A Systematic Review of Studies of Open Source Software Evolution

Hongyu Pei Breivold
Industrial Software Systems
ABB Corporate Research
721 23 Västerås, Sweden
hongyu.pei-breivold@se.abb.com

Muhammad Aufeef Chauhan
Mälardalen University
721 78 Västerås, Sweden
aufeef@gmail.com

Muhammad Ali Babar
IT University of Copenhagen,
Denmark
malibaba@itu.dk

Abstract

Software evolution relates to how software systems evolve over time. With the emergence of the open source paradigm, researchers are provided with a wealth of data for open source software evolution analysis. In this paper, we present a systematic review of open source software (OSS) evolution. The objective of this review is to obtain an overview of the existing studies in open source software evolution, with the intention of achieving an understanding of how software evolvability (i.e., a software system's ability to easily accommodate changes) is addressed during development and evolution of open source software. The primary studies for this review were identified based on a pre-defined search strategy and a multi-step selection process. Based on their research topics, we have identified four main categories of themes: software trends and patterns, evolution process support, evolvability characteristics addressed in OSS evolution, and examining OSS at software architecture level. A comprehensive overview and synthesis of these categories and related studies is presented as well.

Keywords: systematic review, open source software, software evolution, evolvability

1. Introduction

Software evolution reflects "*a process of progressive change in the attributes of the evolving entity or that of one or more of its constituent elements*" [35]. Specifically, software evolution relates to how software systems evolves over time [58]. Software evolution is characterized by inevitable changes in software and increasing complexities, which may lead to huge costs if software cannot easily accommodate changes. Hence, One of the principle challenges in software evolution is the ability of software to evolve over time to meet the changing requirements of stakeholders [39]. In this context, software evolvability is an attribute that "*bears on the ability of a system to accommodate changes in its requirements throughout the system's lifespan with the least possible cost while maintaining architectural integrity*" [47].

With the emergence of the Open Source Software (OSS) paradigm, researchers have access to the code bases of a large number of evolving software systems along with their release histories and change logs. There have been a large number of studies published on OSS characteristics and evolution patterns by examining sequences of code versions or releases using statistical analysis. Meanwhile, the easily accessible data about different aspects of OSS projects also provides researchers with immense number of opportunities to validate the prior studies of proprietary software evolution [32] and to study how evolvability has been addressed in OSS evolution.

The main objective of this research is to systematically select and review published literature in order to build and present a holistic overview of the existing studies on OSS evolution; moreover, a secondary objective is to analyze the literature to find out how software evolvability is addressed during development and evolution of OSS. We are also interested in extracting information on the metrics that researchers use for measuring OSS evolution from different perspectives such as growth patterns, complexity patterns, processes and evolution effort estimation. The detailed research questions include:

- What are the main research themes that are covered in the scientific literature regarding open source software evolution, and analysis and achievement of evolvability-related quality attributes?
- What are the metrics that are used for OSS evolution measurement and analysis, and what are the limitations in using these metrics, if any?

In the rest of the paper, Section 2 describes the research method used. Section 3 presents and discusses the findings from this review. Section 4 discusses validity threats of the review and Section 5 concludes the paper.

2. Research methodology

This research was undertaken as a systematic review [29] which is a process of assessing and interpreting all available research related to a particular research topic. The process

consists of several stages: (i) development of a review protocol; (ii) identification of inclusion and exclusion criteria; (iii) searching relevant papers; and (iv) data extraction and synthesis. These stages are detailed in the following subsections.

Review Protocol was designed based on the Systematic Literature Review (SLR) guidelines [29]. The protocol specifies the background for the review, research questions, search strategy, study selection criteria, data extraction and synthesis of the extracted data. The protocol was developed mainly by one author and reviewed by the other two authors to reduce bias.

Inclusion and exclusion criteria mainly focused on including full papers in English from peer-reviewed journals, conferences, workshops and book chapters published until the end of 2009. We exclude studies that do not cover evolution of OSS, prefaces, articles in the controversial corner of journals, editorials, and summaries of tutorials, panels and poster sessions.

Search strategy was designed to search in a selected set of electronic databases: ACM Digital Library, Compendex, IEEE Xplore, ScienceDirect – Elsevier, SpringerLink, Wiley InterScience and ISI Web of Science. The search terms used for constructing search strings were:

"open source software" OR "libre software" OR "free software" OR "FOSS" OR "F/OSS" OR "F/OSSD" OR "FOSSD" OR "FLOSS" OR "F/LOSS" OR "OSSD".

The selection of studies was performed through a multi-step process:

- Searches in the databases to identify relevant studies by using the search terms;
- Exclude studies based on the exclusion criteria;
- Exclude irrelevant studies based on titles and abstracts;
- Obtain primary studies based on full text read.

TABLE I. DATA EXTRACTION FOR EACH STUDY

Extracted Data	Description
Study identity	Unique identity for the study
Bibliographic references	Author, year of publication, title and source of publication
Type of study	Book, journal paper, conference paper, workshop paper
Focus of the study	Main topic area and aspect of open source software being investigated
Research method used for data collection	Included technique for the design of the study, e.g. case study, survey, experiment, interview to obtain data, observation
Data analysis	Qualitative or quantitative analysis of data
Metrics used	The metrics used in data collection for analysis
Constraints and limitations	Identified constraints and limitations in each study

The searches in electronic databases were performed in two stages. At the first stage, the papers published until the end of 2008 were searched and then a separate complimentary search was performed for 2009 publications. After merging the search results and removing duplicates there were 11,439 papers published until 2008 and 1,921 papers published in 2009. After scanning all the papers by titles and abstracts, 134 papers were selected. In the final stage full paper text was scanned and we selected 41 papers for this review. The paper selection process involved at least two researchers to decide whether to include or exclude a paper. A paper was excluded if both researchers considered it irrelevant. Any disagreement was resolved through discussions and involvement of the third researcher.

Data extraction and synthesis were carried out by reading each of the 41 papers thoroughly and extracting relevant data, which were managed through bibliographical management tool EndNote and Spreadsheets. The data extraction was driven by a form show in Table I. For the data synthesis, we inspected the extracted data for similarities in terms of the focus of the studies in order to define how results could be compared. The results of the synthesis will be described in the subsequent sections.

3. Results

We present that the findings from our SLR on OSS evolution. First we provide some demographic information about the studies, which were included in our SLR. Then we present and discuss the findings from analyzing the data extracted from the reviewed studies in order to answer the research questions which motivated this SLR.

3.1 Overview of the included studies

It has been mentioned that we performed searches in multiple electronic databases. We found that the largest numbers of selected papers (22 papers) were published on OSS evolution from IEEE. The second largest numbers of papers, 9 papers, were published by ACM; while four selected papers were published by John Wiley & Sons in its Journal of Software Maintenance and Evolution. Trend of publications over years shows a positive growth except for year 2008. Only three papers on OSS evolution were published in that year. In year 2009, eleven papers were published showing that a good number of researchers are addressing OSS evolution.

Our review has found that the evolution trends and patterns is the most focused research area with 23 papers published on this topic. There were ten papers on the role of process support in evolution. However, there are quiet few numbers of papers addressing the characteristics of evolvability and architecture, with five and three papers respectively.

We also observed that there were not many researchers who have long term interest in OSS evolution research. There were only three authors Adrea Capiluppi, Gregorio Robles and Israel Herraiz who have more than or equal to three papers on any aspect of OSS evolution.

3.2 Categories of the reviewed studies

As described in the research methodology section, during the data synthesis phase, we examined the papers based on their similarities in terms of research topics and contents in order to categorize the included studies of OSS evolution. Besides classifying the included studies, we also examined the metrics used for assessing OSS evolution as well as the analysis methodology for collected data in each study. After examining the research topics, data analysis and findings addressed in each study, we identified four main categories of themes, one of which is further refined into sub-categories to group primary studies that share similar characteristics in terms of specific research focus, research concepts and contexts. The categories and sub-categories are:

- OSS evolution trends and patterns
  - Software growth
  - Software maintenance and evolution economics
  - Prediction of software evolution
- OSS evolution process support
- Evolvability characteristics
- Examining OSS evolution at software architecture level

These themes and their corresponding sub-categories will be further detailed in the following subsections. For each category of theme, we describe the category and related studies along with the metrics that are used to quantitatively or qualitatively analyze the OSS evolution. Finally an analysis of the studies is discussed with main findings summarized.

3.3 OSS Evolution Trends and Patterns

This category includes studies that focus on investigating OSS evolution trends and patterns. Based on their focus, the studies were further classified into three sub-categories: (i) software growth; (ii) software maintenance and evolution economics; and (iii) prediction of software evolution.

Software growth: The studies in this sub-category mainly focus on software growth and changes using a variety of metrics as shown in Table II.

Software growth modeling can be of interest for developing models to predict software evolution, maintainability and other characteristics [30]. Moreover, many OSS studies focus on utilizing the OSS evolution data to verify Lehman's laws of software evolution [33]; their findings either conform or diverge from the growth behavior of proprietary software. It is essential that the measures of software growth can actually represent and quantify the notion of software growth in order to obtain a reasonable comparison among the results from different studies. However, we noticed that there have been conflicting interpretations of some important operational definitions with respect to the metrics used for measuring software growth patterns. Some examples of the operational definitions that exhibit varying interpretations include *system growth*, *system change*, and *size*, which are discussed below.

TABLE II. SOFTWARE GROWTH METRICS

Study	Metrics
[1]	Number of packages, number of classes, total lines of code, number of statements
[2]	Types of extracted changes: addition of source code modules in successive versions of software; deletion; and modification
[11]	Initial size, current size, modules (folders), modules (files), average module size, days through versions, versions, version rate, delta size
[17]	Source file, source folder, source tree, size, RSN (release sequence number), level number, depth of a folder tree, width of a level, width of a folder tree, files added, modified or deleted
[23] [43]	Lines of code (LOC) in source files as a function of the time in days
[24]	Lines of source code, the number of packages, the changed and unchanged packages
[28]	LOC (lines of code), number of directories, total size in Kbytes, average and median LOC for header and source files, number of modules (files) for each subsystem and for the system as a whole
[30]	Number of LOC added to a file, including all types of LOC, e.g. also commentaries
[41]	Overall project growth in functions over time, overall project growth in LOC over time
[46]	Lines of source code, the number and size of packages
[49]	Lines of code (LOC), executable LOC, lines of code per comment ratio, functions added over each release, number of functions
[50]	Size in number of source code files, number of files handled (added, modified, deleted) between two subsequent releases, average complexity
[52]	Rate of growth with respect to release sequence number
[54]	Module, bugs, bug fixing and requirement implementation
[56]	Source code metrics, e.g., lines of code, number of modules, number of definitions

System growth: is measured by using the metric of percentage growth over time. There exist diverse interpretations of rate of growth. For instance, one assumption in some empirical studies [48, 49] on software evolution, as also suggested by Lehman [32], is to analyze and plot growth data with respect to the release sequence number (RSN). Another interpretation of rate of growth is reflected in [23], which the authors plotted growth rates against calendar dates rather than release numbers. Further, they suggest that plotting according to release numbers would have led to dips in the function curves because development and stable releases follow different behaviors. This interpretation of rate of growth is further confirmed in [52], which shows that due to the new temporal variables introduced by OSS, the rate of growth of OSS should be computed with respect to temporal variables such as the release date. It was also validated that different conclusions can be drawn when software evolution data are analyzed with respect to the release date rather than RSN. Therefore, diverse interpretations of rate of growth can pose a threat in properly interpreting the OSS evolutionary behaviors.

System change: Separating the characterizations of system growth and system change is a challenge [32]. A variety of change metrics can be used. For example, Xie and colleagues [56] used changes to program elements (such as types, global variables, function signatures and bodies) to characterize system change. Cumulative numbers of addition and deletion types of changes to these program elements are plotted. They reported that the majority of changes are made to functions.

It is also possible to count all the different files that have been added, modified and deleted between two subsequent releases in order to measure system changes [50]. In this case, the conventions used for measuring changes can lead to different results in interpreting the OSS evolutionary behaviors, e.g. whether or not taking into consideration of the changes in comment lines or minor changes in a single source line.

Size: Lehman suggests using the number of modules to quantify program size as he argues that this metric is more consistent than considering source lines of code [32]. However, there are different interpretations of a module. For instance, Simmons et al consider modules only at the file level [49]; while Capiluppi [11] studies both at file level and directory level, and discovers different OSS evolutionary behaviors depending on whether they consider directories or files as modules.

Instead of using modules as Lehman suggested, LOC (lines of code) is often used for measuring the size of OSS. For instance, Conley and Sproull [23] used number of uncommented lines of code because as they claim using number of source files would have meant losing some of the full story of the evolution of the system, especially at the subsystem level due to the variation in file sizes. Conly and Sproull also assume that the total number of uncommented LOC grows roughly at the same rate as the number of source files [23]. However, this assumption is not fully validated in a broader scope as it was only verified in some of the largest packages in Debian GNU/Linux [27].

Moreover, the definition of LOC varies as different studies interpret LOC differently, depending on the tools and available data sources used [40]. Koch 's definition of LOC considers all types of files, including comments and documentation [30]. Some other studies [23, 43] counts LOC in two ways: including blank lines and comments in source files (e.g., in .c and .h files) or ignoring blank lines and comments. This kind of counting applies only to source files and ignore other source artifacts such as configuration files, make-files and documentation.

Even the term 'source file' is defined in different ways. For example, [50] considers only files with extension .c as source files. Therefore, for systems involving a variety of source file extensions, different assumptions regarding file extensions and their belonging to the source code or not could lead to different values in size, which would affect the analysis results of different aspects of evolutionary behaviors [45].

Software maintenance and evolution economics: The uncertainties in software evolution arise from, to a certain extent, understanding how OSS would have evolved in terms of costs. Moreover, software evolvability concerns both business and technical perspectives as the choice of maintenance decisions from technical perspective needs to be balanced with economic valuation to mitigate risks. Therefore, another perspective in understanding OSS evolution trends is to analyze how software has evolved in terms of development and maintenance costs. Capra et al [19] analyzes the quality degradation effect, i.e., entropy of OSS by measuring the evolution of maintenance costs over time. The metric used in this study is function points. One assumption in this paper is that the maintenance costs are proportional to the time elapsed between the releases of two subsequent versions. The other study proposes an empirical model to measure evolutionary reuse and development cost which is an indicator of the effect of maintenance decisions made by OSS developers. The metric used is source lines of code (SLOC) [18].

Prediction of software evolution: The OSS history data over time can be utilized to predict its evolution. It has been mentioned that modeling software growth is essential for developing software evolution prediction models. Although there are many studies of monitoring OSS growth, comparatively fewer studies actually utilize the historical evolution data for the purpose of predicting its evolution. We find only three papers in this area. Herraiz et al describe using data from source code management repository to compute size of the software over time [26]. This information is used to estimate future evolution of the project. SLOC is used for counting program text that is not a comment or blank line regardless of the number of

statements or fragments of statements on the line. All lines that contain program headers, declarations, and executable and non-executable statements are excluded. Therefore, the results may vary if other sorts of files are considered.

Ye uses source code changes to indirectly predict the maintenance effort of OSS [57]. The metrics used include lag time between starting a maintenance task and closing the task, source code change at module level (number of modules added, deleted and modified), and source code change at line level (number of source LOC added, deleted and modified) in one maintenance task. Some threats in this study are that all module-level changes are treated in the same manner irrespective of the amount of changes as well as the effort for line-level changes.

Another way to predict OSS evolution has been studied in [42], which describes using data from monthly defect reports to build up time series model that can be used to predict the pattern of OSS evolution defects.

3.4 Evolution Process Support

This category includes studies that focus on OSS evolution support from various perspectives of software development process:

Feedback-driven quality assessment: An approach that is based on remote and continuous analysis of OSS evolution is proposed in [10]. This approach utilizes available data sources such as CVS versioning system repository, commitment log files and exchanged mails in order to provide services that mitigate software degradation and risks. The principle services include growth, complexity and quality control mechanism, feedback-driven communication service, and OSS evolution dashboard service.

Commenting practice: To understand the processes and practices of open source software development, Arafat and Riehle [3] treat the amount of comments in a given source code body as an indicator of its maintainability. They focus on one particular code metric, i.e., the comment density. According to them "commenting" practice is an integrated activity in OSS development and that successful OSS projects follow consistently this practice.

Exogenous factors: Capiluppi and Beecher [12] investigated whether or not an OSS system's structural decay can be influenced by the repository in which it is retained. Based on a comparative analysis of two repositories, they concluded that the repositories in which OSS are retained act as exogenous factors, which can be a differentiating factor in OSS evolvability. Beecher et al [7] extended that work by involving a greater number of repositories and strengthening the results with the formulation of different types of OSS repository along with a transition framework among the various types.

Robles et al [44] describe the problems that can be found when retrieving and preparing for OSS data analysis and present the tools that support data retrieval for OSS evolution analysis: source code, source code management systems, mailing lists, and bug tracking systems. In accordance with this study, Bachmann and Bernstein [5] address the quality of data sources and provides insights into the influencing factors to the quality and characteristics of software process data gathered from bug tracking database and version control system log files. These studies reflect that the analysis of the evolution and history of an open source software and the prediction of its future rely on the quality of data sources and corresponding process data.

Maintenance process evaluation: Koponen presents an evaluation framework for OSS maintenance process [31]. The framework includes attributes for evaluating activity, efficiency and traceability of defect management and maintenance processes.

Evolution model: The traditional staged model [8] represents the software lifecycle as a sequence of stages. Instead of using the model that was built mainly by observing traditional software development, Capiluppi et al [14] revise the staged model for its applicability to OSS evolution.

Configuration management: Asklund and Bendix [4] examine the configuration management process and analyzes how process, tool support, and people aspects of configuration management influences the OSS evolution.

3.5 Evolvability Characteristics

This category includes studies that focus on characteristics that can be considered important for software evolvability.

As indicated in [25], the evolution of open source projects is governed by a sort of *determinism*, i.e., the current state of the project is determined time ago. Their results also show that at least 80% of the sampled projects are short-term correlated. However, a long-term perspective to explicitly address evolvability for the entire software lifecycle is required since the inability to effectively and reliably evolve software systems means loss of business opportunities [9].

Another OSS evolvability characteristic is *code understandability* which is identified in [15]. This study views understandability as a key aspect for maintainability, and takes into account only code structure measures (such as code size, number of macro-modules and micro-modules, size of modules, and average size of modules) for calculating code indistinctness as an indicator of code understandability. Besides *determinism* and *understandability*, complexity and modularity are the other two OSS evolvability characteristics that are frequently described in the reviewed studies of OSS evolution.

Complexity

Complexity is a software characteristic that affects evolvability. Table III shows that a variety of metrics have been used to characterize OSS evolution from software

complexity perspective. According to Table III, McCabe's cyclomatic complexity [36] is the most often used metric. It measures the number of independent paths in the control flow graph. The rationale for using this metric is that the number of control flow paths is correlated to how well-structured the functions are in the program. Another metric is Halstead complexity which measures a program module's complexity directly from source code, with focus on computational complexity. These two complexity measures have different emphasis and therefore can be complementarily used. For instance, Simmons et al [49]'s study found that the McCabe and Halstead complexity metrics yielded contradictory results, which suggest that while the structure complexity decline with successive releases, the complexity of calculation logic increases.

TABLE III. COMPLEXITY METRICS

Study	Metrics
[1] [13] [21]	McCabe's cyclomatic complexity
[16]	System size and the evolving structure of the software
[17]	McCabe's cyclomatic complexity for structural complexity, Halstead Volume for textual complexity
[41]	Overall project complexity, average complexity of all functions, average complexity of functions added
[49]	Overall release complexity and average function complexity using McCabe and Halstead complexity measure
[56]	McCabe's cyclomatic complexity, common coupling and average number of function calls per function

Besides McCabe and Halstead indexes, there are other additional indicators of complexity, both at system and component level, as well as function level:

- *Calls per function* indicate the complexity of functions. It is computed by averaging the number of calls per function for all functions [56].
- *Coupling*, representing the number of inter-module references.
- *Interface complexity*, measuring the sum of input arguments to, and return states from, a function [51]. The number of arguments and state returns has impact on software changeability.
- *Complexity* of some systems may also be found in their data structures rather than in source code [41].

However, we did not find any research papers that explicitly study complexity in terms of coupling, interface complexity and data structure complexity.

Modularity

Modularity is a concept by which a piece of software is grouped into a number of distinct and logically cohesive subunits, presenting services to the outside world through a well-defined interface [6]. Table IV shows that a number of metrics have been used to characterize OSS evolution from modularity perspective. It is also obvious from Table IV that the metrics for modularity are used at different levels. For instance, Liu and Iyer [34], and Simmons et al [49] studied modularity at the class/file level that provides information regarding software functionality. However, Conley and Sproull [21] argue that studying modularity at that level does not capture interface information, i.e., whether classes or files communicate via interfaces, which are used to achieve component independence in modular software. Accordingly, they argue that the package at the module or component level is more appropriate for assessment of software modularity than using classes or files.

TABLE IV. MODULARITY METRICS

Study	Metrics
[21]	Total number of lines of code, number of concrete and abstract classes, afferent and efferent coupling
[24]	Dependencies between packages
[34]	Measured at class/file level
[41]	Correlation between functions added and functions modified
[49]	(Only measured at file level): number of classes, number of files for each release, directory structure and content

Excessive inter-module dependencies have long been recognized as an indicator of poor software design [37] and can diminish the ability to reason about software components in isolation. It becomes also difficult to assess and manage change impacts. Therefore, apart from studying the dependencies between packages [24], inter-module dependency can also be used for achieving modularity, and examining the following kinds of dependencies:

- Class reference: If class A refers to class B, e.g. as in an argument in a method, then A depends on B.
- Invokes: If a function in class A calls a function or a constructor of class B, then A depends on B.
- Inherits: If class A is a subclass of class B, then A depends on B.
- Data member reference: If a function in class A makes reference to a data member of class B, then A depends on B.

However, we did not find any paper that explicitly studies OSS evolution by using the inter-module dependency.

3.6 Examining OSS Evolution at Software Architecture Level

This category includes studies that focus on examining OSS evolution at software architecture level. According to [38], there is a lack of research that investigates the relation between software architecture and OSS, and discusses in details how software architecture is treated in OSS. Godfrey and Tu [23] came up with the similar observations from another perspective, i.e., planned evolution and preventive maintenance may suffer OSS development, which encourages active participation but not necessarily careful reflection and reorganization. The scarcity of studies on architectural level evolution of OSS confirms the above-mentioned observations.

Based on a case study, Nakagawa et al [38] found that software architecture is directly related to OSS quality, and that the knowledge and experience in architecture must be considered in OSS projects. This study also proposes architecture refactoring in order to repair architectures, aiming at improving mainly maintainability, functionality and usability of OSS. A similar approach is described in [53], which explains the process of forward and reverse architectural repair to avoid architectural drift.

There are not many measures proposed for the architectural level evolution. Some variants of the number of calls into and number of calls from a component are used in [12], which addresses the structural characteristics of OSS with respect to the organization of the software's constituent components. This study selects functions as the basic unit for analysis, and three attributes are considered as proxies of static architectural structure, i.e., fan-in, fan-out and instability.

4. Validity Issues

The following types of validity issues need to be considered when interpreting the results from this review.

Conclusion validity [55] refers to the statistically significant relationship between the treatment and the outcome. One possible threat to conclusion validity is bias in data extraction. This was addressed through defining a data extraction form to ensure consistent extraction of relevant data to answering the research questions. The findings and implications are based on the extracted data.

Internal validity [55] concerns the connection between the observed behavior and the proposed explanation for the behavior, i.e. it is about ensuring that the actual conclusions are true. It is a concern for causal or explanatory studies. One possible threat to internal validity is the selection bias. We addressed this threat during the selection step of the review, i.e. the studies included in this review were identified through a thorough selection process which comprises of multiple stages. In the first stage, the first two authors independently selected and reviewed relevant papers from the complete set of papers retrieved on basis of the search strings. Then the selected papers were aggregated. After first set of selected papers was selected, the third author performed random check to validate if it was the right selection of papers.

Construct validity [55] relates to the collected data and how well the data represent the investigated phenomenon, i.e. it is about ensuring that the construction of the study actually relates to the research problem and the chosen sources of information are relevant. The studies identified from the systematic review were accumulated from multiple literature databases covering relevant journals, proceedings and book chapters. One possible threat to construct validity is bias in the selection of publications. This is addressed through specifying a research protocol that defines the research questions and objectives of the study, inclusion and exclusion criteria, search strings that we intend to use, the search strategy and strategy for data extraction. The research protocol and the identified publications have been reviewed by several researchers to minimize the risk of exclusion of relevant studies. Besides, additional reference checking of the identified studies was conducted to guarantee a representative set of studies for the review.

5. Conclusions

This systematic review has identified 41 primary studies of open source software evolution. Based on the research topics of those studies, we have classified them into four main categories of themes: software trends and patterns, evolution process support, evolvability characteristics, and examining OSS at software architecture level. The first category is further refined into three sub-categories: software growth, software maintenance and evolution economics, and prediction of software evolution. A comprehensive overview of these categories, corresponding sub-categories and related studies is discussed. The main findings from this systematic review are:

- Regarding the category of software trends and patterns, most papers focus on using different metrics to analyze OSS evolution over time. Few papers have looked into the economic perspective, e.g., maintenance effort, and few papers utilize the historical evolution data for prediction of OSS evolution and development. In this category, researchers have used various metrics at varying levels of granularities, e.g., class level, file level, and module level to measure OSS evolution. However, this review has also shown that there are diverse interpretations of the same terms, e.g., module, lines of code, rate of growth. This may cause conflicting conclusions that may be drawn from OSS evolution patterns, especially if the studies attempt to make comparisons on the differentiating results though based on using different sets of metrics for measuring.
- Regarding the category of evolution process support, different aspects that appear to have impact on the OSS

evolution process are covered; these aspect include commenting practice, OSS evolution and maintenance evaluation model, structures and quality characteristics of resources such as repositories, mails, bug tracking systems, as well as tools that support data retrieval for evolution analysis.

- Regarding the category of evolvability characteristics, determinism, understandability, modularity and complexity are addressed in the included studies. However, there are more evolvability characteristics that are not covered such as changeability, extensibility, testability, and modifiability. This might also explain the findings in the analysis of OSS evolution trends category that focuses on the evolution history instead of predicting the OSS evolution, because when there is a lack of analysis on OSS evolvability characteristics, it also becomes harder to predict its evolution.
- Regarding the category of examining OSS evolution at software architecture level, we have found that although an increasing amount of attention is being paid to the architecture of software systems due to its recognized role in fulfilling the quality requirements of a system [20], only few papers address OSS evolution at architectural level. Software evolution can be examined at different levels such as architectural level, detailed design and source code level. We have noticed from the review that most papers address OSS evolution at source code level. However, software architectures are inevitably subject to evolution. They expose the dimensions along which a system is expected to evolve [22] and provide basis for software evolution [37]. Therefore, it is of major importance to put more focus on managing OSS evolution and assessing OSS evolvability at the software architecture level besides the code-level evolution.

ACKNOWLEDGMENT

We would like to acknowledge Klaas-Jan Stol -for sharing the repository of papers on OSS for the reported review. Hongyu Pei Breivold acknowledges the Swedish KK-foundation (KKS) through the SAVE-IT project. Aufeef Chauhan would like to acknowledge Prof. Ivica Crnkovic for his supervision. Aufeef also acknowledges the EURECA project (www.mrtc.mdh.se/eureca) funded by the Erasmus Mundus External Coorporation Window (EMECW) of the European Commission for providing resources to conduct a part of his study at Mälardalen University, Sweden.

References

[1] Al-Ajlan, A.: 'The Evolution of Open Source Software Using Eclipse Metrics', International Conference on New Trends in Information and Service Science, IEEE Computer Society, pp. 211-218, 2009.

[2] Ali, S., and Maqbool, O.: 'Monitoring software evolution using multiple types of changes', International Conference on Emerging Technologies, pp. 410-415, 2009.

[3] Arafat, O., and Riehle, D.: 'The commenting practice of open source', Conference on Object Oriented Programming Systems Languages and Applications, ACM, pp. 857-864, 2009.

[4] Asklund, U., and Bendix, L.: 'A study of configuration management in open source software projects', IEE Proceedings-Software, 149, (1), pp. 40-46, 2002.

[5] Bachmann, A., and Bernstein, A.: 'Software process data quality and characteristics: a historical view on open and closed source projects', Proceedings of the Joint International ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol) Workshops, ACM, pp. 119-128, 2009.

[6] Baldwin, C.Y., and Clark, K.B.: 'Design rules: The power of modularity', The MIT Press, 2000.

[7] Beecher, K., Capiluppi, A., and Boldyreff, C.: 'Identifying exogenous drivers and evolutionary stages in FLOSS projects', Journal of Systems and Software, 82, (5), pp. 739-750, 2009.

[8] Bennett, and Rajlich: 'Software maintenance and evolution: a roadmap'. Proceedings of the Conference on The Future of Software Engineering, Limerick, Ireland, 2000.

[9] Bennett, K.H., and Rajlich, V.T.: 'Software maintenance and evolution: a roadmap', The Future of Software Engineering, ACM New York, pp. 73-87, 2000.

[10] Bouktif, S., Antoniol, G., and Merlo, E.: 'A feedback based quality assessment to support open source software evolution: the grass case study', International Conference on Software Maintenance, pp. 155-165, 2006.

[11] Capiluppi, A.: 'Models for the evolution of OS projects', International Conference on Software Maintenance, pp. 65-74, 2003.

[12] Capiluppi, A., and Beecher, K.: 'Structural Complexity and Decay in FLOSS Systems: An Inter-Repository Study', 13th European Conference on Software Maintenance and Reengineering (CSMR), 2009.

[13] Capiluppi, A., Faria, A.E., and 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.

[14] Capiluppi, A., González-Barahona, J.M., Herraiz, I., and Robles, G.: 'Adapting the staged model for software evolution to free/libre/open source software', Ninth International Workshop on Principles of Software Evolution, ACM, 2007.

[15] Capiluppi, A., Morisio, M., and Lago, P.: 'Evolution of understandability in oss projects', Eighth Euromicro Working Conference on Software Maintenance and Reengineering, pp. 58-66, 2004.

[16] Capiluppi, A., Morisio, M., and Ramil, J.F.: 'Structural evolution of an open source system: a case study', Proceedings of the 12th IEEE International Workshop on Program Comprehension, 2004.
[17] Capiluppi, A., Ramil, J.F., and e Informatica, D.A.: 'Studying the evolution of open source systems at different levels of granularity: Two case studies', Proceedings of the 7th International Workshop on Principles of Software Evolution, pp. 113-118, 2004.
[18] Capra, E.: 'Mining open source web repositories to measure the cost of evolutionary reuse', International Conference on Digital Information Management, pp. 496-503, 2006.
[19] Capra, E., Francalanci, C., and Merlo, F.: 'The economics of open source software: an empirical analysis of maintenance costs', International Conference on Software Maintenance, pp. 395-404, 2007.
[20] Clements, P., Kazman, R., and Klein, M.: 'Evaluating Software Architectures: Methods and Case Studies', Addison-Wesley, 2002.
[21] Conley, C.A., and Sproull, L.: 'Easier Said than Done: An Empirical Investigation of Software Design and Quality in Open Source Software Development', 42nd Hawaii International Conference on System Sciences, 2009.
[22] Garlan, D.: 'Software architecture: a roadmap', in Editor (Ed.)^(Eds.): 'Book Software architecture: a roadmap', The Future of Software Engineering, ACM Press New York, pp. 91-101, 2000.
[23] Godfrey, M.W., and Tu, Q.: 'Evolution in open source software: A case study', International Conference on Software Maintenance, pp. 131-142, 2000.
[24] Gonzalez-Barahona, J.M., Robles, G., Michlmayr, M., Amor, J.J., and German, D.M.: 'Macro-level software evolution: a case study of a large software compilation', Empirical Software Engineering, 14, (3), pp. 262-285, 2009.
[25] Herraiz, I., Gonzalez-Barahona, J.M., and Robles, G.: 'Determinism and evolution', International Working Conference on Mining Software Repositories, ACM, pp. 1-10, 2008.
[26] Herraiz, I., Gonzalez-Barahona, J.M., Robles, G., and German, D.M.: 'On the prediction of the evolution of libre software projects', International Conference on Software Maintenance, pp. 405-414, 2007.
[27] Herraiz, I., Robles, G., Gonzalez-Barahona, J.M., Capiluppi, A., and Ramil, J.F.: 'Comparison between SLOCs and number of files as size metrics for software evolution analysis', Conference on Software Maintenance and Reengineering, 2006.
[28] Izurieta, C., and Bieman, J.: 'The evolution of freebsd and linux', Proceedings of the IEEE/ACM International Symposium on Empirical Software Engineering, 2006.
[29] Kitchenham, B., and Charters, S.: 'Guidelines for performing systematic literature reviews in software engineering', Evidence Based Software Engineering, 2007.
[30] Koch, S.: 'Evolution of open source software systems–a large-scale investigation', Journal of Software Maintenance and Evolution: Research and Practice, 2007.
[31] Koponen, T.: 'Evaluation Framework for Open Source Software Maintenance', International Conference on Software Engineering Advances, pp. 52-52, 2006.
[32] Lehman, M.M., Perry, D.E., and Ramil, J.F.: 'On evidence supporting the FEAST hypothesis and the laws ofsoftware evolution', the 5th International Symposium on Software Metrics, pp. 84-88, 1998.
[33] Lehman, M.M., Ramil, J.F., Wernick, P.D., Perry, D.E., and Turski, W.M.: 'Metrics and laws of software evolution-the nineties view', International Symposium on Software Metrics, 1997.
[34] Liu, X., and Iyer, B.: 'Design architecture, developer networks, and performance of Open Source Software projects', Approaches to Information System Development, 2007.
[35] Madhavji, N.H., Fernandez-Ramil, J., and Perry, D.: 'Software Evolution and Feedback: Theory and Practice', John Wiley & Sons, 2006.
[36] McCabe, T.J.: 'A complexity measure', IEEE Transactions on Software Engineering, pp. 308-320, 1976.
[37] Medvidovic, N., Taylor, R.N., and Rosenblum, D.S.: 'An Architecture-Based Approach to Software Evolution', International Workshop on the Principles of Software Evolution, 1998.
[38] Nakagawa, E.Y., de Sousa, E.P.M., de Brito Murata, K., de Faria Andery, G., and Morelli, L.B.: 'Software Architecture Relevance in Open Source Software Evolution: A Case Study', 32nd International Computer Software and Applications Conference, IEEE, pp. 1234-1239, 2008.
[39] Nehaniv, C.L., and Wernick, P.: 'Introduction to Software Evolvability 2007', 3rd International IEEE Workshop on Software Evolvability, 2007.
[40] Park, R.E.: 'Software Size Measurement: A Framework for Counting Source Statements. Software Engineering Institute', Carnegie Mellon University, 1992.
[41] Paulson, J.W., Succi, G., and Eberlein, A.: 'An empirical study of open-source and closed-source software products', IEEE Transactions on Software Engineering, 30, (4), pp. 246-256, 2004.
[42] Raja, U., Hale, D.P., and Hale, J.E.: 'Modeling software evolution defects: a time series approach', Journal of Software Maintenance and Evolution: Research and Practice, 21, (1), pp. 49-71, 2008.
[43] Robles, G., Amor, J.J., Gonzalez-Barahona, J.M., and Herraiz, I.: 'Evolution and growth in large libre software projects', 8th International Workshop on Principles of Software Evolution, pp. 165-174, 2005.
[44] Robles, G., González-Barahona, J.M., Izquierdo-Cortazar, D., and Herraiz, I.: 'Tools for the study of the usual data sources found in libre software projects',

International Journal of Open Source Software and Processes, 1, (1), pp. 24–45, 2009.

[45] Robles, G., Gonzalez-Barahona, J.M., and Merelo, J.J.: 'Beyond source code: The importance of other artifacts in software development (a case study)', The Journal of Systems & Software, 79, (9), pp. 1233-1248, 2006.

[46] Robles, G., Gonzalez-Barahona, J.M., Michlmayr, M., and Amor, J.J.: 'Mining large software compilations over time: another perspective of software evolution', International Workshop on Mining Software Repositories, ACM, 2006.

[47] Rowe, D., Leaney, J., and Lowe, D.: 'Defining systems evolvability-a taxonomy of change', International Conference and Workshop: Engineering of Computer-Based Systems, pp. 541-545, 1994.

[48] Schach, S.R., Jin, B., Wright, D.R., Heller, G.Z., and Offutt, A.J.: 'Maintainability of the Linux kernel', IEE Proceedings-Software, 149, (1), pp. 18-23, 2002.

[49] Simmons, M.M., Vercellone-Smith, P., and Laplante, P.A.: 'Understanding Open Source Software through Software Archaeology: The Case of Nethack', 30th Annual IEEE/NASA Software Engineering Workshop, pp. 47-58, 2006.

[50] Smith, N., Capiluppi, A., and Ramil, J.F.: 'A study of open source software evolution data using qualitative simulation', Software Process Improvement and Practice, 10, (3), pp. 287-300, 2005.

[51] Suh, S.D., and Neamtiu, I.: 'Studying Software Evolution for Taming Software Complexity', 21st Australian Software Engineering Conference, 2010.

[52] Thomas, L.G., Schach, S.R., Heller, G.Z., and Offutt, J.: 'Impact of release intervals on empirical research into software evolution, with application to the maintainability of Linux', Software, IET, 3, (1), pp. 58-66, 2009.

[53] Tran, J.B., Godfrey, M.W., Lee, E.H.S., and Holt, R.C.: 'Architectural repair of open source software', 8th International Workshop on Program Comprehension, pp. 48-59, 2000.

[54] Wang, Y., Guo, D., and Shi, H.: 'Measuring the evolution of open source software systems with their communities', ACM SIGSOFT Software Engineering Notes, 32, (6), pp. 7, 2007.

[55] Wohlin, C., Höst, M., Runeson, P., Ohlsson, M.C., Regnell, B., and Wesslén, A.: 'Experimentation in software engineering: an introduction', Kluwer Academic Pub, 2000.

[56] Xie, G., Chen, J., and Neamtiu, I.: 'Towards a better understanding of software evolution: An empirical study on open source software', International Conference on Software Maintenance, 2009.

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

[58] Yu, L., Ramaswamy, S., and Bush, J.: 'Symbiosis and Software Evolvability', IT Professional, 10, (4), pp. 56-62, 2008.