted the frequent itemsets mining on 14 papers under the topic of bug fixing time prediction. TABLE V shows the result itemsets with the corresponding support. For itemsets above 20% support, the largest frequent itemsets had 7 items, which were selected by 3 out of 14 papers. Since itemsets with 3 and 4 items are all subsets of itemsets with

[2]The detailed descriptions of these metrics can be found on: http://itechs.iscas.ac.cn/cn/membersHomepage/wuhong/metrics.html

TABLE IV
TYPE OF METRIC SET USED BY EACH TOPIC

Topic	Type of Metric Set
Predict indirect effort of the entire project	Project{Size, Task}, Changes{CC, Function}
Predict direct effort of the entire project	Project{Time, Commits, Developer}, Participant{Bug Collaborator}, Community{Contributor}
Predict effort of maintenance activity	Project{Size, Time, Task, Bugs}, Changes{ALL}, Issue Report{ALL}, Participant{Bug Reporter, Bug Collaborator}, Community{workload}
Measure individual contribution	Participant{ALL}, Community{Activity}
Guidelines and discussion	Project{Size}, Changes{CC}

5 items, which means the two results are identical. Therefore, we listed itemsets with 5 items only. Itemsets with 6 items also had the same situation with itemsets that had 7 items so we listed itemsets with 7 items.

The results showed that priority of bug has the highest support with nine studies used it in their estimation models. Priority means how fast it needs to be fixed, which indicates that bug reports that have high priority might be assigned to expert or efficient programmers in order to get fixed faster. Therefore, studies use priority as an indicator to predict bug fixing time. Severity was also commonly used in estimation. Severity means how severe it is affecting the functionality, which relates to the resource assigned to fix the issues. These two metrics are all closely related to the scheduling and resource allocation for bug fixing.

D. RQ3: Studied Projects and Dataset

TABLE VII shows the frequency of the OSS projects used in the selected studies. Mozilla was the most frequently used OSS project in maintenance effort estimation studies. There were six studies selected Mozilla as the studied case. The source code of Mozilla is maintained through CVS, and the issues are managed through Bugzilla. Eclipse was the second commonly selected OSS projects. Many studies also used operation systems as studied cases, such as Linux, Ubuntu, FreeBSD, and Android. Since estimation methods derived from a larger dataset are likely to be more accurate and significant, we investigated the sizes of the datasets that were

TABLE V
FREQUENT METRIC SETS USED IN BUG FIXING TIME PREDICTION

#Item	Metric Set	Support(%)
1	{Priority}	64.29
2	{Priority, Severity}	50
5	{Priority, Severity, Milestone,CC-List, Comments}	35.71
7	{Priority, Severity, Milestone,CC-List, Comments,OS/Platform, Reporting-time}	21.43
	{Priority, Severity, Milestone,CC-List, Comments, Assignee-email, Component}	21.43
	{Priority, Severity, Milestone, CC-List, Comments, Component, OS/Platform}	21.43

TABLE VI
METRICS USED FOR OSS MAINTENANCE EFFORT ESTIMATION

Category	Content	Metric	Frequency
Project	Size	Total_Line, LOC-g, Files, Slice, Total Module, Kernel modules, Kernel_Line, Files-g	5
	Time	Development time, Project fix-time	2
	Task	Task type	2
	Bugs	Fixed bugs	1
	Commits	Amount of commits	1
	Developer	Administrator	1
Changes	CC	CC-based metrics	5
	Function	Function-based metrics	3
	LOC	Code churn, Non-commenting code in changes, ELINE	2
	File	Delta, Files-c	1
	Class	classes in changes	1
	Operator	COPE	1
	Parameter	PCOUNT	1
	Return	RPOINT	1
Issue report	Stakeholder	CC List, Assignee email, Reporter email, Participants, Triager email, Developers, Role Pattern	12
	Info	Priority, Severity, Comments, Component, Description, Milestone, OS/platform, Status, Blocking, Dependable, Resolution, Version, Keywords, Product/Project, Summary, Title, Type, Amount of code in description, Attachments, QA contact, Reproduce	10
	Time	Reporting Time, Open hours, Priority fix-time, Severity fix-time, AssignTime	3
	Similarity	Similarity	3
	Process	Changed attributes	2
	Quality	Readability	1
Participant	Bug Collaborator	Contribution Index (CI), Cumulative Comment Score (CCS), Potential Contributor Index (PCI), Developer's expertise, Developers' time devotes to the project, Merged commits, Assignee Experience, Average past issue lead time	5
	Bug Reporter	Reporter Experience,Degree of Customer Eccentricity (DCE), Duplicate Peers Identied, Reporting Recency, Reporter fix-time, Reporter popularity, Reporter type, Status of Reported bug Index (SRI)	4
	Bug Triager	Accuracy of Reassignment Index (ARI),Reassignment Effort Index (REI)	1
	Bug Owner	Deviation from Median Time to Repair (DMTTR), Priority Weighted Fixed Issues(PWFI), Specialization and Breadth Index(SBI)	1
Community	Contributor	Active programmers	3
	Activity	Community activity	2
	Workload	Open bugs	2

used in the selected studies. Each dataset contains a certain amount of entries or transactions which are identified by bug id along with related metrics. We treated the number of entries/transactions in each dataset as the size. We then categorized the dataset according to the quartile as follows:

- Tier 1: less than 567 entries (Q1)
- Tier 2: between 567 entries OR 4289 entries (Median)
- Tier 3: between 4289 entries OR 14000 entries (Q3)
- Tier 4: more than 14000 entries

Fig. 4 shows the number of studies categorized by datasets size. 21% (6 out of 29) papers used datasets more than 14000 entries while 24% papers used datasets less than 567 entries. With the large number of developers and projects in OSS community, adopting estimation models built on small datasets could threaten the validity of the results.

Most selected studies under the topic of predicting maintenance activity time used bigger datasets but most studies under the topic of predicting maintenance effort of entire projects both directly and indirectly used smaller datasets. This could be a potential threat to generalizing the results for other OSS projects.

E. RQ4: Estimation Methods

In this section, we investigate the different estimation methods that were used in the selected studies along with the strength and weakness of these methods. TABLE IX shows the details of these estimation methods and their mapping to

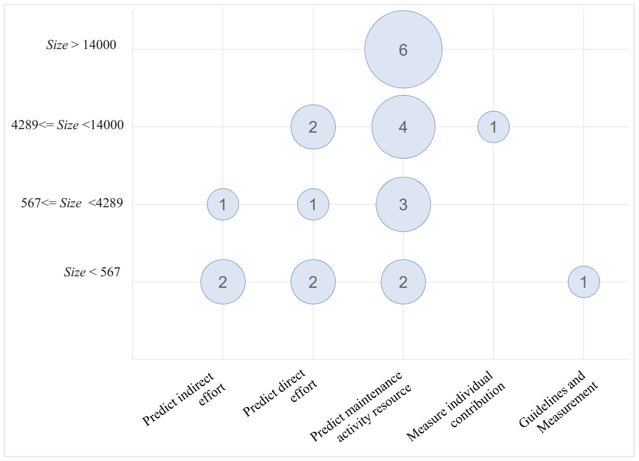

Fig. 4. The number of studies by dataset size

TABLE VII
OSS PROJECTS USED IN THE SELECTED PRIMARY STUDIES

OSS Project	Frequency	Paper
Mozilla	6	[24],[3],[14],[32],[36],[2]
Eclipse	6	[32],[24],[30],[14]
Jboss	3	[38],[16],[37],[29],[2]
Linux	3	[40],[4],[11]
GNOME	3	[21],[14],[2]
Soureforge	2	[12],[20]
Openstack	1	[35]
Filezilla, WxWidgets, Lighttpd	1	[23]
Google Chromium Project	1	[33]
Hibernate, Mule, Spring	1	[37]
Google Android	1	[25]
Ubuntu	1	[6]
FreeBSD	1	[10]
Codehaus	1	[16]
Thunderbird, Add-on SDK	1	[36]
QT,Qpid,Geronimo	1	[13]

the corresponding studies. Among the listed methods, linear regression model was the most frequently used by studies that predicted indirect effort of an entire OSS project. Manpower function method was widely used to predict direct effort of OSS projects. Decision tree, Logistic regression, and Naive Bayes were the three most frequently used methods for predicting effort of maintenance activity. We noticed that *n-fold Cross-Validation* ($n > 1$) was the dominant validation method used in the selected studies. We also extracted and investigated the strengths and weaknesses of different prediction methods as well as the metrics/features used in the method, as illustrated in TABLE IX.

The results show that estimation models for entire projects adopted a diversity of types of metrics while use linear model or classification methods. The estimation models for maintenance activity are opposite. The issue-report related metrics are the main metrics while these models adopted a diversity of methods.

TABLE VIII
ESTIMATION PERFORMANCE FOR PREDICTING INDIRECT EFFORT OF THE ENTIRE PROJECT

Source	PRED (25)	PRED (50)	MMRE	Estimation Context			
				Dataset Size	*Project*	*Target*	*Estimation Method*
[40]	33	60	–	106	Linux Kernel	KLOC	Linear Regression
[40]	53	87	–	106	Linux Kernel	#Module	Linear Regression
[4]	43.9	64.9	44.2	783	Linux Kernel	LOC	Linear Regression

F. RQ5: Estimation Accuracy

Furthermore, we dived into each selected study and examined their evaluation methods and evaluated results. 20 out of the 29 selected paper presented some sort of evaluation methods, whether mathematical or descriptive[3].

Indirect effort prediction of the entire project. Since OSS projects do not have complete maintenance records, there is no direct data for maintenance effort. Yu [40] and Alomari [4] used size-related metrics to indirectly denote the maintenance effort, and then used linear regression method to predict. TABLE VIII shows their estimation accuracy as well as their estimation context. MMRE[4], PRED(25)[5], PRED(50)[6] were the three accuracy metrics used in these selected studies. The average PRED(25) is 43.3% and the average PRED(50) is 70.6%. Only one paper reported MMRE with 44.2. The result shows that paper [40] has the highest accuracy on the project of Linux Kernel in the model of E_Module. Another evaluation method used was descriptive statistics for effort estimations with statistical significance (p-value) [23].

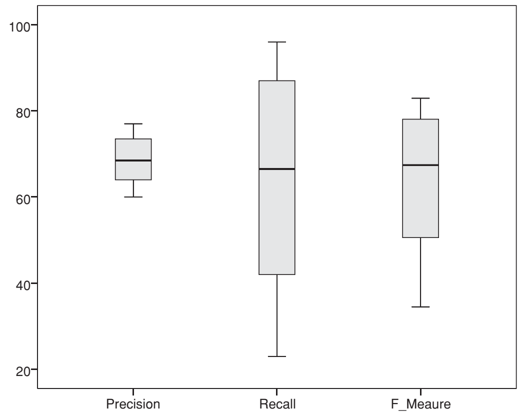

Fig. 5. Performance Distribution of effort prediction on identifying duplicates

Direct effort prediction of the entire project. Several studies used person-month to predict direct effort of the OSS project. Due to the difficulty of obtaining the actual effort data in OSS projects, there were multiple indirect evaluation methods used in papers under this topic, including verifying 3 hypotheses by regression analysis (p-value) [12], observational results [11], and number of commits performed by predicted full-time developers, which consists of a large fraction [35].

Effort Prediction of maintenance activity. There was only one paper that provided an effort estimation model for peer code review, and the model was evaluated through descriptive statistics and distribution [25]. Rakha *et al.* [32] studied the needed effort for duplicate issue identification on four OSS projects. The distribution of prediction performance is shown in Fig.5. The precision varies in a small range from 60% and 80%, while the recall largely ranges from 25% to 95%.

Selected studies on bug fixing time prediction used PRED(25), Accuracy[7], Precision, Recall, and F-Measure as the accuracy metrics. We further looked into the average performance of different estimation methods. As shown in Fig.6, *KNN method has the highest value in terms of PRED(25), Logistic regression method has the highest value both in Accuracy, Precision, Recall, and F-Measure.*

[3]All the detailed information for estimation accuracy can be found: http://itechs.iscas.ac.cn/cn/membersHomepage/wuhong/accuracy.html

[4]Mean Magnitude of Relative Error
[5]Percentage of predictions within 25% of the actual result
[6]Percentage of predictions within 50% of the actual result
[7]Percentage of correct predictions in both positive and negative objectives

TABLE IX
ESTIMATION METHODS

Topic	Sub-topic	Paper ID	Estimation Method	Metrics	Strength & Weakness
Indirect effort prediction of the entire project	Source code based estimation	[40], [4]	Linear regression model	Size,Task	S: Capable of dealing with new projects without historical data W: Rely on factors derived from closed-source projects
	Processes-based estimation	[23]	Classification	CC, Function	S: Intuitive W: Cannot estimate projects without historical data
Direct effort prediction of the entire project	People-based	[21], [12], [20]	Manpower function	Time, Participant, Community	S: Helpful for quantitatively modeling direct effort for long history projects in forms of person-month W: Cannot validate the correctness due to lack of effort data is OSS
		[35]	Classification	Participant,Community	S: Helpful for predicting direct effort in forms of time slot W: The accuracy heavily rely on the results of full-time developer classification
	Activity-based	[11], [5]	Linear model	Commit,Community	S: Reasoning similar to human problem solving approach W: Cannot apply to OSS projects that use CVS or Subversion repository
Effort prediction of maintenance activity	Effort for peer code review	[25]	Non-linear equation	CC, Function, Class, LOC	S: Can deal with patches in different complexity level W: Rely on factors derived from historical data
	Effort for duplicateissues identification	[32]	Classification model	Issue-report related metrics, Participant, Community	S: Can deal with the delayed issue reports W: Prediction performances rely on the pre-defined threshold
	Bug fixing time	[6], [10]	linear regression	Issue-report related metrics	S: Can predict correction time in days W: Accuracy is limited
		[24], [30], [14], [13], [10]	Decision tree	Bugs,Time,Community Issue-report related metrics, Participant,	S: Results can be explained W: Prediction performances rely on both bins and positive objective
		[3]	Support Vector Machine (SVM)	Size,CC,Function, File,LOC,Participant	S: Can predict correction time in days W: Rely on the hypothesize that real effort is distributed evenly
		[38], [16]	K-Nearest Neighbor (KNN)	Issue-report related metrics	S: Ease and flexibility of use W: The accuracy of text similarity will affect the results
		[36]	Apriori & K-means	Issue-report related metrics	S: Results can be easily understand W: Cannot deal with issues that have new collaborators
		[30], [29], [13], [10]	Logistic Regression	Issue-report related metrics Participant	S: Can be easily implemented W: May not work well with short history projects
		[30], [2], [10]	Naïve Bayes	Issue-report related metrics	S: Capable with small dataset W: Rely on historical quartile to determine the target category
		[26]	Distribution Functions	Task	S: Can be easily understand and implement quickly W: Cannot apply to OSS projects that have little history data
		[27]	Average weighted similarity	Issue-report related metrics	S: Intuitive and easy to understand W: Cannot predict correctly on issue reports with low similarity

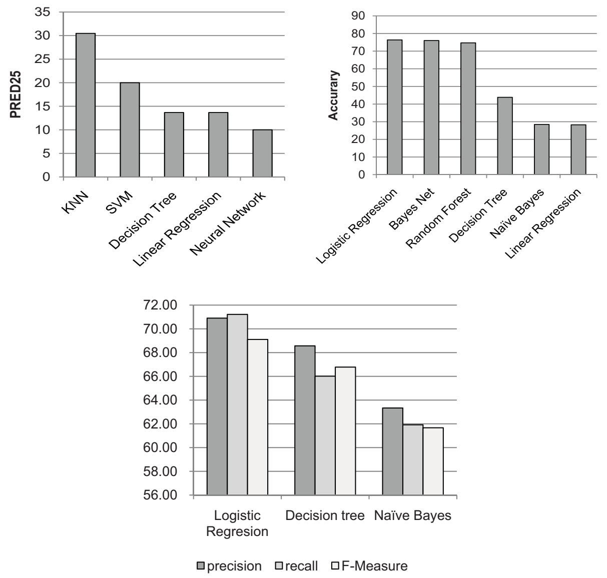

Fig. 6. Average performance of different bug fixing time prediction methods

IV. DISCUSSION AND FUTURE DIRECTIONS

The findings of this systematic literature review for OSS maintenance estimation provide a set of useful organizational results for researchers and OSS teams to better understand the importance of OSS maintenance planning and how to estimate maintenance effort.

New evaluation methods are needed to validate the correctness of these estimation methods. With the growth of more companies developing or collaborating with OSS projects, estimating maintenance effort has become a major interest. More researchers have been focusing on improving the estimation towards the direct effort of OSS projects from both people and activity aspects by developing maintenance effort estimation methods. However, since most OSS projects lack of complete development records and actual effort data, it is very difficult to evaluate and validate the results of these methods by comparing the estimated results with the actual effort. This can be a significant threat to these estimation methods and raises the risks to effectively validate of their results. This is an issue where we need new evaluation methods that can validate the correctness of these estimation methods.

Maintenance cost estimation models of OSS projects are different with general software system. Nguyen [28] surveyed 17 maintenance cost estimation models for general software systems in aspects of phase-level models, release-level models, and task-level models. These models can be applied to estimate the cost of the operation of a software system after it has been delivered, which is important for the trade-off analysis and making investment decisions. He reported the fact that source code based metrics are the main metrics for effort estimation models. However, maintenance cost estimation models of OSS projects are quite different. Various metrics are used in OSS maintenance cost estimation models including project-level metrics, changes related metrics, issue-report related metrics, metrics for participants and community.

Studies that can quantitatively infer OSS maintenance effort from size-related metrics are needed. To mitigate the difficulty of acquiring actual effort data from incomplete development records, Yu *et.al* [40][41][23] focused on predicting the size-related metrics to indirectly estimate the maintenance effort. The strong correlation between size-related metrics and the actual effort has been confirmed in closed-source projects [28]. However there still exists a gap between size-related metrics and time-aware effort for maintaining OSS projects. There is a need for studies that can quantitatively infer OSS maintenance effort from size-related metrics. Furthermore, the effort drivers used in general maintenance effort estimation models can serve as an example to improve OSS maintenance effort estimation. For example, Nguyen [28] developed an extension to COCOMO II [9] size and effort estimation models to capture various characteristics of software maintenance in general through a number of enhancements to the COCOMO II models to support the cost estimation of software maintenance. Some effort drivers such as DATA (Database size), CPLX (Product Complexity), and PVOL (Platform Volatility) in his study might contribute greatly to OSS maintenance effort estimation.

It will be worthwhile to explore the capability model for OSS developers. Since most OSS projects rely on task or issue tracking systems to maintain the projects, recognizing the time of specific maintenance tasks can provide better decision support for task assignment as well as OSS project management. A large amount of studies are devoted to predicting bug fixing time while a small amount focused on other activities such as code review and duplication identification. These studies commonly used metrics from source code changes and issue reports as predictors, which indicate that the prediction results are basically related to the characteristics of the tasks. There are two kinds of targets among these predictions. One is the numerical days of an activity, evaluated by PRED(25) or PRED(50). Another is the bins of categorical time evaluated by accuracy, precision, recall, or f-measure. The prediction results for numerical days are not very satisfying, while the results of categorical bins are relatively high. In OSS projects, the time recorded on issue tracking system and repository may not correlate with the actual effort because the developers are voluntary and self-determined when implementing the tasks. It will be worthwhile to explore the capability model for OSS developers and consider the developer-related metrics to be one of the sources of predictors, as an opportunity to improve the prediction results.

Individual contribution and performance measurement also has been receiving attention. Gousios [15] defined a contribution ratio by considering various type of parameters from the OSS community, and Rastogi *et al*. defined the contribution in terms of four different roles of stakeholder. Since the importance of contribution measurement has been realized, it might be a promising research topic in the coming years.

V. Threats to Validity

Internal Validity: Although the five electronic databases we used for the automatic search have comprehensively included existing literature in the research area, it is not possible to guarantee that all relevant primary studies were selected during the search process. This threat could be mitigated by performing the snowballing technique to find more relevant studies which might be missed during the automatic search process. According to the defined exclusion criteria, gray literature was not included in this study, because gray literature is particularly hard to identify. The inclusion of gray literature could have further increased the validity of our findings.

External Validity: 29 primary studies were selected for conducting this systematic literature review study, which might limit the generalizability of the results. We plan to keep adding new publications to this literature review to mitigate this threat.

VI. Conclusions

In this paper, we surveyed the state of the art in the field of maintenance effort estimation for open source software by conducting a systematic literature review. An extensive search was performed in order to find primary studies, which included searching through five electronic databases and s-nowballing sampling. Out of 3,312 studies found in the electronic databases, 29 primary studies were selected. Data was extracted from these studies and then synthesized and evaluated against a set of defined research questions. Our results showed that the commonly used OSS maintenance effort estimation methods were actual effort estimation and maintenance activity time prediction; the most commonly used metrics and factors for actual effort estimation were source code measurements and people related metrics; the most commonly mentioned activity for maintenance activity time prediction was bug fixing. Accuracy measures and cross validation was used for validating the estimation models. Based on our observations, we pinpointed the opportunities for future studies, as well as the improvements to existing works. We believe that our findings may serve as a good reference for prospective researchers and practitioners in the field of OSS maintenance effort estimation. In the future work, we plan to address the in-depth investigation of estimation accuracy by applying the estimation models to one typical OSS database, and identify benchmarks.

Acknowledgments

This material is based upon work supported by National Natural Science Foundation of China No.91318301, 61432001, 91218302, 61303163. It is also supported by the U.S. Department of Defense through the Systems Engineering Research Center (SERC) under Contract H98230-08-D-0171. SERC is a federally funded University Affiliated Research Center managed by Stevens Institute of Technology. It is also supported by the National Science Foundation grant CMMI-1408909, Developing a Constructive Logic-Based Theory of Value-Based Systems Engineering.

References

[1] Software engineering – software life cycle processes – maintenance. (ISO/IEC 14764:2006), 2006.

[2] W. Abdelmoez, M. Kholief, and F. M. Elsalmy. Bug fix-time prediction model using naïve bayes classifier. In *Computer Theory and Applications (ICCTA), 2012 22nd International Conference on*, pages 167–172. IEEE, 2012.

[3] S. N. Ahsan, J. Ferzund, and F. Wotawa. Program file bug fix effort estimation using machine learning methods for oss. In *SEKE*, pages 129–134. Citeseer, 2009.

[4] H. Alomari. A slicing-based effort estimation approach for open-source software projects. *International Journal of Advance Computational Engineering and Networking (IJACEN)*, 3(8):1–7, 2015.

[5] J. J. Amor, G. Robles, and J. M. Gonzalez-Barahona. Effort estimation by characterizing developer activity. In *Proceedings of the 2006 international workshop on Economics driven software engineering research*, pages 3–6. ACM, 2006.

[6] P. Anbalagan and M. Vouk. On predicting the time taken to correct bug reports in open source projects. In *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, pages 523–526. IEEE, 2009.

[7] J. Asundi. The need for effort estimation models for open source software projects. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 1–3. ACM, 2005.

[8] D. Badampudi, C. Wohlin, and K. Petersen. Experiences from using snowballing and database searches in systematic literature studies. In *EASE '15: Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, pages 17:1–17:10, 2015.

[9] B. W. Boehm, R. Madachy, B. Steece, et al. *Software cost estimation with Cocomo II with Cdrom*. Prentice Hall PTR, 2000.

[10] G. Bougie, C. Treude, D. M. German, and M.-A. Storey. A Comparative Exploration of Freebsd Bug Lifetimes. In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pages 106–109. IEEE, 2010.

[11] A. Capiluppi and D. Izquierdo-Cortázar. Effort estimation of floss projects: a study of the linux kernel. *Empirical Software Engineering*, 18(1):60–88, 2013.

[12] E. Capra, C. Francalanci, and F. Merlo. The economics of community open source software projects: an empirical analysis of maintenance effort. *Advances in Software Engineering*, 2010, 2010.

[13] N. Duc Anh, D. S. Cruzes, R. Conradi, and C. Ayala. Empirical Validation of Human Factors in Predicting Issue Lead Time in Open Source Projects. In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, page 13. ACM, 2011.

[14] E. Giger, M. Pinzger, and H. Gall. Predicting the fix time of bugs. In *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering*, pages 52–56. ACM, 2010.

[15] G. Gousios, E. Kalliamvakou, and D. Spinellis. Measuring developer contribution from software repository data. In *Proceedings of the 2008 international working conference on Mining software repositories*, pages 129–132. ACM, 2008.

[16] A. Hassouna and L. Tahvildari. An effort prediction framework for software defect correction. *Information and Software Technology*, 52(2):197–209, 2010.

[17] M. Jorgensen and M. Shepperd. A systematic review of software development cost estimation studies. *Software Engineering, IEEE Transactions on*, 33(1):33–53, Jan 2007.

[18] M. Jorgensen and M. Shepperd. A systematic review of software development cost estimation studies. *Software Engineering, IEEE Transactions on*, 33(1):33–53, 2007.

[19] B. Kitchenham and S. Charters. Guidelines for performing systematic literature reviews in software engineering. In *Technical report, Ver. 2.3 EBSE Technical Report. EBSE*. 2007.

[20] S. Koch. Effort modeling and programmer participation in open source software projects. *Information Economics and Policy*, 20(4):345–355, 2008.

[21] S. Koch and G. Schneider. Effort, co-operation and coordination in an open source software project: Gnome. *Information Systems Journal*, 12(1):27–42, 2002.

[22] R. G. Kula, K. Fushida, N. Yoshida, and H. Iida. Micro process analysis of maintenance effort: an open source software case study using metrics based on program slicing. *Journal of Software: Evolution and Process*, 25(9):935–955, 2013.

[23] R. G. Kula, K. Fushida, N. Yoshida, and H. Iida. Micro process analysis of maintenance effort: an open source software case study using metrics based on program slicing. *Journal of Software: Evolution and Process*, 25(9):935–955, 2013.

[24] L. Marks, Y. Zou, and A. E. Hassan. Studying the fix-time for bugs in large open source projects. In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, page 11. ACM, 2011.

[25] R. Mishra and A. Sureka. Mining peer code review system for computing effort and contribution metrics for patch reviewers. In *Mining Unstructured Data (MUD), 2014 IEEE 4th Workshop on*, pages 11–15. IEEE, 2014.

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

[27] N. K. Nagwani and S. Verma. Predictive data mining model for software bug estimation using average weighted similarity. In *Advance Computing Conference (IACC), 2010 IEEE 2nd International*, pages 373–378. IEEE, 2010.

[28] V. Nguyen. *Improved size and effort estimation models for software maintenance*. PhD thesis, University of

Southern California, 2010.

[29] M. Ohira, A. E. Hassan, N. Osawa, and K. Matsumoto. The impact of bug management patterns on bug fixing: A case study of eclipse projects. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pages 264–273. IEEE, 2012.

[30] L. D. Panjer. Predicting eclipse bug lifetimes. In *Proceedings of the Fourth International Workshop on mining software repositories*, page 29. IEEE Computer Society, 2007.

[31] K. Petersen, S. Vakkalanka, and L. Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18, 2015.

[32] M. S. Rakha, W. Shang, and A. E. Hassan. Studying the needed effort for identifying duplicate issues. *Empirical Software Engineering*, pages 1–30, 2015.

[33] A. Rastogi, A. Gupta, and A. Sureka. Samiksha: mining issue tracking system for contribution and performance assessment. In *Proceedings of the 6th India Software Engineering Conference*, pages 13–22. ACM, 2013.

[34] M. Riaz, E. Mendes, and E. Tempero. A systematic review of software maintainability prediction and metrics. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 367–377. IEEE Computer Society, 2009.

[35] G. Robles, J. M. González-Barahona, C. Cervigón, A. Capiluppi, and D. Izquierdo-Cortázar. 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*, pages 222–231. ACM, 2014.

[36] M. Sharma. The way ahead for bug-fix time prediction. In *3rd International Workshop on Quantitative Approaches to Software Quality*, page 33.

[37] E. Shihab, Y. Kamei, B. Adams, and A. E. Hassan. Is lines of code a good measure of effort in effort-aware models? *Information and Software Technology*, 55(11):1981–1993, 2013.

[38] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller. How long will it take to fix this bug? In *Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 1. IEEE Computer Society, 2007.

[39] C. Wohlin, R. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Wesslen. *Experimentation in Software Engineering*. Springer Science and Business Media, 2012.

[40] L. Yu. Indirectly predicting the maintenance effort of open-source software. *Journal of Software Maintenance and Evolution: Research and Practice*, 18(5):311–332, 2006.

[41] H. Zeng and D. Rine. Estimation of software defects fix effort using neural networks. In *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*, volume 2, pages 20–21. IEEE, 2004.