Open Source Software Development Process: A Systematic Review

Bianca M. Napoleão, Fabio Petrillo, Sylvain Hallé
Laboratoire d'informatique formelle - Université du Québec à Chicoutimi, Canada

***Abstract*—Open Source Software (OSS) has been recognized by the software development community as an effective way to deliver software. Unlike traditional software development, OSS development is driven by collaboration among developers spread geographically and motivated by common goals and interests. Besides this fact, it is recognized by the OSS community the need to understand OSS development process and its activities. Our goal is to investigate the state-of-art about OSS process through conducting a systematic literature review providing an overview of how the OSS community has been investigating OSS process over past years. We identified and summarized OSS process activities and their characteristics and translated them into an OSS macro process using BPMN notation. As a result, we systematically analyzed 33 studies presenting an overview of the OSS process research and a generalized OSS development macro process represented by BPMN notation with a detailed description of each OSS process activity and roles in OSS environment. We conclude that OSS process can be in practice further investigated by researchers. In addition, the presented OSS process can be used as a guide for OSS projects and be adapted according to each OSS project reality. It provides insights to managers and developers who want to improve their development process even in OSS and traditional environments. Finally, recommendations for OSS community regarding OSS process activities are provided.**

Keywords*—*open source, software development process, systematic literature review

I. Introduction

Nowadays, Open Source Software (OSS) represents an important force in the current scenario of software industrial production [43]. According to Balali *et al.* [4], there are many examples of projects that were developed and maintained through open source initiatives such as Linux, Open Office, and Mozilla Firefox. Moreover, companies that have a restrictive business model are publishing products and artifacts via OSS [31].

Due to the significant attention it has gained over the years, Open Source (OS) development has been widely recognized as an effective way to deliver software [23], [31]. The OS development principle is based on continuous interaction in a participatory user community motivated by common goals and interests; it introduces non-conventional ways to develop, test, and maintain software [12], [16].

Several studies present investigations on how the OSS process differs from traditional Software Engineering (SE) processes [14], [32], [38]. As stated by Scacchi *et al.* [38], OSS projects have complex development processes executed by weakly coordinated contributors and developers spread geographically. Also, as stated by Fuggetta [15], it is difficult to detect aspects of OSS development compared to other traditional software development practices. As a matter of fact, OSS projects have received criticism for not making their development process clear. Therefore, discovering and mapping OSS process and its activities is a recognized need by the OSS community [17], [38]. As stated by Hashmi *et al.*, sharing clear and open description of OSS processes with the possibility of reuse and adaptation is certainly of extensive interest to the OSS practitioners. One obstacle to developers starting to collaborate in an OSS is precisely the lack of a formal description of the different activities and phases performed in an OSS project [42].

In this study, we investigate the current state-of-art regarding OSS process by conducting a Systematic Literature Review (SLR), which provides an overview of how the OSS community has been investigating the OSS process over past years. We identify and summarize OSS process activities and their characteristics, as well as translate the OSS process into a macro process through the BPMN notation due to this notation is commonly adopted to modeling software process that considers process behavioral and structural proprieties [5], [11].

Our results provide a valuable overview of the OSS development process and show that this process can be further investigated by researchers. In addition, the presented OSS macro process can be used as a guide for OSS projects and also be adapted according to the OSS project reality. The mapped process and the description of its activities provide insights to managers and developers who want to improve their development process in OSS and even traditional environments. Finally, we present recommendations for the OSS community regarding OSS process activities.

The remainder of this study is organized as follows. Section II describes the design of the study. Section III presents our results answering the proposed research questions. Section IV discusses our findings. Recommendations regarding OSS development process are provided in Section V. Section VI presents threats to validity to this study. Finally, Section VII concludes our study including directions for future research.

II. Study Design

This section covers the SLR method performed to understand the main contributions of the state-of-art regarding the OSS development process. An SLR is a means of identifying, evaluating, and summarizing available research related to a particular research topic [22]. In this context, an SLR was appropriate to summarize existing evidence regarding OSS development process. We did not opt to perform an empirical

study directly because the main idea of this study is to summarize what was already investigated in the literature to provide a basis for future empirical investigation.

Two other studies investigated the state-of-art on OSS process. Acuña *et al.* [3] performed a systematic mapping identifying what activities OSS process models contain and grouped them according to its focuses (concept exploration, software requirements, design, maintenance and evaluation). They conclude that the use of primary studies is the start point to analyze and propose an open source process model for the OSS community. In this study, we use primary studies evidence to summarize and propose an OSS macro process. Crowston *et al.* [6] presented a quantitative summary of articles categorized into issues pertaining to inputs, processes (software development and social processes), emergent states and outputs. However, the study does not investigate OSS development process activities and their characteristics in detail or even mapped them in a unified process, as we present.

To conduct this study, we followed the guidelines for performing SLR developed by Kitchenham *et al.* [22]. The SLR steps were conducted by three researchers: one research worked in the execution of the study design and through weekly meetings, all researchers reviewed the work's progress and discussed any disagreements about the decisions during the SLR conduction.

A. Research Questions

Our goal is to understand how the OSS community has been investigating the OSS process over past years, identifying and summarizing OSS process activities and their characteristics, as well as translating the OSS process into a macro process using BPMN notation. We translated our research goal into two Research Questions (RQs).

RQ1: How has the OSS community been investigating the OSS process over the past years?

This question aims to analyze the intensity of scientific interest on OSS process by understanding how the OSS community has been studying and exploring the OSS process. A categorized map from previous publications is provided.

RQ2: What activities do OSS processes contain? What are their main characteristics?

This question intends to identify OSS process activities and summarize their main characteristics to understand the processes behind the OSS projects. An OSS macro process is presented and explained in detail.

The adopted search strategy includes an automated search and a snowballing search [45]. We performed the automated search in order to detect relevant primary studies related to OSS development process. To perform the automated search, we developed a search query and we ran a search pilot test as recommended by Kitchenham *et al.* [22]. The study control group, used to calibrate our search string, was based on included studies from Acuña *et al.* [3] secondary study since this study aimed to investigate the OSS development process as well. Our search query is presented following.

```
(("open source" OR "free source") AND
("development process" OR "software process"))
```

We chose to run our search query on the most renowned Software Engineering (SE) digital libraries [22]: *IEEE Xplore, ACM Digital Library, Scopus* and *Web of Science*. *Scopus* and *Web of Science* were chosen because they index studies of several international publishers, including *Springer*, *Wiley-Blackwell*, *Elsevier*, *IEEE Xplore* and *ACM Digital Library*; although not necessarily the most recent conference proceedings. Therefore, we opted for searching at *IEEE Xplore* and *ACM Digital Library* individually because they are considered the two-key publisher-specific resources which together covers the most important SE and computer science journals and conferences [22]. We executed the search query in three metadata fields: title, abstract and keywords. The search query was also adapted to meet specific search criteria (e.g. syntax) of each digital library. Details about the automated search execution are available at: *https://bit.ly/2ZaOFCB*.

The selection criteria are organized into two Inclusion Criteria (IC) and four Exclusion Criteria (EC):

- **IC1:** The study must address OSS development process or models in open source projects;
- **IC2:** The study must be a peer-reviewed study.
- **EC1:** The study is just published as an abstract;
- **EC2:** The study is not written in English;
- **EC3:** The study is an older version of another study already considered;
- **EC4:** The study addresses process information considering hybrid projects instead of only OSS projects.

As illustrated in Figure 1, a total of 1721 items were returned from the automated search execution. Then, we removed all duplicated studies and conference announcements, totaling 1229 studies. Next, we read the papers' title, abstract and keywords and applied the selection criteria (IC and EC) on these fields which reduced our number to 61 candidate studies. Finally, the selection criteria were applied considering the reading of each study's full text, resulting in a set of 24 included studies from this stage. This step was performed by one author, and revised by a second author.

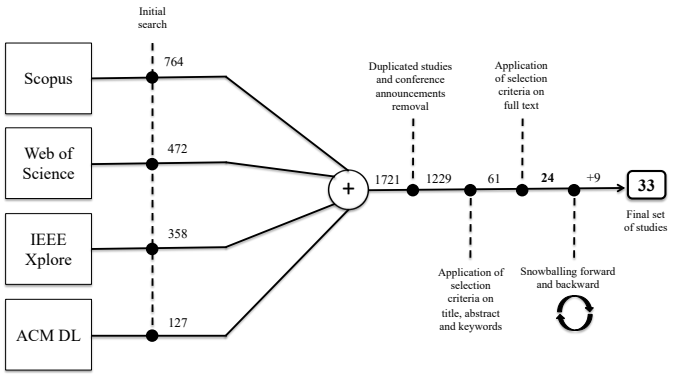

Figure 1: Search strategy process

The results from the automated search showed that the number of primary studies investigating the OSS process has decreased over the last years. For this reason, we decided to

perform forward and backward snowballing to avoid missing relevant studies. We followed the guidelines proposed by Wohlin [45] to perform the snowballing technique.

We used the 24 relevant included studies from our automated search process as our seed set (starting set) for performing three iterations in the snowballing technique. We performed the forward and backward snowballing techniques. Forward snowballing is an approach that considers the analyzed studies' citations, while backward snowballing considers each study's reference list aiming to find other relevant studies [45]. The studies' citations were extracted with support of search engines, such as *Google Scholar*, the *ACM Digital Library* and *IEEE Xplore*. In each snowballing iteration, we applied IC and EC criteria first on title, abstract and keywords, and next on full text. We performed three snowballing iterations, stopping its execution because at the last iteration, no more relevant study was detected. One author performed the snowballing technique, and when doubts about the inclusion or exclusion of a paper were raised, it was discussed with all authors in the weekly meetings. As a result from both snowballing techniques, we added nine more relevant studies in our final set of included studies, totaling 33 studies. Figure 1 illustrates these findings.

B. Data Extraction and Analysis

In order to extract all relevant data to answer our RQs, we created a data extraction form based on our RQ goals. In other words, the fields of the data extraction form considered the data needed to answer our RQs, for instance, OSS process activities described in the paper, description of each activity, roles, etc. The data were extracted by one researcher and reviewed by an experienced researcher who already worked with several OSS projects. As mentioned before, the findings were discussed and reviewed by all authors every week. For example, to answer RQ1 and define the main topic addressed by each paper, the researchers read the papers and in the meetings though consensus the papers' categories were defined.

The data synthesis was performed through a combination of content analysis from the extracted data categorizing the findings into broad thematic categories as well as a narrative synthesis. The extracted information was interpreted and analyzed considering the authors' knowledge and experience in OSS projects. It is worth mentioning that the OSS macro process presented in this paper is a summary of synthesized evidence from primary studies.

III. Results

This section presents our RQs answers based on the results from our included studies data extraction and synthesis.

RQ1: How has the OSS community been investigating the OSS process over the past years?

To analyze the intensity of scientific interest on OSS process, we analyzed the distribution of publications over past years addressed by our included studies. Figure 2 illustrates our results.

As shown in Figure 2, the highest number of publications regarding the OSS process was published in the years 2004 (three studies), 2005 (five studies) and 2006 (four studies)

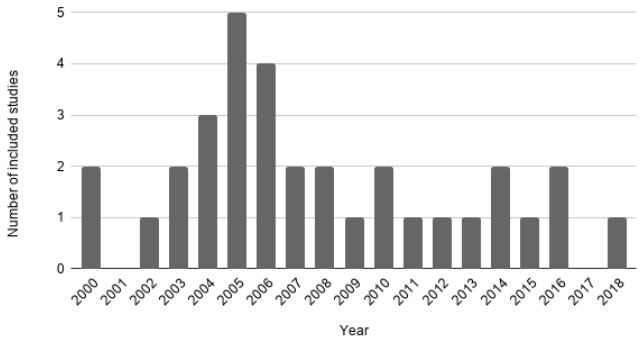

Figure 2: Distribution of publications (included studies) by year

representing 36.36% (12 studies) from our set of included studies. There were no included studies published in 2001 and 2017. Also in Figure 2, it is possible to observe that the interest in OSS process or specific activities about OSS process is still being investigated over the years. For example, continuous integration is a process activity that has been recently adopted in OSS projects [33].

Through automatic search, the two most recent included studies found were from 2014 and 2016. As mentioned before, we perform a snowballing forward search in order to detect other relevant studies (mainly more recent studies) addressing the OSS process.

We categorized the included studies according to their main addressed topic and their publication type (journal, conference, workshop or symposium). The main topic categorization emerged from the full text reading and discussions among the authors. Table I presents these findings sorted by main topic.

Main topic	Publication type	Included Studies
General OSS process	Journal	[38] [41] [24] [15] [37] [13] [36] [9] [28]
General OSS process	Conference	[27] [42] [32] [8] [26] [40] [14] [12] [39] [46] [20] [7] [23] [25] [29]
General OSS process	Workshop	[19] [17]
Quality assurance	Journal	[44]
Quality assurance	Symposium	[21]
Bug	Symposium	[35]
Bug	Workshop	[18]
Testing	Conference	[1] [2]
Continuous integration	Conference	[33]

Table I: List of included studies sorted by main topic

As Table I shows, the most significant part of the included studies were published in conferences (17 studies – 51.51%) followed by journals (10 studies – 30.30%), workshops (3 studies – 9.09%) and symposiums (2 studies – 6.06%). Publications from conferences and journals include some of the most renowned SE venues such as the *International Conference on Software Engineering*, *Journal of Information and Software*

Technology, *Journal of Systems Software* and *Transactions on Software Engineering*.

The majority of the included studies address the OSS process generally. They present information and characteristics regarding the whole OSS process activities and roles. On the other hand, there are studies that address specific activities (sub-processes) or practices in OSS environment, namely testing, quality assurance, bugs and continuous integration (see Table I).

RQ2: What activities do OSS processes contain? What are their main characteristics?

OSS development is characterized by collaborative and voluntary software development carried out by several developers distributed geographically around the world [15], [25], [37]. The distributed aspect in OSS development allows OSS developers to work from the most different locations and even hardly ever or never meeting one each other face-to-face; ordinarily they coordinate and exchange experiences and project information only online (for instance, via emails and bulletin boards) [27]. As stated by Mockus *et al.* [27], [28], the main OSS process characteristics are: (i) vast number of volunteer developers involved in the project; (ii) openness for the developer to choose the work they want to do (tasks are not assigned); (iii) the system under development has no explicit design, the system is built collaboratively; and (iv) there is no project plan, calendar or even an established delivery list.

To understand the OSS development process phenomenon, firstly we identify and summarize the activities contained in each included study into a macro process using the BPMN notation [30] as well as providing a detailed explanation of each activity contained in this OSS development macro process. The BPMN macro process emerged from the analysis of the extracted data. The process was built interactively where during the weekly meetings the authors revised and discussed about the findings. Secondly, we analyze what are the roles played by developers in OSS projects. These results are presented in Section III-1, III-2 and III-3, respectively.

1) OSS Process Activities: We focused on identifying what are the process activities performed by the OSS developers and consolidating evidence into a macro process. Figure 3 presents the OSS macro process represented in BPMN notation. BPMN is a standard business process modeling notation that can be applied to modeling software processes [5]. According to Dumas and Pfahl [11], it offers a valuable spectrum of constructs that can be used to capture software development activities considering process behavioral and structural proprieties in closed and OS environments.

As shown in Figure 3, the OSS process starts with a requirements elicitation step where a developer can report or work on new features or fix defects in an OSS project. These requirements can be even reported or consulted in several sources: developers' mailing lists, project webpage, wiki roadmap or newsgroups and bug reporting or tracking systems. Looking into these sources, a developer discovers that a problem exists or validates if a problem that he/she is facing was reported or not by others. In the sequence, the developer selects a problem to work on it or assigns it as a suggestion to another developer that could contribute to the problem. The following step is to identify a solution to this problem and consequently develop a code solution and commit this code to the project repository. After committing the code, several tests can be performed by the project community. If a test fails, the results should be posted in a mailing list, bug reporting or tracking system (this depending on the bug or issues source used in the development project). On the other hand, if all tests pass, a code review can be performed in order to apply code best practices, improve the code's quality including aspects such as performance, security or consistency.

Some OSS projects adopted a continuous integration practice. Even if this practice is not applied, the next step can be to freeze the project and perform tests on a frozen version or perform a project pre-release to get users' feedback. A pre-release is not a mandatory practice, but it is commonly adopted by several OSS projects. On the one hand, if major issues are found they should be reported. On the other hand, if the project is ready to release, the next step is to write user instructions (if needed) and finally perform a final release.

2) OSS process activities characteristics: In the following, we explain in detail each OSS process activity and its characteristics, as illustrated in Figure 3.

Requirements elicitation – Unlike traditional software processes, requirements elicitation in OSS projects is based on the knowledge and professional experience of core developers [29], [39]. According to Dillon *et al.* [8], in OSS development there is no formal requirement process. However, requirements in OSS projects are done basically through two main tasks: (i) development of new features and (ii) fixing defects [9], [19], [35], [46]. Requirements in OSS projects emerge from the communication and discussion among OSS community members [25], [29]. New features are normally defined by core developers who control the project architecture and direction of development; for example, they look into the project wiki and decide on how and when to start a feature development [23]. Nonetheless, in some cases, individuals committers can also add a new feature or create a team to work on a large project specifically [9]. It can be said that OSS requirements are community-built requirements [8].

The main requirement inputs and sources in OSS are developers mail lists [9], [13], [17], [19], [26]–[28], [36], [37], [44], [46] where developers report issues, give suggestions and discuss bugs [29]; project webpage and/or wiki page and/or roadmap list and/or newsgroup [7], [9], [17], [19], [23], [27], [36], [37], [39], [46] where project information with focus on end users is made available; bug report databases (e.g. BUGDB and GNATS) [8], [9], [17], [19], [21], [27], [28], [36], [37], [42], [44] and bug tracking systems (e.g. Bugzilla, RedMine and Trac) [17]–[21], [26], [28], [35], [37], [39], [44] where defects are reported, tracked and discussed.

Select or assign a problem to work on it – OSS developers usually self-select projects and tasks (e.g. bug or issues) that they will work on [12], [38]. Besides, OSS developers are also end users, so they tend to work on issues reported by themselves or features that they developed. In other words, they tend to work on code areas where they are most familiar [27], [28]. In some projects, there is "code ownership", which means part of the code is created or maintained consistently by a specific(s)

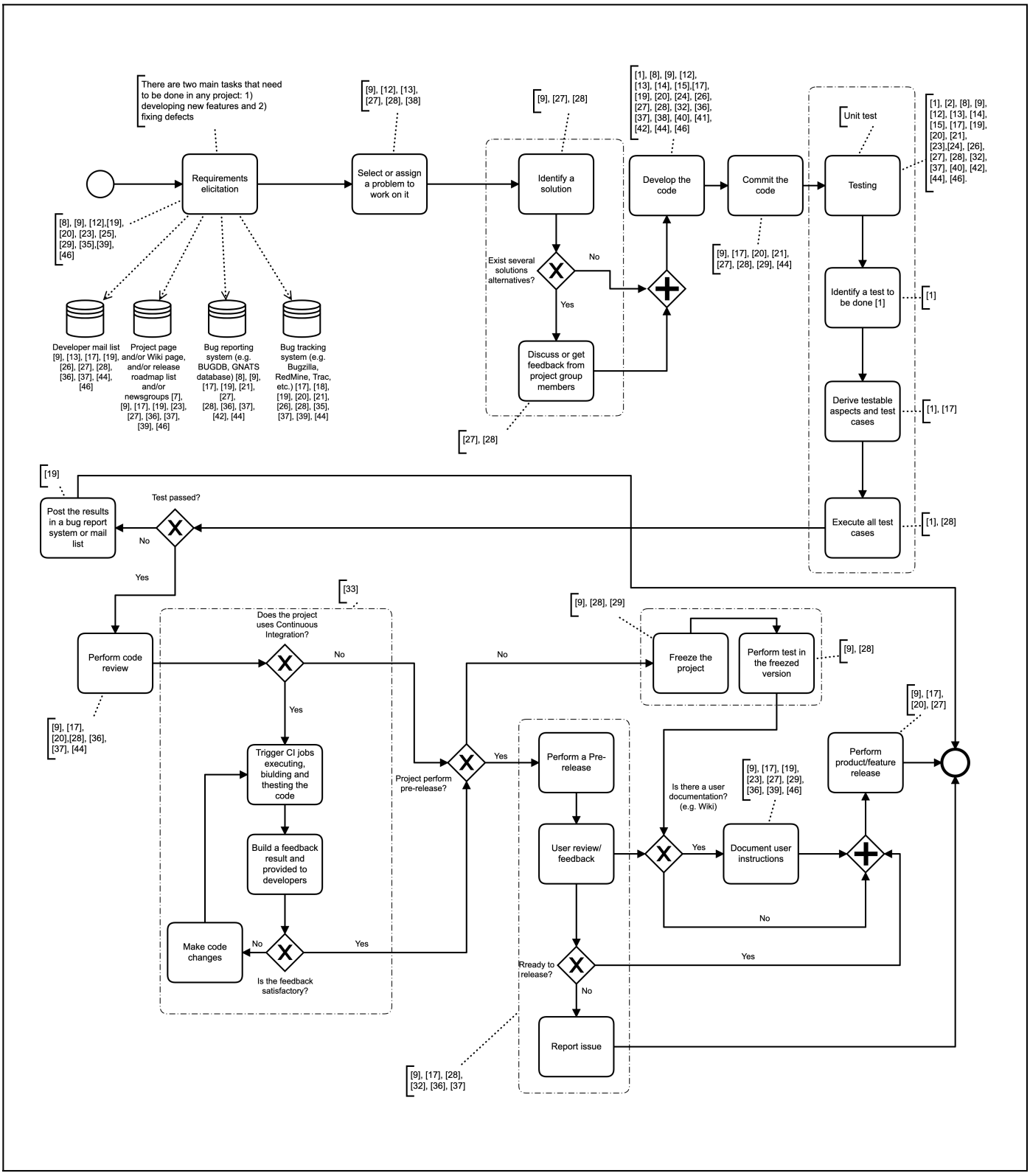

Figure 3: OSS macro process shown in the BPMN notation

developer(s). This fact does not prevent another developer from contributing to this part of the project, but there are some respect in the community for these "senior" contributors with experience in this project area [27]. However, in other OSS projects, there is no "code ownership". The developers are just required to ask for code review from developers who maintained this part of the code before committing it [9].

In OSS development there is a "skill meritocracy": the core developers play a leadership role where the most skillful developers respond to them. Consequently, other developers who contribute to the project respect their decisions about the project direction and activities lead the project more cohesive centralized [12], [29]. Moreover, less skillful developers do not have the possibility to change the code base; they can develop or test the solution and send it to the corresponding assigned committer [9], [29]. Notwithstanding, in the majority of the cases the developers select and assign their own tasks. Hence, on the FreeBSD project, a developer (committer) can assign a pull request to another developer who should be able to solve the problem [9]. Regarding "OSS 2.0", a movement to leverage the OSS community for business, paid developers are assigned to work in OSS projects in order to find a solution to project's open problems [13].

Identify a solution – Once a developer is assigned to work on a problem, the next step is to try to identify a solution for it. The challenge most faced at this stage is to decide among the several solutions which one is better. In some cases, when a user reported a bug, for instance, he/she proposed a solution together [9]; other developers could also have suggested alternative solutions [27], [28]. When many solutions are available, the developer forwards the solutions to the community (e.g. mail list or Bugzilla), aiming to get feedback from the other project group members before deciding to develop a solution [27], [28].

Develop the code – This process activity consists basically in writing code for a solution [13] which is an essential activity in any software development process. Therefore, almost all included papers mention this activity [1], [8], [9], [12]–[15], [17], [19], [20], [24], [26]–[28], [32], [37], [38], [40]–[42], [44], [46]. According to Mockus *et al.* [27], driven by passion, OSS developers write code with more care and creativity. OSS projects have coding guidelines consisting in coding best practices to assist the coding activity [36], [42].

Commit the code – Commit the code consists of adding source code changes to the project repository in order to make it available to the community [9], [20], [27]. The authority to commit code is different in each project. As stated by Hashmi *et al.* [17], the big part of OSS projects allow core developers (most experienced developers) to commit (more than 90%); over 60% of the projects allow commits from developers who frequently contribute in the project and just over 25% of the projects allow passive developers to commit [29]. In the FreeBSD project [9], a committer is a developer who has the authority to commit changes in the project's CVS repository. A developer can assume the role of committer after actively contributing to the project for the last 18 months, or if the core developer team gives him/her this privilege; however this developer will have a mentor to supervise his/her work until he/she has gained experience and becomes reliable [9]. It is important to highlight that the committer is not necessarily the code author [29]: a committer could be just the developer responsible for committing changes officially in the code repository [17], [21], [28], [44].

Another observed OSS strong process characteristic is peer review activities. Peer review is the evaluation of a product work by other [44]. There are different peer review activities such as testing, code review, parallel debugging among others. These activities have been performed distributed and asynchronously by millions of developers in several OSS projects [17], [42]. As reported by Wang *et al.* [44], peer review is a huge advantage from community involvement in OSS projects because they have "many eyeballs" (Eric Raymond's "Linus Law" [34]) looking for problems, resulting in bugs found and fixed rapidly. Next, we address the two identified peer review activities: testing and code review.

Testing – Testing is an essential activity in any software development process. OSS community has the advantage of great user involvement and a structured approach to handle bugs [17]. These points collaborate with test activities leading to produce a high-quality software product [2]. The huge majority of the included papers mentioned software testing activity [1], [2], [8], [9], [12]–[15], [17], [19]–[21], [23], [24], [26]–[28], [32], [37], [40], [42], [44], [46]. Unit test is the most common test approach in OSS development [17] followed by pre-release testing [9], [17], [28]. Regarding multiple test techniques, a survey conducted by Hashmi *et al.* [17], showed that the majority of the respondents mentioned the adoption of functional testing including some form of system testing, regression test, integration test or acceptance testing. On the other hand, Senyard *et al.* [42] affirm that the regression test is not performed in OSS projects or is not mandatory [17], [28].

The study performed by Abdou *et al.* [1] outlined four OSS testing activities. In the following, they are described in their order of execution: (i) Identify a test to be done – There is no prioritization of the features that need to be tested, the features to be tested are selected independently based on developers' interests; (ii) Derive testable aspects and test cases – The testable aspects are simply based on specific attributes of interest from developers who decide to test the feature. The test cases are derived by determining pre-conditions, selecting input values and defining acceptance criteria. Moreover, approximately half of the projects analyzed by Hashmi *et al.* [17] use a type of documented test cases; (iii) Execute all test cases – This activity is performed through the test execution in a local source code copy [28]. Considering that it is a manual activity, it depends on the human factors (developer expertise and judgment) [28]; (iv) Review or vote until accepting the test – After executing the test, the developer commits the test results or posts the results on a bug report system (e.g. Bugzilla) or mailing lists.

The second more acknowledged form of test in an OSS environment is a pre-release test (also known as system test). It consists of introducing a release candidate to the OSS project community and let them test and make fixes until a core developer team decides that the system is ready for a final release [9]. It is important to mention that OSS users act as beta testers in OSS projects [32].

Code review – Code review is one of the most important peer review activities in the OSS development process. Its most important element is code inspection by other developers [17]. The code review practice is mentioned in the studies [9], [17], [20], [28], [36], [37], [44]. According to Hashmi *et al.* [17], code review is performed asynchronously and distributively and it is done before and after a source code is to be committed in the project repository. However, in [9] the developer is required to ask for a code review before committing in a code portion that is maintained by other committers.

As reported by Wang *et al.* [44], the code review activity can be divided in there steps: (i) pre-review, which is performed by developer's code author and a committer; (ii) review, which is performed by a developer acting as a code reviewer, verifier and approver; and (iii) post-review, performed by a submitter. Not just core developers perform code review, and other developers can assume this role. However, the post-review step is usually performed by core developers since they have the power to perform project releases.

Continuous Integration (CI) – Continuous integration was addressed by a single included study. This activity is a process that integrates code changes automatically in a shared repository responsible for compiling, building and executing test cases every time a code change is submitted [33]. In this study, the authors conclude that continuous integration is beneficial for OSS since after its adoption, the collaboration in larger OSS projects as well as the bug and issues resolution increased. A typical continuous integration process has the following steps: (i) A developer commits his/her code changes in a repository maintained by a version control system (e.g. Github); (ii) These commits trigger CI jobs on a CI tool which executes, builds and tests the code; (iii) this step results into a build feedback that is provided to developers (via e-mail and/or cellphone alerts). Based on this feedback, developers can make code changes if needed and then repeat the process until no more changes are necessary and then moving forward to the next OSS process activities.

Perform a partial release or pre-release – Some OSS projects perform a pre-release before the final release in order to make sure that the project is ready to be released. Pre-releases are usually expressed as tentative alpha, beta, candidate or stable releases [36], [37]. The main objective of a pre-release is to get user feedback [17], [28]. Potdar *et al.* [32] states that users are the best beta testers. When an issue is detected, it is reported to the project community and a new development cycle is started. A pre-release is tested and fixed until the core development team decides that the system is ready for a final release [9].

Document user instructions – In some OSS projects is common to have a place to present project's user documentation; for example, project webpage, wiki, release roadmap lists and newsgroups [9], [17], [19], [23], [27], [29], [36], [37], [39], [46]. Before a final release is made available, these pages must be updated, reporting the project changes and feature modifications [23].

Freeze the project – In order to release a real stable project version, the project is frozen [9], [28], [29] which means that only serious bug fixes [28], [29] and security repairs are allowed in this period [9]. In other words, just a bit of polishing is allowed [28]. During the freeze period, widespread testing is performed until the release gets ready [9], [28]. The freeze period is typically just a few days (e.g. 15 days) [9], [28].

Perform product/feature release – Release management is a crucial part in OSS projects [17]. This activity is extremely important since it is the moment to make developers collaboration results available to the OSS community. All releases are performed in a coordinated way. OSS projects have a central person or a team that controlled the official project releases [17], [20], [27]. When it is close to a release point, a member of the core development team (most experienced project developers) assumes the role of release manager [9], [17], [27]. The survey conducted by Hashmi *et al.* [17] showed that more than half of OSS project leaders have full release authority and under 20% give to the core development team the authority to release. On the other hand, it is clear that the consensus of the core development team is the basis for a release.

Regarding the release time-frame, the results presented in [17] study demonstrated that one-third of the OSS projects considered in their study release every six months and approximately 10% every quarter. However, the study concludes that the release frequency is ultimately up to core developers decision.

After a code release, only small code updates addressing specific problems in the last release are allowed (hotfixes) [29].

3) OSS process roles: In summary, the roles in OSS development can be divided into two major group visions: users and developers [44]. However, it is known that OSS developers are users too [12], [37]. The onion model and the layered community structure are the best-known examples of role organization in OSS projects. They are similar and present the two major group visions: (i) developers – core developers and others developers who contribute actively in the project (such as active developers and co-developers) or occasionally (such as peripheral developers); (ii) users – active users who report bugs and passive ones who only use the software [44]. In this research, we identified and categorized the roles mentioned in our set of included studies. The roles identified are:

Core developer – Core developers are the most experienced developers responsible for controlling and managing an OSS product [21]. In other words, they control and guide the project development direction, the project architecture and perform release management [1], [9], [12], [27], [38]. For example, when a project release needs to be done, usually one of the most experienced core developers volunteers himself/herself to perform the code release [9], [17], [27]. In the majority of OSS projects, core developers are the project founders [12]. According to the repository mining analysis performed by Scotto *et al.* [41], core developers represent 20% of the total developers and they develop approximately 80% of the project's code [1]. a project release needs to be done,

Contributor or active developer – A contributor or an active developer is responsible for submitting patches [1], [37], [44] and in some cases, review or revise code developed by other developers [44]. They have a direct impact on the project software development since their contributions to the code

base are essential to the project evolution [21]. More than half of the projects surveyed by Hashmi *et al.* [17] showed that active developers have the ability to commit code. The core development team can give the privilege to commit to an active developer [10]. However, their submissions are reviewed or checked by a core developer [9], [10].

Active users – Active users are OSS practitioners who test, report bugs and use cases to the project, but they do not write code [1], [9], [44].

Passive users – A passive user only uses the OSS software [44] and/or follows the project to stay informed about its development [1]. As mentioned by Hashmi *et al.* [17] and detected during the conduction of this study, not all types of roles presented in this study existing in all OSS projects also their names may vary. It is important to mention that the role of a volunteer in an OSS project may change over time [12]. For example, a very participative committed contributor can become a core developer.

The huge majority of developers in an OSS environment are volunteers. However, in some cases paid developers are assigned to work on OSS projects when a company is promoting or participating in an OSS project [13], [41].

IV. Discussion

In the last years, OSS processes have not been generally investigated. However, specific OSS activities or practices have been explored by researchers [1], [21], [33], [35]. This fact shows that OSS researchers are more interested in investigating OSS activities individually to understand these activities and possibly apply them in other OSS projects or even in SE traditional projects, if possible; [6] and to improve these activities in OSS development environment [17]. It is recognized by the OSS community the need of understanding the OSS process [17], [38]. The OSS processes followed by OSS projects were not mapped or even well documented. The existing documentation in OSS can be divided in two categories: (i) user documentation that are user instructions documented in wikis, project webpages, newsgroups and release roadmap lists; and (ii) developer documentation that are messages from mailing lists, bugs reports, code guidelines, etc. [17], [26].

As stated by Lavazza *et al.* [25], OSS is characterized by an unstructured working environment. However, our study shows OSS projects have an organizational sense. Most of the OSS projects started with a “seed" code [12], [42]. In other words, the majority of OSS projects started with the characteristic of a cathedral development style [42]. In contrary to what is stated in the "Bazaar" model [34] which is an OSS model connoted as chaotic but effective, large OSS projects have a systematic organization sense and care about process. For example, before the development of Apache Server project started, the main developers tried to solve process issues first. They were aware that a unique development process to make decisions was needed to organize work performed by developers spread around the world without any organizational tie [27]. Besides, modification in OSS projects is not performed in an ordered way; developers focus on a modification per time (they do not mix different kinds of modifications). It is worth highlighting that successful OSS projects have governance rules that developers must follow. For example: restricted access to the source code repository; permission to commit to the project's shared repository for release and redistribution (they need to send modifications to core developers that will evaluate and decide if they accept or not the contributions) [37], [41]; and guidelines and code style standards [42].

Regarding testing process activities, automated testing is just addressed by one included study [26] that concludes that OSS projects spend little time effort on implementing automated tests. The authors attribute the reason for it to the OSS development nature, which consists of a community that frequently provides bug reports. However, this study was published almost 15 years ago, so other investigation criteria can be applied to study automated tests specifically.

As mentioned in Section III, roles in OSS process are guided by meritocracy. The organization's roles are crucial to organize and support project contribution management, for example, core developers act as quality assurance members in order to maintain project architecture as well as product owners guiding the product development.

Several studies addressed by our set of included studies report process information about extremely successful OSS projects such as Apache Server [1], [20], [27], [28], [44], FreeBSD [9], Mozilla Firefox [1], [20], [28], [44], Moodle [23], ILIAS [23] and NetBeans IDE [19], [20], [39], [44]. Moreover, we systematically analyzed the state-of-art of existing research on OSS process. Based on that, we conclude the macro process and process activities characteristics described in this work generalized OSS process. Nevertheless, it is important to make clear that every OSS project has its process particularities. For example, each OSS project team adapt its process activities (including execution order) and roles according to external factors (size of the community, organization funding the development efforts, etc.) which are more environmental than structural.

Contrarily, a large number of OSS projects are stagnant [37]. This main failure reason is a lack of interest from the OSS community driven by low project usefulness and attractiveness [12], [37]. As reported by Ezeala *et al.* [12], detailed documentation is fundamental to attract contributors to collaborate in OSS projects. The absence of formal project activities and different steps performed during the project execution is a barrier to incentive collaboration in OSS process [42]. The process map and its activities characteristics description presented in our study are based on primary academic sources. However, it can be used as basis for the conduction of empirical studies to investigate OSS process in a more practical view. We believe that our results can contribute to improve the understanding of OSS process activities and consequently help to mitigate OSS project failure.

V. Recommendations

This study provides a generalized OSS process based on literature evidence. The synthesis from the included primary sources leads us to provide some recommendations to the OSS community. They are described following.

Our results suggest that there is a process with activities and roles well defined behind OSS project success. Also, we conjecture that an organized process facilitates project development and improves collaboration and engagement. Therefore, we recommend the OSS community clearly document and keep up-to-date their process on their documentation page or wiki to clarify for everyone their project process activities and roles. Diagrams such as BPMN are an interesting form of presenting these process activities since they are easy to read and understand.

Another core point in the OSS environment is peer review activities. Testing and code review are activities that need to be systematically integrated into an OSS process. Continuous integration is an activity with the potential to facilitate not just the integration of the code changes from multiple contributors but improve peer review practices. For example, CI best practices like Test-Driven Development (TDD) (an activity that needs to be further explored in the OSS context) and pull requests with code reviews can increase the code assertion, reduce technical debt and consequently increase delivery quality. In this context, we strongly advise considering CI practices in OSS projects.

Last but not least, project communication channels such as mails lists, bug tracking systems, web pages, wikis, etc. are the primary sources of OSS requirements. Best practices for their use can facilitate problem-solving and requirements elicitation.

VI. Threats to Validity

We report on the main threats to validity of our study as well as the adopted mitigation strategies.

Construct validity – This threat is related to the construction and execution of the SLR study protocol. In order to mitigate this limitation, our SLR protocol followed the guidelines of [22] and [45]. We rigorously selected relevant studies following defined selection criteria. Also, we performed a search pilot test as recommended by [22] and adopted a study control group to calibrate our search string. During the search strategy execution and data extraction analysis, we performed weekly meetings among all authors whose work progress was reviewed and any disagreement on results was discussed.

External validity – The most potent external threat to the validity of our study is that our set of included studies is not large enough to represent the state-of-art of the OSS process. In order to mitigate this threat, our search strategy included an automated search on the most renowned SE digital libraries complemented with snowballing technique. Besides, our selection criteria considered the inclusion of the only peer-reviewed studies and focused exclusively on publications that addressed the OSS process in some way discarding hybrid development process data. We are aware that some specific OSS process evidence could be missed by applying our selection criteria. Thus, we focused only on papers that address the OSS process or models explicitly. As the main goal of this study is to analyze the OSS process accurately, summarize results from explicit evidence is crucial to avoid bias in our results.

Conclusion validity – This threat addresses the conclusions made for building of the presented OSS macro process. The OSS macro process was built considering extracted evidence from the included primary studies, and the diagram does not reflect all details described in the text. Thus, the diagram does not represent a hard process, but the most representative activities that we found in the studies. For instance, our model has a step after committing, but testing is done at different moments that it is hard to represent in a single process diagram clearly. To mitigate possible imprecision, we performed weekly meetings to discuss among the authors the findings extracted from the 33 included papers. During these meetings, we reviewed the OSS process and made improvements on it continuously. A final review meeting was performed in order to produce and review the final version of the OSS macro process.

VII. Conclusions

In this study, we presented an overview of how the OSS community has been investigating the OSS process over the past few years and proposed a macro OSS development process with a detailed description of its process activities characteristics through a narrative description. Our study extracted and systematically analyzed OSS studies and translated these findings using BPMN notation.

In summary, our results showed that the OSS process has not been generally investigated by OSS researchers in the last years and led us to conjecture that successful open source projects follow structured software development processes. The presented OSS macro process can be used as a guide for OSS projects and also be adapted according to the OSS project reality as well as provide insights to managers and developers who want to improve their development process even in OSS and traditional environments. Finally, we presented recommendations for OSS community regarding OSS process activities.

As future work, we intend to apply the findings of our study in more recent OSS projects and provide a validation of the proposed OSS macro process though a practical point of view analyzing OSS projects process activities, their characteristics and understanding how roles are involved in each activity, fact that still not clear yet in the literature. Furthermore, we intend to investigate how researchers perform OSS process analysis in OSS projects, including approaches, techniques and tools they have used to retrieve OSS process information.

References

[1] T. Abdou, P. Grogono, and P. Kamthan. A conceptual framework for open source software test process. pages 458 –463, 07 2012.

[2] T. Abdou and P. Kamthan. A knowledge management approach for testing open source software systems. *2014 IEEE 33rd International Performance Computing and Communications Conference, IPCCC 2014*, 01 2015.

[3] S. Acuna, J. Castro, O. Dieste Tubío, and N. Juristo. A systematic mapping study on the open source software development process. volume 2012, pages 42–46, 01 2012.

[4] S. Balali, I. Steinmacher, U. Annamalai, A. Sarma, and M. A. Gerosa. Newcomers' barriers... is that all? an analysis of mentors' and newcomers' barriers in oss projects. *Comput. Supported Coop. Work*, 27(3-6):679–714, Dec. 2018.

[5] A. L. N. Campos and T. Oliveira. Software processes with bpmn: An empirical analysis. In J. Heidrich, M. Oivo, A. Jedlitschka, and M. T. Baldassarre, editors, *Product-Focused Software Process Improvement*, pages 338–341, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[6] K. Crowston, K. Wei, J. Howison, and A. Wiggins. Free/libre open-source software development: What we know and what we do not know. *ACM Computing Surveys - CSUR*, 44:1–35, 02 2012.

[7] R. A. De Carvalho and R. De Campos. A development process proposal for the erp5 system. In *2006 IEEE International Conference on Systems, Man and Cybernetics*, volume 6, pages 4703–4708, Oct 2006.

[8] T. Dillon and G. Simmons. Semantic web support for open-source software development. pages 606 – 613, 01 2009.

[9] T. Dinh-Trong and J. Bieman. The freebsd project: a replication case study of open source development. *Software Engineering, IEEE Transactions on*, 31:481– 494, 07 2005.

[10] T. Dinh-Trong and J. M. Bieman. Open source software development: a case study of freebsd. In *10th International Symposium on Software Metrics, 2004. Proceedings.*, pages 96–105, Sep. 2004.

[11] M. Dumas and D. Pfahl. *Modeling Software Processes Using BPMN: When and When Not?*, pages 165–183. Springer International Publishing, Cham, 2016.

[12] A. Ezeala, H. Kim, and L. A. Moore. Open source software development: Expectations and experience from a small development project. In *Proceedings of the 46th Annual Southeast Regional Conference on XX*, ACM-SE 46, pages 243–246, New York, NY, USA, 2008. ACM.

[13] B. Fitzgerald. The transformation of open source software. *MIS Quarterly*, 30(3):587–598, 2006.

[14] B. Fitzgerald and P. Ågerfalk. The mysteries of open source software: Black and white and red all over? 01 2005.

[15] A. Fuggetta. Open source software—an evaluation. *Journal of Systems and Software*, 66(1):77 – 90, 2003.

[16] M. Glassman and M. J. Kang. Teaching and learning through open source educative processes. *Teaching and Teacher Education*, 60:281 – 290, 2016.

[17] Z. Hashmi, S. Shaikh, and N. Ikram. Methodologies and tools for oss: Current state of the practice. *ECEASST*, 33, 01 2010.

[18] A. Ihara, M. Ohira, and K.-i. Matsumoto. An analysis method for improving a bug modification process in open source software development. *7th International Workshop Principles of Software Evolution and Software Evolution (IWPSE-Evol)*, pages 135–144, 01 2009.

[19] C. Jensen and W. Scacchi. Experiences in discovering, modeling, and reenacting open source software development processes. In *Proceedings of the 2005 International Conference on Unifying the Software Process Spectrum*, SPW'05, pages 449–462, Berlin, Heidelberg, 2005. Springer-Verlag.

[20] C. Jensen and W. Scacchi. Role migration and advancement processes in ossd projects: A comparative case study. pages 364–374, 06 2007.

[21] A. Khanjani and R. Sulaiman. The process of quality assurance under open source software development. In *2011 IEEE Symposium on Computers Informatics*, pages 548–552, March 2011.

[22] B. Kitchenham, D. Budgen, and P. Brereton. *Evidence-Based Software Engineering and Systematic Reviews*. Chapman & Hall/CRC, " ", 2015.

[23] A. Krishnamurthy and R. O'Connor. An analysis of the software development processes of open source e-learning systems. volume 364, pages 60–71, 06 2013.

[24] S. Kumar, R. Ranjan, and A. Trivedi. A new paradigm for open source software development. *Advances in Intelligent Systems and Computing*, 243:913–918, 12 2014.

[25] L. Lavazza, S. Morasca, D. Taibi, and D. Tosi. Applying scrum in an oss development process: An empirical evaluation. In A. Sillitti, A. Martin, X. Wang, and E. Whitworth, editors, *Agile Processes in Software Engineering and Extreme Programming*, pages 147–159, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[26] M. Michlmayr. Software process maturity and the success of free software projects. In *Proceedings of the 2005 Conference on Software Engineering: Evolution and Emerging Technologies*, pages 3–14, Amsterdam, The Netherlands, The Netherlands, 2005. IOS Press.

[27] A. Mockus, R. T. Fielding, and J. Herbsleb. A case study of open source software development: the apache server. In *Proceedings of the 2000 International Conference on Software Engineering. ICSE 2000 the New Millennium*, pages 263–272, June 2000.

[28] A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Trans. Softw. Eng. Methodol.*, 11(3):309–346, July 2002.

[29] K. Neulinger, A. Hannemann, R. Klamma, and M. Jarke. A longitudinal study of community-oriented open source software development. pages 509–523, 06 2016.

[30] OMG. Business Process Model and Notation (BPMN), Version 2.0, January 2011.

[31] G. Pinto, I. Steinmacher, L. F. Dias, and M. Gerosa. On the challenges of open-sourcing proprietary software projects. *Empirical Software Engineering*, 23(6):3221–3247, Dec 2018.

[32] V. Potdar and E. Chang. Open source and closed source software development methodologies. 01 2004.

[33] A. Rahman, A. Agrawal, R. Krishna, and A. Sobran. Characterizing the influence of continuous integration: empirical results from 250+ open source and proprietary projects. *Proceedings of the 4th ACM SIGSOFT International Workshop on Software Analytics - SWAN 2018*, 2018.

[34] E. S. Raymond. *The Cathedral and the Bazaar*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1st edition, 1999.

[35] H. Rocha, G. de Oliveira, M. T. Valente, and H. Marques-Neto. Characterizing bug workflows in mozilla firefox. In *Proceedings of the 30th Brazilian Symposium on Software Engineering*, SBES '16, pages 43–52, New York, NY, USA, 2016. ACM.

[36] W. Scacchi. Free and open source development practices in the game community. *IEEE Softw.*, 21(1):59–66, Jan. 2004.

[37] W. Scacchi. Free/open source software development: Recent research results and methods. *Advances in Computers*, 69:243–295, 12 2007.

[38] W. Scacchi, J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani. Understanding free/open source software development processes. *Software Process: Improvement and Practice*, 11:95–105, 03 2006.

[39] W. Scacchi, C. Jensen, J. Noll, and M. Elliott. *Multi-Modal Modeling, Analysis, and Validation of Open Source Software Development Processes*, pages 51–65. 01 2008.

[40] W. Schroeder, L. Ibanez, and K. Martin. Software process: the key to developing robust, reusable and maintainable open-source software. pages 648 – 651 Vol. 1, 05 2004.

[41] M. Scotto, A. Sillitti, and G. Succi. An empirical analysis of the open source development process based on mining of source code repositories. *International Journal of Software Engineering and Knowledge Engineering*, 17:231–247, 04 2007.

[42] A. Senyard and M. Michlmayr. How to have a successful free software project. In *11th Asia-Pacific Software Engineering Conference*, pages 84–91, Nov 2004.

[43] I. Steinmacher, C. Treude, and M. A. Gerosa. Let me in: Guidelines for the successful onboarding of newcomers to open source projects. *IEEE Software*, 36(4):41–49, July 2019.

[44] J. Wang, P. C. Shih, Y. Wu, and J. M. Carroll. Comparative case studies of open source software peer review practices. *Information and Software Technology*, 67:1 – 12, 2015.

[45] C. Wohlin. A snowballing procedure for systematic literature studies and a replication. In *18^{th} Int. Conference on Evaluation and Assessment in Software Engineering (EASE'14)*, pages 321–330, 2014.

[46] Y. Yamauchi, M. Yokozawa, T. Shinohara, and T. Ishida. Collaboration with lean media: How open-source software succeeds. In *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*, CSCW '00, pages 329–338, New York, NY, USA, 2000. ACM.