information

MDPI

Review

Frequent Releases in Open Source Software: A Systematic Review

Antonio Cesar Brandão Gomes da Silva [1], Glauco de Figueiredo Carneiro [1,*], Fernando Brito e Abreu [2] and Miguel Pessoa Monteiro [3]

[1] Programa de Pós-Graduação em Sistemas e Computação (PPGCOMP), Universidade Salvador (UNIFACS), Salvador 41770-235, Brazil; antoniocesar01@gmail.com
[2] Departamento de Ciências e Tecnologias da Informação (ISTA), Instituto Universitário de Lisboa (ISCTE-IUL), 1649-026 Lisbon, Portugal; fba@iscte-iul.pt
[3] Departamento de Informática (DI), Faculdade de Ciências e Tecnologia (FCT), Universidade NOVA de Lisboa (UNL), 2829-516 Caparica, Portugal; mtpm@fct.unl.pt
* Correspondence: glauco.carneiro@unifacs.br; Tel.:+55-71-3330-4630

Received: 26 June 2017; Accepted: 31 August 2017; Published: 5 September 2017

Abstract: *Context:* The need to accelerate software delivery, supporting faster time-to-market and frequent community developer/user feedback are issues that have led to relevant changes in software development practices. One example is the adoption of Rapid Release (RR) by several Open Source Software projects (OSS). This raises the need to know how these projects deal with software release approaches. *Goal:* Identify the main characteristics of software release initiatives in OSS projects, the motivations behind their adoption, strategies applied, as well as advantages and difficulties found. *Method:* We conducted a Systematic Literature Review (SLR) to reach the stated goal. *Results:* The SLR includes 33 publications from January 2006 to July 2016 and reveals nine advantages that characterize software release approaches in OSS projects; four challenge issues; three possibilities of implementation and two main motivations towards the adoption of RR; and finally four main strategies to implement it. *Conclusion:* This study provides an up-to-date and structured understanding of the software release approaches in the context of OSS projects based on findings systematically collected from a list of relevant references in the last decade.

Keywords: rapid release; systematic literature review; open source software projects

1. Introduction

The effectiveness and success of practices adopted by Open Source Software (OSS) projects has aroused the interest of the Software Engineering research community [1–4]. Understanding how such software projects work will enable companies to draw lessons from reported best practices and apply them in their internal projects [5,6].

According to [7], there are more than 400 active OSS distributions. Due to the ever growing competition, distributions need to release new features and versions in shorter time frames. To achieve this, they rely on hundreds of volunteers to integrate the latest versions and bug fixes of the tens of thousands of integrated upstream components [7].

Many projects issue a new release after implementing a certain set of features and/or fixing a set of pending bugs [8], which involves numerous challenges. Alternatively, some projects adopt a time-based strategy, in which releases are planned for a specific date. Traditional release strategies have associated problems that can be overcome by time-based release management [1,9].

In this paper, we review published studies following a systematic literature review (SLR) guideline that address the growing interest of OSS projects in the adoption of frequent deployment (aka Frequent Release) approaches [1,10]. In contrast to an expert review using ad hoc literature selection, an SLR is a

methodologically rigorous review of research results based on evidence from best quality scientific studies on a specific topic or research question [11]. This interest is a response to the need of providing shorter time-to-market in response to customer requests, thus improving their satisfaction [10].

The rest of this paper is organized as follows. Section 2 presents the problem statement/scope and emphasizes the differences between this Systematic Literature Review (SLR) and others that focus on similar topics. Section 3 outlines the research methodology. Section 4 discusses the results of the SLR. Concluding remarks, as well as scope for future research, are provided in Section 5.

2. Software Release Main Related Concepts

A release is a version of a system that has been delivered to customers (or other users in an organization) for use [12]. A version is an instance of a configuration item that differs, in some way, from other instances of that item. Versions always have a unique identifier, which is often composed of the configuration item name plus a version number [12].

There are a variety of strategies for deciding when to release a new version of a piece of software, and different projects may employ different techniques [1]. Software projects such as the Linux Ubuntu plans in advance to release on a certain date, and the preceding development effort is aimed at producing a release on this prearranged date. This characterizes the time-based releases. Another important concept is the regular cycles where each release is delivered at regular intervals. On the other hand, a release classified as continuous flow is the one that has the ability to bring valuable product features to customers on demand and at will (deployment), in series or patterns with the aim of achieving continuity and in significantly shorter cycles than traditional lead-times, from a couple of weeks, to days or even hours [13].

The term rapid release was coined by agile methodologies such as Extreme Programming (XP) [14] to highlight likely benefits that shorter software release cycles may offer to companies and end users. In this scenario, companies would expect faster feedback regarding new features and bug fixes, and releases could become slightly easier to plan. This would emphasize the differences between short-term and long-term planning releases [15]. With rapid releases, developers are not under pressure to complete features because of an approaching release date, and can focus on quality assurance every week instead of every couple of months [15]. Moreover, the higher number of releases tend to provide more market visibility for the companies. Customers also experience benefits in the form of faster access to new features, bug fixes and security updates [15].

According to [16], several web-based organizations, both on the server side (e.g., Facebook and Google) and the client side (e.g., Google Chrome and Mozilla Firefox), have adapted their development processes towards rapid release models. Instead of working for a long period of time (e.g., months) on a major new release, these companies limit the time between two subsequent releases to a short period of time, such as a couple of weeks, days or, in specific cases, hours to bring their latest features to customers faster [16]. Rapid release cycles not only provide faster user feedback but are also easier to plan due to their smaller scope [14]. They also have important repercussions on software quality [16].

3. Problem Statement and Scope

Release strategies used by OSS communities can be classified as feature-based and time-based [1]. The former issue a release when a set goals have been fulfilled, e.g., a certain set of features has been implemented and/or a set of pending bugs have been fixed. The latter issue releases according to a preset schedule (aka "roadmap"). However, evidence from the literature suggest a preference for time-based strategies among OSS projects [1,17]. Indeed, fixed release dates and intervals allow a better internal planning, so that additional patchwork and differentiating features may be added in time for product shipment to users [17].

The scope and coverage of this SLR differ significantly from previous reviews. During the carrying out of this study, we found one SLR and one systematic mapping focusing on rapid releases and testing [18] and continuous deployment in software intensive products and services [13]. However,

none of these studies is focused on the relationship between the issues involved, as targeted by the present SLR. The literature review reported in [18] suggests that Rapid Release (RR) is a prevalent industrial practice, even in some highly critical domains of software engineering, and this originates from successful approaches such as agile, open source and lean software development. The authors also conclude that empirical studies proving evidence of the claimed advantages and disadvantages of RR are still scarce [18].

4. Research Methodology

This section provides a short description of the methodology applied for the phases of planning, conducting and reporting the review. In contrast to a non-structured review process, a Systematic Literature Review (SLR) [11,19] reduces bias and follows a precise and rigorous sequence of methodological steps to review the scoped literature. SLRs rely on well-defined and evaluated review protocols to extract, analyze, and document results as the steps conveyed in Figure 1.

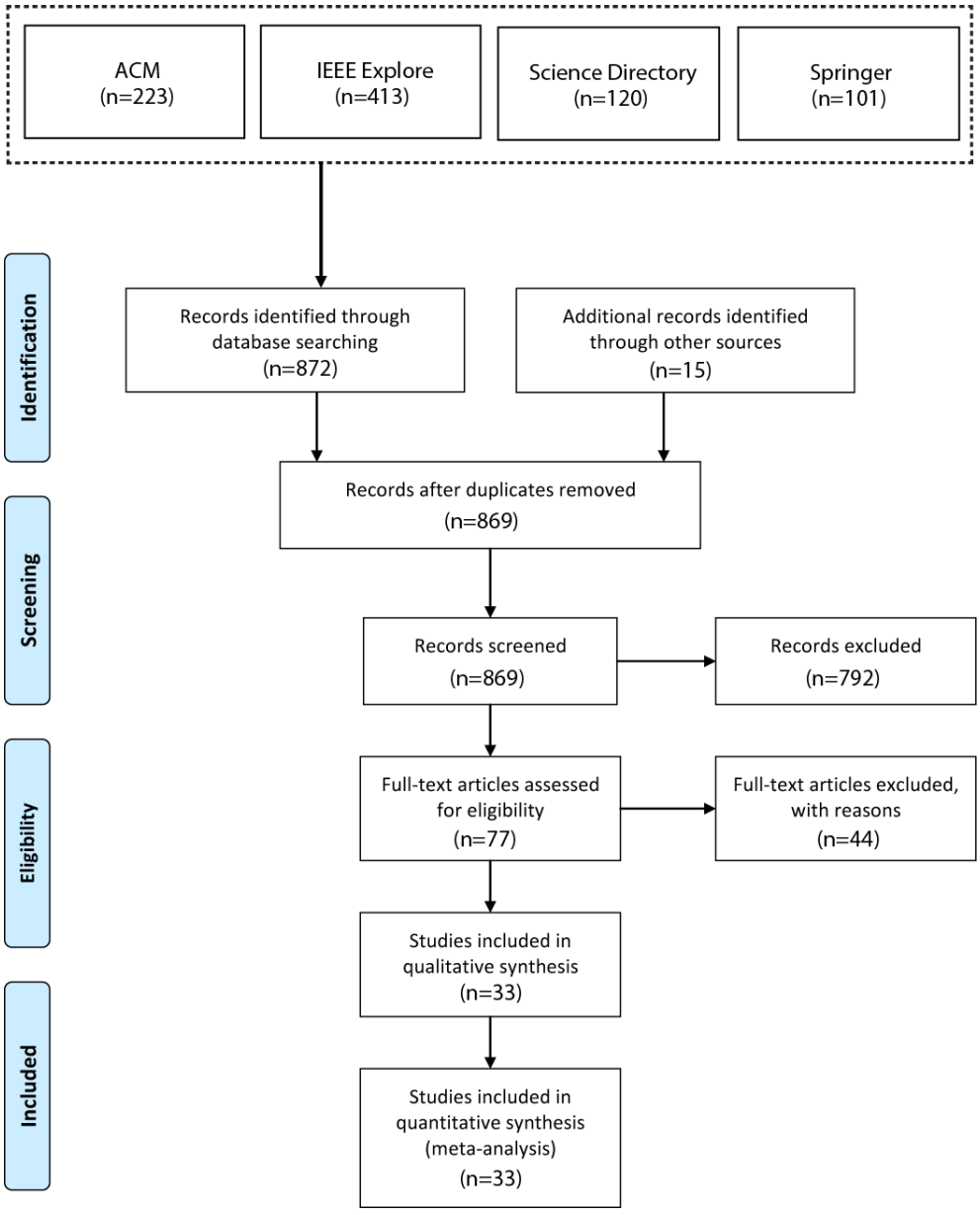


Figure 1. Stages of the study selection process.

Identify the needs for an SLR. Search for evidence in the literature on how OSS projects decide on the adoption and use of RR and what are the motivations behind this adoption; strategies applied, as well as advantages and difficulties reported in those scenarios.

Specifying the research questions. We aim to answer the following research questions by conducting a methodological review of existing research: **RQ1**. *What is the time scale applied to frequent software releases in the context of OSS projects?* Knowledge of the definition of frequent software releases from the adopters' perspective and how it has been implemented by the OSS community can be used as a reference for potential new adopters. **RQ2**. *What are the main motivations for the implementation of frequent software releases in the context of OSS projects?* Identifying goals, proposals and motivations for the use of frequent software releases in OSS projects will help practitioners to better characterize their needs and therefore provide conditions suitable for a successful adoption of these practices. **RQ3**. *What are the main strategies adopted by practitioners to implement frequent software releases in the context of OSS projects?* Knowledge of effective strategies adopted by practitioners is a starting point for building a body of knowledge regarding the use of frequent software releases. **RQ4**. *What are the main advantages and challenges related to the adoption of frequent software releases in the context of OSS projects?* Knowledge of successful strategies and problems raised by the implementation of frequent software releases will contribute to software development organizations and other OSS projects to be more confident in the adoption of that practice.

These four research questions are not independent from each other. In the case of **RQ2** and **RQ3**, specific motivations can lead to the selection of a specific strategy. Moreover, there is the possibility that a specific strategy can bring specific advantages and challenges in the adoption of RR, revealing a close relationship between **RQ3** and **RQ4**.

4.1. Searching for Study Sources

Based on the above research questions, keywords were extracted and used to search the primary study sources. Regarding the time frame, we considered papers published in journals and conferences from January 2006 to January 2017. Although some studies commenting on the use of rapid releases were published before 2006, such as [20,21], we decided to focus on the last decade to provide an up-to-date and structured understanding of the software release approaches in the context of OSS projects.

The following search string uses the same strategy as described in [22]:

((“rapid release” or “fast release” or “frequent release” or “release history” or “quick release”) and (“oss” or “open source” or “open source software”))

4.2. Selection of Primary Studies

The following steps guided the selection of primary studies.

Stage 1—Search string results automatically obtained from the engines—Submission of the search string to the following repositories: ACM Digital Library, IEEE Xplore, Science Direct and Springer Link. We selected these libraries due to their relevance as sources of publications in software engineering [23]. The search was performed using the specific syntax of each database, considering only the title, keywords, and abstract. The search was configured in each repository to select only papers carried out within the prescribed period. The automatic search was complemented by a manual search to obtain a list of studies from journals and conferences. Duplicates were discarded.

Stage 2—Read titles and abstracts to identify potentially relevant studies—Identification of potentially relevant studies, based on the analysis of title and abstract, discarding studies that are clearly irrelevant to the search. When there was some doubt about whether a study should be included or not, it was included for consideration at later stages.

Stage 3—Apply inclusion and exclusion criteria upon reading the introduction, methods and conclusion—Selected studies in previous stages were reviewed, by reading the introduction,

methodology section and conclusion. Afterwards, inclusion and exclusion criteria were applied. At this stage, whenever doubts arose preventing a conclusion, the study was read in its entirety.

Inclusion Criteria: IC1—The publications should be "journal" or "conference" and written in English. IC2—Works involve an empirical study or present "lessons learned" (experience report). IC3—If several journal articles reporting the same study are found, the most recent article is included. IC4—Articles that address at least one of the research questions.

Exclusion Criteria: EC1—Studies not focused on Rapid Releases. EC2—Studies merely based on expert opinion without providing sound evidence. EC3—Publications that are earlier versions of the latest published work. EC4—Publications published before January 2006 (note: publications after January 2017 are also not included in this SLR because this was the month where the collection process ended).

Stage 4—Obtain primary studies and make a critical assessment of them—A list of primary studies was obtained and later subjected to critical examination using the following quality criteria, suggested by Dyba and Dingsoyr [24].

Quality Criteria: QC1—Is the paper based on research (or is it merely a "lessons learned" report based on expert opinion)? QC2—Is there a clear statement of the aims of the research? QC3—Is there an adequate description of the context in which the research was carried out? QC4—Was the research design appropriate to address the aims of the research? QC5—Was the data collected in a way that addressed the research issue? QC6—Is there a clear statement of the findings?

4.3. Data Extraction

All relevant information on each study was recorded on a spreadsheet. This information was helpful to summarize the data and map them to its sources. The following data were extracted from the studies: (i) name and authors; (ii) type of article (journal, conference, workshop); (iii) aim of the study; (iv) research question; (v) scenario(s); (vi) results and conclusions; (vii) benefits; (viii) limitations and challenges. We analyzed the extracted data quantitatively in an effort to answer our research questions.

4.4. Data Synthesis

This synthesis aims at grouping findings from the studies in order to: identify the main concepts, motivations and strategies for the implementation, as well as advantages reported from the projects and associated challenges and issues, regarding the four research questions (**RQ1, RQ2, RQ3** and **RQ4**). Other information is synthesized whenever necessary. We use the meta-ethnography method [25] as a reference for the process of data synthesis.

4.5. Conducting the Review

We started the review with an automatic search, followed by a manual search, to identify potentially relevant studies and afterwards applied the inclusion/exclusion criteria. The first tests using automatic search began in December 2015. We had to adapt the search string in some engines, taking care to preserve its primary meaning and scope. The manual search consisted of studies published in conference proceedings and journals that were included by the authors while searching the theme in different repositories. These studies were likewise analyzed regarding their titles and abstracts. Figure 1 conveys them as 15 studies which, compared with 872 studies selected through the automatic search, represent approximately 1.7% of all papers. This indicates that the automatic search provided the vast majority of the analyzed papers of this SLR. We tabulated everything on a spreadsheet so as to facilitate the subsequent phase of identifying potentially relevant studies. Figure 1 presents the results obtained from each electronic database used in the search, which resulted in 872 articles spanning all databases.

4.6. Potentially Relevant Studies

Results obtained from both the automatic and manual search were included on a single spreadsheet. Papers with identical title, author(s), year and abstract were discarded as redundant. At this stage, we registered a total of 872 articles, namely 857 from the automated search plus 15 from the separate manual search (*Stage 1*). We next read their titles and abstracts to identify relevant studies, resulting in 77 papers (*Stage 2*). At (*Stage 3*), we read the introductions, sections on methodology and conclusions and applied the quality criteria, yielding 33 studies considered for the subsequent stage. At (*Stage 4*), answers to the four research questions—**RQ1**, **RQ2**, **RQ3** and **RQ4**—were obtained. Table 1 presents an overview of the selection process per public data source.

Table 1. Selection process overview per public data source.

Public Data Source	Search Result	Relevant Studies	Search Effectiveness
ACM	223	11	4.9%
IEEE	413	10	2.4%
ScienceDirect	120	4	3.3%
Springer	101	3	2.9%
Manual Inclusion	15	5	33.3%

5. Results and Discussions

This section presents the SLR results to answer research questions **RQ1, RQ2, RQ3** and **RQ4**. All selected studies are listed in the (Table A1) and referenced as "*S*" followed by the number of the paper.

As a result of the analysis of the selected studies, we organized key elements related to the four research questions as shown in Figure 2. The figure provides an overview of issues and concepts related to each RQ along with selected studies that discussed them. The nodes are numbered to identify the elements in the structure according to the RQ it is related to. Node 1 represents concepts related to the definition of RR, in this case **RQ1**: (1.1) *Regular Cycle*, (1.2) *Continuous Flow* and (1.3) *Short Release without regular cycle*. Node 2 represents the motivations to implement RR, in this case **RQ2**: (2.1) *OSS Attractiveness and Increase of Participants*, (2.2) *Maintenance and Increase of Market Share*. Node 3 represents four possible strategies to implement RR (**RQ3**): (3.1) in the context of *Test Driven Development*, through *Automated Release Process* (3.2), *Continuous Delivery* (3.3) and through *Time-Based Releases*.

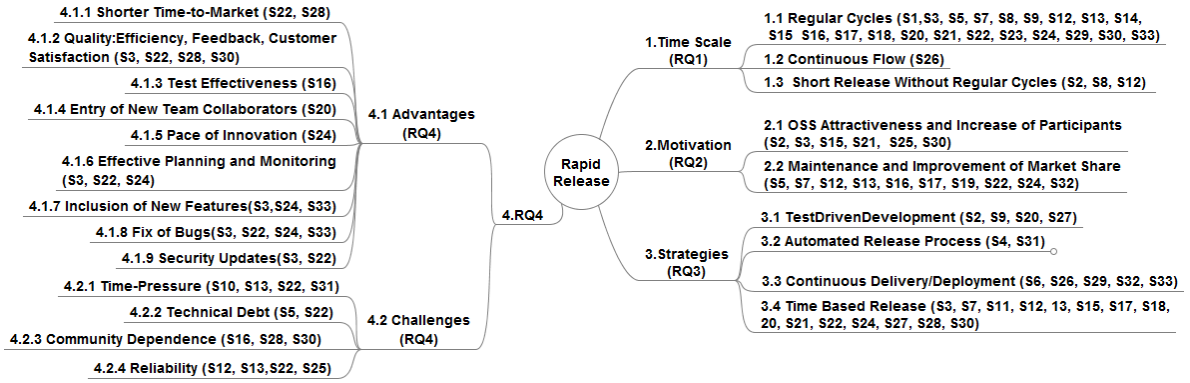


Figure 2. Findings from the selected studies.

The following advantages were identified in the literature (**RQ4—part 1**): *Shorter Time to Market* (4.1.1), *Quality: Efficiency, Feedback, Customer Satisfaction* (4.1.2), *Test Effectiveness* (4.1.3) *Entry of New Team Collaborators* (4.1.4), *Pace of Innovation* (4.1.5), *Effective Planning and Monitoring* (4.1.6), *Inclusion of New Features* (4.1.7), *Fix of Bugs* (4.1.8), and *Security Updates* (4.1.9). The challenges (**RQ4—part 2**)

reported were dealing with *Time Pressure* (4.2.1), *Technical Debt* (4.2.2), *Community Dependence* (4.2.3) and *Reliability* (4.2.4).

Figure 3 conveys the selected studies and the respective research questions they focus on. The figure conveys that 23 studies addressed issues related to **RQ1**, 16 studies discussed **RQ2** issues, 24 studies were related to **RQ3** and finally 15 papers addressed **RQ4** issues. Figure 4 depicts the temporal distribution of the selected studies.

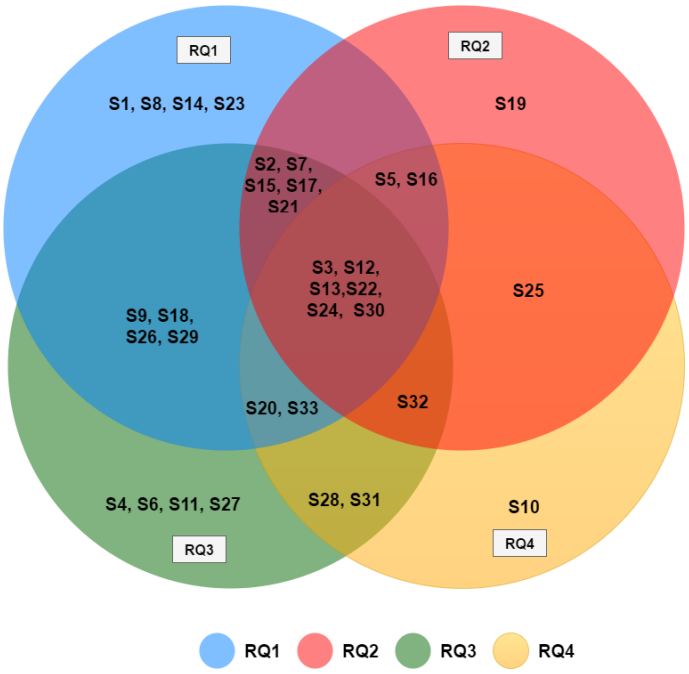
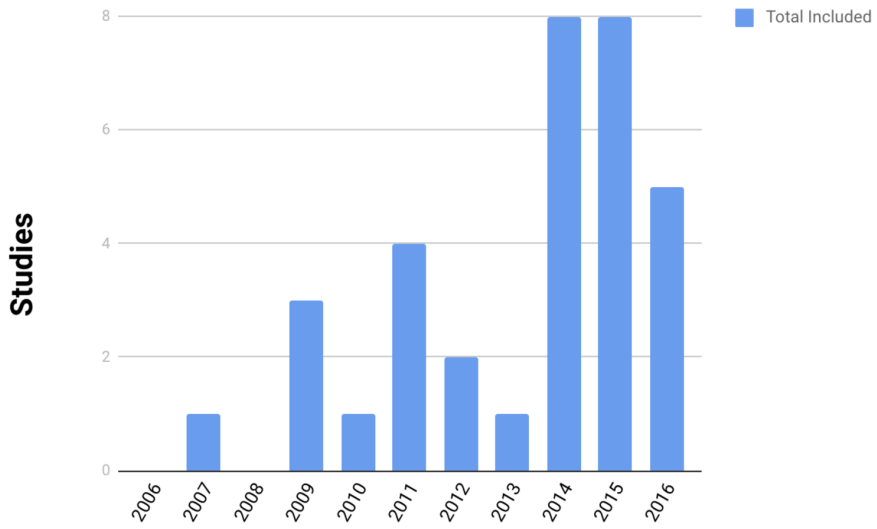


Figure 3. Selected studies per research question (**RQ**).



	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016
Total Included	0	1	0	3	1	4	2	1	8	8	5

Figure 4. Timeline distribution of papers.

5.1. Time Scale of Software Releases in OSS Projects

We identified three important concepts related to the *time scale applied to frequent software releases in the context of OSS projects* (**RQ1**) according to findings from the selected studies. The concepts are *Regular Cycles*, i.e., releases delivered at regular intervals (**S1**, **S3**, **S5**, **S7**, **S8**, **S9**, **S12**, **S13**, **S14**, **S15**, **S16**, **S17**, **S18**, **S20**, **S21**, **S22**, **S23**, **S24**, **S29**, **S30**, **S33**); *Short Release without Regular Cycles*, frequent releases whose time intervals at which new versions of the software project are provided are not regular, mentioned in (**S2**, **S8**, **S12**); and finally the *Continuous Flow*, related to the ability of the release to bring valuable product features to customers on a demand basis, as mentioned in (**S26**). Figure 5 represents the distribution of papers among the aforementioned three types of frequent release time scales. In the following paragraphs, we discuss the contribution of each of the mentioned papers to **RQ1**.

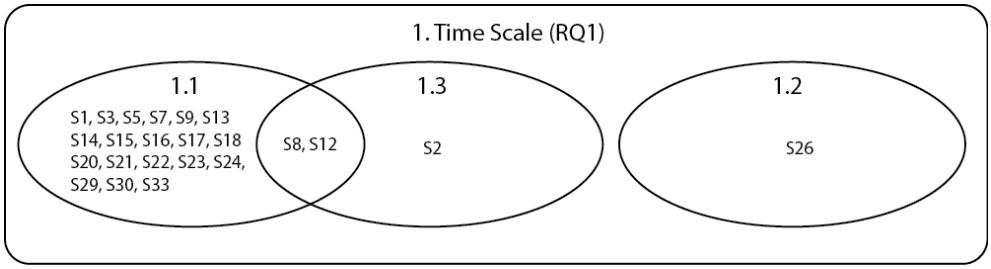


Figure 5. Types of time scale (**RQ1**).

Considering the commonalities identified in the studies grouped in items 1.1, 1.2 and 1.3 of Figure 2, we observed that rapid release cycles tend to be measured in hours, days, weeks or few months. The studies covered refer to projects whose time interval between releases is always less than six months. Another common characteristic is that frequent releases are always associated with short development cycles, using time as a basis rather than features in demand.

Regarding the differences identified in the time scale approach in the items 1.1, 1.2 and 1.3 of Figure 2, we observed that in papers that refer to a preset schedule, processes are concerned in defining each stage in a precise way. On the other hand, papers referring to non regular cycles do not stipulate a concrete date for the next release—even when the referred process is a mature one. One example is Debian Linux, which documents all its processes in its repository and keeps it up to date.

In the case of *regular cycles* and according to **S30**, many projects moved towards a time-based release management strategy based on the early successful experiences of projects such as GNOME [21]. This increases the exposure of the software and may lead to better feedback. However, sufficient development done in release intervals appears to be essential to pursue a time-based release management strategy. A large number of time-based Free and open-source software (FOSS) projects have chosen a release interval of up to six months. Major Linux distributions, such as Fedora, are examples of this practice.

S15 recognizes that the advent of open source software development resulted in a development model unencumbered by those traditional business constraints, and empowered by (potentially) thousands of developers and an even greater number of testers. Under this model, consecutive releases can take place in the same day. In line with this scenario, **S8** highlighted the time line of the Chrome OSS project releases and identified their frequency as between one and two months between each release. **S16** also highlights that frequent releases delivers working software from every couple of weeks to every couple of months, with a preference for the shorter timescale. **S16** also notes that, in RR, the modifications on each release are very limited, which makes the number of faults per release very small, compared to the traditional releases.

In **S1**, the authors also use data from the Chrome web browser OSS project to examine its release history, bug reporting and fixing data, and usage statistics. They note that Chrome provides evidence

of a fast evolving system, with short version cycles and a user base that very quickly adopts new versions as they become available (due largely to Chrome's mandatory automatic updates). **S5** uses clone detectors to understand the evolution and to compute the overall change distribution between all existing releases of Firefox between June 2011 and February 2013. The authors measured similarity between consecutive versions. The approach gives insights into how a change of release cycle may influence the evolution of a software, showing a tendency of increased risk in fast release cycles even if the code modifications are in general smaller. **S7** investigates the consequences and impact of rapid-release methodology on the security of the Mozilla Firefox browser. The resulting data shows that Firefox RR does not result in higher vulnerability rates. The pattern exhibited by vulnerability disclosure in Firefox is the result of would-be attackers having to re-learn and re-adapt their tools in response to a rapidly changing code base. The papers **S3**, **S13**, **S17**, **S21**, **S22**, **S23**, **S24** and **S33** discuss the methodology adopted by Mozilla Firefox and/or Google Chrome that moved from a traditional software release approach based in 12–18 months to a delivery of new releases every six weeks. **S18** reports that Google Plus can release new changes in 36 h and Facebook.com releases twice a day on weekdays. **S20** highlights that the Defense Information Systems Agency (DISA) has historically operated on 18–36 months release cycles and is now striving towards delivering smaller components in 30–60–90 day release cycles.

In **S29**, it is mentioned that recent studies have started to evaluate the concept of continuous delivery and rapid releases from the perspective of software quality. **S9** highlights that Agile teams strive to deliver working software at frequent intervals ranging from two to four weeks. **S14** mentions that the age of the project and the time between releases is likely to affect the size of the development team and the quantity of effort that can be contributed for each release.

Considering the possibility of continuous flow, **S26** mentions that rapid and continuous software engineering refers to the "organizational capability to develop, release and learn from software in rapid parallel cycles, such as hours, days or very few weeks".

In the case of short releases without regular cycles, **S2** presents an approach augmented with a number of engineering best practices specifically tailored to attain a weekly release cycle for a hosted software product. It is noted that the problem of managing rapid releases for a hosted product is likely to become even more important in the future, cutting across vast tracts of enterprise IT in addition to publicly hosted "Software as a Service" products.

In **S12**, the authors confirm once more that releasing every few weeks is typically referred to as a rapid release cycle, while releasing monthly or yearly is typically referred to as a traditional release cycle. The authors also conclude that 98% of addressed issues in systems with rapid release cycle (in this case Firefox project) were delayed by one or more releases. On the other hand, projects with traditional releases (in this case ArgoUml and Eclipse projects), only address 34 to 60% of their issues in one or more releases. This provides a preliminary evidence that OSS projects with RR tend to address issues earlier than projects with more traditional release rates.

5.2. Motivations for Software Releases in OSS Projects

We identified two *main motivations for the implementation of frequent software releases in the context of OSS projects* (**RQ2**), which are the *project attractiveness/increase of participants* (**S2**, **S3**, **S15**, **S21**, **S25**, **S30**) and *maintenance and increase of market share* (**S5**, **S7**, **S12**, **S13**, **S16**, **S17**, **S19**, **S22**, **S24**, **S32**). Figure 6 represents the distribution of papers among these two motivation types. In the following paragraphs, we will discuss the contribution of each of the mentioned papers to **RQ2**.

Considering the commonalities identified in the studies grouped in items 2.1, 2.2 of Figure 2, we observed that OSS projects tend to stay afloat due to participation of volunteers. For this reason, it is common for them to use processes that are attractive to collaborators. It was also possible to conclude that participating in certain projects can enhance the status of their collaborators. It was also possible to find some evidence suggesting that projects that adopted rapid releases improved their commercial success performance relative to their competitors.

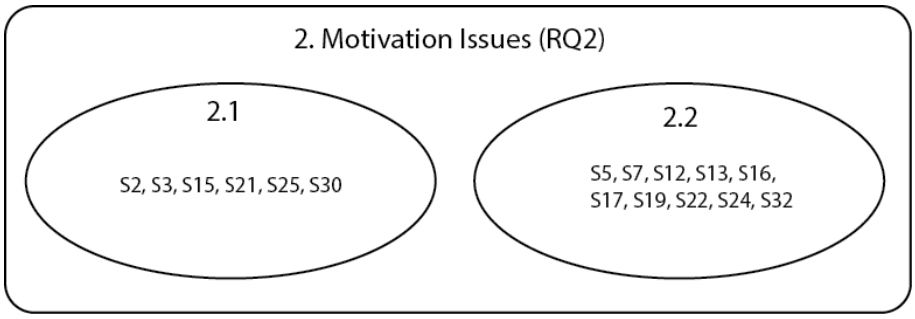


Figure 6. Types of motivation issues (**RQ2**).

There are reports of projects that increased the number of participants where the majority are volunteers motivated by attractiveness and challenges associated with short releases activities **S30**. Moreover, it raises the possibility of user feedback in a shorter feasible time interval. The maintenance and increase of market share occurs as a result of the pace of changes and hence new features provided by the software.

S15 and **S30** note an increase in the number of participants, most of which originate from the open source community. The OSS model matured and participation in open source projects can add status to its participants as well. **S30** calls this model "volunteer-oriented". **S2** highlights the pressure from customers, which encourages greater care on the part of the project's team, to meet the demands of customers. **S15** highlights the further development of the OSS model, which, due to its characteristics, encourages time-based releases. **S3** reports that a shorter release cycle provides various benefits to both companies and end users. Companies get faster feedback about new features and bug fixes, and releases become slightly easier to plan (short term vs. long-term planning). Customers benefit as well, since they have faster access to new features, bug fixes and security updates. **S21** says that one of the key properties of open source projects is that community members are voluntarily engaged in a project. **S25** reports that there is a widespread recognition across software industry that open source projects can produce software systems of high quality and functionality, and OSS development is based on a relatively simple idea: the original core of the OSS system is developed locally, then a prototype system is released, so that other programmers can freely read, modify and redistribute that system's source code.

S5 reports that some software projects adopted rapid releases to compete better and with faster solutions. **S32** points out that to maintain their competitive advantage, software intensive companies need to deliver valuable product features to customers considerably faster than before, if not close to real-time, while embracing business changes and pursuing economic efficiency. **S7** remarks that the survival of a software project entails the frequent introduction of new features. **S12** highlights that users and contributors may become frustrated when an addressed issue is not integrated in an upcoming version. **S13** notes that the main motivation for moving to rapid (or short) release cycles is time-to-market. **S19** points out that frequent system releases are performed in line with user expectations for greater responsiveness and shorter cycle times.

S17 notes that a switch to faster cycles was driven by the need to meet the demands of users, in line with the evolution of web standards and competition in the browsers market. **S24** points out that the Firefox project adopted the rapid release model to be able to compete with Chrome, which already used it. The fast expansion of mobile platforms resulted in increased competition between applications, making rapid releases very important for the applications to remain competitive. **S16** highlights that Firefox was already losing market share to Chrome when it decided to adopt the rapid releases.

Finally, **S17** notes that the advancement of mobile platforms brought a big incentive to rapid releases in that the greater ease of access resulted in an increased competition between applications. This made rapid releases more important for the application or service to stay competitive.

5.3. Strategies to Adopt Software Releases in OSS Projects

We also identified four *main strategies adopted by practitioners to implement frequent software releases in the context of OSS projects* (**RQ3**). These strategies are the *time-based release* strategy, discussed in the studies **S3**, **S7**, **S11**, **S12**, **S13**, **S15**, **S17**, **S18**, **S20**, **S21**, **S22**, **S24**, **S27**, **S28**, **S30**; the *automated release process* discussed in **S4** and **S31** as an essential aspect to support frequent releases; the *Test Driven Development* as an interesting option for companies that practice rapid releases approaches (**S2**, **S9**, **S20**, **S27**); and, finally, the *continuous delivery/deployment* discussed in (**S6**, **S26**, **S29**, **S32**, **S33**). Figure 7 represents the distribution of papers among these four strategies to implement software releases.

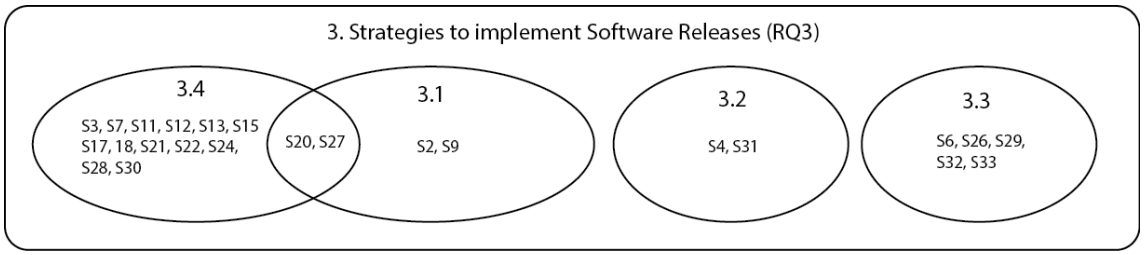


Figure 7. Types of strategies to implement software releases (**RQ3**).

According to **S28**, while a time-based release strategy provides several benefits, it is important to realize that it does not necessarily benefit all projects. The first step is to decide whether or not a project is suitable for a time-based release strategy. In this case, it is not advisable to implement regular releases if there has been little work done that would warrant a new release. The key point is that a successful implementation of time-based releases is based on trust among its contributors and the release manager, as well as on appropriate control structures that should be accepted by developers. Once the time-based release is chosen, it is time to determine the release interval. To this end, **S28** identifies five factors that affect the choice of interval: (i) regularity and predictability; (ii) nature of the project and its users; (iii) commercial factors; (iv) cost and effort; and (v) network effects (need to synchronize the project release schedule with the schedules of other projects from which it can leverage benefits).

S3, **S7**, **S13**, **S21**, **S22**, **S24** describe the Mozilla strategy, which is characterized by each release of Firefox to completely replace the previous one. In addition, every new version goes through the following 4-release-channels workflow: *Nightly*: integrates new resources from developers' source code repositories as soon as they are ready. *Aurora*: inherits the characteristics of Nightly every six weeks. Resources that need more development are disabled and left for the following import cycle. *Beta*: receives just the new features from Aurora that are scheduled for the management of the new version of Firefox. *Main*: receives the mature version of Beta. This switching between channels is defined every six weeks (**S12**, **S17**). **S11** also highlights in Mozilla the code review as a basic mechanism for the validation and deployment of patches. **S13** calls this process the *train model*, as each new feature and error correction flows along the repositories. **S21** notes that the community support is basically comprised of volunteers. **S27** points that the Firefox project was forced to hire extra testing resources and cut on regression testing coverage. **S18** describes the Chrome strategy, which, like Mozilla, goes through an interval of six weeks with code transitions between three channels: *Development*, *Beta* and *Stable*. When development of a new release begins, previous releases switch to their subsequent channels.

S15 notes that the global dispersion of co-developers means that the code can be tested 24 h a day. **S30** explains that a time-based strategy specifies a date for the release well in advance and a schedule is made public, so people can plan accordingly. Prior to the release, there is a cut-off date on which all features are evaluated as regards stability and maturity. A decision is then made as to whether the features can be included in the release or whether they should be postponed to the following release.

S4 points out that when a release process is well controlled (i.e., repeatable) and smooth (i.e., automated when possible), organizations can afford short release cycles.

S2 notes that it is fundamental for an effective Quality Assurance (QA) process to use some automated test tool in an integrated manner, so as to support frequent releases. **S9** stresses that focus on testing is critical for small teams that support quick releases. **S20** highlights that management began to recognize TDD (*Test Driven Development*) strategic discriminators against competition with a greater assurance of receiving high quality software.

S26 reports a systematic mapping based on continuous delivery and notes that continuous evolution of features drive this process, enabling, for example, the monthly delivery of releases. **S29** also lends strength to the idea that continuous deliveries are a recent phenomenon that is suitable for the rapid release of modern applications. **S6** starts by pointing out the difference between continuous integration, which focuses on the automation of the build process on a central server, and continuous delivery, which extends the previous approach to all workflows needed for the test and deployment of a new build, consequently simplifying the release of software and enabling shorter feedback cycles between developers and customers.

5.4. Advantages and Challenges of Software Releases in OSS Projects

Advantages. The main positive points associated to rapid releases are: *quick return on customer needs, rapid delivery of new features, quick bug fixes, immediate release security patches* (**S3**, **S22**, **S24**, **S33**), *increased efficiency* (**S22**), *entry of new collaborators* (**S20**), and *greater focus on quality on the part of developers and testers* (**S16**, **S22**, **S28**, **S30**).

S22 presents a semi-systematic review that associates rapid releases to testing. It reports that frequent releases driven by testing allows for a greater and quicker feedback in each release, and also increases the incentive for developers to deliver a quality product. Noteworthy is the increased efficiency due to greater time-pressure. The study also adds to the list of benefits that narrower tests—due to the reduced test time—also allows for deeper testing. Narrower tests are also easier to manage.

S16 highlights the ease of planning and testing, since the tests are more focused and run more frequently, easing the monitoring of progress and quality (**S3**, **S22**, **S24**).

S24 proposes the increase in the pace of innovation, since the high rate of releases encourages the team to continuously attempt new solutions and new tools. In addition, a greater rate of releases provides more marketing opportunities for the company.

S28 notes that short release intervals allow for more competitive OSS projects compared to proprietary designs, since rapid release brings significant advantages over competitors using the traditional cycle. For example, the *Beta* cycle corresponds to several new product versions arriving to the market. Another factor raised is the enhancement of reputation and employee satisfaction, as they see their code quickly being used by users.

S30 notes a trend regarding the increasing maturity of the practice of OSS, with an increase in their significance and economic potential. OSS projects are increasingly adopting rapid releases so as to allow for greater quality and sustainability. The quicker feedback that these practices allow also provides more information on what parts of the software are more in need of attention.

S20 reports on a case study in which the team—which initially was of small size and taking just a small office—increased to 20 developers as a consequence of the software project success after adoption of rapid releases and took almost the whole of the floor.

Challenges and Issues. The main negative points are: reliability of new versions (**S12**, **S13**, **S22**, **S25**), increase in the "technical debt" (**S5** , **S22**), pressure felt by employees (**S10**, **S22**) and community dependence (**S16**, **S28**, **S30**).

In its systematic review, **S22** presents the main weak points of rapid releases. On one hand, tests become more focused, but, on the other hand, it also becomes practically impossible to test all possible options. In addition, the short time available does not allow for test quality requirements,

e.g., performance. Another point mentioned is increased pressure on the team, which can lead to exhaustion. Finally, there is an increase in technical debt, since it allows for less time for activities such as refactoring. Neglect of those issues risks compromising the quality of the software and negatively impact organizations in the long run.

S25 also confirms that reliability requirements for the software require a delay in releases, to allow for enough time for testing. **S12** points that *Addressed Issues* are usually delayed in a rapid release cycle. Finally, **S13** reports that short cycles and shorter deadlines entail shorter times for testing, which, in the long run, may yield less stable released versions. Likewise, **S5** also claims that adoption of rapid releases makes it difficult to maintain re-engineering activities, which, in the long run, results in the increase of technical debt.

S31 presents a study that concludes that the traditional division between operators and developers that takes place in many organizations is a major obstacle for fast and frequent releases of applications.

In his study, **S16** claims that the difficulty in attracting a large number of volunteers for the test community makes the tests in rapid releases more deadline oriented. **S28** claims that projects are maintained exclusively by volunteers require a significant planning effort to cope with periods of shortage of volunteers, e.g., Christmas.

Table 2 lists the top ten papers included in the review according to Google Scholar citations. These papers are evidence of the relevance of the issues discussed in this SLR and the influence these papers exert on the literature as can be confirmed by their respective citation counts. The table presents an overview of the distribution of the most relevant papers according to the addressed research questions. In the following paragraphs, we briefly describe these papers. Papers **S3**, **S12**, **S13**, **S22**, **S24** and **S30** address the four **RQ**s of this SLR. **S3** is the top cited paper and is referenced by 57 publications.

Table 2. Top ten cited papers according to Google Scholar.

Studies	Cited By	Research Question
S3	62	RQ1, RQ2, RQ3 and RQ4
S25	40	RQ2 and RQ4
S11	30	RQ3
S19	21	RQ2
S21	19	RQ1 and RQ3
S1	18	RQ1
S14	17	RQ1
S6	15	RQ3
S10	13	RQ4
S2	12	RQ1, RQ2 and RQ3

Paper **S3** reports that the results of migrating the Firefox browser to versions that had a shorter release cycle, compared with those having a traditional release cycle, were: (1) shorter release cycles where users do not experience significantly more post-release bugs; (2) bugs fixed more quickly and (3) users experience these bugs earlier during software execution (the program crashes sooner). The case study reporting the Mozilla Firefox adoption of RR, its consequences, challenges and impacts is a relevant example and reference of how OSS projects deal with the RR approach. Paper **S3** also uses data from the Mozilla Firefox project transition to RR.

In **S25**, the authors recognize that multiple releases are expected to maintain a sufficient number of volunteers and to attract newcomers. On the other hand, a difficulty arises in that RR strategy and OSS reliability seem to be in conflict with each other. In order to support the selection of the optimal version-updating, the authors propose a decision model based on multi-attribute utility theory (MAUT). The application example shows that the proposed decision model can assist management to make a rational decision based on its own scenarios. In **S11**, the authors examine the process of incrementally submitting and integrating patches into Mozilla Firefox OSS project.

In **S19**, the authors propose a novel framework for the semantic integration of data from a variety of data sources and tool support to allow the efficient data collection, even in projects with frequent iterations. They recognize that frequent system releases are performed in line with user expectations for greater responsiveness and shorter cycle times. Moreover, they also identify that OSS teams routinely develop complex software products in distributed settings and with rather lightweight processes and project documentation. In this context, project managers and task leaders need data collection services as foundation for the timely overview on progress, cost, and quality of the project activities, similar to a data warehouse for analyzing business processes. They analyze data from two OSS projects and compare the proposed framework with a traditional approach. A major result is that the new approach seems well suited to make data collection for project monitoring 30–50% more efficient, particularly if data sources evolve during the project. As a result, they propose a framework for the semantic integration of data from a variety of data sources and tool support to allow the efficient data collection, even in projects with frequent iterations.

6. A Preliminary Guideline for the Adoption of Rapid Releases

To present a preliminary guideline for the adoption of Rapid Releases, we suggest the use of findings presented at Figure 2. First of all, potential companies need to answer the following questions: *Is the company interested in increasing its market share? To which extent does the company want to promote the attractiveness of their software projects and to increase the number of participants in these projects?* The answers to these two questions can indicate the suitability of the rapid release approach for such companies. In this case, companies that plan to increase their market share (node 2.2 from Figure 2) and promote the attractiveness of their software projects (node 2.1 from Figure 2), as well as the number of their participants (node 2.1 from Figure 2) can consider the rapid release approach as an effective option. The next step is the strategy to implement and manage the rapid releases. Evidence from selected studies indicates four possible strategies for this purpose as indicated in nodes 3.1–3.4 from Figure 2: test-driven development, automated releases, continuous delivery and time-based releases, respectively. Another key factor for the implementation of rapid releases is the time scale. Nodes 1.1–1.3 present three options as follows: regular cycles, continuous flow and short releases without regular cycles, respectively. Finally, nodes 4.1 and 4.2 from Figure 2 list potential advantages and challenges in the rapid releases adoption.

OSS Projects Reported in the Selected Studies

Table 3 presents a selected list of OSS projects that were identified in the studies analyzed in this SLR, focusing on the implementation of software release practices. These projects may be a reference for companies that plan to adopt rapid releases. In the following paragraph, we describe these projects.

Table 3. Projects-Studies-Releases-Time Scale.

OSS-Project	Study	Release Time	Time Scale
Apache Cocoon (P1)	**S19**	Not Informed in the paper	SRWRC
Apache Tomcat (P2)	**S19**	Not Informed in the paper	CF, SRWRC
ArgoUml (P3)	**S12**	26 weeks	SRWRC
Debian Linux (P4)	**S28**, **S30**	15–18 months	CF, SRWRC
Eclipse (P5)	**S12**	6 weeks	RC, CF
Gnome (P6)	**S25**, **S28**, **S30**	6 months	CF, SRWRC
GNU Compiler (P7)	**S8**, **S28**, **S30**	6 months	CF, SRWRC
Google Chrome (P8)	**S1**, **S8**, **S18**, **S24**, **S29**	6 weeks	RC, CF
Linux Kernel (P9)	**S15**, **S18**, **S28**, **S30**	2–3 months	CF, SRWRC
Mozilla Firefox (P10)	**S1**, **S3**, **S5**, **S7**, **S12**, **S16**, **S17**, **S21**, **S22**, **S23**, **S24**, **S29**	6 weeks	RC, CF
Open Office (P11)	**S28**, **S30**	3–6 months	CF, SRWRC
Plone (P12)	**S28**, **S30**	6 months	CF, SRWRC
Subversion (P13)	**S8**	<=6 months	CF, SRWRC
X.org (P14)	**S28**, **S30**	6 months	CF, SRWRC

Cycle = RC, Continuous Flow = CF, Short Release without Regular Cycle = SRWRC.

The table contains the name of the project, the study that reported its practices and the respective release time adopted by each software project. In the following sentences, we will briefly describe each of these projects. The software project P1 (Apache Cocoon), referenced in **S19**, is a Spring-based framework (since version 2.2 of Cocoon) built around the concepts of separation of concerns and component-based development. Cocoon implements these concepts around the notion of component pipelines, with each component on the pipeline specializing on a particular operation. Software project P2 (Apache Tomcat) is an open source implementation of the Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket technologies. P3 (ArgoUml) is open source uml modelling tool. P4 (Debian Linux) is a free operating system (OS) that is widespread used throughout the world. P5 (Eclipse) is an Integrated Development Environment (IDE) supported by the Eclipse Foundation, an open source community of tools, projects and collaborative working groups. P6 (GNOME) is a Windows-like desktop system that provides a suite of applications on Linux, FreeBSD, IRIX and Solaris. P7 (GNOME Compiler) is a compiler collection that was originally written as the compiler for the GNU (GNU's Not Unix) operating system. P8 (Google Chrome) is a freeware web browser developed by Google. P9 (The Linux kernel) is a Unix-like computer operating system kernel. P10 (Mozilla Firefox) is also a a freeware web browser developed in the context of the Mozilla project. It was well known for its adoption of the rapid release approach against the traditional approach [15]. P11 is a free and open source collection of Office tools developed in the context of the Apache project. P12 (Plone) is an open source content management system built on top of the open source application server Zope and the accompanying Content Management Framework. P13 (Subversion) is an open source version control system developed as a project of the Apache Software Foundation. P14 (X.Org) project provides an open source implementation of the X Window System.

7. Threats to Validity

The following types of validity issues were considered when interpreting the results from this review.

Conclusion validity. There is a risk of bias in the data extraction. This was addressed by 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. One possible threat is selection bias. We addressed this threat during the selection step of the review, i.e., the studies included in this review were identified by means of a thorough selection process comprising multiple stages.

Construct validity. The studies identified in the SLR were accumulated from multiple literature databases covering relevant journals and proceedings. One possible threat is bias in the selection of publications. This is addressed by 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 a strategy for data extraction. Another potential bias is related to the fact that OSS projects were investigated purely by looking at scientific/academic publications. This has the possibility of not drawing a complete picture because not all OSS projects that consider agility in software release practices could have their respective information and experience published in scientific/academic papers. The goal is not to obtain an exhaustive list of OSS projects, but at least the ones that are more representative in the OSS community. Table 3 presents these OSS projects.

8. Conclusions

This paper presented a Systematic Literature Review (SLR) to identify the relevant issues towards the adoption of Rapid Releases in Open Source Software (OSS) projects. Far from being anecdotal, evidence collected and discussed in this SLR has the goal of gaining and sharing insight from the literature so that the OSS community can have more confidence and hence be better able to make a decision on or towards the adoption of Rapid Releases. The primary contribution of this paper is to

reveal the motivations behind the adoption of RR, strategies applied and identification of the potential advantages and difficulties faced in this regard.

Developers of OSS projects are aware that changes to requirements on different abstraction levels can arrive at any moment and this has an impact on specific features of the application. This can affect the source code and consequently the application to be deployed.

The adoption of RR in OSS projects takes into account that these projects usually have users all over the world who eagerly download each new version as soon as it is released, and test it as thoroughly as they can. The global dispersion of users means that the code can be tested 24 h a day [26].

Many OSS projects tend not to be adept at traditional release strategies that deliver a new version of the software based on a set of new features or defect fixes [1]. The traditional release strategy can lead to potential drawbacks such as long, unpredictable release cycles where certain features may never be finished, and, consequently, the release may not take place at all [1].

We have in fact identified these motivations, strategies, advantages and difficulties in other software projects that are not OSS but have the goal to meet customers' expectations. For this reason, many of the lessons learned from OSS projects can also be adopted in other types of software projects regarding the use of rapid releases.

As future work, we are considering using the results of this study to build a meta-model for the mining of open source bases in view of gathering data that lead to assessments of the quality of projects adopting the *Frequent Release* approach.

Author Contributions: Antonio Cesar B. G. da Silva and Glauco de Figueiredo Carneiro together searched for eligible papers from the publication databases and read the eligible papers carefully. Glauco de Figueiredo Carneiro, Miguel P. Monteiro and Fernando Brito e Abreu wrote the paper. All authors have read and approved the final manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

The list of the selected papers of this Systematic Literature Review is provided here as follows.

Table A1. List of selected studies.

ID	Author, Title	Venue	Year
S1	Olga, B., Ian, D., Michael, G., A tale of two browsers.	ICSE	2011
S2	Puneet, A. Continuous SCRUM: agile management of SAAS products.	ISEC	2011
S3	Khomh, F., Dhaliwal, T., Ying, Z., and Adams, B., Do Faster. Release Improve software Quality? An Empirical Case Study of Mozilla Firefox.	MSR	2012
S4	Noureddine Kerzazi and Foutse Khomh, Factors impacting rapid releases: an industrial case study.	ESEM	2014
S5	Thierry Lavoie and Ettore Merlo. How much really changes?: A case study of firefox version evolution using a clone detector.	ICSE	2013
S6	S M Didar Al Alam, S M Shahnewaz, Dietmar Pfahl, Guenther Ruhe. Introduction of continuous delivery in multi-customer project courses.	ICSE	2014
S7	Sandy Clark, Michael Collis, Matt Blaze and Jonathan M. Smith, Moving Targets: Security and Rapid-Release in Firefox.	CCS	2014
S8	Joe F., Shobe, Md Yasser Karim, Motahareh Bahrami Zanjani and Huzefa Kagdi on mapping releases to commits in open source systems.	ICSE	2014
S9	Anil Shankar, Honray Lin, Hans-Frederick Brown and Colson Rice, Rapid Usability Assessment of an Enterprise Application in an Agile Environment with CogTool.	CHI	2015
S10	Johannes Wettinger, Uwe Breitenbucher and Frank Leymann Standards-Based DevOps Automation and Integration Using TOSCA.	UCC	2014
S11	Mehrdad Nurolahzade, Seyed Mehdi Nasehi, Shahedul Huq Khandkar and,Shreya Rawal, The role of patch review in software evolution: an analysis of the mozilla firefox.	FSE	2009

Table A1. *Cont.*

ID	Author, Title	Venue	Year
S12	Daniel Alencar da Costa, Surafel Lemma Abebe, Shane Mcintosh, Uira Kulesza and Ahmed E. Hassan, An Empirical Study of Delays in the Integration of Addressed Issues.	ICSME	2014
S13	Souza, R. , Chavez, C. and Bittencourt, R.A. Do Rapid Releases Affect Bug Reopening? A Case Study of Firefox.	SBES	2014
S14	Conley, C.A. and Sproull, L. Easier,Said than Done: An Empirical Investigation of Software Design and Quality in Open Source Software Development.	HICSS	2009
S15	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.	IET	2009
S16	Hemmati, H. , Zhihan Fang and Mantyla, M.V. Prioritizing Manual Test Cases in Traditional and Rapid Release Environments.	ICST	2015
S17	Souza, R., Chavez, C. and Bittencourt, R.A. Rapid Releases and Patch Backouts: A Software Analytics Approach.	IEEE	2015
S18	Md Tajmilur, Rahman and Peter C. Rigby Release Stabilization on Linux and Chrome.	IEEE	2015
S19	Biffl,S., Sunindyo, W.D. and Moser, T. Semantic Integration of Heterogeneous Data Sources for Monitoring Frequent-Release, Software Projects.	CISIS	2010
S20	Cohan, S. Successful Integration of Agile Development Techniques within DISA.	AGILE	2007
S21	Baysal, O., Kononenko, O., Holmes, R. and Godfrey, M. W., The Secret Life of Patches: A Firefox Case Study.	WCRE	2012
S22	Mantyla M, Adams B, Khomh F, Engstrom E and Petersen K. On rapid releases and software testing: a case study and a semi-systematic, literature review.	ESSE	2015
S23	Souza,R, Chavez C and Bittencourt R. Patch rejection in Firefox: negative reviews, backouts, and issue reopening.	SBES	2015
S24	Adams B, Khomh F, Dhaliwal T and Zou Ying Understanding the impact of rapid releases on software quality.	ESSE	2015
S25	Xiang Li, Yan Fu Li , Min Xie, Szu Hui Ng. Reliability analysis and optimal version-updating for open source software.	GSE	2011
S26	P Rodriguez, et al. Continuous deployment of software intensive products and services: A systematic mapping study.	JSS	2016
S27	MV, Mantyla, K Petersen and, TOA Lehtinen, Time Pressure: A Controlled Experiment of Test Case Development and Requirements, Review.	ICSE	2014
S28	M, Michlmayr, B Fitzgerald and KJ Stol, Why and How Should Open Source Projects Adopt Time-Based Releases?.	IEEE	2015
S29	B Adams, M Michlmayr Modern Release Management in a Nutshell: Why Researchers should Care.	SANER	2016
S30	B Fitzgerald, S McIntosh. Time-Based Release Management in Free/Open Source (FOSS) Projects.	LERO	2011
S31	J Wettinger, U Breitenbücher, O Kopp, F Leymann. Streamlining DevOps automation for Cloud applications using TOSCA as standardized metamodel.	FGCS	2016
S32	P Rodríguez, A Haghighatkhah, L Ellen, S Teppola, T Suomalainen, J Eskeli , T Karvonen , P Kuvaja , J Verner, M Oivo Continuous deployment of software intensive products and services: A systematic mapping study.	JSS	2016
S33	T Karvonen, W Behutiye, M Oivo, P Kuvaja Systematic Literature Review on the Impacts of Agile Release Engineering Practices.	IST	2016

References

1. Michlmayr, M.; Fitzgerald, B.; Stol, K.J. Why and how should open source projects adopt time-based releases? *IEEE Softw.* **2015**, *32*, 55–63.
2. Fitzgerald, B. The transformation of open source software. *MIS Q.* **2006**, *30*, 587–598.
3. Gonzalez-Barahona, J.M.; Izquierdo-Cortazar, D.; Maffulli, S.; Robles, G. Understanding how companies interact with free software communities. *IEEE Softw.* **2013**, *30*, 38–45.

4. Gonzalez-Barahona, J.M.; Robles, G. Trends in Free, Libre, Open Source Software Communities: From Volunteers to Companies. *Inf. Technol.* **2013**, *55*, 173–180.
5. Stol, K.J.; Fitzgerald, B. Inner Source—Adopting Open Source Development Practices in Organizations: A Tutorial. *IEEE Softw.* **2015**, *32*, 60–67.
6. Rigby, P.C.; Cleary, B.; Painchaud, F.; Storey, M.A.; German, D.M. Contemporary peer review in action: Lessons from open source development. *IEEE Softw.* **2012**, *29*, 56–61.
7. Adams, B.; Kavanagh, R.; Hassan, A.E.; German, D.M. An empirical study of integration activities in distributions of open source software. *Empir. Softw. Eng.* **2015**, *21*, 960–1001.
8. Fogel, K. *Producing Open Source Software: How to Run a Successful Free Software Project*; O'Reilly Media Inc.: Sebastopol, CA, USA, 2005.
9. Michlmayr, M.; Fitzgerald, B. Time-based release management in free and open source (FOSS) projects. *Int. J. Open Source Softw. Process.* **2012**, *4*, 1–19.
10. Feitelson, D.G.; Frachtenberg, E.; Beck, K.L. Development and deployment at Facebook. *IEEE Internet Comput.* **2013**, *17*, 8–17.
11. Kitchenham, B.; Charters, S. *Guidelines for Performing Systematic Literature Reviews in Software Engineering, Version 2.3*; EBSE Technical Report; Keele University: Keele, UK; University of Durham: Durham, UK, 2007.
12. Sommerville, I. *Software Engineering*, 10th ed.; Pearson: Harlow, UK, 2015.
13. Rodríguez, P.; Haghighatkhah, A.; Lwakatare, L.E.; Teppola, S.; Suomalainen, T.; Eskeli, J.; Karvonen, T.; Kuvaja, P.; Verner, J.M.; Oivo, M. Continuous deployment of software intensive products and services: A systematic mapping study. *J. Syst. Softw.* **2017**, *123*, 263–291.
14. Beck, K. *Extreme Programming Explained: Embrace Change*; Addison-Wesley Professional: Boston, MA, USA, 2000.
15. Khomh, F.; Dhaliwal, T.; Zou, Y.; Adams, B. Do faster releases improve software quality?: An empirical case study of Mozilla Firefox. In Proceedings of the 9th IEEE Working Conference on Mining Software Repositories, Zurich, Switzerland, 2–9 June 2012; pp. 179–188.
16. Mäntylä, M.V.; Adams, B.; Khomh, F.; Engström, E.; Petersen, K. On rapid releases and software testing: A case study and a semi-systematic literature review. *Empir. Softw. Eng.* **2015**, *20*, 1384–1425.
17. Michlmayr, M.; Hunt, F.; Probert, D. Release management in free software projects: Practices and problems. In *Open Source Development, Adoption and Innovation*; Springer: New York, NY, USA, 2007; pp. 295–300.
18. Mantyla, M.V.; Khomh, F.; Adams, B.; Engstrom, E.; Petersen, K. On rapid releases and software testing. In Proceedings of the 2013 29th IEEE International Conference onSoftware Maintenance (ICSM), Eindhoven, The Netherlands, 28–29 September 2013; pp. 20–29.
19. Brereton, P.; Kitchenham, B.A.; Budgen, D.; Turner, M.; Khalil, M. Lessons from applying the systematic literature review process within the software engineering domain. *J. Syst. Softw.* **2007**, *80*, 571–583.
20. Horne, N.T. Open source software licensing: Using copyright law to encourage free use. *Ga. State Univ. Law Rev.* **2000**, *17*, 863.
21. Madey, G.; Freeh, V.; Tynan, R. The open source software development phenomenon: An analysis based on social network theory. In Proceedings of the AMCIS 2002 Proceedings, Dallas, TX, USA, 9–11 August 2002; p. 247.
22. Chen, L.; Babar, M.A. A systematic review of evaluation of variability management approaches in software product lines. *Inf. Softw. Technol.* **2011**, *53*, 344–362.
23. Zhang, H.; Babar, M.A.; Tell, P. Identifying relevant studies in software engineering. *Inf. Softw. Technol.* **2011**, *53*, 625–637.
24. Dyba, T.; Dingsoyr, T. Empirical studies of agile software development: A systematic review. *Inf. Softw. Technol.* **2008**, *50*, 833–859.
25. Noblit, G.W.; Hare, R.D. *Meta-Ethnography: Synthesizing Qualitative Studies*; SAGE: Thousand Oaks, CA, USA, 1988; Volume 11.
26. Thomas, L.; Schach, S.R.; Heller, G.Z.; Offutt, J. Impact of release intervals on empirical research into software evolution, with application to the maintainability of Linux. *IET Softw.* **2009**, *3*, 58–66.

© 2017 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).