Evolution Prediction and Process Support of OSS Studies: A Systematic Mapping

Article · May 2017

DOI: 10.1007/s13369-017-2556-5

2 authors:

Ghulam Rasool
COMSATS University Islamabad
32 PUBLICATIONS **166** CITATIONS
SEE PROFILE

Nancy Fazal
University of Eastern Finland
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Arab J Sci Eng (2017) 42:3465–3502
DOI 10.1007/s13369-017-2556-5

RESEARCH ARTICLE - COMPUTER ENGINEERING AND COMPUTER SCIENCE

Evolution Prediction and Process Support of OSS Studies: A Systematic Mapping

Ghulam Rasool[1] · **Nancy Fazal**[1]

Received: 3 November 2016 / Accepted: 11 April 2017 / Published online: 4 May 2017

Abstract Open source software (OSS) evolution is an important research domain, and it is continuously getting more and more attention of researchers. A large number of studies are published on different aspects of OSS evolution. Different metrics, models, processes and tools are presented for predicting the evolution of OSS studies. These studies foster researchers for contemporary and comprehensive review of literature on OSS evolution prediction. We present a systematic mapping that covers two contexts of OSS evolution studies conducted so far, i.e., OSS evolution prediction and OSS evolution process support. We selected 98 primary studies from a large dataset that includes 56 conference, 35 journal and 7 workshop papers. The major focus of this systematic mapping is to study and analyze metrics, models, methods and tools used for OSS evolution prediction and evolution process support. We identified 20 different categories of metrics used by OSS evolution studies and results show that SLOC metric is largely used. We found 13 different models applied to different areas of evolution prediction and auto-regressive integrated moving average models are largely used by researchers. Furthermore, we report 13 different approaches/methods/tools in existing literature for the evolution process support that address different aspects of evolution.

Keywords OSS · Open source · FLOSS · Systematic mapping · Evolution prediction · OSS evolution

✉ Ghulam Rasool
grasool@ciitlahore.edu.pk

[1] Department of Computer Science, COMSATS Institute of Information Technology, Lahore, Pakistan

1 Introduction

"Software Evolution" is the phenomena of software change over years and releases, since inception (concept formation) to the decommissioning of a software system [1]. Software continues to evolve after shipment of its first version, and it continuously evolves because requirements emerge when it is used and errors are fixed. Therefore, a key issue that organizations face relates to managing and implementing changes to their software systems as they have huge investments on their key assets. According to Ratzinger et al. [43], improving software maintenance and development is time consuming and costly task, with direct financial impact. Large and well-known companies such as Microsoft and IBM reported multi-billion dollar expenses on the development of software applications in every single year. Many estimates show that the software evolution and its maintenance cost is at least 50–90% or even more than that of overall cost of a software system [24,25]. In order to reduce cost, managers and developers should take into consideration the driving factors of software evolution. Active steps should be taken to make transformations easy and ensuring that software systems do not crumble for successful evolution [3].

Lehman and his colleagues presented well-known studies on software evolution 35 years ago in the mid-1970s. These studies have resulted in eight laws of software evolution. These laws were the result of empirical studies of wide-scale software systems evolution [6]. Laws of software evolution were actually developed in the perspective of closed source software systems, e.g., software applications that were running on IBM's OS/360.[1] Large and mature open source projects came later. With the evolution of open

[1] http://wiki.cs.pdx.edu/cs562-winter2013/schedule/CS562_Shaun_Brandt_slides.pdf.

source systems, researchers got access to an extensive number of continually evolving software applications for research and it led to a way of new concerns in empirical studies of software system evolution [1]. Many commercial products are developed using open source software components, and the trend of adopting open source components is escalating. According to a survey report, Apache is used by more than 58% web servers [26]. Linux is the best example of open source software that shattered monopoly of Microsoft. Nine out of 10 super computers of world run Linux.[2] The prominent companies such as Google and large public bodies, like the French Ministry of Finance, base their information technology infrastructures on FOSS components [65]. It is also reported through the empirical data of 1000 US-fortunate companies that adaption of OSS components by large US companies is significant and it is continuously increasing [2]. This has lead to a renewed interest in the empirical study of software evolution [1].

To espouse an OSS component effectively, organizations need knowledge of the project development, composition and possible risks associated with its use, due to its unconventional and complex development process and evolution history. This, in turn, calls for building reliable prediction models and methods supporting error prediction, measuring maintenance effort, predicting refactoring, size, quality attributes, change propagation and cost of OSS projects [47]. These prediction methods and models help software project managers to direct their resources into the areas with the highest impact on the bottom line, supporting rational and timely resource allocation to the evolution process. It also enables adequate process control in maintenance activities which guarantee that software does not decay and ensures successful evolution. Evolution process support is also highly desirable because managing the software evolution for large open source software applications is a major challenge. Explicit project planning, lack of documentation, distributed development teams, frequent turnover of volunteers make open source software applications hard to maintain [61]. A large number of prediction models, methods, tools and metrics are presented by researchers in last two decades for evolution prediction and evolution process support of OSS studies that demand systematic mapping to categorize and summarize different aspects of OSS studies.

Open Source Software (OSS) evolution is a significant research domain, and it is continuously gaining momentum over years among research community [5,6]. The ability to evolve software rapidly and reliably is a major challenge for software engineering and software process managers also need to be able to understand the driving factors of evolution. There have been a large number of studies published on OSS evolution because with the emergence of the OSS paradigm, researchers are provided with the wealth of data for open source software evolution analysis. These efforts have resulted in an ample set of research results for which there is a need for up-to-date comprehensive overviews and literature surveys that summarize and structure the existing body of knowledge. To the best of our knowledge, there exist four SLRs (Systematic Literature Reviews) [5,6,20,21] addressing OSS evolution studies in different contexts. These SLRs are very specific and focus on limited aspects of open source evolution studies.

The systematic mapping study presented in this paper is different from previous SLRs based on scope, research questions, timeframe and comprehensiveness. The scope of our study is generic as compared with previous SLRs that cover very specific topics on OSS evolution studies. For example, SLRs [5,20] are from same group of authors and focus of authors is to identify and structure research on open source software evolution. The authors analyzed different metrics, methods and datasets used for the evolution of open source projects. SLR [6] is a conference paper with 41 primary studies. The authors classify research on open source projects into four categories: software trends and patterns, evolution process support, evolvability characteristics, and examining OSS at software architecture level. They list different metrics used for the evolution of OSS studies but they do not discuss relationship between these metrics and evolution features of OSS studies. The scope of SLR [21] is limited to fault prediction studies with specific focus on metrics, methods and datasets. The authors conclude that method level metrics and machine learning algorithms are largely used by the researchers for fault prediction of open source projects. Our systematic mapping study is generic in scope and cover evolution prediction and evolution process support for open source projects. Secondly, we focus on two generic research questions regarding OSS evolution, i.e., OSS evolution prediction and OSS evolution process support. The evolution process support is not discussed by previous SLRs. The comparison of research questions of our systematic mapping with research questions of the state-of-the-art SLRs is presented in Sect. 2. Thirdly, the timeframe for our systematic mapping is most contemporary which include studies from 2000 to 2015. Fourthly, all previous SLRs summarize/discuss metrics used for evolution prediction but they did not discuss relationship between these metrics and evolutionary features of OSS projects. Finally, we applied automated search and snowballing technique to avoid chances of missing relevant studies. SLRs [6,20,21] applied only automated search using different keywords to select primary studies. The authors in [6] also applied automated search method, and then they manually verified the studies selected during the automated search.

[2] https://www.ibm.com/developerworks/community/blogs/6e6f6d1b-95c3-46df-8a26b7efd8ee4b57/entry/attractions_and_challenges_of_open_source15?lang=en.

The paper is organized as: Related work is discussed in Sect. 2. The methodology adopted for systematic mapping is presented in Sect. 3. Results derived from study are presented and discussed in Sect. 4. Validity threats are discussed in Sect. 5. Section 6 concludes the whole study and suggests future outline.

2 Related Work

Initially, the study of software evolution was pioneered by Lehman and Belady on the releases of IBM OS/360 (operating system) [9]. It opened the way for many other works [10,32], where Lehman's laws of software evolution were devised extended, and modified. A comprehensive systematic review is presented on evolutions of Lehman's laws of software evolution [22]. There are controversial studies presented on Leman's laws of software evolution by different researchers in last 15 years. The major focus of authors in this review is to highlight how and when the laws and field of software evolution is evolved. Moreover, the authors also discuss the validity of laws of software evolution and how they are perceived by the research community. For the region of OSS systems, numerous works explicitly deal with the theme of software evolution. Godfrey and Tu [32] analyzed the most well-known Linux operating system kernel. They measured its size in LOC (lines of code) since its release 1.0 and found that growth pattern follows super-linear rate. These results contradict the theory of software evolution, [23] which implies that system growth decline can be best modeled by using a linear or an inverse square rate. On the other side, Paulson et al. [34] make use of a linear approximation, and results find no difference in the growth pattern of OSS and proprietary software packages. They further analyzed three well-known OSS (Linux, Apache and GCC) and three proprietary systems. Robles et al. [13] reproduced the findings of Godfrey and Tu [32] using new datasets but they found similar results.

The first large study on OSS evolution presented by Capiluppi et al. [14] involved 406 projects. The authors implied more focus on 12 alive projects out of the set of 406 projects. They observed that 97% of projects do not change size or have been changed for less than 1% over 6 months. They also realized that number of modules grows, but stable value is found for evolving size in large and medium size of projects.

Robles-Martinez et al. [15] studied and analyzed the evolution of MONO (an open source implementation of .Net framework). They applied a methodology to their work on the basis of the utilization of datasets freely available in the form of the Concurrent Versions System (CVS) [16], source code control system, commits, authorship and size in Lines of Code (LOC). The findings of the authors suggest relatively different growth rate for studied modules, but generalization of their results for other models requires investigation. Nakakoji et al. [33] studied the OSS projects evolution and took an open view in further inspecting the evolution of related communities. By performing experiments on four case studies, they associated system evolution to the community evolution on the hierarchical basis of different versions of the system. They categorize the projects into three different types. The author in [7] presented a tremendous discussion on OSS evolution, with an evaluation of closed and open source project studies. On the basis of his analysis, he further identifies that the laws of software evolution not hold for super-linear growth in the software size. Further, he also emphasizes the significance of the accessibility of data on OSS projects that provide great aid for conducting further studies on software engineering and software evolution. An empirical study on the evolution of seven well-known OSS projects was performed by Guowu Xie et al. [3]. They investigate Lehman's evolution laws and OSS evolution facets beyond the framework of Lehman's law. Further, Young Lee et al. [18] also reported an empirical investigation of the evolution of JFreeChart, an OSS system. They concluded that the evolution of JFreeChart shows consistency with 1st, 2nd and 6th laws of software evolution proposed by Lehman. Another finding [13] also analyzed a number of OSS projects including Linux Kernel. Results showed that the exceptional occurrence of super-linearity, a linear growth pattern, is being followed in many systems. The results obtained were found different from the ones that exist in traditional studies of software evolution. Smith et al. [28] presented a study based on data from 25 open source projects, and they concluded that there is a relationship between size and complexity for long-time sustainability of open source software projects. Skoulis et al. [29] presented a study on database evolution of open source software and compared their findings with Lehman's Law of software evolution. The authors conclude that open source databases evolve over time with long period of calmness interrupted by different types of maintenance and a lack of complexity increase. Cosentino et al. [4] presented a study to highlight status of research on GitHub. The authors selected 93 papers and came up with a number of findings but the study is limited with only GitHub.

Summarizing the ample of research results, a systematic review [6] on OSS evolution studies analyzed 41 papers published till 2009 and classified the reviewed studies into four categories, i.e., OSS evolution trends and patterns, OSS evolution process support, evolvability characteristics and examining OSS evolution at software architecture level. The authors focused on the metrics that are used for OSS evolution measurement and analysis. Regarding the category of software trends and patterns, most papers focus on using different metrics to analyze OSS evolution over time. A few papers looked into the economic perspective. Regarding the category of evolution process support, different aspects that appear to have an impact on the OSS evolution process are covered. With the category of evolvability characteristics, determinism, understandability, modularity and complexity are addressed in the included studies. The authors found that minimum attention is paid to architecture evolution of software systems. Furthermore, the findings of another systematic review published in 2013[5] aimed at identification and structuring of research on the evolution of OSS projects. A set of 101 articles (21 journal and 80 conference articles) were selected for this review that address the facets, dimensions, research approaches, datasets and tools for metric data collection of OSS projects. Besides that, authors also addressed the portfolio of OSS projects analyzed for evolution studies, concerns about OSS evolution study trends, contributions made to analyze the evolution of software, evolution of organization or community, interdependency in the evolution of the software and organization and validation of research approaches and results of the articles.

To the best of our knowledge, there has not been any study conducted toward OSS evolutionary aspects identified in [6] except evolution prediction. The authors in [20] present a systematic literature review on open source prediction methods by focusing on both code and community dimension. The SLR focuses on four key research questions: focus of studies, datasets used for evolution prediction, methods used for prediction and metrics used for evolution prediction. According to [20], evolution prediction has been identified as the most focused research area. Finally, a systematic review on software fault prediction studies based on 74 primary studies is presented by authors in [21]. The focus of SLR was to evaluate method level metrics used for fault prediction. The authors concluded that method level metrics and machine learning algorithms are widely used by researchers for fault prediction. They also found that most papers on software fault prediction are published in journal of Transactions on software engineering, software quality journal, journal of systems and software and empirical software engineering journal. The summarized information about four discussed SLRs and comparison with our systematic mapping study is presented in Table 1.

3 Mapping Methodology

A systematic mapping is a method of identifying, categorizing and summarizing all available research results relevant to a particular research area [8,19,27]. Summarizing the state of the art, identifying the research gaps and providing an overview on a certain area is very helpful for existing and new researchers (e.g., MPhil and PhD students). Systematic mapping studies focus on generic and coarser grained overview of published results as compared to SLRs that focus on specific research questions and detailed analysis of results. A comparison between systemic maps and systemic reviews is presented by Petersen et al. [8]. The recent guidelines for systematic maps are also presented by Petersen et al. [27].

Figure 1 depicts the mapping process followed by us, and it is based on the guidelines developed by [8,17,27,30]. We start by developing the study protocol, which once accepted is followed by identifying research questions, search strategy, selection of relevant studies. The selected studies are used to extract, synthesize, map and report results.

3.1 Study Protocol

OSS evolution is an imperative research realm, and it is continuously getting more and more attention of researchers. A large number of published studies on OSS evolution resulted in an ample set of research results for which there is a need to summarize and structure the existing body of knowledge.

3.2 Research Questions

It is essential to formulate research questions for the systematic mapping study. Table 2 presents the research questions, description of research questions and motivation.

3.3 Search Strategy

The main reason for developing a search strategy is to go through a systematic process of finding studies which are related to research questions in the selected searched venues. Moreover, identifying primary studies should be unbiased [34]. We applied automated and manual search strategy to ensure that no studies related to our research questions are missed. The following steps are performed in search strategy:

1. Select search string
2. Select search venues
3. Define criteria to include or exclude a study
4. Define quality assessment criteria
5. Data extraction

3.4 Search String

Keywords are identified and noted based on some trial searches. The main reason for doing this trial search is due to the large number of keywords and terminologies used in software engineering and OSS research. We identified keywords that are relevant to the current research when formulating the search string. We also analyzed keywords used by previous SLRs for the selection of keywords. The time span of our research is 2000–2015. The search terms used for constructing search strings are: “open source software” OR “open

Table 1 Comparative overview of features of existing SLRs

References	Time- span	Primary studies	NCPS	Research questions	Focus
[6]	NM till 2009	41	26	NM	The aim of SLR is to provide overview of studies on open source software evolution with respect to growth, complexity and modularity metrics
[5]	Jan 2000–Jan 2013	101	25	Q1: Which facets of OSS projects were explored and what statistical distribution the articles have in those facets? Q2: What are the dimensions of OSS projects explored under each study facet? Q3: What are the research approaches followed in the studies? Q4: What are the datasets or data sources of OSS projects mostly exploited in evolution studies? Q5: What metric suits are evaluated and what tools are used for metric data collection? Q6: What is the portfolio of projects analyzed for evolution studies and what are their domains? Q7: Does the concern on "OSS evolution study" follow an increasing trend? Q8: What contributions are made in literature to analyze the evolution of software? Q9: What contributions are made in literature to analyze the evolution of organization or community? Q10: What contributions are made in literature to analyze the interdependency? Q11: How are the research approaches and results of the articles typically validated?	The focus of SLR is to identify and structure research on evolution of OSS projects
[20]	1980–Jun 2011	36	12	Q1: What are the main focuses/purpose of the study? Q2: What are the datasets of the OSS projects exploited in prediction? Q3:Which kind of methods are used in predicting OSS projects? Q4:Which kind of metrics are used in predicting OSS projects?	The focus of SLR is to analyze datasets, methods and metrics used for open source software evolution

Table 1 continued

References	Time- span	Primary studies	NCPS	Research questions	Focus
[21]	1990–2007	74	4	Q1: Which journal is the dominant software fault prediction journal? Q2: What kind of datasets are the most used for fault prediction? Q3: What kind of datasets are the most used for fault prediction after year 2005? Q4: What kind of methods are the most used for fault prediction? Q5: What kind of methods are the most used for fault prediction after year 2005? Q6: What kind of metrics are the most used for fault prediction? Q7: What kind of metrics are the most used for fault prediction after year 2005? Q8: What is the percentage of publications published after year 2000?	The focus of SLR was to classify studies with respect to metrics, methods, and datasets that have been used in fault prediction papers
Our Mapping Study	2000–2015	98	–	Q1: How has OSS evolution Prediction been addressed in the existing literature and applied in practice? Q2: What is the state of the art in OSS evolution research to support OSS evolution process support?	The focus of study is to map state-of-the-art research on OSS studies to metrics, methods, models, tools and processes used for evolution prediction and evolution process support of OSS studies

NM not mentioned, *NCPS* number of primary studies common with our Study

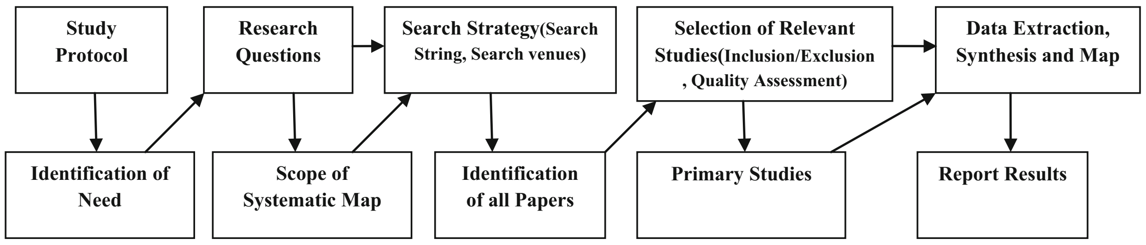
Study Protocol
Identification of Need
Research Questions
Scope of Systematic Map
Search Strategy(Search String, Search venues)
Identification of all Papers
Selection of Relevant Studies(Inclusion/Exclusion, Quality Assessment)
Primary Studies
Data Extraction, Synthesis and Map
Report Results

Fig. 1 Mapping process

source” OR “Libre software” OR “OSS” OR “F/OSS” OR “FLOSS” AND “Evolution” AND “Prediction”.

3.5 Searched Venues

The below-mentioned digital libraries are selected as search venues:

1. IEEE Xplore
2. ACM
3. Science Direct
4. Springer Link
5. ISI Web of Science
6. International Conference on Open Source Systems (2010 –2015)
7. Wiley Online Library (Journal of Software : Evolution and Process 2000–2015)

Three largest and most commonly used databases in the software engineering field are IEEE, ACM Digital Library and ISI Web of Knowledge. After referencing existing SLRs, we extended the search venues with Springer Link and Science Direct in order to get more reliable studies. Unlike existing SLRs, we have also included International Conference

Table 2 Research questions

Research questions	Description and motivation
RQ1: How has OSS evolution Prediction been addressed in the existing literature and applied in practice?	The OSS history data over time can be utilized to predict its evolution [6]. It is useful for organizations and software project managers interested for planning, effort estimation, predicting quality and stability, supporting rational and timely resource allocation to the evolution prediction metrics, models and tools
RQ2: What is the state of the art in OSS evolution research to support OSS evolution process support?	Free and Open Source Software (FOSS) systems are facing the issues of community-centric management such as it is not only driven by differences of current capability and environment demands but it also depends on security issues and frequent releases of components. Evolution management of these systems is crucial for the organizations. OSS community has proposed number of tools and methods to support the up gradation of these systems for which there is a need to summarize the existing research results and identify future directions

on Open Source Systems (ICOSS) and "Journal of Software: Evolution and Process" as they are explicitly dedicated toward Evolution studies. Since research results are indexed and cited in these databases, we opted to search in these databases by using a well-formulated search string.

Conducting the mapping study involves searching the identified digital libraries and databases using the search string. We used a reference management tool known as "Zotero" in which papers were automatically downloaded from databases into the Zotero[3] library. This step was automatic for each of the databases except "International Conference on Open Source Systems" (ICOSS) for which papers were manually downloaded and imported into the Zotero library. It resulted in a total of 1559 papers. After removing duplicates, 1124 papers were left. We reviewed title and abstract of each paper and applied inclusion/exclusion criteria. This step reduced the number of papers to 241 from which further 161 papers were discarded after full text scan. We searched out 86 papers after this step. We also applied snowballing method on the selected primary studies and

[3] https://www.zotero.org.

Table 3 Initial hits

Searched venues	Matches found
IEEE Xplore	$313 + 18 = 331$
ACM	$49 + 6 = 55$
Springer link	57
Science direct	82
ISI Web of science	725
ICOSS	108
Journal of Software: Evolution and Process	162
IEEE Transactions on Software Engineering	30
Others	9
Total:	1559

found 12 more relevant papers. In total, 98 papers including 35 journal, 56 conference and 7 workshop papers were selected for data extraction. The reported primary studies are shown in Appendix A. Table 3 depicts the initial matches found from the searched venues. Figure 2 explains selection of primary studies and stages involved in retrieving the studies.

3.6 Inclusion/Exclusion and Scope Determination

The inclusion and exclusion criteria are used to identify the studies that are relevant to the research questions. In this study, we included the studies published between 2000 and 2015. The following are the criteria for including a study.

The following criteria is used to select a paper:

1. Articles published between 2000 and 2015.
2. Is the article written in English?
3. Is the article available in full text?
4. Is the article peer reviewed?
5. The article presents/discusses/analyses metrics, models, process and tools used for evolution prediction of OSS studies (RQ1).
6. The article discusses OSS evolution process support (RQ2).

Articles in the controversial corners of Journals, Editorials, Workshops, Book chapters, Summaries of tutorials, Panels and Poster Sessions are excluded. We also excluded articles that are published as short papers on tools used for the evolution of open source projects.

The study selection criteria for reviewing primary studies are based on the selection criteria. The inclusion /exclusion criteria are used to select the articles that are more related to this study. This criterion was defined in order to get unique articles that are related to the study. After reading the title and abstract, primary studies are selected. When the deci-

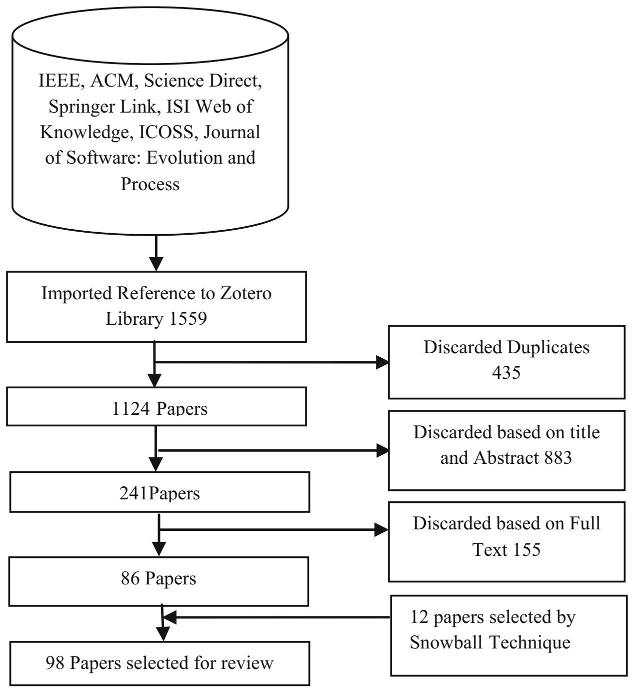
IEEE, ACM, Science Direct, Springer Link, ISI Web of Knowledge, ICOSS, Journal of Software: Evolution and Process
Imported Reference to Zotero Library 1559
Discarded Duplicates 435
1124 Papers
Discarded based on title and Abstract 883
241Papers
Discarded based on Full Text 155
86 Papers
12 papers selected by Snowball Technique
98 Papers selected for review

Fig. 2 Selection of primary studies

sion cannot be made after reading the title and abstract, the introduction and conclusion are further reviewed for the inclusion/exclusion purpose.

3.7 Study Quality Assessment

In addition to general inclusion/exclusion criteria, it is generally considered important to assess the "quality" of primary studies. It is a means of investigating whether quality differences provide an explanation for differences in the study results [11]. We used the method of checklist for assessing the quality of primary studies. The questions for checklist are formulated based on our two main research questions and data extracted through data extraction sheet given in Table 4. However, questions of checklist are developed ad hoc by the authors of this paper and their validation may be criticized. The checklist questions are given blow:

1. Q1:Does the article specifically consider OSS Evolution?
2. Q2: Does the article consider Evolution Prediction?
3. Q3: Does the article consider Evolution Process Support?

Table 4 Attributes for data extraction sheet

Attribute	Sub attribute	Brief description
General	Article type, Source, Year of Pub.	Conference, Journal, Book
	Article Name, Keywords, Author	Digital Library, Journal, Conferences
Focus of the study	Software Evolution	OSS evolution prediction, OSS evolution process support
OSS projects studied	OSS projects	List of OSS projects studied
	Domains	Application domain of OSS projects
Methodology	Methods	Concrete methods applied
	Models	Type of metrics used
	Tools used	Existing tools, algorithms used for study
	Tools implemented	Tools implemented for the study
Data sources	Source code	CVS/SVN, Code base
	Contributions	Bug tracking systems, Change log
	Communication	Chat history, mailing list archive,
	External sources	Ohloh, Sourceforge, github
Evolutionary parameters studied		size, defects, refactoring etc.
Metrics studied	Types of metrics	SLOC, Class metrics, Graphic metrics
Results	Measures of evolution	Qualitative, Quantitative
	Prediction classification	
	Summary	Other findings
	Constraints/Limitations	
Validation Process		Validation process for study

4. Q4: Are the research aims clearly specified?
5. Q5: Are the findings of study credible?

We define the scale of Yes, No and Partial to answer above checklist questions. We assign a score of “1”, “0” and “0.5” to each question. “0.5” scale is assigned to a particular question based on our judgment after full text scan of an article. The minimum score for selection of a primary study is 3 and above.

3.8 Data Extraction and Synthesis

The data extraction step consists of examining closely the contributions and extracting all the relevant data required to answer the research questions. A preliminary data extraction sheet in Excel is used for this purpose. Prior to the start of the data extraction process, we performed a pilot data extraction on a set of randomly selected 10 papers to identify and characterize the set of attributes. Second, this list of attributes was refined further into a number of specific sub-attributes to get a precise description of each of the general attributes and fine tune the findings on the research questions to increase their reusability. For example, sub-attributes “methods”, “models” and “tools” are intuitively generalized to methodology.

The results obtained from the primary studies and extracted data are inserted into the data extraction sheet. This data extraction sheet explains how the data are extracted from the primary studies and give us a clear description and relation of the data to the research questions. The final attribute framework for data extraction sheet is shown in Table 4. The extracted data are analyzed for consistency. Data attributes are mapped with the research questions and quality assessment checklist.

4 Results and Discussion

We present and discuss results extracted from selected primary studies in this section.

4.1 Overview of Studies

The overview of primary studies based on different aspects related to OSS evolution prediction and process support obtained from the data extraction sheet is presented in this subsection.

4.1.1 Publication Venues

We map selected primary studies with publication venues as shown in Table 5. It is clear from results that maximum studies on open source software evolution are published in the journal of Transactions in Software Engineering, Journal of Software Evolution and Process and International Conference on Software Maintenance.

4.1.2 Trend of Publication Year

The trend of publication year for primary studies reported in this study is presented in Table 6. We can see that maximum studies on evolution prediction are published in years 2007, 2014 and 2015. It also reflects that trend of researchers on predicting different aspects of open source projects is escalating in coming years. Similarly, maximum studies on evolution process support are published in the year 2009.

4.1.3 Mapping of Keywords

Mapping of keywords with primary studies is common feature for systematic mapping studies. Keywords give information about relevancy of a research paper with the main topic covered. We mapped major keywords used by different authors of primary studies in Table 7. These keywords may be used by researchers to define search strings that are used to search articles related with the main topic. Keywords may also be used to identify subtopics of open source evolution studies. We can see from Table 7 that keywords ”software evolution”, “open source” and “defect prediction” are used by maximum primary studies. We want to clarify that authors of some papers do not list keywords, and we used keywords from titles of such papers.

4.1.4 Trend of Authors

Ranking of researchers is also common practice while performing systematic mapping studies. Table 8 shows the authors with more than two publications from our selected primary studies. The common publications from same group of authors are counted separately in Table 8.

4.1.5 Evolution Prediction Aspects

We present mapping of primary studies with our research questions as shown in Table 9. We found that 80 papers from primary studies focus on evolution prediction and 18 papers focus on evolution process support.

4.1.6 Experimental Case Studies

We analyzed open source projects that are selected by researchers as case studies for evolution prediction and evolution process support. Out of 98 primary studies, Linux, Apache, Mozilla and Eclipse open source projects are commonly selected by 15, 15, 13 and 8 studies as shown in Table 10. The major motivation for the selection of open source case studies is their free availability and continuous

Table 5 Mapping of publication venues

Journal/Conference	Papers	Journal/Conference	Papers
Journal (TSE)	[34,84,86–94,103]	Conference (ASE)	[70]
Journal (Soft Evo Pro)	[39,41,54,67–69,122–124]	Conference (MSCR)	[71]
Conference (ICSM)	[32,35,42,61–63,108,115,116]	Conference (MSR)	[73,120,121,126]
Conference (ICSE)	[40,60,76,82,83]	Conference (SEAA)	[74]
Conference (CSMR)	[36,38,49,59,109,112]	Conference (WCE)	[75]
Conference (SEN)	[37,114]	Conference (PMSE)	[77]
Journal (IEE Software)	[105,106]	Journal (JIPS)	[78]
Conference (ESME)	[43,72,113]	Conference (WASET)	[79]
Journal (JSS)	[44,99]	Conference (MSR)	[80]
Journal (IST)	[45,65]	Conference (SSM)	[81]
Workshop (IWMSR)	[48,53]	Conference (ESEM)	[85]
Workshop (IWPSE)	[33,107,110,129]	Conference (OOPL)	[95]
Conference (Testing)	[46]	Workshop (FSE)	[96]
Journal (Software Quality)	[47]	Journal (IJOSSP)	[97]
Conference (ICSTE)	[50]	Journal (ESE)	[98]
Conference (WCRE)	[51]	Conference (IWPSE)	[100]
Conference (ICPC)	[52]	Conference (EIS)	[101]
Conference (OSS)	[55]	Conference (EAIT)	[102]
Conference (ICSEA)	[56]	Journal (ToSEM)	[104]
Conference (CSSP)	[57]	Conference (ICSM)	[108]
Journal (TMIS)	[58]	Conference (WCSMR)	[109]
Journal (SCP)	[64]	Conference (ISSDM)	[111]
Conference (ICACT)	[66]	Journal (IJHIT)	[118]
Conference (ICEMIS)	[119]	Conference (ISSRE)	[127]
Conference (ICACCI)	[128]	–	–

evolution. It reflects that most authors performed experiments on different open source projects and comparison of results of different studies become arduous. There is a need of common corpus regarding evolution prediction of open source projects that can be used by the researchers for comparing results.

4.1.7 Aspects Explored

Interests of researchers toward predicting and supporting evolution process of OSS projects are shown in Tables 11 and 12. Research interest toward predicting OSS projects has dominantly focused defect prediction. Change propagation, ,maintenance effort and SOC (self-organized criticality) have got slightly better attention. The rest of the aspects are addressed considerably very low. In the context of OSS evolution process support, researchers paid major attention to evolution models and exogenous factors contributing to support evolution process. Maintenance support is the second largest aspect, and fault detection and change propagation are the third largest explored aspects.

4.1.8 Distribution of Datasets

Overview of the datasets used for evolution prediction and evolution process support is shown in Table 13. OSS development produces repositories consisting of source code, bug reports, mailing lists, change logs, forums, wikis, etc. Due to such wide variety of data sources, we classify them into different categories, such as mailing lists, SVN/CVS, bug tracking system, change log and external sources. According to our analysis, the highest utilized data sources are source code version control systems (38%) and Bug tracking systems (29%) with the maximum exploration count. These sources are mainly used for fault or defect prediction. But the two sources, communication channels and knowledge sources (e.g., mail, chat, wikis), are yet to get attention confirming the fact that OSS community dynamics was not explored in prediction studies. SVN/CVS is again highly utilized data source in evolution process support studies, but the ratio of studies not specifically mentioning any data source is very high in this context.

Table 6 Trend of publication year for primary studies

Year/Aspect	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	Total
Evolution prediction	1	1	3	3	4	6	7	8	6	5	3	4	5	5	8	11	80
Evolution process support	0	0	1	0	0	1	2	1	1	5	0	3	1	2	1	0	18

Table 7 Mapping of keywords with primary studies

Keywords	# of papers	References
Software evolution	42	[32,33,35–38,40–43,45,51,60–67,96,98,99,101–103,107–122,127]
Open source/FOSS/Floss	41	[32,33,36,37,41,47,55,59,64–66,74,77,81,82,85,94,96,97,99–102,104–107,109–116,118–121,124,125,129]
Defect/fault prediction	27	[40,41,46,48,49,51,57,69–78,81,82,85–88,90,91,124,126]
Empirical study	16	[36,39,40,42,45,46,54,68,70,75,77,78,80,89,92,101,128]
Software metrics	7	[44,56,74,77,85,97,118,127]
Software maintenance	5	[39,41,44,68,84]
Software quality	7	[40,73–75,80,118,119,128]
Object-oriented metrics	4	[54,75,79,84]
Bayesian networks/classifiers	3	[58,90,93]
Genetic algorithm	2	[58,79]
Change propagation	2	[52,62]
Machine learning	2	[78,88]
Liberal	2	[35,37]
Software reuse	2	[68,84]

Table 8 Top authors of papers from primary studies

Author	# of papers	References
Capiluppi	8	[59,96,99,108–110,112,115]
Robles	7	[35,37,38,96–98,117,121]
Herraiz	5	[35,37,38,96,97,121]
Gonzalez-Barahona	5	[35,37,38,98,117,121]
Di Ruscio, Pelliccione	3	[60,64,65]
Bernstein	3	[48,49,100]
Malhotra	2	[75,78,128]
Ratzinger	2	[43,51]
Gall	2	[49,51]
Olague	2	[54,84]
Beecher	2	[59,99]
Song	2	[88,91]
Poshyvanyk	2	[63,86]
Pinzger	2	[48,51]
Alenezi	2	[118,119]

Table 9 Mapping of Primary Studies with Evolution Aspects

Evolution aspect	Papers
Evolution prediction	[32–55,57,58,70–94,101–104,106–129]
Evolution process support	[56,59–69,95–97,99,100,105]

Table 10 Commonly used case studies

Case study/project	Papers
Linux	[32–34,39,49,59,86,94,98,99, 106,107,113,117,124]
Apache	[34,47,56,66,74,75,78,81,83,100, 104,119–121,127]
Mozilla	[32,46,48,49,55,56,80,86,94,97, 104,110,116]
Eclipse	[36,40,44,49,66,77,85,102]
Postgre sql	[33,35,42,80,102]
GNOME	[37,47,66,97,100]
GCC	[11,34,42,62,107]
FreeBSD	[35,42,53,62]
NASA dataset	[82,88,89,91]
AgroUML	[43,51,85,129]

Table 11 Aspects explored for OSS evolution prediction

Aspect	No. of studies
Defect prediction	40
Size	11
Change propagation	11
Maintenance effort	6
Development effort prediction	2
SOC	23
Refactoring	3
Contribution	1
Evolution	
Clone evolution	1
Stability prediction	2

4.1.9 Distribution of Methods

Figure 3 depicts that around 62% of the articles used statistical methods such as regression, time series analysis, correlation analysis, Pearson coefficient, Spearman's rank correlation, principal component analysis and Bayesian belief network. Statistical analysis methods are found to be more reliable for predicting different aspects for OSS studies as compared with other methods. The second large set of methods employed falls into the category of machine learning algorithms. Other methods such as mathematical models, probability methods, Chaos theory and SRGM's (Software Reliability Growth Models) have very low exploration.

Table 12 Aspects explored for OSS evolution process support

Aspect	% of studies
Evolution models	22.2
Exogenous factors	16.6
Maintenance support	11.1
Defect prediction	11.1
Change propagation	11.1
Configuration management	5.5
Feature location	5.5
Growth, complexity and control	5.5
Possible evolutionary paths	5.5
Expert developer recommendation	5.5

Table 13 Datasets for evolution prediction and evolution process support

Aspect	Data sources	% of studies
Evolution prediction	SVN/CVS	38
	BTS	29
	Mailing list	5
	Change log	7
	External source	27
Evolution process support	SVN/CVS	32
	BTS	9
	Not mentioned	45
	External source	14

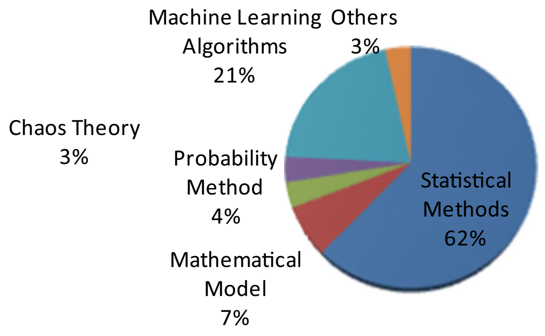

Fig. 3 Methods for evolution prediction

4.1.10 Distribution of Metrics

The distribution of metrics used in research for predicting evolution in OSS projects is shown in Table 14. It can be seen that SLOC got the highest priority, i.e., 26% among all the metric suites studied. The second largest number of metrics used is class level metrics, i.e., 9% . There also exist studies that do not specifically mention metric suites studied.

Table 14 Metrics for evolution prediction

Metrics	% of studies	Metrics	% of studies
SLOC	26	Modification	2
Class level	9	Evolution	5
Complexity level	5	Value series	1
Graph level	1	Bug fix	1
File level	3	Repository	1
Project level	2	Age of a project	1
SNA	4	No of clones	1
Dependency	1	Structural	2
Historic	3	Network	3
People	3	Not Mentioned	22
Architecture	2	Bug report	2

Table 15 OSS projects domains

OSS projects domains	% of studies
Application software	20
Operating systems	11.4
Database	6.6
Frameworks	11.4
IDE	4.9
Open office	4.9
Web applications	4.9
Programming	4.9
Libraries	4.9
Desktop environment	3.3
Others	27.5

4.1.11 OSS Projects Domain studied

Table 15 presents the domains of studied OSS projects in our selected primary studies. The largest domains of OSS projects explored for evolution prediction are application software and operating systems, i.e., 18.0 and 11.4%, respectively. There exist projects of other domains, and we place them in the category of "Others".

4.1.12 Validation Process

Figure 4 presents the validation process adopted by different studies for evolution prediction and its process support. Results show that articles not addressing validation process are considerably higher in both contexts, i.e., evolution prediction and evolution process support. The internal validation got a slightly better attention of researchers as compared to the external and construct validation.

4.1.13 Context of the Fault Prediction Models

Studying context of the fault prediction studies help us to understand the generalization and applicability of models. For that purpose, we extracted the data against the origin of systems and the programming languages for which models have been developed and tested. It is visible from statistics presented in Table 16 that a large portion of data being used in fault prediction studies comes from OSS. Similarly, we can see in Table 17 that Java language dominates in the studies. C/C++ has got almost equal ratio. Other languages include PHP, Ruby, Perl, Python and assembly languages. The information about types of projects and programming languages for selected primary studies is given in Tables 16 and 17.

4.2 Metrics for Evolution

We discuss and analyze metrics studied, the context in which they are studied and their subsequent results in this subsection for OSS evolution prediction and process support. Following metrics are used:

4.2.1 SLOC (Source Lines of Code) Metrics

Herraiz et al. [35] used the SLOC metric to compute the size of FreeBSD, NetBSD and PostgreSQL. They collected time series data for the size of each case study (these series were predicted based on past values). The degree of internal autocorrelation (internal autocorrelation means that a value in the future depends on some values in the past). Therefore, a model based on past values can be obtained to forecast the future. Herraiz et al. [35] in another study used SLOC (excluding blank and comment line) metric to compute the

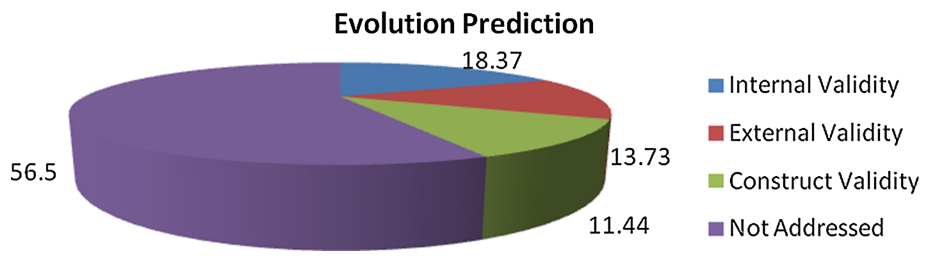

Fig. 4 Validation process for evolution prediction and evolution process support

Table 16 Types of datasets

Datasets used	Number of papers	% of studies
OSS	55	56
NASA	5	5
Industrial	6	7
Closed source	4	4
Not clear	16	16

Table 17 Types of programming languages

Languages used	Number of papers	% of studies
C / C++	23	23
JAVA	40	40
Others	9	9
Not given	25	25

size of 3821 projects to evaluate the concept of SOC (self-organized criticality) in dynamics of software evolution. Yu et al. [39] used SLOC for calculating the size of 121 subversions of Linux in the form of the following measures such as: E-Line (summation of KLOC added, deleted, and modified), T-Line (Total size of the system measured in KLOC) and K-Line (Kernel size of the system measured in KLOC). Ratzinger et al. [43] used linesAdded, linesModified, or linesDeleted relative to the total LOC (lines of code) of a file as features of evolution data while predicting future refactoring. Additionally, they regard large changes as double of the LOC of the average change size and small changes as half of the average LOC of a specific file. Gonzalez-Barahona et al. [98] used SLOC to analyze the evolution of different releases of Debian over a period of nine years (1998-2007). The authors realized that stable releases of Debian increase double in size approximately after every 2 years. Knab et al. [48] applied source code metrics to compute the size of seven releases of Mozilla, with the purpose of predicting defect density. The selected metrics are: linesOfCode (Lines of code), nrVars (Number of variables), nrFuncs (Number of functions), incomingCallRels (Number of incoming calls), outgoingCallRels (Number of outgoing calls), incomingVarAccessRels (Number of incoming variable accesses) and outgoingVarAccessRels (Number of outgoing variable accesses). It is observed that line of code metric is not the only indicator to detect problematic files. Hence, LOC metric has little predictive power with regard to defect density. Finally, SLOC metric is used for detection of faults and other evolution aspects from OSS and other types of software applications by a number of researchers [32,34,73,79,80,83,85,88,90,92,94,101,110,111,113,117,118].

4.2.2 Dependency Metrics

Mirarab et al. [52] proposed a novel technique for predicting change propagation using Bayesian belief networks as a probabilistic tool to make such predictions. The technique relies on dependency metrics using static analysis and change history extracted from Version Control Repository. The authors extract the information about the dependencies between the software elements using "Dependency Finder" (an open source java program) from all the significant revisions of Azureus2, an open source Java system. The results reflect that developers can predict change propagation at early and later stages of development and maintenance.

4.2.3 People Level Metrics

Ratzinger et al. [43] constructed *authorCount* metric as a feature for evolution data while predicting refactoring for software systems. They assume that the number of authors of files influence the way software is developed. Herraiz et al. [37] used number of people who have sent at least one message for a given month, number of people who have reported at least one bug for a given month, number of people who have committed at least one change to the versioning system for a given month while predicting the evolution of contributions.

4.2.4 File Level Metrics

To measure how often a file was involved during development with the introduction of other new files, Ratzinger et al. [43] used *coChangeNew* metric and *coChangedFiles* metric to measure the files changed together. Ekanayake et al. [49] used file level metrics while investigating the reasons why the prediction quality is so fluctuating due to the altering nature of the bug (or defect) fixing process by adopting the notion of a concept drift. The study computed features on file level such as *revision* (Number of revisions), *activityRate* (Number of revisions per month), *grownPerMonth* (Project grown per month), *totalLineOperations* (Total number of line added and deleted), etc. Further, [42] used logical and structural changes at source file level while presenting empirical evidence for the existence of fractal structures in software evolution.

4.2.5 Bug Fix Metrics

Ratzinger et al. [43] used bug fix metrics such as bugfixCount (is computed by enumerating all changes to source code) as well as bugfixLinesAdded, bugfixLinesModified and bugfixLinesDeleted in relation to the base measures such as the number of lines of code added, modified, and deleted from a file. These metrics are used as features of evolution data of software systems while constructing prediction model for refactoring using data mining algorithm.

4.2.6 Bug Report Metrics

Knab et al. [48] applied bug report metrics to compute the size of seven releases of Mozilla, with the purpose of predicting defect density. The selected metric is nrPRs (Number of problem reports). Similar to modification metrics, bug report metrics also provide better accuracy for predicting defect densities. Herraiz et al. [37] used bug reports for maintenance purpose as they are good indicator of the fixes and found strong correlations between the number of messages, the number of bugs and the number of commits and that the activity of reporting bugs or making a commit generates more activity in the mailing lists.

4.2.7 Modification Metrics

Knab et al. [48] selected modification metrics to compute the size of seven releases of Mozilla, with the purpose of predicting defect density. The selected metrics are nrMRs (Number of modification reports) and sharedMRs (Number of shared modification reports). The shared modification reports metric (sharedMRs) represents the number of times a file has been checked into the CVS repository together with other files. It was observed that modification metrics provides satisfactory accuracy for predicting defect densities. Ratzinger et al. [43] used *addingChanges*, *modifyingChanges*, and *deletingChanges* per author and per file for providing input to the defect prediction of files. They also used *changeCount* metric in relation to the number of changes during the entire history of each file.

4.2.8 Repository Metrics

Herraiz et al. [37] presented an approach that describe the evolution of the size and cost associated with the development using versioning system, mailing lists and bug tracking system of a large libre software(GNOME). The cost associated with the development predict applied number of commits to the versioning system (CVS) for each month during the lifetime of the project for source code production as they are a good indicator of coding activity. Results reflect that relationship between activity and participation are a good indicator for evolution prediction.

4.2.9 Age of Project Metrics

Herraiz et al. [38] used the age of project metric as "SF.net age" and "CVS age" parameters, they indicate the age of project in months. "SF.net age" is the number of months that the project has been stored in SF.net. "CVS age" is the difference in months between the dates of the last and first commit in the CVS while exploring the contradiction that evolution of libre software projects is governed by sort of determinism.

4.2.10 Graph Metrics

Bhattacharya et al. [40] used graph metrics to construct predictors for bug severity, high-maintenance software parts, and failure-prone releases. Three graph metrics are as follows:

1. NodeRank: A graph metric akin to PageRank can predict bug severity; it assigns a numerical weight to each node in a graph, to measure the relative importance of that node in the software.
2. Modularity Ratio Metric: Standard software engineering practice suggests that software design that exhibits high cohesion and low coupling actually provides number of benefits. These benefits not only lead to easy software understanding but they also support testing and evolution. Therefore, it defines modularity ratio of A module as ratio among values of cohesion and coupling: Modularity Ratio(A) = Cohesion(A) / Coupling(A), where Cohesion(A) refers to the number of variable references in module A; similarly, Coupling(A) is the total number of inter-module calls in module A. It is used to for forecasting modules that will sustain high maintenance effort.
3. Edit distance: Over time, while investigating how does program structure changes, Bhattacharya et al. [40] introduced a metric for capturing the number of changes made between vertices and edges among two graphs, i.e., between consecutive releases. It is used to predict failure-prone.

4.2.11 Class Level Metrics

Shatnawi et al. [44] applied class level metrics that can predict class error-proneness and the prediction can be used to accurately group error-prone classes subsequently while examining the three releases of Eclipse project. The authors applied the UBR (univariate binary regression) analysis to investigate whether the eleven metrics (CBO, CTA, CTM, RFC, WMC, DIT, NOC, NOAM, NOOM, NOA, and NOO) are significant predictors of class error-proneness. A metric was significantly associated with class error-proneness if its P value was less than 0.05. Results showed that:

Significant predictors—CBO (Coupling between objects), RFC, CTA (Coupling through data abstraction), CTM (Coupling through message passing), WMC (weighted methods complexity)

Good predictors—NOOM (Number of overridden methods), NOAM (Number of added methods), NOA (Number of attributes), NOO (Number of operations)
Bad predictors—DIT (Depth of inheritance hierarchy), NOC (Number of child classes)
It was concluded that as class error probability got smaller and smaller in the post-release evolution of the system, it became more and more difficult to predict where the errors were likely to occur by using the metrics.

Puri et al. [79] used the Chidamber & Kemerer metrics suite while predicting the faulty modules in jEdit. Malhotra et al. [75] focused on detecting relation between object-oriented metrics and fault-prone classes by using Chidamber and Kemerer java metrics (ckjm). English et al. [77] used Chidamber and Kemerer metric suite for identifying error-prone classes in open source software systems. Malhotra et al. [78, 128] used Object Oriented metrics for predicting error-prone classes and change prediction in different open source projects. Yuming et al. [89] used Chidamber and Kemerer metric suite for predicting fault-proneness of NASA datasets. Tibor et al. [94] also used Chidamber and Kemerer metrics to show how fault-proneness of the source can be carried out. Results have shown LCOM as good metric, DIT as untrustworthy and NOC for not to be used at all for prediction of fault-proneness. Olague et al. [84] studied three object-oriented software metric suites, i.e., Chidamber and Kemerer (CK) metrics, Abreu's Metrics (MOOD), and Bansiya and Davis' Quality Metrics (QMOOD) for predicting OO fault-prone classes. The detail information about Chidamber and Kemerer (CK) metrics suite is presented by Chidamber et al. [31].

4.2.12 Evolution Metrics

Jacek et al. [51] applied data mining techniques for value series based on the evolution attributes for measuring defect density and defect prediction. Daily data points of these evolution attributes were captured over a period of 2 months to predict the defects in the subsequent 2 months for a project. The results show that by utilizing series of these attributes accurate prediction models can be built with high correlation coefficients (between 0.716 and 0.946). The evolution attributes used were measured at file level. These evolution attributes/metrics were: *Lines Added, Lines Deleted, Number of Changes, Number of Authors, Author Switches, Commit Messages, With no Message, Number Bugfixes, Bugfix Lines Added, Bugfix Lines Deleted, Couplings, CoChanged Files, CoChanged New Files, Transaction Lines Added, Transaction Lines Deleted, Transaction Bugfix Lines Added, Transaction Bugfix Lines Deleted.* The approach is evaluated on two open source and one commercial project. The *a*uthors concluded that the number of authors and number of commit messages to versioning system are excellent predictors for defect density as compared with other metrics. Saini et al. [102] applied fuzzy logic-based technique to predict number of commits of open source software projects based on the past history. The authors validated their approach on three open source projects and concluded that their approach outperforms as compared with ARIMA models. Santiago et al. [121] presented empirical study to determine evolution of Apache using the ratio of total messages to the mailing lists to total number of commits metrics. They name this metric as intensive metric as it is independent of the size of project and its activity. Charrada et al. [122] presented a recent approach to identify the impact of source code changes on the requirements. The authors evaluated their approach on three case studies. They identify source code changes and then determine their impact on the requirements using commit metric. Threm et al. [127] applied software normalized compression distance metric to measure the difference between evolving versions of Apache. The authors conclude that program level, architecture level and information level metrics are important to measure evolutionary stability of software artifacts.

4.2.13 Historical Metrics

Illes-Seifert et al. [45] explored the relationship between several historical characteristics of a file and defect count as knowledge about particular characteristics of software that are indicators of defects. Defect count is a very valuable metric for testers as it helps them to focus the testing effort and to allocate their limited resources appropriately. Nine open source Java projects across different versions were analyzed for this purpose. The selected historical metrics characteristics are as: *"Defect history"* of a file concerns previously found defects, *"Release history"* of a file concerns the time between two releases at which a HT ("History touch" is one of the commit actions where changes made by developers are submitted) occurs. *"Change history"* of a file comprises the number, size and author(s) of the HTs performed to that file. *"File age"* according to the age files is classified as follows:

Newborn: A file is newborn at its birthday.
Young: All files that are not older than the half of a systems' age and that are not classified as Newborn.
Old: All files that are older than or equal to the half of a systems' age.

Results showed that file age is a good indicator of its defect count. However, there is no other indicator that persists across all projects in an equal manner.

Mirarab et al. [52] proposed a novel technique for predicting Change Propagation using Bayesian belief networks as a probabilistic tool to make such predictions. The technique relies on dependency metrics and change history information extracted from Version Control Repository. To make history information useful for change propagation prediction, it is extracted in the form of "co-changes". The situation where system elements change concurrently refers as co-change, whereas the concept of co-change and change propagation differs from one another. It can be very helpful for predicting propagation because elements that change simultaneously in the past, gives us assurance of their future change together and will be more probable of transmitting among elements.

Askari et al. [53] used change history data while developing three probabilistic models to predict which files will have changes or bugs. The authors use historical records from source control repositories of large software systems. These events are file changes to fix bugs, or to add new features or change existing features. These events are extracted from the history of the software.

4.2.14 Project Metrics

Ekanayake et al. [49] used project level metrics while investigating the reasons of fluctuating prediction quality because of the changing nature of the bug fix method using the idea of "concept drift". With the intention of uncovering features that may be used for predicting type of period when a software project acts as pointer respected to the use of defect predicting models. As a result, regression model was made for predicting the AUC of bug prediction model. AUC (area under curve) is a facet of prediction model at project level; therefore, at project level these features are used for learning the prediction model. Some of the project level metrics used are: Number of revisions as revision, Project grown per month as grownPerMonth, Total number of line added and deleted as totalLineOperations, Total number of bugs fixed in every type as bugFixes, etc. The concept of project and products metrics is also applied for defect prediction by Wahyudin et al. [74].

4.2.15 Number of Clones in Each Version

Shawky et al. [50] presented an approach for modeling clones evolution in open source software projects. The authors adapt chaos theory for predicting clones in new versions of a software system. For this purpose, they use the *number of clones* in each version and analyze them as a time series data. For clones identification, they use CloneDR tool, which is a tool that detects and aids the removal of duplicate code.

4.2.16 Value Series Metrics

Jacek et al. [51] construct final value series at file level containing relative measures ordered by time from the absolute values of evolution attributes while generating models for measuring defect densities and defect prediction. The relative measures used to create value series per file for each day are: *LinesAdd, LinesDel, ChangeCount, Authors, AuthorSwitches, CommitMessages, WithNoMessage, BugfixCount, BugfixLinesAdd, BugfixLinesDel, CoChangeCount, CoChangedFiles, CoChangedNewFiles, TLinesAdd, TLinesDel, TBugfixLinesAdd and TBugfixLinesDel.* Results showed that *Authors, CommitMessages* and*TLinesAdd* are best predictors for defect prediction.

4.2.17 Complexity Metrics

Olague et al. [54] studied the effectiveness of nine object-oriented (OO) software complexity metrics for predicting defective object-oriented (OO) classes in Rhino (an open source Java-based software project). The metrics used are as follows: weighted methods per class (WMC), weighted methods per class McCabe (WMC McCabe), McCabe cyclomatic complexity, average maximum cyclomatic complexity of a single method of a class (CCMax), standard deviation method complexity (SDMC), Number of instance methods (NIM), Number of trivial methods (NTM). Results have shown that studied complexity metrics relates well fault-proneness of object-oriented (OO) class. Dejaeger et al. [90] used complexity metrics on NASA datasets while predicting faults in Open Source Software systems. Alenezi et al. [118] applied McCabe metric to predict size and complexity of five open source software projects.

4.2.18 SNA Metrics

Biçer et al. [57] used social network metrics on issue repositories while predicting defects for the first time. The authors collected metrics that are used in Social Network Analysis (SNA) from communication networks of two projects, RTC and Drupal. General SNA metrics of developer networks based on issue reports, which can be extracted from almost all issue repository systems, are used. The metrics used are: *Betweenness Centrality, Closeness Centrality, Barycenter Centrality, Degree Centrality, Group Degree Centrality Index, Density, Diameter, Clustering Coefficient, Bridge* and *Characteristic Path Length*. Results showed that using social network metrics from issue repositories can dramatically improve the prediction performance and reduce inspection cost as compared to the churn metrics. Premraj et al. [85] used Network metrics for predicting defects. They compared network metrics and code metrics. They concluded that network metrics offer no advantage over code metrics. Nzeko'o et al. [125] applied social network analysis to analyze developers and users mailing lists for four open source software projects. The authors analyzed threads in developers and mailing lists. Zhang et al. [129] presented an empirical study to identify core developer of AgroUML project by using developers mailing lists and community mailing lists. The authors found that developers within same module communicate closely and frequently with each other.

4.2.19 Structural Software Metrics

To predict stability of OSS projects, authors [58] reported 18 structural software metrics at the OO-class level. The structural metrics belong to four categories, namely complexity, inheritance, cohesion, and coupling. Alenezi et al. [119] applied structural object-oriented metrics to measure modularity evolution using complexity, coupling and cohesion metrics.

4.2.20 Architectural Change Metrics

Duc et al. [120] presented an empirical study of architectural changes on 14 open source projects using architectural change metrics for the purpose of measuring architectural changes. The authors consider architectural changes at system and component levels. They applied two new architectural metrics for their study: a2a (architecture to architecture), a system level metric and cvg (cluster coverage) a component level metric. Kouroshfar et al. [126] presented a study to determine the impact of software architecture evolution on quality of software using architectural metrics. These metrics are IMC(intra-module co-changes) and CMC(cross-module co-changes). The authors conclude that crosscut multiple architectural modules are more correlated with defects than co-changes that are localized in the same module.

We identified 20 different categories of metrics that are used to predict different aspects of OSS studies as presented in Table 18. We can see from Table 18 that SLOC metric is mostly used by different authors to predict different aspects of OSS studies. This metric is used to predict size and defects by most authors. The class level metrics are used for defect prediction, bug severity and class error-proneness by different authors as shown in Table 18. We found some contradictions on metric's predictive power. For example, SLOC is evaluated as good predictor in [35] and evaluated as bad predictor for carrying out defect densities in [48]. Similarly, in [48], bug report metrics provide satisfactory accuracy for predicting defect densities, whereas [37] uses bug reports for maintenance purpose and declare them as good indicator of the fixes and found strong correlations between the number of messages. Similarly, it is observed in [48] that modification metrics also provide satisfactory accuracy for predicting defect densities. Moreover, it is also revealed that file age is a good indicator for defect count. The authors in [94] declared SLOC as good metric and CBO as best metric for defect prediction while they found DIT as untrustworthy. They also realized that NOC cannot be used at all. The authors in a study [96] compared CK (Chidamber and Kemerer), QMOOD (Quality Metrics for Object-Oriented Design) and MOOD (Metrics for Object-Oriented Design) metrics to predict defects and results showed that CK metrics are more reliable for fault prediction.

Finally, we present summarized facts about metrics, types of these metrics and level at which these metrics are computed and applied by researchers in Table 19. It is visible from results in Table 19 that most metrics are computed at file level and very little attention is paid to compute metrics at class level.

4.3 Models for Evolution Prediction

This subsection discusses proposed models, the context in which models are used and their strengths and limitations. Summarized information about evolution prediction models is presented in Table 20.

4.3.1 Predicting Size

To obtain the prediction of future values, Herraiz et al. [35] applied ARIMA model (ARIMA models are linear combinations of past values of the series, weighted by some coefficients) on three long-lived projects (FreeBSD, NetBSD and PostgreSQL). They obtained regression and ARIMA models for these projects which predict growth of these projects in the future. Ruohonen et al. [123] presented empirical study based on time series trends to evaluate evolution of two open source software projects. The results of study can be interpreted against laws of software evolution.

4.3.2 Predicting Indirect Maintenance Effort Models

The study [39] constructed two indirect maintenance effort models for the Linux project (121 recent versions of Linux). Software maintenance can be organized into three categories: software support utilization, change in the source code (can be further categorized into two types if function changes), and updating documentation. It has considered one category of software maintenance, i.e., source code change. From the activity-based view, changes in source code can be categorized as corrective, perfective, or adaptive. The activity performed on artifact for removing enduring defects but leaving the anticipated semantics unchanged is known as "corrective maintenance". Including enhancive maintenance, adaptive and perfective maintenance are activities performed with the intention of implementing functionality change for changing software properties such as security, usability, performance etc. or for adapting new platform (software or hardware).

Table 18 Evolution metrics and aspects explored for prediction

	Aspects explored for prediction														
Metric type	Size	Main. E	Refac	Defect P.	Class EP	EP	Bug S	CDA	CP	Defect C	CaP	EC	SOC	CE	ES
SLOC	[32,34,35,98,101,107, 110,111,113,116,117]	[38,106]	[39]	[43,73,79,80,83,85,88, 90,92,94]					[107,118]			[101]	[48]		
Class level metrics				[79,94–96]	[44,75,77]	[78]	[79,94–96]				[128]				
Complexity metrics				[90]		[54]					[110,112,114]				
Source code-based graphs		[40]		[40]			[40]								
File level metrics			[43]					[49]					[42]		
Project level metrics				[74]				[49]							
SNA metrics				[57,85]							[125]	[129]			
Dependency metrics									[52]						
Historical metrics									[52]	[45]	[53]				
Bug report metrics				[48]								[37]			
People level metrics			[43]							[104]		[37]			
Modification metrics			[43]	[48]											
Evolution metrics		[122]		[51]							[102,121]				[126]
Value series metrics				[51]											
Bug fix metrics			[43]												
Repository metrics												[37]			
Age of project metrics													[38]		
No. of clones														[50]	
Structural S/W metrics		[119]	[119]											[58]	[58]
Architectural change metrics				[126]					[120]		[120]				

Main. E.: maintenance effort; Refac: refactoring; Defect P: defect prediction; Class EP.: class error-proneness; E.P.: error-proneness; Bug S.: bug severity; CDA: concept drift analysis; CP: change propagation; Defect C.: defect count; CaP: change prediction; EC: evolution of contribution; SOC: self-organized criticality; CE: clone evolution; ES: evolutionary stability

Table 19 Metrics and their applications

Metric type	Study ID	How metrics and Sub-metrics are applied?	Level
SLOC	[32,35,38,39,43,49,59,73, 79,80,83,85,88,90,92,94, 98,101,110,111,117,118]	E-Line (summation of KLOC added, modified and deleted), T-Line (system size measured in KLOC) and K-Line (kernel size measured in KLOC), lines Added, Modified or Deleted related to the total LOC (lines of code) of a file, nrVars (Number of variables), nrFuncs (Number of functions), incomingCallRels (Incoming calls), outgoingCallRels (Outgoing calls), incomingVarAccessRels (Incoming variable accesses) and outgoingVarAccessRels (Outgoing variable accesses, LT (lines of code in a file before the change), AvgLineCode and CountLine, LOC_total, LOC_blank, Number_of_lines and LOC_comments	File level
Class level metrics	[44,75,77–79,94–96,103,128]	CBO, CTM, NOOM, RFC, DIT, NOO, NOC, NOAM, CTA, NOA, and WMC. Chidamber and Kemerer metric suite: Coupling between Objects (CBO), Lack of Cohesion (LCOM), Number of Children (NOC), Depth of inheritance (DOI), weighted Methods per Class (WMC), Response for a class (RFC), Number of Public Methods (NPM), Lack of cohesion in methods (LCOM3), Data Access Metric (DAM), Measure of Aggregation(MOA), Cyclomatic Complexity (CC), Inheritance Coupling (IC) Measure of Functional Abstraction (MFA), Cohesion Among Methods of Class (CAM), Coupling Between Methods (CBM), Average Method Complexity (AMC), QMOOD metrics, MOOD metrics	Class level
Complexity metrics	[54,90,112]	Weighted methods per class (WMC), weighted methods per class McCabe (WMC McCabe), McCabe cyclomatic complexity, Average Maximum cyclomatic complexity of a single method of a class (CCMax), standard deviation method complexity (SDMC), Number of instance methods (NIM), Number of trivial methods (NTM)	Class and File level
Source code-based graphs	[40]	NodeRank, Modularity Ratio and Edit distance	Not mentioned

Table 19 continued

Metric type	Study ID	How metrics and Sub-metrics are applied?	Level
File level metrics	[42,43,49]	coChangeNew metric and coChangedFiles metric for measuring the files changed together, Number of revisions as revision, Number of revisions per month as activityRate, Project grown per month as grownPerMonth, Total number of line added and deleted as totalLineOperations at file level	File level
Project level metrics	[49,74]	Number of revisions as revision, Project grown per month as grownPerMonth, Total number of line added and deleted as totalLineOperations and bugFixes	Not mentioned
SNA metrics	[57,85,125,129]	Betweenness Centrality, Closeness Centrality, Barycenter Centrality, Degree Centrality, Group Degree Centrality Index, Density, Diameter, Clustering Coefficient, Bridge and Characteristic Path Length. Size, Ties, Pairs, Density, Weak Comp, n Weak Comp, TwoStepReach, ReachEfficency, Brokerage, nBrokerage, EgoBetween, nEgoBetween, mailing list, community lists	Not mentioned
Dependency metrics	[52]	Not mentioned	Not mentioned
Historical metrics	[45,52,53]	Defect history, Release history, Change history, File age, co-changes	File level
Bug report metrics	[37,48]	nrPRs (Number of problem reports)	File level
People level metrics	[37,43,104]	authorCount, Number of people sent one message at least for a given month, Number of people reported one bug at least for a given month, Number of people Committed change at least once to the versioning system	Not mentioned
Modification metrics	[43,48]	nrMRs (Number of modification reports), sharedMRs (Number of shared modification reports), addingChanges, modifyingChanges, and deletingChanges and changeCount metric	File level
Evolution metrics	[34,51,102,121]	Lines Added, Lines Deleted, Number Changes, Number Authors, Author Switches, Commit Messages, With no Message, Number Bugfixes, Bugfix Lines Added, Bugfix Lines Deleted, Couplings, CoChanged Files, CoChanged New Files, Transaction Lines Added, Transaction Lines Deleted, Transaction Bugfix Lines Added, Transaction Bugfix Lines Deleted, Commits, No of messages	File level

Table 19 continued

Metric type	Study ID	How metrics and Sub-metrics are applied?	Level
Value series metrics	[51]	LinesAdd, ChangeCount, LinesDel, AuthorSwitches, Authors, CommitMessages, BugfixCount, WithNoMessage, BugfixLinesDel, BugfixLinesAdd, CoChangedFiles, CoChangeCount, CoChangedNewFiles, TLinesDel, TLinesAdd, TBugfixLinesDel and TBugfixLinesAdd	File level
Bug fix metrics	[43]	bugfixCount, bugfixLinesModified, bugfixLinesAdded and bugfixLinesDeleted	File level
Repository metrics	[37]	Not mentioned	Not mentioned
Age of project metrics	[38]	“SF.net age” and “CVS age”	Not mentioned
No. of clones	[50]	Not mentioned	Not mentioned
Structural S/W metrics	[58]	Not mentioned	Class level
Architectural change metrics	[120]	A2a, cvg, CMC, IMC	System and component level

Table 20 Evolution prediction models

Proposed models	Aspect	Study ID	Results
Regression models, time series-based ARIMA models	Size	[35]	Time series-based ARIMA models can predict the growth in the next year of a project with lower error than regression models. The performance of proposed models may vary, if the projects are not long lived and their size is not large
Source-code-based maintenance effort models	Indirect maintenance effort models	[39]	Model 1. E-Line = a1NC + a2NO Model 2. E-Module = a1T-Module + a2K-Module + *a3NC* + a4NO 1. Model 2 outperforms Model 1 2. The proposed models may be inapplicable in the situation where we do not know what tasks are included in the next revisions
Probabilistic models (MLE, RED, RED-Co, REDCC), Change prediction model	Predicting Change	[53, 128]	REDCC is slightly better than RED, and both of these considerable better than MLE Change prediction model is presented based on different techniques

Table 20 continued

Proposed models	Aspect	Study ID	Results
Time series models	Defect count	[41]	Results show that the mean square error (MSE), mean absolute percentage error (MAPE) and mean absolute deviation (MAD) for the ARIMA(0,1,1) model are all better (lower) than the competing ARIMA(1,1,0) model
ARIMA(0,1,1)			
ARIMA(1,1,0)			
ARIMA (1,1,0)(1,1,0) model	Change request prediction	[36]	The model is able to make reliable forecasts with up to 1 year in advance, concerning the number of future change requests. For all the error statistics, it is the model leading to the smallest errors
MLR (multiple linear regression) model	Class error-proneness	[44]	Not specifically mentioned
BBN models, BDM	Change propagation	[52]	The overall accuracy of BDM and BHM lies in the same range, but BDHM clearly shows better results
BHM BDHM			
Linear regression models	Fault prediction	[46]	Results on mozilla and MP projects showed that QALP score appear to produce useful information in an environment where the coding style is homogeneous
SRGM	Fault prediction	[47]	Results on seven open source software projects show that SRGM model is most reliable for residual fault prediction
Stability prediction models	Stability prediction	[58]	Results on a large-scale software show that stability prediction models outperform state-of-the-art approaches
Software clustering	Fault prediction	[70]	Results indicate that the models we build using software clusters perform better than those built on the classes in system
Predicting field defects	Predicting field defects	[81]	It is impossible to predict field defects by extending Weibull model, which indicates the significance of metrics-based field defects prediction models
Efficacy of process and code metrics	Defect prediction	[83]	Results have shown code metrics less stable. Whereas, process metrics outperform in all cases

Table 20 continued

Proposed models	Aspect	Study ID	Results
Comparison of network and code metrics	Defect prediction	[85]	Network metrics perform better as compared to the code metrics for predicting defects
Bayesian network model	Development effort prediction	[93]	The proposed model can handle missing data very efficiently and is capable of providing uncertainty in predicted values
Fuzzy time series	Commits prediction	[102]	The proposed model forecasts number of commits per month and authors claim that their method outperforms as compared with ARIMA models

4.3.3 Predicting Change

Askari et al. [53] proposed three probabilistic models for predicting suspicious files having changes or bugs. The primary model is maximum likelihood estimation (MLE). It is a model that is used for counting changes or bugs happened to all files and normalizing counts for computing distribution of probability. The next model is Reflexive Exponential Decay (RED) in which any modification to a file leads to an increment in the predictive rate of modification in that file and crumbles exponentially. Another model is called RED-Co-Change. It not only increases the predictive rate, but also increases rate for other files having the relation of previous co-changes, as a result of each modification to a given file. The models were tested empirically by two approaches on six OSS (open source systems), i.e., Top Ten List evaluation and Information theoretic evaluation. Results show that REDCC is somewhat better than RED, and both of them are considerably better than MLE as declared by Top Ten List evaluation. Malhotra et al. [128] presented a change prediction model based on machine learning and search-based techniques. The authors conclude that search-based techniques outperform as compared to statistical and machine learning techniques.

4.3.4 Predictors for Bug Severity, Maintenance Effort and Defect Count Bug Severity Predictor

Bhattacharya et al. [40] presented a new approach that makes use of graph-based metrics for predicting bugs severity. The project managers review and allocate severity rank to bug whenever it is reported on the basis of how much severe affect it has on the program. They used NodeRank for identifying important functions and modules, i.e., when buggy, carries a large probability of exhibiting high-severity bugs. Bug severity predictors improve the quality of software by implying more focus on checking and verifying efforts on high severity parts of software systems. By using six OSS projects, i.e., Firefox, Blender, MySQL, VLC, Samba, and OpenSSH for analyzing the hypothesis "*Higher NodeRank functions and modules are inclined to bugs of high severity"* was validated. They also concluded that "*The node degree is not a good indicator for bug severity prediction*".

4.3.5 Maintenance Effort Predictor

According to Bhattacharya et al. [40], leading cause of high software maintenance cost such as refactoring or adding new functionality is obscurity related with source code changes. They intend to recognize modules difficult to change by new module-level metric known as *Modularity Ratio*. For maintenance effort measurement, number of commits are divided by the agitated eLOC. For effort estimation, it is largely used metric. They validated the approach using dataset of four programs, i.e., Firefox, Blender, MySQL and VLC for analysis and validated the hypothesis "Modules of higher *ModularityRatio* have low maintenance effort." It was also found that for a module, as its ModularityRatio increases, its maintenance effort decreases, which shows that software structure improves.

4.3.6 Defect Count Predictor

According to Bhattacharya et al. [40], stable and highly organized development team produces higher-quality software as compared to software offered by disconnected and highly turnover teams. Hence, it was studied that how does stable teams and composite structures led to high collaboration level, which results in high software quality. They theorize that low defect count time periods are resulted in a case of established development teams. They make use of Eclipse bug reports and Firefox for building developer collaboration graphs. They analyzed 129,053 bug reports (May 1998 to March 2010) for Firefox and reflected on bugs numbers from 1 to 306,296 (October 2001 to March 2010) for Eclipse. Further, they also validated the hypothesis that *"With the Increase in edit distance in Bug-based Developer Collaboration graphs* (If a bug is cannot be resolved by developer D1, the bug is reassigned to D2 i.e., second developer and a directed edge is added from D1 to D2 in graph)*results in an increase of defect count"*.

Raja et al. [41] proposed the single time series model, ARIMA(0,1,1), for accurately predicting software evolution defect patterns. It make comparison of two models, i.e., ARIMA(0,1,1) and ARIMA(1,1,0). ARIMA(0,1,1) was declared as a most appropriate model. For evaluating the precision of the model, the holdout observations are made. Results reveal that the mean square error (MSE) shows absolute percentage error (MAPE) and mean absolute deviation (MAD) for the ARIMA(0,1,1)model are lower (better) than the opposing ARIMA(1,1,0) model. This excellent performance holds constantly for all projects and time spans within the holdout samples. It is computationally competent, understandable, and simple to apply MODEL using a partial amount of data.

4.3.7 Software Change Request Prediction Models

Goulão et al. [36] proposed that ARIMA (1,1,0)(1,1,0) model for long-term prediction of the overall number of change requests is valid and more accurate for predicting the evolution of change requests in Eclipse than the other non-seasonal models. The model is able to make reliable forecasts with up to 1 year in advance, concerning the number of future change requests. For all the error statistics, ARIMA is the model leading to the smallest errors. The Ljung-Box Q test indicates, through its high significance value, that the model is suitable and well-adjusted to the time series. This model is considered appropriate for estimating future change requests.

4.3.8 Class Error-Proneness Prediction Models

Shatnawi et al. [44] proposed MLR (multiple linear regression) models for predicting class error-proneness in three versions 2.0, 2.1 and 3.0 of Eclipse (an industrial-strength system that is continuously evolving with thousands of classes). The study uses eleven class level metrics (CBO, CTA, CTM, RFC, WMC, DIT, NOC, NOAM, NOOM, NOA, and NOO) and used UMR (univariate multinomial regression) analysis to investigate whether eleven metrics were associated with the three error severity categories (high, medium, low). A metric was significantly associated with an error severity category if its P value in the UMR analysis was less than 0.05. Likelihood ratio test (LRT) was used to verify the overall significance of the model.

4.3.9 Software Fault Prediction

Binkley et al. [46] applied measure referred to as a *QALP* score that make use of techniques from information retrieval to predict faulty modules. The study undertook two projects: one open source software, i.e., Mozilla and one proprietary software, i.e., MP (a software written for a business application in a mid-size enterprise). Data were extracted from the Bugzilla for Mozilla which assign each bug to a set of classes. For MP fault data were collected in a similar in-house database. Linear mixed-effects regression models are used to analyze the data. The proposed approach first computes the structural measures for each module (each Mozilla Class and MP file). The second step involves computing the QALP score including three steps: the first step breaks the source into modules, and the second step separates each module into comments and code. The final phase applies language processing techniques to improve the efficiency and accuracy of cosine similarity. The linear mixed-effects regression models were applied for predicting the defects in Mozilla and MP. Code inspection of Mozilla proved the QALP score as an ineffective measure for faults prediction, while, for the second studied program, QALP score shows the inverse correlation with defect rate that makes it an effective component of a fault-predictor. Results have also shown that the QALP score proves to produce useful information in an environment of homogeneous coding style. The reason is extreme difference of programmer's ranges and forecasting ability by using QALP scores. Moreover, lack of inward-looking comments in Mozilla leads to the mediocre model for it.

A method for predicting enduring defects of the OSS projects by selecting SRGM (Software Reliability Growth model) which best predicts the residual defects of an OSS is proposed [47]. The applied method selects the best model to help in decision making on when to stop testing and deploy the software. The author proved the reliability of proposed model through empirical results on seven open source software projects.

Rossi et al. [55] presented a study for determining and discussing patterns of failure occurrences in three OSS projects to be used for reliability behavior prediction of upcoming releases by making use of traditional method of Software Reliability Growth for forecasting failures occurrences. Results show that in all the cases, a predetermined pattern is tracked for failure occurrences.

4.3.10 Predicting Change Propagation

Mirarab et al. [52] proposed three BBN (Bayesian belief networks) models for predicting change propagation, and they differ in the source of information they used. These models are: Bayesian Dependency Model (BDM—uses dependency information only), Bayesian History Model (BHM—uses history information only), and Bayesian Dependency and History Model (BDHM—uses both dependency and history information).

Results show that, for IR metrics, the overall accuracy of BDM and BHM lies in the same range but BDHM clearly shows better results. Moreover, the average point-biserial correlation of BDHM is better than those of BDM and BHM. However, unlike the mean values, BDM's average is better than BHM's.

4.3.11 Stability Prediction Models

The authors presented stability prediction models using a combination of Bayesian classifiers to predict the stability of OSS components [58]. The proposed models are capable to explain links between architectural aspects of a software component and its stability behavior in the context of OSS. The approach is implemented using generic algorithm, and it is tested on a large-scale system (Java API). Results show that this approach outperforms state-of-the-art approaches while predicting quality and stability of OSS components.

We present summarized information about rest of evolution prediction models in Table 20.

We realize that time series-based ARIMA models have been largely used by the research community that cover aspects such as size, defect count and change request prediction. These models predict growth of project with better accuracy as compared with other models, but their accuracy varies depending on size of a project. We revealed that as modularity ratio of module increases, its maintenance effort decreases which leads to the improvement in software structure. The maintenance effort predictors should focus on the increase in modularity ratio of modules instead of focusing other attributes such as SLOC, etc. The time period in software development that shows stable development teams results in low defect count because reassigning bug from one developer to another results in an increase of its defect count [40]. SRGMs (Software Reliability Growth Models) have been used by two selected primary studies that cover fault prediction of OSS projects [47,55]. The evolution prediction models are found to be constrained to the size of projects because the performance of a model may vary if a project is not long lived or the size is not large. Lack of prior knowledge of future tasks involved may also degrade model's performance.

Defect prediction models are prominently used for evolution prediction of open source and close source software projects. We present comparative overview of feature of defect prediction models used for evolution prediction of OSS studies in Table 21.

4.4 Approaches for Evolution Process Support

This subsection discusses proposed methods, tools and approaches available for OSS evolution process support and their strengths/limitations. Summarized information about evolution process support approaches is presented in Table 22.

4.4.1 EVOSS (Evolution of Free and Open Source Software)

The authors presented EVOSS, a tool for the evolution management of OSS (Open Source Software) systems [60]. It is composed of "simulator" and "fault detector" component. The simulator is used to forecast failures before the effects made to the actual system and the fault detector component is used for discovering discrepancies in a system configuration model. It leads to the improvement of existing tools, capable of forecasting partial number of upgrade faults. This tool is a collection of command line tools and can be initiated from a Linux shell. It is tested in installations of real Linux distribution and this practice demonstrates that EVOSS tool deals with the strengths and limitations of package managers. Di Cosmo et al. [64] also proposed an approach called EVOSS (Evolution of free and Open Source Software), that depends on model-driven technique for improving upgrade predictions in FOSS products. For making prediction upgrades more precise, EVOSS deals with both static and dynamic facets of upgrades. The approach also promotes simulation upgrades for failures prediction before they make actual effects to the real system.

4.4.2 Feedback-Driven Quality Assessment

Ouktif et al. [61] proposed the continuous study of open source software projects for evolution examination using accessible possessions of code repository (CVS), exchanged mails and commitment log files. Monitoring evolution consists of three primary services such as complexity, growth, and quality control mechanism. The method of designing a environment of software evolution, a general CASE tool which makes it probable to incorporate functions for the support of quality improvement and risk alleviation is proposed. The anticipated approach was applied to GRASS (a large-scale open source GIS). The developed plugins allowed GRASS maintainers effectively carrying out automated maintenance and receiving feedback on performed evolution activities.

4.4.3 Adaptive Change Propagation Heuristics

Malik et al. [62] discussed that propagating changes accurately is vital for the evolution of complex open source software systems for avoiding the bugs. In the past, numerous

Table 21 Features of defect prediction models

Study ID and proposed models/approaches	Project type and domain	Programming languages	Methods used	Data sources used	Metrics used	Validation process
[40] Time series models ARIMA(1,1,0) ARIMA(0,1,1)	OSS	C, C++	Correlation Granger causality test	SVN/CVS Bug Tracking System	Source code base Graph Metrics Node rank Modularity Ratio Edit Distance	Not Addressed
[44] MLR (multiple linear regression) model	OSS IDE	JAVA	Statistical analysis Regression analysis (Binary regression-univariate, multivariate)	Bug Tracking System, Changelog	Class level metrics: Chidamber&Kemerer metrics L&K (Lorenz and Kidd's eleven metrics) Li's metric suite for OO programming	Internal and External validity addressed
[46] QALP score	OSS, Web browser	C, C++	Not mentioned	Bug Tracking System	Not specifically mentioned	Not addressed
[47,55] SRGM's (Software Reliability Growth Models)	OSS	C, C++, JAVA	SRGM	Sourceforge, Bugzilla Bug Tracking System	NM	Construct Validity
	Web server					
	Web browser					
[70] Software Clustering	OSS	C++, JAVA	Multivariate Linear Regression	PROMISE repository	NM	Internal, External, Construct and Conclusion validity
[71] CODEP (COmbined DEfect Predictor)	OSS	C++, JAVA	PCA analysis Logistic regression Bayesian network Receiver Operating Characteristic	NM	NM	Addressed
[72] Data selection Procedure, Feature Subset Selection		NM		NM	Object Oriented Metrics	Addressed

Table 21 continued

Study ID and proposed models/approaches	Project type and domain	Programming languages	Methods used	Data sources used	Metrics used	Validation process
[73] Universal Defect Prediction Model	OSS	NM	Confusion matrix, Area Under Curve (AUC), Wilcoxon rank sum tests	Sourceforge and Google code	Code and Process metrics	Internal, External, Construct and reliability threats
[75] Defect Prediction Model	OSS Web Server	JAVA	Multivariate logistic regression analysis, multivariate analysis, ROC analysis		Chidamber and Kemerer java metrics (ckjm)	Internal threats
[76] Cost-sensitive discriminative dictionary learning (CDDL) approach	NASA	NM	statistical test, i.e., Mcnemar's test	NM	NM	Not addressed
Merging information from CVS repository and Bugzilla database	OSS JDK	JAVA	Univariate logistic regression analysis	CVS, Bug Tracking System	Chidamber and Kemerer java metrics (ckjm)	Internal, External and Construct validity
[78] Prediction using statistical and machine learning methods	OSS Web Server	JAVA	Logistic regression ANN, random forest, bagging, boosting, ROC analysis	NM	Object Oriented CK metrics and QMOOD metrics	Not addressed
[79] Genetic Algorithm-based approach	OSS	JAVA	NM	NM	Chidamber&Kemerer metrics suite	Not Addressed
[80] JIT (Just-In-Time) Defect Prediction	OSS	JAVA, Perl, C, Ruby	SZZ algorithm Spearman correlation tests, AUC	VCS	Three different LA, LD, and LT (Lines of code in a file before the change) metrics to measure the size dimensions	Internal, External Validity addressed

Table 21 continued

Study ID and proposed models/approaches	Project type and domain	Programming languages	Methods used	Data sources used	Metrics used	Validation process
[86] Conceptual Cohesion of Classes	OSS	C++	Regression analysis, univariate and multivariate logistic regression analysis methods, LSI-based coherence measurement	Bugzilla	NM	NM
[87] Defect Decay Model: ED3M	NM	NM	Exponential Peeling, Nonlinear regression	NM	NM	Not Addressed
[88] Novel benchmark Framework	NM	NM	Different machine learning schemes	NASA and PROMISE repository	LOC	Not Addressed
[90] Bayesian network learners	NM	NM	BAYESIAN NETWORK CLASSIFIERS, K2 algorithm, MMHC algorithm, ROC, AUC, Bonferroni-Dunn tests	NASA repository	LOC, McCabe Halstead, Miscellaneous metrics	Not Addressed
[91] Association rule mining method	NM	NM	Association rule mining, SQL	NM	NM	Not Addressed
[92] Negative Binomial Regression Model	Industrial software projects	NM	Negative binomial reg. model	Change log	LOC	Not Addressed
[124] Model Driven	OSS	JAVA	Queries	Repository (OCL Queries)	No of Packages	Not Addressed

Table 22 Summary of evolution process support approaches/models

Proposed approach	Aspect	Study ID	Results
EVOSS	Fault detector	[60]	The approach is validated by applying it to Fedora and Debian-based systems consisting of ≈1400 installed packages
Feedback-driven Quality Assessment	Remote and continuous analysis of open source software to monitor evolution	[61]	In real time, GRASS maintainers have successfully carried out automatic maintenance and the feedback of evolution activities performed
Adaptive Change Propagation Heuristics	Change Propagation	[62]	An empirical case study conducted on four large and complex open source systems: FreeBSD (an operating system), GCC (a compiler), GCluster (a clustering framework) and PostgreSQL (a database) results demonstrated that adaptive change propagation heuristics shows 57% considerable progress as compared to the out-performing static change propagation heuristics
SE^2 Model	Feature location (FA), Software change impact analysis (IA), and Expert developer recommendation (DR)	[63]	The general accuracy of correctly recommended developers is between 47 and 96% and between 43 and 60% for bug reports and feature requests respectively
A Two-dimensional Classification Model of OSS	Degree of maturity/visibility of organizational structure governing development projects	[66]	The model provides understanding of the current status and possible evolutionary paths
KERIS	Extensible modules as the basic building blocks for software	[67]	It allows extensibility of linked systems by restoring sub-modules with well-matched releases without the need to re-link the whole system. A compiler prototype for KERIS is also implemented
Lean–Kanban approach	Reduce the average time needed to complete maintenance requests	[68]	The results reveal that effective modeling and simulation is possible, by putting actors and events in use. Moreover, maintenance process in which flow of issues is gone through series of tasks, correctly reproduce key statistics of real data

Table 22 continued

Proposed approach	Aspect	Study ID	Results
FLOSS Staged Evolution Model	FLOSS software life cycle	[96]	Not mentioned
Maintenance Process Eval	Evaluation of OSS maintenance	[56]	Results showed that most of the projects were stabilizing and maintenance was active in all the projects because most of the defect reports were resolved
Exogenous Factors	a) Low inclusion of a project in repository increases its success?	[99]	1. A general framework relates several types of FLOSS repositories and gives a better perspective of describing the diversity of results, i.e., success of the normal OSS project
	b)The quality of data sources	[100]	2. It shows OSS open source software evolution analysis and history, and that the forecasting its future depends on data sources quality and consequent process data
	c)Comparative analysis between FLOSS repositories	[59]	3. Results have shown that during project evolution OSS repositories act as exogenous factors
	d)SS data analysis	[97]	4. Results revealed that largely used systems in libre software projects are SCM and offer detailed information about software development. Where, mailing lists and forums are generally core communication channels being used in libre software projects
FAULTTRACER	Ranks program edits according to their suspiciousness	[69]	FAULTTRACER lessen the changes to be examined manually by more than 50% on data sets of real regression faults, and more than 60% on seeded faults data set while comparing method-level changes in comparison to existing ranking heuristic
Commenting practice	Understanding the processes and practices of open source software development	[95]	It was realized that the average comment density varies by programming language but remains constant on several other dimensions
Configuration management	Configuration management process to analyze open source software projects	[105]	It was concluded that configuration management process has potential benefits for coordinating and synchronizing different activities of geographically located personals for OSS projects

heuristics have been proposed for change propagation. The authors proposed adaptive change propagation heuristics. These are meta-heuristics which are combinations of different previously researched heuristics for overall performance improvement (recall and precision) of change propagation heuristics. An empirical case study conducted on four large and complex open source systems: FreeBSD (an operating system), GCC (a compiler), GCluster (a clustering framework) and PostgreSQL (a database). Results demonstrate that adaptive change propagation heuristics shows 57% considerable progress as compared to static change propagation heuristics.

4.4.4 *SE^2*

Kagdi et al. [63] proposed "SE^2 " an integrated approach, for supporting three important tasks of software evolution and maintenance such as: software change impact analysis (IA), expert developer recommendation (DR) and feature location (FA). This approach is comprised of evolutionary and conceptual associations concealed in both software artifacts, i.e., structured and unstructured. For analyzing and deriving these relationships, Mining Software Repositories (MSR) and Information Retrieval (IR)-based techniques are used. A single framework is used to support all the three tasks. To assess IA, evaluations are conducted on number of changes in open source systems httpd, Apache, iBatis, KOffice and ArgoUML. Results showed that considerable enhancement in recall and precision values can be obtained by joining two couplings. Further, an introductory assessment of DR based on change requests of three OSS open source systems Eclipse, KOffice and ArgoUML was conducted. The general accuracy of correctly recommended developers is between 47 and 96% and between 43 and 60% for bug reports and feature requests respectively.

4.4.5 Two-Dimensional Classification Model of OSS

The author [66] proposed two-dimensional classification model, i.e., Institution (shows the degree of maturity/visibility of organizational structure governing development projects) and industrial (shows the degree of industrial solution goals. If the second is high, it shows the industrial attention to the completeness of the package. If it is low, it shows the focus on the development of each component) and it can be used in designing the evolution of an OSS project in order to best utilize corporate engagement. The model provides understanding of the current status and possible evolutionary paths.

4.4.6 KERIS

Matthias Zenger [67] presented the programming language KERIS, a Java extension with precise support for software evolution. Extensible modules are introduced as essential building blocks for software. These modules are composed in a hierarchy and reveal system architecture. Module design differs in the sense that modules are not linked manually. Rather, it gets inferred. The KERIS module assembly and refinement method are not constrained to the unexpected extensibility of atomic modules. It also allows extensibility of already systems linked by replacing selected sub-modules with companionable versions without re-linking the whole system. He has also implemented a compiler prototype for KERIS, where compiler reads KERIS source code and make standard Java class files for classes as well as modules.

4.4.7 Simulating the Software Maintenance Process

Concas et al. [68] presented simulation studies showing that Lean–Kanban method is very helpful because it reduces the average time needed for completing maintenance requests. They developed a process simulator for simulating maintenance processes which do not use a WIP limit. Two case studies are further performed that make use of real maintenance data taken from a Microsoft project and from a Chinese software firm. Results reflect that effective modeling and simulating is possible by using actors and events and maintenance process in which processing of flow of issues is through a set sequence activities.

4.4.8 Exogenous Factors

Beecher et al. [99] evaluated how the enclosure of a project in repository affects its "success". They selected six repositories and selected 50 projects from each repository. They further studied four process and product characteristics of the selected projects. While testing for existence of same results when studying FLOSS repositories, results showed that it is not just the repositories that differ in context of product and process characteristics (or both), but the two groups have also shown considerable changes among them. The first group (KDE and Debian) constantly differ when compared to a second group (Savannah, SourceForge and RubyForge). Afterward, it was revealed that two repositories (Debian and KDE) got considerably good outcomes as compared to the second group.

The authors address the quality of data sources that give detail of the dynamics influencing software process data characteristics and quality taken from version control system [100]. The results reflect that evolution analysis, open source software history and future forecasting are dependent on the quality of data sources and subsequent data processes.

Capiluppi et al. [59] presented a relative study on two repositories of FLOSS (SourceForge and Debian). Architectural structure of selected projects from every single repository was assessed. They concluded that a repository itself is influenced by the same structure as an exogenous factor. Robles et al. [97] also address the issues initiated while preparing and eliciting data analysis for open source software projects. They also proposed tools for supporting data extraction and analysis.

4.4.9 FLOSS Staged Evolution Model

It is a conventional staged model that characterizes the life cycle of software into series of steps. Despite using a model which is founded after examining the customary development of software, Capiluppi et al. [96] modified the staged model for its appliance on open source software evolution. The proposed model consists of four stages: initial development, evolution changes, servicing and phase out. Results show the general commonalities among commercial and FLOSS evolutionary behavior such as a point of stabilization with less functionality added is found in evolutionary behavior some FLOSS where initial development opts to be super-linear. Apart from the resemblance, three points of difference are also found such as: releases availability, changeover between evolution and servicing stage and changes made to the model is a probable switch among phases of evolution and phase out.

4.4.10 FAULTTRACER

Zhang et al. [69] presented a new approach, FAULTTRACER, for ranking program edits depending upon their suspiciousness for diminishing developers attempt in examining the influence of changes manually. It uses spectrum-based fault localization technique that supposes statements primarily executed by failed tests are more suspicious. Further, an experimental study was conducted using 23 versions of four Java programs from Software Infrastructure Repository. FAULTTRACER lessen the changes to be examined manually by more than 50% on data sets of real regression faults, and more than 60% on seeded faults data set while comparing method-level changes in comparison to existing ranking heuristic. Fault localization component of FAULTTRACER is found 80% more efficient as compared to traditional spectrum-based fault localization. FAULTTRACER is integrated with Eclipse IDE and is also implemented as a toolkit freely accessible.

4.4.11 Commenting Practice

In order to understand the practices and processes of OSS development, Arafat et al. [95] used source code comments as an indicator of maintainability. The comment density, i.e., code metric was focused. Commenting practice is an incorporated task in open source software development and that this project is consistently being followed in successful OSS projects. They also found that the average comments density remains constant in several other dimensions but varies by the programming language.

4.4.12 Maintenance Process Evaluation

Koponen et al. [56] proposed a framework for OSS maintenance evaluation. It includes several features for type, activity and quality of the maintenance evaluation. It has been evaluated with five case studies. Results of these case studies have shown that this framework can be used for evaluating maintenance of OSS though manual evaluation is very demanding and time consuming because of large datasets. Case studies show that most of the projects are stable and maintenance is active in all the projects as usually reported defects are resolved. Furthermore, maintenance tasks are not even traceable in all case studies; and usually changes even do not refer to reported defects in most all case studies. Therefore, results showed that maintenance does not make efficient use of Version Management System and defects.

We report 13 different approaches/models/tools available for evolution process support that cover different aspects such as fault detection, change propagation, evolution models, exogenous factors. All approaches have been tested and validated on well-known sets of OSS projects by different researchers. We conclude that SCM systems are largely used in free software projects and give detailed information about software development, while forums and mailing lists are generally the main channels of communication used in open source software projects. It has also been identified that FLOSS repositories act as exogenous factors for the evolution of OSS projects.

5 Validity Threats

Validity is the degree to measure accuracy of a conclusion. It is concerned with the accuracy of our measurement and the quality of guidelines followed for this systematic mapping study. The merits and demerits of the study with respect to the validity of the outcome can be assessed based on the discussion of validity threats. Significant threats to validity are choice of research databases, selection of primary studies, data extraction and conclusion validity.

5.1 Choice of Research Databases

The selection of limited research databases for contributions could be seen as a limitation of our systematic mapping study. We selected most relevant database libraries plus proceedings of International Conference on Open Source Systems and Journal of Software: Evolution and Process. However, it is possible that some relevant studies are missed which are published in different other venues.

5.2 Selection of Primary Studies

The main motive of conducting a systematic mapping studies is to identify as many empirical studies as possible that cover state-of-the-art research relevant to the context of the main topic. For this, a total of seven databases were searched and a large number of studies were retrieved. The search string was formulated and refined with combination of different keywords. This search string was useful in extracting the most relevant studies needed for the review. Inclusion/exclusion criteria are applied on the articles followed by a quality assessment criterion to find out the related articles which mainly focused on the target topic. However, 98 selected studies which qualified our assessment criteria may still have chances about their selection due to involvement of only two authors of this paper for the selection of primary studies that might make our results unreliable. It may be also possible that we missed papers that should be included in 98 primary studies. We additionally applied a checklist method to determine the quality of selected primary studies which reduces the threat of researcher's biases for the selection of primary studies. Furthermore, we selected primary studies from academic indexing services which has limited our data source to academic contributions, expect with contributions that are co-authored with industrial researchers.

5.3 Data Extraction

During the final data extraction step, 98 primary studies are analyzed and different pieces of information are elicited. Due to the high variability on the quality, location and level of detail of the information that was provided in the contributions, (findings, research methods and other type of information) identifying the different pieces of data within the document was highly difficult. This situation may lead to missing information due to the fact that only two authors of this study performed the task of data extraction. Due to large variations of metrics, models, methods and tools used for evolution of OSS studies in different contexts, we were not able to perform meta-analysis of the primary studies. Furthermore, data are extracted from the primary studies with respect to our research questions, which is primarily about evolution prediction of OSS studies and evolution process support. Most studies focused evolution prediction at code level. It is worthwhile to investigate evolution studies at architecture, design and requirement levels.

5.4 Conclusion Validity

Conclusion validity is the degree to measure relationship between the reality and findings of an experiment. Conclusion validity of our study refers to the degree to which the findings and conclusions of our systematic mapping are empirically sound and reliable. We tried to mitigate this threat by collecting data about primary studies is a structured way and derived conclusions following a rigorous analysis process. In Sect. 6, we present conclusions about our study, including summarization of results, research trends and future research directions. Our conclusions are based on statistical data extracted from academic literature. A threat to conclusion validity may that that we did not compare our findings with industrial surveys and questionnaires.

6 Summary and Conclusion

We present a systematic mapping of OSS evolution and process support studies in this paper. We undertake two aspects of evolution studies, i.e., OSS Evolution Prediction and OSS Evolution Process Support in the context of two research questions. The targeted time period of our research is from January 2000 to 2015. Automated and manual search for the relevant literature resulted in 1513 papers that were imported into Zotero library. Ninety-eight papers are selected for final review after discarding duplicates, title/abstract and full text scan. Eighty papers are related to OSS Evolution Prediction, and eighteen papers focus on OSS Evolution Process support.

In response to our RQ 1, the results reflect that OSS Evolution Prediction studies have largely focused on *Defect Prediction*. A large number of methods, models, metrics and data sources are presented for predicting defects of OSS studies. Other aspects of evolution prediction are: change propagation, size, refactoring, maintenance effort, contribution evolution, SOC and clone evolution. SVN/CVS is found to be the largest dataset used. Researchers employed statistical methods (62%) and machine learning algorithms (21%) for predicting different aspect of OSS studies. We found that only 15% studies performed experiments on common case studies (Linux, Apache) for evolution prediction and evolution process support of OSS studies as shown in Table 10. We identified 20 categories of metrics used for the evolution prediction of different aspects of open source projects, and SLOC is found to be extensively used metric (26% studies). SLOC metrics is used for predicting size, defects, maintenance effort, change propagation, refactoring, evolution of contribution and self-organized criticality. Researchers have contradictions on the predictive power of metrics used for the evolution of OSS studies. These contractions are discussed in Sect. 4.2. There is a need of further research to empirically *evaluate predictive power* of different metrics. Most metrics are applied on the file level for the evolution prediction of OSS studies as highlighted in Table 19. We also analyzed that class level metrics are applied by few studies but method-level metrics are applied by none of the selected primary studies. Moreover, we also reveal that code level metrics are applied by most researchers for evolution prediction of OSS studies. Little attention is paid to *requirements, design and architectural* level metrics for predicting evolution of OSS studies.

A number of models have been proposed that focus on different aspects of OSS prediction. We analyzed that time series-based ARIMA models are largely used that cover aspects such as size, defect count, maintenance effort and change request prediction. These models predict growth rate of projects with better accuracy but their accuracy vary depending on the size of a project. They also make reliable prediction with less error rate. In response to RQ2, we found that the OSS studies focusing on evolution process support used different methods, tools and approaches for OSS evolution process support. OSS Evolution process support studies have usually focused on evolution models, exogenous factors, maintenance support, fault detection and change propagation aspects. A very little effort has been paid to the other aspects such as *Configuration Management, Growth, Complexity and Control, possible evolutionary paths* etc. SVN/CVS is again found to be the largest explored dataset. The detailed explanation about these aspects is presented in Sect. 4.4.

In both cases of evolution prediction and evolution process support, the reviewed articles admitted the necessity of external validity, whereas the ratio of articles not addressing validation process is considerably higher (56% for evolution prediction and 68% for evolution process support). We realized that vast heterogeneity of evolution prediction models building data make their evaluation difficult. *Generalization of evolution prediction models* regardless of their applicability on project size requires the attention of researchers.

Future research should also focus on predicting change propagation, size, refactoring, maintenance effort, contribution evolution, SOC and clone evolution besides defect prediction. The architectural and requirements change evolution is also undermined area that needs attention of researchers. The tools and approaches proposed for evolution also admit the necessity of external validation. Future research should focus on the above mentioned issues and try to make them more generalized regardless of the domain of OSS projects.

References

1. Fernandez-Ramil, J.; Lozano, A.; Wermelinger, M.; Capiluppi, A.: Empirical studies of open source evolution. In: Software Evolution, pp. 263–288. Springer (2008)
2. Spinellis, D.; Giannikas, V.: Organizational adoption of open source software. J. Syst. Softw. **85**(3), 666–682 (2012)
3. Xie, G.; Chen, J.; Neamtiu, I.: Towards a better understanding of software evolution: an empirical study on open source software. Proc. Softw. Maint. ICSM **2009**, 51–60 (2009)
4. Cosentino, V.; Luis, J.; Cabot, J.: Findings from GitHub: methods, datasets and limitations. In: Proceedings of the 13th International Workshop on Mining Software Repositories, pp. 137–141 (2016)
5. Mahbubul Syeed, M.M.; Hammouda, I.; Systä, T.: Evolution of open source software projects: a systematic literature review. J. Softw. **8**, 2815–2829 (2013)
6. Breivold H.P.; Chauhan M.A.; Babar M.A.: A systematic review of studies of open source software evolution. In: Software Engineering Conference (APSEC), 2010 17th Asia Pacific, pp. 356–365 (2010)
7. Koch, S.: Software evolution in open source projects—a large scale investigation. J. Softw. Maint. Evol. Res. Pract. **19**(6), 361–382 (2007)
8. Petersen, K.; Feldt, R.; Mujtaba, S.; Mattsson, M.: Systematic mapping studies in software engineering. In: 12th International Conference on Evaluation and Assessment in Software Engineering, pp. 71–80 (2008)
9. Lehman, M.M.; Belady, L.A.: Program evolution-processes of software change. Academic Press, London (1985)
10. Lehman, M.M.; Ramil, J.F.: Rules and tools for software evolution planning and management. Ann. Softw. Eng. **11**, 15–44 (2001)
11. Software Engineering Group. Department of Computer Science, Guidelines for performing Systematic Literature Reviews in Software Engineering. EBSE Technical Report EBSE-2007-01, Version 2.3. 9 (July, 2007)
12. Ivarsson, M.; Gorschek, T.: Technology transfer decision support in requirements engineering research: a systematic review of REj. Requir. Eng. **14**(2009), 155–175 (2009)
13. Robles G.; Amor J.J.; Gonzalez-Barahona J.M.; Herraiz I.: Evolution and growth in large libre software projects. In: Proceedings of Eighth International Workshop on Principles of Software Evolution (IWPSE 2005), pp. 165–174 (2005)
14. Capiluppi, A.; Lago, P.; Morisio, M.: Evidences in the evolution of OS projects through Changelog analyses. In: Proceedings 3rd Workshop on Open Source Software Engineering, 25th International Conference on Software Engineering, pp. 19–24 (2003)
15. Robles-Martinez, G.; Gonzalez-Barahona, J.M.; Centeno-Gonzalez, J.; Matellan-Olivera, V.; Rodero-Merino, L.: Studying the evolution of libre software projects using publicly available data. In: Proceedings of 3rd Workshop on Open Source Software Engineering, 25th International Conference on Software Engineering, pp. 111–116 (2003)
16. Fogel, K.: Open source development with cvs. Coriolis Open Press, Scottsdale (1999)
17. Kitchenham, B.; Brereton, O.P.; Budgen, D.; Turner, M.; Bailey, J.; Linkman, S.: Systematic literature reviews in software engineering-a systematic literature review. Inf. Softw. Technol. **51**(1), 7–15 (2009)
18. Lee, Y.; Yang, J.; Chang, K.H.: Metrics and evolution in open source software. In: Seventh International Conference on Quality Software (QSIC 2007), pp. 191–197 (2007)
19. Kitchenham, B.A.; Budgen, D.; Brereton, O.P.: Using mapping studies as the basis for further research—A participant–observer case study. Inf. Softw. Technol. **53**(6), 638–651 (2011)
20. Syeed, M.M.; Kilamo, T.; Hammouda, I.; Systä, T: Open source prediction methods: a systematic literature review. In: IFIP International Conference on Open Source Systems, pp. 280–285 (2012)
21. Catal, C.; Diri, B.: A systematic review of software fault prediction studies. Expert Syst. Appl. **36**(4), 7346–7354 (2009)

22. Herraiz, I.; Rodriguez, D.; Robles, G.; Gonzalez-Barahona, J.M.: The evolution of the laws of software evolution: a discussion based on a systematic literature review. ACM Comput. Surv. (CSUR) **46**(2), 28 (2013)
23. Lehman, M.M.; Ramil, J.F.: An approach to a theory of software evolution. In: Proceedings of the 4th International Workshop on Principles of Software Evolution, pp. 70–74 (2001)
24. Banker, R.D.; Datar, S.M.; Kemerer, C.F.; Zweig, D.: Software complexity and maintenance costs. Commun. ACM **36**(11), 81–94 (1993)
25. Obdam, T.S.; Rademakers, L.W.M.M.; Braam, H., Eecen, P.: Estimating costs of operation & maintenance for offshore wind farms. In: Proceedings of European Offshore Wind Energy Conference, pp. 4–6 (2007)
26. Farber, D.: Six barriers to open source adoption, ZDNet Tech Update, March (2004) [WWW document]
27. Petersen, K.; Vakkalanka, S.; Kuzniarz, L.: Guidelines for conducting systematic mapping studies in software engineering: an update. Inf. Softw. Technol. **64**, 1–18 (2015)
28. Smith, N.; Capiluppi, A.; Ramil, J.F.: A study of open source software evolution data using qualitative simulation. Softw. Process: Improv. Pract. **10**(3), 287–300 (2005)
29. Skoulis, I.; Vassiliadis, P.; Zarras, A.V.: Growing up with stability: how open-source relational databases evolve. Inf. Syst. **53**, 363–385 (2015)
30. Keele, S.: Guidelines for performing systematic literature reviews in software engineering. pp. 1–57, Technical report, EBSE Technical Report EBSE-2007-01 (2007)
31. Chidamber, S.R.; Kemerer, C.F.: A metrics suite for object oriented design. IEEE Trans. Softw. Eng. **20**(6), 476–493 (1994)

Primary Studies

32. Godfrey, M.W.; Tu, Q.: Evolution in open source software: A case study. In: Proceedings of International Conference on Software Maintenance, pp. 131–142 (2000)
33. Nakakoji, K.; Yamamoto, Y.; Nishinaka, Y.; Kishida, K.; Ye, Y.: Evolution patterns of open-source software systems and communities. In: Proceedings International Workshop on Principles of Software Evolution, pp. 76–85 (2002)
34. Paulson, J.W.; Succi, G.; Eberlein, A.: An empirical study of open-source and closed-source software products. IEEE Trans. Softw. Eng. **30**(4), 246–256 (2004)
35. Herraiz, I.; Gonzalez-Barahona, J.M.; Robles, G.; German, D.M.: On the prediction of the evolution of libre software projects. In: Proceedings of IEEE International Conference on Software Maintenance, pp. 405–414 (2007)
36. Goulão, M.; Fonte, N.; Wermelinger, M.; Brito e Abreu, F.: Software evolution prediction using seasonal time analysis: a comparative study. In: Proceedings of 16th European Conference on Software Maintenance and Reengineering, pp. 213–222 (2012)
37. Herraiz, I.; Robles, G.; Gonzalez-Barahona, J.M.: Towards predictor models for large libre software projects. ACM SIGSOFT Softw. Eng. Notes **30**(4), 1–6 (2005)
38. Herraiz, I.; Gonzalez-Barahona, J.M.; Robles, G.: Determinism and evolution. In: Proceedings of the Working Conference on Mining Software Repositories, pp. 1–10 (2008)
39. Yu, L.: Indirectly predicting the maintenance effort of open source software. J. Softw. Maint. Evol. Res. Pract. **18**(5), 311–332 (2006)
40. Bhattacharya, P.; Iliofotou, M.; Neamtiu, I.; Faloutsos, M.: Graph-based analysis and prediction for software evolution. In: Proceedings of the 34th International Conference on Software Engineering, pp. 419–429 (2012)
41. Raja, U.; Hale, D.P.; Hale, J.E.: Modeling software evolution defects: a time series approach. J. Softw. Maint. Evol. Res. Pract. **21**(1), 49–71 (2009)
42. Wu, J.; Holt, R.C.; Hassan, A.E.: Empirical evidence for SOC dynamics in software evolution. In: Proceedings of International Conference on Software Maintenance, pp. 244–254 (2007)
43. Ratzinger, J.; Sigmund, T.; Vorburger, P.; Gall, H.: Mining software evolution to predict refactoring. In: First International Symposium on Empirical Software Engineering and Measurement, pp. 354–363 (2007)
44. Shatnawi, R.; Li, W.: The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process. J. Syst. Softw. **81**(11), 1868–1882 (2008)
45. Illes-Seifert, T.; Paech, B.: Exploring the relationship of a file's history and its fault-proneness: an empirical method and its application to open source programs. Inf. Softw. Technol. **52**(5), 539–558 (2010)
46. Binkley, D.; Feild, H.; Lawrie, D.; Pighin, M.: Software fault prediction using language processing. In: Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION, pp. 99–110 (2007)
47. Ullah, N.: A method for predicting open source software residual defects. Softw. Quality J. **23**(1), 1–22 (2014)
48. Knab, P.; Pinzger, M.; Bernstein, A.: Predicting defect densities in source code files with decision tree learners. In: Proceedings of the International Workshop on Mining Software Repositories, pp. 119–125 (2006)
49. Ekanayake, J.; Tappolet, J.; Gall, H.C.; Bernstein, A.: Tracking concept drift of software projects using defect prediction quality. In: Proceedings of 6th IEEE International Working Conference on Mining Software Repositories, pp. 51–60 (2009)
50. Shawky, D.M.; Ali, A.F.: Modeling clones evolution in open source systems through chaos theory. In: Proceedings of 2nd International Conference on Software Technology and Engineering (ICSTE), vol. 1, pp. V1-159 (2010)
51. Ratzinger, J.; Gall, H.; Pinzger, M.: Quality assessment based on attribute series of software evolution. In: Proceedings of 14th Working Conference on Reverse Engineering, pp. 80–89 (2007)
52. Mirarab, S.; Hassouna, A.; Tahvildari, L.: Using bayesian belief networks to predict change propagation in software systems. In: Proceedings of 15th IEEE International Conference on Program Comprehension, pp. 177–188 (2007)
53. Askari, M.; Holt, R.: Information theoretic evaluation of change prediction models for large-scale software. In: Proceedings of the 2006 International Workshop on Mining Software repositories, pp. 126–132 (2006)
54. Olague, H.M.; Etzkorn, L.H.; Messimer, S.L.; Delugach, H.S.: An empirical validation of object oriented class complexity metrics and their ability to predict error prone classes in highly iterative, or agile, software: a case study. J. Softw. Maint. Evol. Res. Pract. **20**(3), 171–197 (2008)
55. Rossi, B.; Russo, B.; Succi, G.: Modelling failures occurrences of open source software with reliability growth. In: IFIP International Conference on Open Source Systems, pp. 268–280 (2010)
56. Koponen, T.: Evaluation framework for open source software maintenance. In: Proceedings of International Conference on Software Engineering Advances, pp. 52-52 (2006)
57. Biçer, S.; Bener, A.B.; Çağlayan, B.: Defect prediction using social network analysis on issue repositories. In: Proceedings of the International Conference on Software and Systems Process pp. 63–71 (2011)
58. Bouktif, S.; Sahraoui, H.; Ahmed, F.: Predicting stability of open-source software systems using combination of Bayesian classifiers. ACM Trans. Manag. Inf. Syst. (TMIS) **5**(1), 1–26 (2014)

59. Capiluppi, A.; Beecher, K.: Structural complexity and decay in FLOSS systems: An inter-repository study. In: Proceedings of 13th European Conference on Software Maintenance and Reengineering, pp. 169–178 (2009)
60. Di Ruscio, D.; Pelliccione, P.; Pierantonio, A.: EVOSS: A tool for managing the evolution of free and open source software systems. In: Proceedings of 34th International Conference on Software Engineering (ICSE), pp. 1415–1418 (2012)
61. Ouktif, S.; Antoniol, G.; Merlo, E.; Neteler, M.: A feedback based quality assessment to support open source software evolution: the GRASS case study. In: Proceedings of 22nd IEEE International Conference on Software Maintenance, pp. 155–165 (2006)
62. Malik, H.; Hassan, A.E.: Supporting software evolution using adaptive change propagation heuristics. In: Proceedings of IEEE International Conference on Software Maintenance, pp. 177–186 (2008)
63. Kagdi, H.; Gethers, M.; Poshyvanyk, D.: SE 2 model to support software evolution. In: Proceedings of 27th IEEE International Conference on Software Maintenance, pp. 512–515 (2011)
64. Di Cosmo, R.; Di Ruscio, D.; Pelliccione, P.; Pierantonio, A.; Zacchiroli, S.: Supporting software evolution in component-based FOSS systems. Sci. Comput. Program. **76**(12), 1144–1160 (2011)
65. Di Ruscio, D.; Pelliccione, P.: Simulating upgrades of complex systems: the case of Free and Open Source. Softw. Inf. Softw. Technol. **56**(4), 438–462 (2014)
66. Yamakami, T.: A two-dimensional classification model of OSS: towards successful management of the evolution of OSS. In: Proceedings of 13th International Conference on Advanced Communication Technology (ICACT), pp. 1336–1341 (2011)
67. Zenger, M.: Keris: evolving software with extensible modules. J. Softw. Maint. Evol. Res. Pract. **17**(5), 333–362 (2005)
68. Concas, G.; Lunesu, M.I.; Marchesi, M.; Zhang, H.: Simulation of software maintenance process, with and without a work in process limit. J. Softw. Evol. Process **25**(12), 1225–1248 (2013)
69. Zhang, L.; Kim, M.; Khurshid, S.: FaultTracer: a spectrumbased approach to localizing failure-inducing program edits. J. Softw. Evol. Process **25**(12), 1357–1383 (2013)
70. Scanniello, G.; Gravino, C.; Marcus, A.; Menzies, T.: Class level fault prediction using software clustering. In: Proceedings of IEEE/ACM 28th International Conference on Automated Software Engineering (ASE), pp. 640–645 (2013)
71. Panichella, A.; Oliveto, R.; De Lucia, A.: Cross-project defect prediction models: L'Union fait la force. In: Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), pp. 164–173 (2013)
72. He, Z.; Peters, F.; Menzies, T.; Yang, Y.: Learning from open-source projects: An empirical study on defect prediction. In: Proceedings of ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 45–54 (2013)
73. Zhang, F.; Mockus, A.; Keivanloo, I.; Zou, Y.: Towards building a universal defect prediction model. In: Proceedings of the 11th Working Conference on Mining Software Repositories, pp. 182–191 (2014)
74. Wahyudin, D.; Schatten, A.; Winkler, D.; Tjoa, A.M.; Biffl, S.: Defect Prediction using Combined Product and Project Metrics-A Case Study from the Open Source Apache MyFaces Project Family. In: Proceedings of 34th Euromicro Conference on Software Engineering and Advanced Applications (2008)
75. Malhotra, R.: A defect prediction model for open source software. In: Proceedings of the World Congress on Engineering, vol. 2 (2012)
76. Jing, X.Y.; Ying, S.; Zhang, Z.W.; Wu, S.S.; Liu, J.: Dictionary learning based software defect prediction. In: Proceedings of the 36th International Conference on Software Engineering, pp. 414–423 (2014)
77. English, M.; Exton, C.; Rigon, I.; Cleary, B.: Fault detection and prediction in an open-source software project. In: Proceedings of the 5th International Conference on Predictor Models in Software Engineering, p. 17 (2009)
78. Malhotra, R.; Jain, A.: Fault prediction using statistical and machine learning methods for improving software quality. JIPS **8**(2), 241–262 (2012)
79. Sandhu, P.S.; Dhiman, S.K.; Goyal, A.: A genetic algorithm based classification approach for finding fault prone classes. World Acad. Sci. Eng. Technol. **60**, 485–488 (2009)
80. Fukushima, T.; Kamei, Y.; McIntosh, S.; Yamashita, K.; Ubayashi, N.: An empirical study of just-in-time defect prediction using cross-project models. In: Proceedings of the 11th Working Conference on Mining Software Repositories, pp. 172–181 (2014)
81. Li, P.L.; Herbsleb, J.; Shaw, M.: Finding predictors of field defects for open source software systems in commonly available data sources: a case study of openbsd. In: 11th IEEE International Symposium on Software Metrics, p. 10 (2005)
82. Caglayan, B.; Bener, A.; Koch, S.: Merits of using repository metrics in defect prediction for open source projects. In: FLOSS'09. ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development, pp. 31–36 (2009)
83. Rahman, F.; Devanbu, P.: How, and why, process metrics are better. In: Proceedings of the 2013 International Conference on Software Engineering, pp. 432–441 (2013)
84. Olague, H.M.; Etzkorn, L.H.; Gholston, S.; Quattlebaum, S.: Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes. IEEE Trans. Softw. Eng. **33**(6), 402–419 (2007)
85. Premraj, R.; Herzig, K.: Network versus code metrics to predict defects: A replication study. In: International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 215–224 (2011)
86. Marcus, A.; Poshyvanyk, D.; Ferenc, R.: Using the conceptual cohesion of classes for fault prediction in object-oriented systems. IEEE Trans. Softw. Eng. **34**(2), 287–300 (2008)
87. Haider, S.W.; Cangussu, J.W.; Cooper, K.M.; Dantu, R.: Estimation of defects based on defect decay model: ED^3 M. IEEE Trans. Softw. Eng. **34**(3), 336–356 (2008)
88. Song, Q.; Jia, Z.; Shepperd, M.; Ying, S.; Liu, J.: A general software defect-proneness prediction framework. IEEE Trans. Softw. Eng. **37**(3), 356–370 (2011)
89. Zhou, Y.; Leung, H.: Empirical analysis of object-oriented design metrics for predicting high and low severity faults. IEEE Trans. Softw. Eng. **32**(10), 771–789 (2006)
90. Dejaeger, K.; Verbraken, T.; Baesens, B.: Toward comprehensible software fault prediction models using bayesian network classifiers. IEEE Trans. Softw. Eng. **39**(2), 237–257 (2013)
91. Song, Q.; Shepperd, M.; Cartwright, M.; Mair, C.: Software defect association mining and defect correction effort prediction. IEEE Trans. Softw. Eng. **32**(2), 69–82 (2006)
92. Ostrand, T.J.; Weyuker, E.J.; Bell, R.M.: Predicting the location and number of faults in large software systems. IEEE Trans. Softw. Eng. **31**(4), 340–355 (2005)
93. Pendharkar, P.C.; Subramanian, G.H.; Rodger, J.A.: A probabilistic model for predicting software development effort. IEEE Trans. Softw. Eng. **31**(7), 615–624 (2005)
94. Gyimothy, T.; Ferenc, R.; Siket, I.: Empirical validation of object-oriented metrics on open source software for fault prediction. IEEE Trans. Softw. Eng. **31**(10), 897–910 (2005)
95. Arafat, O.; Riehle, D.: The commenting practice of open source. In: Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications, pp. 857–864 (2009)

96. Capiluppi, A.; González-Barahona, J.M.; Herraiz, I.; Robles, G.: Adapting the staged model for software evolution to free/libre/open source software. In: Proceedings of Ninth International on Principles of Software Evolution: In Conjunction with the 6th ESEC/FSE Joint Meeting, pp. 79–82 (2007)
97. Robles, G.; González-Barahona, J.M.; Izquierdo-Cortazar, D.; Herraiz, I.: Tools for the study of the usual data sources found in libre software projects. Int. J. Open Sour. Softw. Process. (IJOSSP) **1**(1), 24–45 (2009)
98. Gonzalez-Barahona, J.M.; Robles, G.; Michlmayr, M.; Amor, J.J.; German, D.M.: Macro-level software evolution: a case study of a large software compilation. Empir. Softw. Eng. **14**(3), 262–285 (2009)
99. Beecher, K.; Capiluppi, A.; Boldyreff, C.: identifying exogenous drivers and evolutionary stages in FLOSS projects. J. Syst. Softw. **82**(5), 739–750 (2009)
100. Bachmann, A.; Bernstein, A.: Software process data quality and characteristics: a historical view on open and closed source projects. In: Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol) Workshops, pp. 119–128 (2009)
101. Aversano, L.; Di Brino, M.; Guardabascio, D.; Salerno, M.; Tortorella, M.: Understanding enterprise open source software evolution. Procedia Comput. Sci. **64**, 924–931 (2015)
102. Saini, M.; Kaur, K.: Software Evolution Prediction Using Fuzzy Analysis. In: Proceedings of Fourth International Conference on Emerging Applications of Information Technology (EAIT), pp. 349–354 (2014)
103. Aoki, A.; Hayashi, K.; Kishida, K.; Nakakoji, K.; Nishinaka, Y.; Reeves, B.; and Yamamoto, Y.: A case study of the evolution of Jun: an object-oriented open-source 3d multimedia library. In: Proceedings of the 23rd International Conference on Software Engineering, pp. 524–533 (2001)
104. Mockus, A.; Fielding, R.T.; Herbsleb, J.D.: Two case studies of open source software development: Apache and mozilla. ACM Trans. Softw. Eng. Methodol. **11**(3), 309–346 (2002)
105. Asklund, U.; Bendix, L.: A study of configuration management in open source software projects. IEE Proc. Softw. **149**(1), 40–46 (2002)
106. Schach, S.R.; Jin, B.; Wright, D.R.; Heller, G.Z.; Offutt, A.J.: Maintainability of the Linux kernel. IEE Proc. Softw. **149**(1), 18–23 (2002)
107. Bauer, A.; Pizka, M.: The contribution of free software to software evolution. In: Sixth International Workshop on Principles of Software Evolution, pp. 170–179 (2003)
108. Capiluppi, A.: Models for the evolution of OS projects. In: Proceedings of International Conference on Software Maintenance, pp. 65–74 (2003)
109. Capiluppi, A.; Morisio, M.; and Lago, P.: Evolution of understandability in oss projects. In: Proceedings of Eighth Euromicro Working Conference on Software Maintenance and Reengineering, pp. 58–66 (2004)
110. Capiluppi, A.; Ramil, J.F.; e Informatica, D.A.: Studying the evolution of open source systems at different levels of granularity: Two case studies. In: Proceedings of the 7th International Workshop on Principles of Software Evolution, pp. 113–118 (2004)
111. Al-Ajlan, A.: The evolution of open source software using eclipse metrics. International Conference on New Trends in Information and Service Science, pp. 211–218 (2009)
112. Capiluppi, A.; Faria, A.E.; Ramil, J.F.: Exploring the relationship between cumulative change and complexity in an open source system. European Conference on Software Maintenance and Reengineering (CSMR) (2005)
113. Izurieta, C.; and Bieman, J.: The evolution of freebsd and linux. In: Proceedings of the IEEE/ACM International Symposium on Empirical Software Engineering (2006)
114. Wang, Y.; Guo, D.; Shi, H.: Measuring the evolution of open source software systems with their communities. ACM SIGSOFT Softw. Eng. Notes **32**(6), 7 (2007)
115. Capiluppi, A.; Morisio, M.; Ramil, J.F.: The evolution of source folder structure in actively evolved open source systems. In: Software Metrics, International Symposium on, pp. 2–13 (2004)
116. Fischer, M.; Pinzger, M.; Gall, H.: Populating a release history database from version control and bug tracking systems. In: Proceedings of International Conference on Software Maintenance, pp. 23–32 (2003)
117. Robles, G.; Gonzalez-Barahona, J.M.; Michlmayr, M.; Amor, J.J.: Mining large software compilations over time: another perspective of software evolution. International Workshop on Mining Software Repositories, pp. 3–9 (2006)
118. Alenezi, M.; Almustafa, K.: Empirical analysis of the complexity evolution in open-source software systems. Int. J. Hyb. Inf. Technol. **8**(2), 257–266 (2015)
119. Alenezi, M.; Zarour, M.: Modularity measurement and evolution in object-oriented open-source projects. In: Proceedings of the International Conference on Engineering & MIS, p. 16 (2015)
120. Le, D.M.; Behnamghader, P.; Garcia, J.; Link, D.; Shahbazian, A.; Medvidovic, N.: An empirical study of architectural change in open-source software systems. In: Proceedings of the 12th Working Conference on Mining Software Repositories, pp. 235–245 (2015)
121. Gala-Pérez, S.; Robles, G.; González-Barahona, J.M.; Herraiz, I.: Intensive metrics for the study of the evolution of open source projects: Case studies from Apache Software Foundation projects. In: Proceedings of the 10th Working Conference on Mining Software Repositories, pp. 159–168 (2013)
122. Ben Charrada, E.; Koziolek, A.; Glinz, M.: Supporting requirements update during software evolution. J. Softw. Evol. Process **27**(3), 166–194 (2015)
123. Ruohonen, J.; Hyrynsalmi, S.; Leppänen, V.: Time series trends in software evolution. J. Softw. Evol. Process **27**(12), 990–1015 (2015)
124. Di Ruscio, D.; Pelliccione, P.: A model driven approach to detect faults in FOSS systems. J. Softw. Evol. Process **27**(4), 294–318 (2015)
125. Nzeko'o, A.J.N.; Latapy, M.; Tchuente, M.: Social Network Analysis of Developers' and Users' Mailing Lists of Some Free Open Source Software. In: Proceedings of IEEE International Congress on Big Data, pp. 728–732 (2015)
126. Kouroshfar, E.; Mirakhorli, M.; Bagheri, H.; Xiao, L.; Malek, S.; Cai, Y.: A Study on the Role of Software Architecture in the Evolution and Quality of Software. In: Proceedings of the 12th Working Conference on Mining Software Repositories, pp. 246–257 (2015)
127. Threm, D.; Yu, L.; Ramaswamy, S.; Sudarsan, S.D.: Using normalized compression distance to measure the evolutionary stability of software systems. In: Proceedings of 26th International Symposium on Software Reliability Engineering (ISSRE), pp. 112–120 (2015)
128. Malhotra, R.; Khanna, M.: Mining the impact of object oriented metrics for change prediction using Machine Learning and Search-based techniques. In: Proceedings of International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 228–234 (2015)
129. Zhang, W.; Yang, Y.; Wang, Q.: Network analysis of oss evolution: An empirical study on argouml project. In: Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution, pp. 71–80 (2011)