Empirical Software Engineering (2024) 29:97
https://doi.org/10.1007/s10664-024-10482-4

Can instability variations warn developers when open-source projects boost?

Rafael Capilla[1] · **Victor Salamanca**[1] · **Alejandro Valdezate**[2] · **Gregorio Robles**[1]

Accepted: 28 March 2024

Abstract
Although architecture instability has been studied and measured using a variety of metrics, a deeper analysis of which project parts are less stable and how such instability varies over time is still needed. While having more information on architecture instability is, in general, useful for any software development project, it is especially important in Open Source Software (OSS) projects where the supervision of the development process is more difficult to achieve. In particular, we are interested when OSS projects grow from a small controlled environment (i.e., the *cathedral* phase) to a community-driven project (i.e., the *bazaar* phase). In such a transition, the project often explodes in terms of software size and number of contributing developers. Hence, the complexity of the newly added features, and the frequency of the commits and files modified may cause significant variations of the instability of the structure of the classes and packages. Consequently, in this article we analyze the instability in OSS projects, especially during that sensitive phase where they become community-driven. Our results show that instability metrics can be easily obtained in such type of transitions. We also observed from our case studies that instability metrics can help finding out the balance between adding new functionality and performing refactoring. As a conclusions we state that instability metrics offer relevant information in the transition phase from the cathedral to the bazaar.

Keywords Instability · Open source software · Evolution · Software architecture · Cathedral · Bazaar

Communicated by: Jin L.C. Guo and Raula Gaikovina Kula

This article belongs to the Topical Collection: *Registered Reports*

✉ Rafael Capilla
rafael.capilla@urjc.es

Victor Salamanca
victorsalamanza@gmail.com

Alejandro Valdezate
valdezate@gmail.com

Gregorio Robles
gregorio.robles@urjc.es

[1] Universidad Rey Juan Carlos, Mostoles, Madrid, Spain

[2] International University of La Rioja, Logroño, Spain

Published online: 14 June 2024

1 Introduction

Software projects evolve over time to cope with changing requirements and maintenance operations. Le et al. point out that this evolution may cause an architectural mismatch between design and code, leading to architectural drift and erosion when the design diverges from the implementation or when a sub-optimal code violates architectural principles (Le et al. 2016). Typically, software architectural mismatches may impact negatively on the descriptive architecture (e.g., architectural drift) and on the prescriptive architecture (e.g., architectural erosion). This divergence may lead to architectural decay that must be estimated using different metrics and prediction models such as discussed by Garcia et al. (2022).

Today, Open Source Software (OSS) projects are no exception to the instability of the architecture. Given its nature, the risk of architecture instability is often higher as the design is seldom explicit (Brown and Wilson 2011) and the development team is heterogeneous and prone to a high developer turnover (Lin et al. 2017). Even in industrial OSS projects, such as OpenStack, with a highly professionalized development team, having an overall picture is difficult to obtain, as developers come from many different companies, each with their own interests (Teixeira et al. 2016; Zhang et al. 2016). Among the OSS lifecycle, we have identified a phase where architecture instability is a major risk. This happens when projects grow from a small size with a few developers to a community-driven project with hundreds of contributors (Capiluppi and Michlmayr 2007), where the activities follow a self-organized (stigmergic) pattern (Robles et al. 2005). During that transition phase, the automatisms among developers that were possible during the early stages of the project are not possible. And anyhow, becoming a community-driven project is a sign that the project attracts much interest, and that external effort, if conveniently integrated into the project, can set the project in another level (Zhou et al. 2017; Tan et al. 2020). It is in such scenarios where having metrics (and tools) on project instability that point out to risky parts in the source code would be very valuable. Also, developers could be aware of parts of the project that need a further look and possibly action. Additionally, other stakeholders, like external companies willing to invest in a project, would have information on the risk of having high architectural instability.

In this article, we propose to analyze several OSS projects to identify those releases before becoming a community-driven project (i.e., to enter the *bazaar* phase), and to evaluate them for architectural instability. Our aim is to find out what parts exhibit higher instability values hampering its evolution. Our main contribution in this research is to analyze instability evolution trends in OSS projects during the transition from the cathedral to the bazaar and how changes (i.e., bug fixes, refactorings and addition of functionality) may impact on instability variations.

For our study, we have adopted an exploratory case study methodology. The rationale for doing so is that in the registered report (Valdezate et al. 2022) that described the planned study, we specified a set of inclusion criteria (based on threshold values, elaborated in Section 3.1), and thus at the time of applying them we do not know: (i) what projects are going to meet these inclusion criteria, and (ii) what results those projects are going to give in the instability analysis. Thus, having the possibility to analyze these projects in detail (through case study research) will allow to gain more insight into understanding the usefulness of instability metrics.

The structure of this paper is as follows. Section 2 describes our motivation for this research work. Section 3 details the study design, presents the research questions and offers execution plan. In Section 4 we discuss software metrics used to compute instability in classes,

components and packages as related background of our study. Next, Sections 5 and 6 describe our results answering the research questions, and in Section 7 we discuss our findings and implications for practitioners. In addition, we outline the related work in Section 8 and we discuss the threats to validity in Section 9. Finally, we draw conclusions in Section 10.

2 Motivation

The OSS development bazaar model is a collaborative approach to software development in which a large number of people contribute (Mockus et al. 2002; Dinh-Trong and Bieman 2005). It is based on the idea that the best way to create high-quality software is to give everyone the opportunity to contribute. This means that a large community of people is constantly reviewing and improving the software. The bazaar model is also based on the fact that most bugs are found and fixed by the people using the software. Some of the most well-known OSS projects, such as the Linux kernel, the Apache HTTP server, and the MySQL database, follow this model. The bazaar model is often contrasted with the cathedral model, in which the software is developed by a small team of experts.

According to Capiluppi and Michlmayr (2007), OSS projects start in a *cathedral* phase where a small number of developers collaborate to achieve the main goal of the project. In the *cathedral* phase, releases produce small-size software following the *release early* paradigm where developers are invited to publish their software even in its initial stages, offering the first evolution history of the project. In this phase, we can consider that the instability of the releases is limited by the fact that there is a small group of contributors. If the project achieves to attract the interest of other developers and users, and a significant number of developers engage into the project (e.g., to add new functionality), the project ends up in the *bazaar* phase. The software architecture in this phase typically tends to stabilize as the project matures, as it has been reported for long-lived OSS projects (Gonzalez-Barahona 2014). We hypothesize that it is in the transition from the *cathedral* to the *bazaar* phases where instability grows as the result of an increasing number of changes and contributors.

The idea of becoming a bazaar-driven project is very relevant in OSS. Capiluppi and Michlmayr state that a project's success is often related to the number of developers it can attract (Capiluppi and Michlmayr 2007): a larger developer community (the "bazaar") identifies and fixes more software bugs and adds more features through a peer review process. In fact, only successful OSS projects make the transition from a traditional, closed project (the *cathedral*) to a community project (the *bazaar*). According to Senyard et al. it is impossible to launch an OSS project directly in the bazaar phase (Senyard and Michlmayr 2004); in their view, *catehdral* and *bazaar* are not diametrically opposed, as Raymond originally suggested (Raymond 2001), but can be complementary phases within the same OSS project. So, they point out that the initial phase of OSS projects has all the characteristics of cathedral-style development, i.e., the initial phase of an OSS project does not take place in the context of a community of volunteers. Thus, the first phase of developing an initial implementation is carried out by an individual or a small team working in isolation from the community. All features of cathedral-style development (e.g. requirements gathering, design, implementation and testing) are present and executed in the typical cathedral architectural style, i.e., the work is carried out by an individual or a small work team isolated from the community.

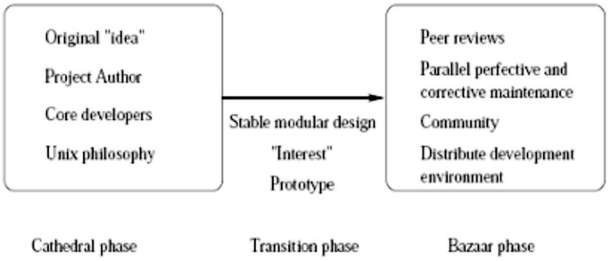

Fig. 1 Transition from the *cathedral* phase to the *bazaar* phase. Taken from Capiluppi and Michlmayr (2007)

In order to become a high quality and useful product, Senyard et al. argue that an OSS project has to make a transition from the cathedral to the bazaar phase (as depicted by the arrow in Fig. 1) (Senyard and Michlmayr 2004).

In this phase, users and developers continuously join the project writing code, submitting patches and correcting bugs. This transition is associated with many complications: it is argued that the majority of OSS projects never leave the cathedral phase and therefore do not access the vast amount of resources of manpower and skills the OSS community offers (Munaiah et al. 2017). In fact, while there are many examples of successful projects in the bazaar phase, most free software projects never leave the cathedral phase and never access the resources of a community of co-developers. A 2002 study by Krishnamurthy found that when examining the 100 most active projects on Sourceforge, a minority of mature OSS projects examined are bazaars (Krishnamurthy 2002). He found that only 19% had more than 10 developers, while 22% had only one developer. Koch examined the entire SourceForge archive and only 1.3% have more than 10 programmers (Koch 2007). We argue that knowing the behavior of the instability over time in a transition phase is of major interest for different stakeholders. For instance, many companies would like to be early promoters of incipient technologies as this can be an important technological advantage for the future; having this information may help these companies to take better, informed decisions on how to allocate their effort or assets.

The transition requires a drastic restructuring of the project, particularly in the way it is managed. The first question, as Senyard et al., is how the code should be distributed (Senyard and Michlmayr 2004. They claim that there are a variety of management styles that can be used in the bazaar phase. However, they all have important characteristics in common. While in the cathedral phase the project is clearly controlled by the project author, during the transition this control should be weakened and responsibility should be transferred to the project community.

The shift between phases is a risky situation, because as projects significantly increase the number of developers, which may produce a loss of control and the appearance of misalignments. For instance, Fig. 2 shows the evolution in number of weekly committers for the Catroid and Hadoop projects, as offered by GitHub. For instance, for Catroid we can observe this shift between the cathedral and bazaar phases in 2011-2012 (although there is a second peak in 2013-2014). For Hadoop, the transition phase can be observed during 2011. Before it, Catroid and Hadoop count with a reduced group of participants (around 5); after it, the number of contributors is always above 20 with peaks well above 50.

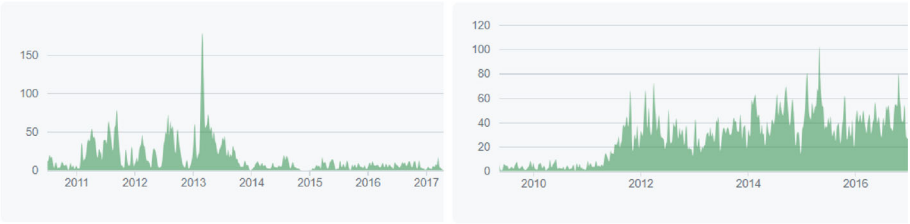

Fig. 2 Evolution of the number of committers (per week) for Catroid (left) and Hadoop (right) projects. Source: GitHub contributors graphs

3 Study Design

To determine whether an OSS project follows the bazaar or cathedral model, several characteristics of the project must be considered. In the past, projects that follow the cathedral model were assumed to (i) have a relatively small centralized development team, (ii) have a well-defined development process, and (iii) release new versions infrequently. In contrast, projects in the bazaar model (i) have a large, distributed development team, (ii) have a less defined or no development process, and (iii) release new versions frequently. It should be noted that nowadays, due to the widespread use of software development platforms such as GitHub that make code easily available, release frequency is no longer considered important as the latest version of the code is available at any time.

Initially, we selected a set of projects randomly to test if the fulfill with the cathedral-bazaar model but we had to exclude some of them because of the complexity and because we didn't find clear transitions from cathedral to bazaar models. Then, we started looking for projects with high popularity and checked their evolution looking for peaks where the number of developers and commits exploded. After, we selected those that have a significant time frame 12 months before and after the peak showing evidences of both the cathedral (6 and 12 months before the peak) and the bazaar (6 and 12 months after the peak) models.

Therefore, for the purposes of our study, we will work with thresholds that clearly delineate projects that are in one phase or another. As noted in the research literature, cathedral projects consist of small teams, particularly those commonly found in traditional industrial settings (Brooks 1995). In this sense, several studies have analyzed the most suitable team size for software development. The number of developers in a software development team can vary greatly depending on the size and complexity of the project. However, research suggests that the ideal team size is between 5 and 10 developers (McConnell 2006; Hoegl 2005; Bhowmik et al. 2015). The results of a study by Rodríguez et al. showed that there are statistical correlations between team size, effort, productivity, and project duration (Rodriguez et al. 2012); projects with an average team size of 9 or more people are less productive than projects below this number. For this reason, we considered projects to be in the cathedral phase if they have fewer than 10 active developers in a given period.

According to previous studies, the number of active developers of a project in the bazaar phase must then be greater than 10 developers (Krishnamurthy 2002; Koch 2007). To avoid time spikes and noisy data and to ensure that a project has reached the bazaar phase, in our research we set the minimum number of developers working on a project at the same time (i.e., in a month) to 50 developers. With this number of developers, we can be sure that the project has reached the bazaar phase.

The transition phase we are interested in must be limited in time, as we want to examine projects that undergo major change in a short period of time. In this way, we avoid projects growing organically and therefore not being affected by the sudden appearance of many developers who want to collaborate on the project. On the other hand, it is known that OSS projects suffer a large turnover in community-oriented projects (Robles and Gonzalez-Barahona 2006). Lin et al. have shown that in OSS industrial projects, developers spend a limited amount of time on projects and that at any given time, 50% of developers are no longer active after two or three years (Lin et al. 2017). Ferreira et al. found that 104 (59.7%) of the 174 projects they analized have an annual turnover of at least 30%, 46 (26.4%) projects have an average annual turnover of 50% and only 10 (5.7%) projects have an average annual turnover of less than 10% (Ferreira et al. 2020). We have therefore limited the transition period to one year.

Regarding instability, we have been inspired for this article by a previous work by Carrillo and Capilla (2018) that describes an instability metric to estimate the ripple effect of design decisions. We will use the formula described by Martin (1994) to compute the instability values of OSS projects. To advance the state of the art, we will investigate if OSS projects exhibit more instability during the transition from the cathedral to the bazaar, and how the changes performed affect the variations of instability. With this aim, we will conduct an exploratory case study (Runeson and Host 2009; Yin 2014) in several OSS projects to uncover the estimation and evolution of instability measures. We will therefore address the following research questions:

- **RQ1. Can we estimate instability variations during the transition from the cathedral to the bazaar in OSS projects ? Rationale:** With this research question, we attempt to provide trends of the evolution of the architectural instability during the transition phase. Thus, we plan to analyze evolution trends of the instability values. We expect that as the number of developers contributing to the project grows, so does first the functionality added to the project and the erosion of the software architecture. We will test this hypothesis by means of statistical test.
- **RQ2. How do new functionality, bugs and refactorings affect the instability in OSS projects when they shift from the *cathedral* to the *bazaar* phase?**
  Rationale: Changes to the project may affect the architectural instability of the project, especially if most of these changes are based on adding and removing classes and relationships between classes. Our hypothesis is that during the transition from the ***cathedral*** to the ***bazaar*** this occurs frequently. Therefore, in this research question we will investigate how such changes affect the instability values during the transition period. We expect i) to see many changes introducing new functionality by novel developers, thus being a source of instability, and ii) to see a lower amount of refactoring that would mitigate the effect of introducing that newer functionality.

Statistical analysis: In order to discover if there is correlation between instability and number of classes and edges, we run a Spearman correlation test with a Python script using the Scipy library. The instability value is the dependent variable, and the number of classes and edges of a snapshot are the independent variables. Hence, we defined following hypotheses:

- H0: There is no significant association between instability and the number of classes and edges of a snapshot.
- H1: There is a significant association between instability and the number of classes and edges of a snapshot.

3.1 Execution Plan

According to the ACM guidelines (Ralph 2021), we will follow an exploratory case study for the experiment design. We will select snapshots of several OSS projects described in the dataset section. To apply the instability metrics, we will compute the instability of each project at the class level and their dependencies. Hence, we will adopt following protocol:

1. We will select a set of OSS projects where we can identify a transition from the *cathedral* to the *bazaar* (Raymond 2001). Thus, we have to define three aspects: i) identify the cathedral phase, ii) identify the bazaar phase, and iii) specify the time interval for the change.
   - As for i) and ii), we have looked at the scientific literature for any type of definition in this regard, but have not found anything. Our position is that this can be done merely based on the number of committers in a given time period. As this has not been previously researched, we propose two tentative numbers that we find reasonable at this point: we expect for the cathedral phase less than 10 committers in a month, while for the bazaar phase it should be more than 50 committers in a month.
   - As for iii), we think that a reasonable time span is in the range of 6 to 12 months from a specific reference date. We will therefore start looking for projects where conditions i) and ii) apply in 12 months, using a sliding window algorithm.
   - To offer some visual evidence of our decision, we have taken two projects as examples. Figure 2 shows the evolution of committers (on a weekly basis) of the Catroid and Hadoop projects, respectively, as taken from GitHub. We can observe that in both cases around 2011 there is a transition phase between the cathedral and the bazaar. We also can observe how in the case of Hadoop the high activity has been maintained since then, while for Catroid there is more variance.
2. We will analyze the instability for different snapshots of several OSS projects to investigate the differences of instability values when new functionality is added. For this aim we will perform the following sub-steps:
   (a) According to the cathedral and bazaar phases, we will select those periods where we observe a significant activity of developers.
   (b) We will use the Scitools Understand[1] software to obtain the dependencies between the classes of each of the releases selected. Scitools Understand is a static analysis code tool for Java projects that is commonly utilized in industry (among others by NASA, Toyota, Amazon) and has been frequently used in the research literature, e.g., (Moore et al. 2016; Kim 2017; Gupta et al. 2021; Malhotra et al. 2016; Benkoczi et al. 2020; Zhang et al. 2013).
   (c) We will transform the data obtained into a format that can be read by an algorithm we developed to compute the instability values of the snapshots.
3. We will apply a statistical analysis of the results using the Spearman correlation test to find if there is correlation between the instability values and the number of classes and dependencies added.

Deviations from the registered report Here we listed some changes from the original execution plan (Valdezate et al. 2022) but without affecting to the research questions and results as well:

[1] https://www.scitools.com/

1. We used Scitools Understand instead of the ARCADE[2] tool (Laser et al. 2020) to compute the number of classes and edges, because support for ARCADE has been discontinued. In addition, Scitools Understand does not require to compile the projects, while ARCADE needs to do it. We did a sanity check between the tools with an older analysis that we had done with ARCADE and the measures that Scitools Understand offers are equivalent.
2. The time span was finally set 6 and 12 months before and after a reference date from the transition from the cathedral to the bazaar. For each project we have taken a date (which we call the *reference date*) approximately in the middle of the transition phase from the cathedral to the bazaar. In addition to the instability values for the snapshot of the project at the reference date, we have analyzed snapshots of the repository 6 and 12 months before (in the cathedral phase) and 6 and 12 months after (in the bazaar phase) the reference date, having in total 5 points for each project, two before and two after the reference date.
3. To compute instability, we used snapshots of the git repository at a given point in time (i.e., the status of a project at a given commit) instead of releases, because this offered several advantages. First, it allowed to find a snapshot that is close to the time we are looking for, because such projects have many commits daily, but releases are more separated in time. Second, having exactly the same time window for different projects enables to perform comparisons among them.
4. The ratios of modified files to predict instability changes are no longer needed, as we have other measures (such as changes performed) that describe better what is happening during the transition from the cathedral to the bazaar.
5. We have been unable to find developers of the selected projects available to be interviewed. After several contacts, we did not obtain positive answers. The only two developers who answered just stated they were too busy or not interested.

3.2 Tools

We will use Scitools Understand to compute the number of classes and dependencies between classes (i.e., edges) as these numbers are needed to compute the instability values using our own algorithm is based on Martin's formula. Scitools Understand is a customizable integrated development environment (IDE) that allows the analysis of static code through a variety of visual, documentation and metric tools. With Scitools Understand we can generate information reports on the dependencies between classes that we will use later to obtain the instability of a project. With Scitools Understand we do not need to compile the OSS project.

3.3 Dataset

To investigate the instability in real OSS projects,[3] we will select OSS projects that comply with the following criteria:

1. Have a repository in GitHub and are not forks
2. Are written in Java
3. We can identify a transition phase between the cathedral and bazaar phases
4. Can be analyzed using Scitools Understand

[2] https://bitbucket.org/joshuaga/arcade

[3] The datasets generated during and/or analysed during the current study are available from the corresponding author on reasonable request.

The projects considered in our analysis are Apache Kafka,[4] Jenkins,[5] Google Guava,[6] Apache Dubbo,[7] and Apache PDFBox.[8]

We have selected these five open-source projects as they fulfill the criteria set for the cathedral-bazaar transition phase. Finding projects that fulfilled our criteria was not as easy as we thought in advance, so we had to try several approaches to reach that number.

Our initial idea was to find those projects from the most active Java projects on GitHub. We naively thought that most of the projects that become very relevant (i.e., had a high number of stars (Borges and Valente 2018)) would have such a transition phase. We therefore searched for lists of very successful Java projects on GitHub and chose the one published by IssueHunt, a funding platform for OSS projects, in the well-known Medium online publishing platform entitled "50 Top Java Projects on GitHub".[9] However, only three (Jenkins, Guava and Dubbo) of the listed 50 projects fulfilled our criteria, as most of the projects in the list never achieved a high number of contributors.

Our next step consisted in mining Boa (Dyer et al. 2015), the ultra-large-scale software repository and source-code mining. We therefore wrote a query in the domain-specific language of Boa asking for Java projects by number of committers.[10] The output contained 14.64 million projects, which we ordered in decreasing order by means of a simple shell command using `sort`. We then inspected the projects in order. After 250 projects, we had not found any candidate that fulfilled our criteria. Even though these projects had more than 150 contributors, they either did not show a clear transition phase, or stopped abruptly.

Considering the projects that we had identified already in the first step, we saw that two of them are part of Apache. As we were interested in having case studies rather than a representative sample of Java projects, we thought it would be a good idea to search for the remaining two projects among the ones under the Apache Software Foundation umbrella. This is because the ASF has a special program where projects can be nurtured, in fact moving from a cathedral-style development to a bazaar-like one. This is supported by previous research by Yang et al., who report that "more than half of the projects [in their sample of 292] tr[ied] to join the ASF with motivations related to fostering a community, strengthening the project's outcome, increasing interactions with other OSS projects in the ASF, and boosting technical development" (Yang et al. 2022), which is in line with what we intend to investigate. Thus, we searched the Apache Software Foundation page on GitHub, which contains more than 2,400 repositories,[11] and considered projects in the list ordered by relevance (i.e., number of stars) until we got the two remaining ones: Kafka and PDFBox.

The selected projects are briefly presented next.

3.3.1 Kafka

Apache Kafka is an OSS message brokering project developed by LinkedIn and donated to the Apache Software Foundation. The project aims to provide a unified, high-performance, and low-latency platform for real-time manipulation of data sources. It can be seen as a

[4] https://github.com/apache/kafka

[5] https://github.com/jenkinsci/jenkins

[6] https://github.com/google/guava

[7] https://github.com/apache/dubbo

[8] https://github.com/apache/pdfbox

[9] https://medium.com/issuehunt/50-top-java-projects-on-github-adbfe9f67dbc

[10] https://boa.cs.iastate.edu/boa/?q=boa/job/public/102538

[11] https://github.com/apache

massively scalable publish-subscribe message queue conceived as a distributed transaction log, which makes it attractive for enterprise application infrastructures.

3.3.2 Jenkins

Jenkins is an OSS automation server written in Java. It is based on the Hudson project and is, depending on the vision, a fork of the project or simply a name change. Jenkins helps automate part of the software development process through continuous integration and facilitates certain aspects of continuous delivery. It supports version control tools such as CVS, Subversion, Git, Mercurial, Perforce, and Clearcase, and can run Apache Ant and Apache Maven-based projects, as well as Windows batch programs and console scripts.

3.3.3 Guava

Guava is a set of OSS common libraries for Java, developed primarily by Google engineers. It includes new collection types (such as multimap and multiset), immutable collections, a graph library, and utilities for concurrency, I/O, hashing, caching, primitives, strings, among others. It is widely used on most Java projects within Google, and widely used by many other companies as well.

3.3.4 Dubbo

Dubbo is high-performance, lightweight, Java-based RPC framework. It was donated to the Apache Foundation by Alibaba, and offers an easy-to-use, high-performance WEB and RPC framework with builtin service discovery, traffic management, observability, security features, tools and best practices for building enterprise-level microservices.

3.3.5 PDFBox

Apache PDFBox is an OSS pure-Java library that can be used to create, render, print, split, merge, alter, verify and extract text and meta-data of PDF files.

4 Instability Metrics

In the following subsections, we detail and compare different approaches to compute the instability in software systems and for different kinds of artifacts. We group them accordingly to the type of element to which an instability formula is applied. We provide a comprehensive comparison of the metrics discussed across three subsections to clarify the evolution and applicability of each metric. In our research, we employed one of the instability metrics introduced in the literature to calculate the instability of key elements in the five selected open-source projects.

4.1 Instability Metrics in Packages

Software architecture packages are high-level entities commonly used to describe subsystems or to group related functionality. In many complex systems, some packages depend on

others. For instance, the Linux system often requires additional packages when installing and configuring new functionality. As a consequence, a set of dependencies is established between packages and the modifications in one of these packages affect other related packages. Alves et al. perform a comparison of code querying languages and tools based on an implementation of the instability metric defined by Martin (1994), and on the number of classes outside the package that depends on classes inside the package (i.e., AfferentCoupling) and on the number of classes inside the package that depend upon classes outside the package (i.e., EfferentCoupling) (Alves et al. 2011). One early experience analyzing the architectural instability of Eclipse releases following Martin's formula is discussed by Wermelinger et al. (2011), where the authors investigate the evolution of instability variations according to the dependencies that violate the Stability Dependency Principle (SDP).

Alenezi and Khellah (2015) suggest ways to estimate package stability metrics to measure the changes affecting the stability of the architecture and measure the changes that happen during system evolution. The authors estimate the instability of two consecutive releases due to changes in the packages and they provide an aggregate measure of the system instability as the average of the sum of the package instabilities. In addition, Baig et al. (2019) suggest a package stability metric (PSM) based on the changes between package contents and the relationships inside the package. The proposed metric estimates the maintenance effort and computes package stability based on three dimensions: content, internal package connection, and external package connections, as well as eight properties (Alshayeb et al. 2011) and four types of relationships between classes. More recently, Fontana et al. (2017) mention that a package is less stable if it depends on an unstable related package, and they suggest a metric so-called "Degree of Unstable Dependency" as the ratio between the number of dependencies that makes a package unstable (i.e., BadDependency) and the total number of dependencies.

Table 1 summarizes the metrics discussed above. Finally, Baig et al. (2019) suggest a new package stability metric based on the changes between package contents and intra- and inter-package connections that validate empirically in five open-source programs. The authors found a negative correlation between the proposed metric and the maintenance effort and a positive correlation between the package stability metrics based on changes in lines of code and class names (Baig et al. 2019).

Table 1 Overview of instability metrics for software packages

Authors	Instability metric	Instability formula		Instability between versions
Alves et al. (2011)	Package Instability	$I = \frac{EfferentCoupling}{AfferentCoupling+EfferentCoupling}$	(1)	No
Alshayeb et al. (2011)	Property Stability	$Stab_{Property} = \frac{Unchanged_{Property}}{Number_{Property}}$	(2)	No
Alenezi and Khellah (2015)	Aggregate System Instability Change	$ASIC(v) = \frac{\frac{1}{K}\sum_p (I_{v-1}-I_v)_p + \frac{1}{N}\sum_p I_p}{2}$	(3)	Yes
Fontana et al. (2017)	Degree of Unstable Dependency	$DoUD = \frac{BadDependencies}{TotalDependencies}$	(4)	No
Baig et al. (2019)	Package Stability Metrics	$PSM = \frac{PCS+IPIS+EPIS}{3}$	(5)	No

4.2 Instability Metrics in Components

With respect to the metrics that estimate the instability in software components, we can highlight several approaches. Chawla et al. propose a new quality model (SQMMA) to compute the stability of a software component using a weighted formula in terms of the number of subclasses, depth of tree hierarchies, the coupling between objects, components invoked, and entry/exit points (Chawla and Chhabra 2015). The weights for each parameter are assigned using the Analytic Hierarchy Process (AHP) method. In addition, other quality attributes, like modifiability, are computed in terms of stability assigned to a specific weight. The authors compare their results of the stability values with the defect density, so more stable components exhibit less change density.

Works like the one by Threm et al. analyze evolutionary instability metrics between two different versions by measuring the distance between software components (Threm et al. 2015). The authors define version stability and branch stability metrics using distances and they compute the structural stability of an artifact for a given period. They also provide an aggregate stability indicator to measure the entire evolution of an artifact.

Aversano et al. (2018) suggest a family of instability metrics for measuring the evolution of the architecture instability across multiple releases. They investigate different factors influencing the instability of core components as an indicator of good reusability. To analyze this instability, they propose the so-called Design Instability (DI) metric, aimed to evaluate the changes performed on the architecture between two releases. The proposed formula described in Table 2 includes the components that have changed (c_Comp), added (a_Comp), or removed (r_Comp) in version N+1 with respect to version N and according to the total number of components (n_Comp) in version N. The metric suggests the so-called Calls Instability (CI) index as the changes of the interactions between the software components in release N with respect to release N+1. The summary of these metrics is shown in Table 2.

4.3 Instability Metrics in Classes

One of the seminal works suggesting the idea of stability in object-oriented (OO) design was authored by Martin (1994), who stated a way to measure the instability between OO classes as *"the number of classes inside a particular category that depends upon classes outside this category (i.e., efferent coupling) and divided by the sum of the efferent coupling plus the number of classes outside a category that depend upon classes within this category (i.e., afferent coupling)"*.

Table 2 Overview of instability metrics for components

Authors	Instability metric	Instability formula		Instability between versions
Chawla and Chhabra (2015)	Stability	$Stability = -0.19 * Subclasses - 0.21 * Coupling - 0.20 * Hierarchies - 0.18 * EntExt - 0.21 * Communication$		No
Threm et al. (2015)	Version Stability	$VS(m) = 1 - \frac{\sum_{i=1}^{p} NCD(S_m, S_n)}{p}$	(6)	Yes
Aversano et al. (2018)	Calls Instability	$CI = \frac{a\_Iter_{N+1} + r\_Iter_{N+1}}{t\_Iter_N + a\_Iter_{N+1} + a\_Iter_{N+1}}$	(7)	Yes

Li et al. (2000) suggest three different instability metrics named System Design Instability (SDI), Class Implementation Instability (CII), and System Implementation Instability (SII), which are used to estimate the evolution of OO systems via the analysis of the instability of object-oriented designs and the classes that have changed. Regarding the SDI metric, the authors conclude that the instability of the project examined is higher in the early stages of the development phase. In addition, Ratiu et al. (2004) consider a class is stable with respect to a measurement of a previous version if there is no change in that measurement. They provide a formula to estimate the stability of classes applied to a class history, which is computed as the fraction of the number of versions in which a class changes over the total number of versions.

A complementary work, described in Alshayeb and Li (2005), refines and extends Li et al.'s work (Li et al. 2000) in order to incorporate the number of inheritances of classes that have changed between two consecutive versions. Another approach, by Alshayeb et al. (2011) describe a metric to estimate class instability by considering the class relationships and class design. The authors identified class stability factors and they came up with eight class properties used to measure class stability, dividing the unchanged properties by the total number of properties. The authors validated empirically the proposed formula in two industrial client-server systems. The results provided a more accurate estimator of class stability than previous approaches.

Finally, Ampatzoglou et al. (2015) suggested a way to predict class instability in GoF design patterns using a probability formula that relies on an estimation of the ripple effect measure (REM) computed using the number of method calls between classes, the number of methods, and attributes, and the number of polymorphic methods. In this way, the coupling and cohesion between classes participating in instability estimation are major factors to correlate the instability between classes for a given design pattern. The summary of the instability metrics for classes is shown in Table 3.

5 Results About Instability Variations (RQ1)

In this section, we report the results alongside our observations for RQ1 (Can we estimate instability variations during the transition from the cathedral to the bazaar in OSS projects?). In both research questions, we computed the instability of classes and dependencies for each project releases using Martin's formula (Martin 1994) but we provide a finer-grained analysis rather than solely computing the instability of the overall project releases. In this way, we calculate the instability of the classes and dependencies added and removed between releases (RQ1) as well as the instability of new and removed functionality (RQ2) for each project snapshot. In addition, we did not use the instability metrics from Section 4 referred to components and packages because these do not offer the required precision to analyze the classes and dependencies of the projects selected. Also, from the instability metrics discussed in Table 3, Martin's formula was proven the best choice while others did not suit to our research goals or exhibit a certain divergence in the results indicated by the authors.

5.1 Instability Variations of Committers

This subsection describes the results of computing the instability variations of the project releases when they transition from cathedral to bazaar models. We provide this information as we need to prove that there are significant instability variations in the data between releases

Table 3 Over view of instability metrics for classes

Authors	Instability metric	Instability formula		Instability between versions
Martin (1994)	Instability	$I = \frac{Ce}{Ce+Ca}$	(8)	No
Li et al. (2000)	Class Implementation Instability	$CII = \frac{LOC_{N+1} - LOC_N}{LOC_N} * 100$	(9)	Yes
Ratiu et al. (2004)	Stability	$Stabi_{1..n}(C, M) = \frac{\sum_{i=2}^{n} Stabi_i(C,M)}{n-1}$		Yes
		$Stabi_i(C, M) = \begin{cases} 1, & M_i(C) - Mi - 1 = 0 \\ 0, & M_i(C) - Mi - 1 \neq 0 \end{cases}$		
Alshayeb et al. (2011)	Stability Class	$Stability_{CLASS} = \frac{Stab_{ClassAL} + Stab_{Inteface}}{properties_{CLASS}}$		No
		$+ \frac{Stab_{Inhr} + Stab_{Mthd}}{properties_{CLASS}}$		
		$+ \frac{Stab_{Var} + Stab_{varAl}}{properties_{CLASS}}$		
		$+ \frac{Stab_{MthdAL} + Stab_{Body}}{properties_{CLASS}}$		
Ampatzoglou et al. (2015)	Ripple Effects Measurement	$REM = \frac{NDMC + NOP + NPrA}{NOM + NA}$	(10)	No

and that the snapshots are chosen adequately in order to demonstrate there is a transition from cathedral to bazaar based on how much instability is in the classes. This implies significant activity during along the evolution of the project for a certain period of time.

In the following tables we show the results computing the instability of a selection of snapshots of the aforementioned projects. We consider a snapshot the status of the project in its git repository at a given date (i.e., we checkout the repository for that date, e.g., November 1st 2015 for Kafka). For each project we computed the overall instability of the snapshot (third column) and the instabilities of the artifacts added and removed between two snapshots (except for the first snapshot) in order to show how much instability appears during refactoring or when new functionality between two consecutive snapshots (fourth and fifth columns) is added. The last two columns show the number of classes and edges computed using *Scitools Understand*, as we need this information to compute the instability of the snapshots. In all the projects analyzed we find that the instability starts to increase in the transition phase; as prior versions exhibit lower instability.

Kafka and Jenkins Both projects exhibit a transition phase as the first two snapshots (i.e., first two rows in Table 4 for Kafka and Table 5 for Jenkins) have lower instability values than the reference date (i.e., third row). Additionally, in both projects the instability values shown 6 and 12 months after the reference date grow, as more complexity has been added to the project (reflected by higher values of the number of classes and edges).

As we can observe in Fig. 3, the left side shows the activity of the committers of Kafka while the right side shows the activity in Jenkins. The solid black lines indicate the reference date in both projects, while the blue and orange lines show the snapshots during the transition phase (as already said, two before and two after). As both projects exhibit similar patterns: the instability is lower before the reference date (i.e., in the cathedral phase) and it grows after it (during the bazaar phase) when both projects exploit and increase their functionality is confirmed by an increase in the number of classes.

Table 4 Instability of Kafka snapshots

Commit	Year	Instability of the snapshot	Instability of removed classes and dependencies	Instability of new classes and dependencies	Number of classes	Number of edges
ed5d7cee	Nov-14	0.5408	—	—	284	862
9e71ba41	May-15	0.5521	0.0655	0.1598	334	1,195
f851d258	**Nov-15**	0.6137	0.0642	0.47	1,030	4,096
0fdaf497	May-16	0.6288	0.2780	0.4130	1,518	6,488
dbb0a2e0	Nov-16	0.6517	0.0552	0.1774	1,843	8,116

The bold numbers indicate the reference date of the releases to set the period of time chosing project snapshots

Guava Trend shown in Table 6, the behavior of the instability values is similar to Kafka and Jenkins, but with a small decrement in the fourth snapshot (six months after the reference date). However, the decrease is so small, that we can consider these results and the instability behaves in a similar way to the two previous projects.

Dubbo Results shown in Table 7 exhibit a different pattern of the instability values compared to the three previous projects. As reference date we chose December 2018. The snapshots in the bazaar phase exhibit slightly lower instability values than the snapshots in the cathedral phase. We also see that the number of classes did not increase much, while the number of dependencies (i.e., edges) grew significantly. This is because, according to Martin's instability formula, a bigger number in the sum of afferent and efferent dependencies between classes leads to lower instability values. Nevertheless, the instability values grow again after December 2018 and the project showed an increasing activity in the number of classes (i.e., new functionality) and dependencies.

Figure 4 shows the reference date chosen in black, and the activity of the committers for the first two snapshots shown in blue lines. A lower activity in the last two snapshots can be seen from the lines in orange. While the number of classes increases when the project moves from the cathedral to the bazaar, as reflected in a significant increase of functionality (new classes varying from 2,082 to 2,871) and dependencies between classes (i.e., from 7,430 to 10,492), the instability grows in parallel with this activity.

PDFBox Instabilities shown in Table 8 have a different trend from all previous cases. The instability values of the first two snapshots (in the cathedral phase) are slightly higher than the for the snapshot in the reference date (January 2015), exhibiting a similar pattern to Dubbo. However, the two subsequent snapshots continue with the decreasing trend of instability.

Table 5 Instability of Jenkins snapshots

Commit	Year	Instability of the snapshot	Instability of removed classes and dependencies	Instability of new classes and dependencies	Number of classes	Number of edges
e0d5fcd5	Oct-10	0.6870	—	—	2,123	9,401
d3c4c750	Apr-11	0.6949	0.0207	0.0788	2,287	10,267
4c1d0aa0	**Oct-11**	0.7024	0.0848	0.0838	2,273	10,173
069070b6	Apr-12	0.7080	0.0275	0.0823	2,468	10,974
2b8d7813	Oct-12	0.7104	0.0280	0.0714	2,623	11,500

The bold numbers indicate the reference date of the releases to set the period of time chosing project snapshots

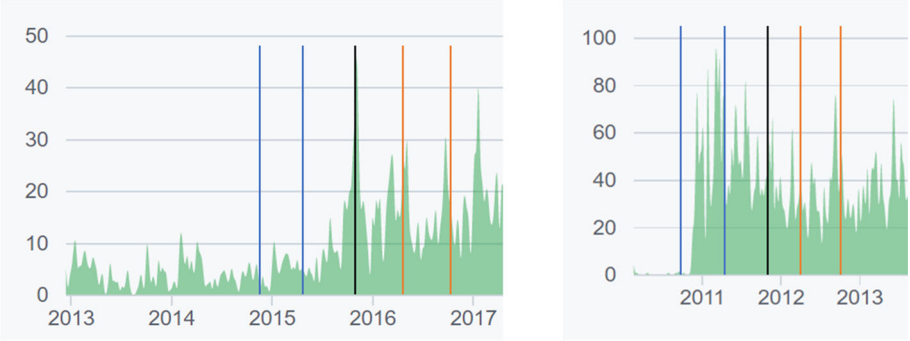

Fig. 3 Activity of committers of Kafka and Jenkins projects. The vertical lines show the periods of the snapshots where we analyzed the instability of the releases. The black line represents the starting date for the first snapshot while the blue and orange lines represent the snapshots 6 and 12 months before and after respectively

Table 6 Instability of Guava snapshots

Commit	Year	Instability of the snapshot	Instability of removed classes and dependencies	Instability of new classes and dependencies	Number of classes	Number of edges
adf2ce79	Oct-11	0.7359	—	—	3,122	12,062
a351697f	Apr-12	0.7487	0.1292	0.2979	4,158	15,269
d63bcfad	**Oct-12**	0.7603	0.0999	0.2149	5,083	18,164
88ec3b70	Apr-13	0.7599	0.1032	0.1565	5,549	20,221
72a9270f	Oct-13	0.7638	0.0431	0.0983	6,311	22,652

The bold numbers indicate the reference date of the releases to set the period of time chosing project snapshots

Table 7 Instability of Dubbo snapshots

Commit	Year	Instability of the snapshot	Instability of removed classes and dependencies	Instability of new classes and dependencies	Number of classes	Number of edges
23732dc5	Dec-17	0.6827		—	1,856	6,545
563347be	Jun-18	0.6750	0.1277	0.1461	2,000	6,737
d708271a	**Dec-18**	0.6695	0.6581	0.6343	2082	7,430
be501691	Jun-19	0.6780	0.0537	0.1719	2532	9,245
91e339fe	Dec-19	0.6722	0.0799	0.1414	2871	10,492

The bold numbers indicate the reference date of the releases to set the period of time chosing project snapshots

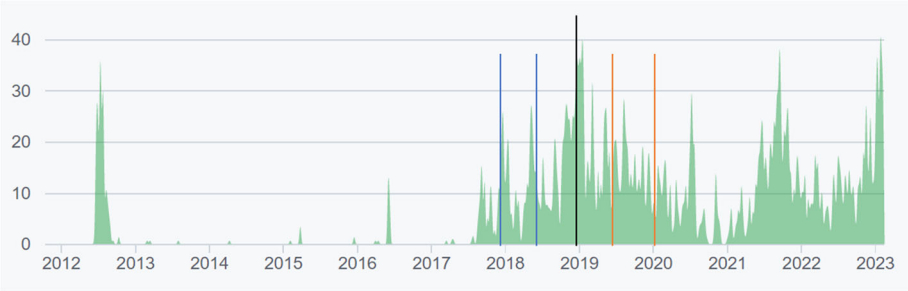

Fig. 4 Activity of committers of the Dubbo project

In a similar vein, we show in Fig. 5 the time-frame selected for the cathedral-bazaar transition. The blue lines display the activity before the reference date and the orange lines show the activity after it. We can see that in this second phase the project activity tends to stabilize, and its instability as well. In this project the number of classes and dependencies grows to a small extent. What we can observe is that, even when there is an increase in the number of committers, the instability values remain almost stable and decreasing a bit. This hints to possible activities by the project to maintain the architecture, for instance by performing refactoring; we can verify it in RQ2 when we consider the different type of changes carried out.

5.2 Statistical Results

We show the results of the Spearman correlation test and the p-values in Table 9. Spearman's correlation test is a valuable tool for analyzing monotonic relationships between variables when the assumptions of parametric tests are violated or when the data is not normally distributed, skewed, or contains outliers. It is also suitable for analyzing ordinal data and provides an easy-to-interpret correlation coefficient. The effect size is a quantitative measure of strength of a phenomenon (in our case, the strength of a relationship). In the case of the Spearman correlation, the correlation factor is itself a measure of the effect size being 1 a perfect (positive, and -1 a negative) relationship while 0 would be no relationship at all. A monotonic relationship is a relationship that does one of the following: (1) as the value of one variable increases, so does the value of the other variable; or (2) as the value of one variable increases, the other variable value decreases.

Table 8 Instability of PDFBox snapshots

Commit	Year	Instability of the snapshot	Instability of removed classes and dependencies	Instability of new classes and dependencies	Number of classes	Number of edges
89f7d669	Jan-14	0.6511	—	—	1,101	5,596
e74493eb	Jul-14	0.6553	0.1164	0.1393	1,143	5,604
7fcc7236	**Jan-15**	0.6435	0.1047	0.1087	1,164	5,868
73c9f556	Jul-15	0.6379	0.0475	0.1012	1,278	6,367
882d7646	Jan-16	0.6357	0.0477	0.0817	1,363	6,795

The bold numbers indicate the reference date of the releases to set the period of time chosing project snapshots

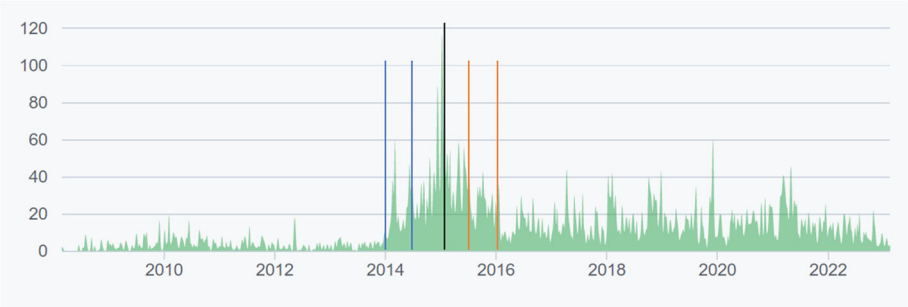

Fig. 5 Activity of committers of the PDFBox project

We found a high and strong positive correlation between the instability value and the number of classes and dependencies in four of the analyzed projects projects ($p<0.05$), all except for Dubbo. Three projects show a positive correlation (i.e., Kafka, Jenkins and Guava), while two (e.g., Dubbo and PDFBox) have a negative correlation. The negative correlation might be due to architecture maintenance activities (i.e., refactoring tasks in order to reduce the instability); we will further study that question in RQ2.

5.3 Summary and Take-Away Messages

We found that instability values provide a richer amount of information than the number of new classes and edges. In all projects under study, new functionality has been added, resulting in a higher number of classes and more edges among them. For four of the five projects we have seen that the instability correlates strongly with classes and edges. However, in the other two projects, we do not find a significant correlation (Dubbo) or the correlation that we find is in the opposite direction (PDFBox). This may be because the projects have made specific efforts to maintain instability under control, for instance by devoting more effort to refactorings. RQ2 will verify this hypothesis. Nevertheless, we learned from RQ1 that the instability metrics capture better the architectural risk than just considering new functionality or any other of its proxies (new classes, new edges, or added LOC). Some take-away messages are as follows:

– Instability variations prove in some cases significant activity of committers which alongside with the graphs and for specific periods indicate evidences of cathedral to bazaar transitions. Therefore, developers can focus on selected periods in the transitions to analyze better how instability varies.

Table 9 Spearman correlation between instability values and classes and dependencies

Projects	Spearman correlation	p-value
Kafka	0.999	**1.404e-24**
Jenkins	0.899	**0.037**
Guava	0.899	**0.037**
Dubbo	-0.499	0.391
PDFBox	-0.899	**0.037**

The bold numbers indicate the significant values of the statistical results being p-value less than 0.05

Table 10 Instability of new and removed functionality of Kafka snapshots

Snapshot	Year	Instability of removed classes and dependencies	Instability of new classes and dependencies	Variation of instability	Number of classes	New classes between snapshots
ed5d7cee	Nov-14	—	—	—	284	—
9e71ba41	May-15	0.0655	0.1598	0.024	334	50
f851d258	**Nov-15**	0.0642	0.47	0.3176	1,030	696
0fdaf497	May-16	0.2780	0.4130	0.1328	1,518	488
dbb0a2e0	Nov-16	0.0552	0.1774	0.0312	1,843	325

The bold numbers indicate the reference date of the releases to set the period of time chosing project snapshots

- As not all projects behave the same for given time periods, it cannot be proven –at least in some cases– that the instability variations imply a transition from the cathedral to the bazaar model even if this could be true.
- The correlation between instability values and the classes and dependencies could be either positive or negative. In the case of positive correlation we should understand the project is start exploiting as a consequence of a significant activity of developers, mainly adding new functionality. However, in the case of negative correlation, the activity could be more concerned with refactoring tasks in order to reduce the instability of the classes.
- Instability variations seem to be a good indicator in most cases to show the activity of a project according to the different tasks of the committers so we can analyze different trends in their activity along the evolution of each project.

6 Results About Instability Impacted by New Functionality, Bugs, and Refactoring (RQ2)

In a similar vein as in the previous section, we discuss here how the instability variations are affected by new functionality, bugs, and refactoring operations. Regarding the new functionality, it is interesting to observe how the new functionality impacts the instability of project when a project explodes to a bazaar model. With regard to the case of fixing bugs and perform significant refactoring operations, the instability of the project is affected due to the classes that change when errors must be fixed or when a project demands a significant reorganization of their classes. In these last two cases the variation in the instability values may suggest the importance of such activities (Tables 10 and 11).

Table 11 Instability of new and removed functionality of Jenkins snapshots

Snapshot	Year	Instability of removed classes and dependencies	Instability of new classes and dependencies	Variation of instability	Number of classes	New classes between snapshots
e0d5fcd5	Oct-10	—	—	—	2,123	—
d3c4c750	Apr-11	0.0207	0.0788	0.0057	2,287	164
4c1d0aa0	**Oct-11**	0.0848	0.0838	0	2,273	0
069070b6	Apr-12	0.0275	0.0823	0.0065	2,468	195
2b8d7813	Oct-12	0.0280	0.0714	0.0042	2,623	155

Note that as the variation of instability for the third snapshot is negative, it has been set to zero
The bold numbers indicate the reference date of the releases to set the period of time chosing project snapshots

6.1 Instability of New Functionality

In this section we discuss our results to uncover how the addition of new functionality, and the activity on bug fixing and refactoring may affect instability of the projects analyzed.

Impact of New Functionality According to the results in RQ1 regarding the instability computed for the new classes and dependencies (fifth column in the tables), we observe that the variations of the instability (VoI) between two consecutive snapshots can be computed as the instability of the new elements added divided by the increment of elements between two consecutive snapshots, such as as described in he following equation:

$$VoI = \frac{NewElements * InstabilityNewElements}{IncrementElementsSnapshots} \tag{11}$$

Regarding the results of the tables above, we show in Fig. 6 the trends of the instability variations between snapshots of new classes and dependencies. As can be observed, the orange line in the graphics in the right is very similar to the instability trend shown in the orange line in the graphics in the left. Therefore, estimating the trend of new classes and dependencies between snapshots, we can somehow estimate what the instability trend would be.

Regarding the Guava project, results are shown in Table 12 and Fig. 7. Guava shows a similar trend to Jenkins regarding new functionality. Nevertheless, with respect to the evolution of instability, the trend decreases abruptly in a similar way as in Kafka; the curve shows the evolution of the increment of instability values between snapshots to be a bit different than in Kafka and Jenkins, as it exhibits a decreasing pattern until it stabilizes.

In a similar vein, we provide the results of the other two projects, shown in Table 13 (Dubbo) and Table 14 (PDFBox).

According to the results, shown in Fig. 8, Dubbo exhibits a growth of new functionality caused by more classes and dependencies between some snapshots, and the instability of these

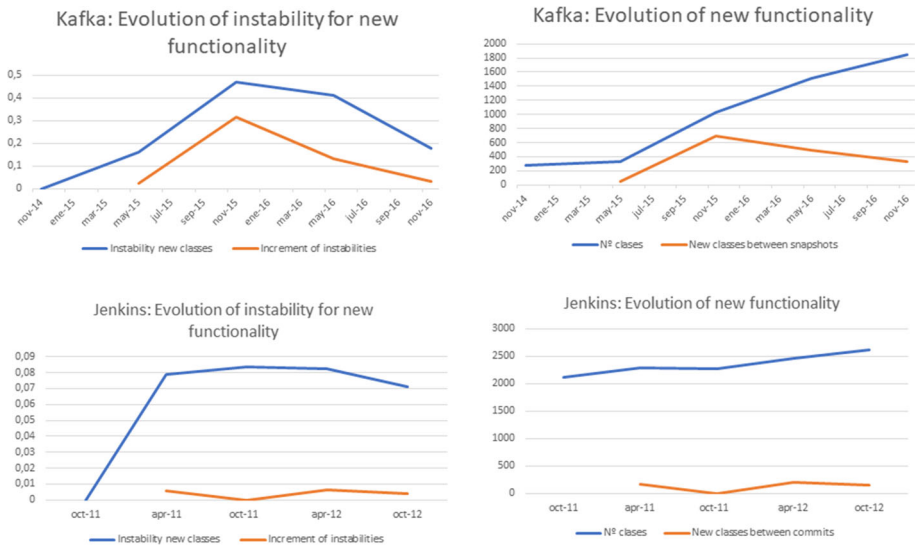

Fig. 6 Evolution of the instability trend of new elements

Table 12 Instability of new and removed functionality of Guava snapshots

Snapshot	Year	Instability of removed classes and dependencies	Instability of new classes and dependencies	Variation of instability	Number of classes	New classes between snapshots
adf2ce79	Oct-11	—	—	—	3,122	12,062
a351697f	Apr-12	0.1292	0.2979	0.0742	4,158	15,269
d63bcfad	**Oct-12**	0.0999	0.2149	0.0391	5,083	18,164
88ec3b70	Apr-13	0.1032	0.1565	0.0131	5,549	20,221
72a9270f	Oct-13	0.0432	0.0983	0.0118	6,311	22,652

The bold numbers indicate the reference date of the releases to set the period of time chosing project snapshots

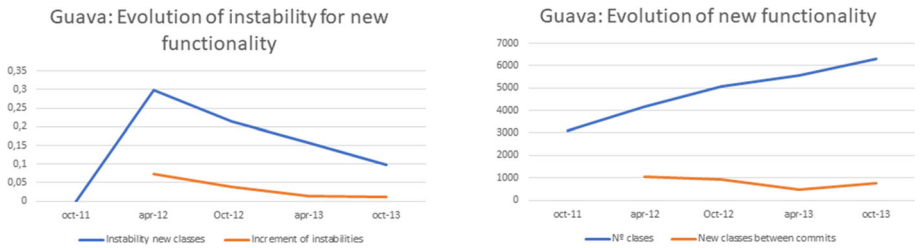

Fig. 7 Instability trends between snapshots of new elements of Guava project

Table 13 Instability of new and removed functionality of Dubbo snapshots

Snapshot	Year	Instability of removed classes and dependencies	Instability of new classes and dependencies	Variation of instability	Number of classes	New classes between snapshots
23732dc5	Dec-17	—	—	—	1,856	—
563347be	Jun-18	0.1277	0.1461	0.0105	2,000	144
d708271a	**Dec-18**	0.6581	0.6343	0.6343	2,082	82
be501691	Jun-19	0.0537	0.1719	0.1719	2,532	450
91e339fe	Dec-19	0.0799	0.1414	0.1414	2,871	339

The bold numbers indicate the reference date of the releases to set the period of time chosing project snapshots

Table 14 Instability of new and removed functionality of PDFBox snapshots

Snapshot	Year	Instability of removed classes and dependencies	Instability of new classes and dependencies	Variation of instability	Number of classes	New classes between snapshots
89f7d669	Jan-14	—	—	—	1,101	—
e74493eb	Jul-14	0.1164	0.1393	0.0051	1,143	42
7fcc7236	**Jan-15**	0.1047	0.1087	0.0019	1,164	21
73c9f556	Jul-15	0.0475	0.1012	0.009	1,278	114
882d7646	Jan-16	0.0477	0.0817	0.005	1,363	85

The bold numbers indicate the reference date of the releases to set the period of time chosing project snapshots

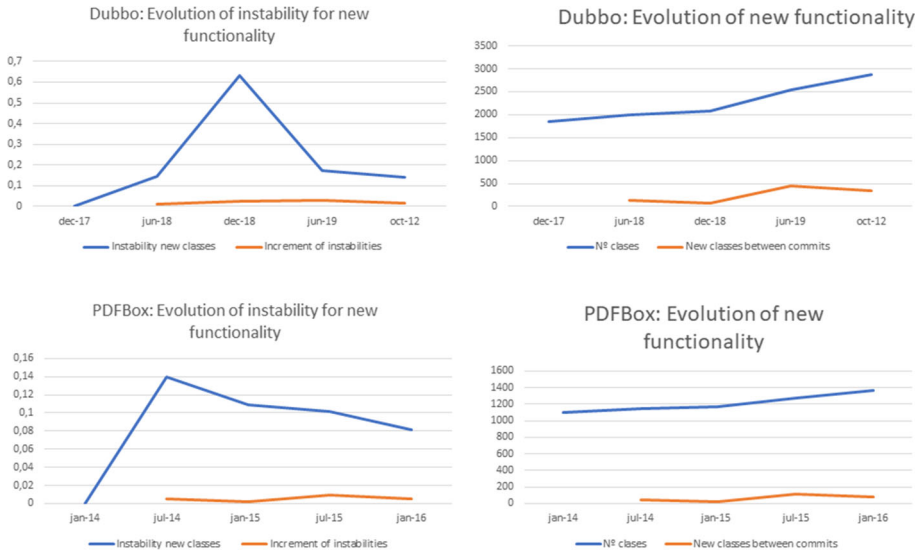

Fig. 8 Evolution of the instability of new elements in Dubbo and PDFBox projects

new classes shows a peak in December 2018. However, the curve that reflects the increment of instability values is more flat than the trend of new classes between snapshots. As presented in RQ1, the instability of this project shows a different pattern – but, surprisingly, while the overall instability for that snapshot is lower than for the other snapshots, the instability of the new classes is much higher, which at this point hints to major refactorings even if new functionality was added.

In the case of PDFBox, the behavior of adding new functionality tends to decrease after the second snapshot accordingly to the overall instability values of the project, showing a more predictable trend. The curves indicating the new classes between snapshots and the increments of instability values are quite similar.

6.2 Instability of Bugs and Refactorings

In order to evaluate the impact of bugs, refactorings and combinations of both, we searched in the commit messages between the snapshots for keywords that indicate what kind of task has been done. For the sake of completion, we also searched for keywords that hint to the addition of new functionality. Then, we used the `Perceval` tool (Duenas et al. 2018) to retrieve the log messages from the git repositories. This are provided in JSON files, that are parsed, extracting the message and the date. The date is used to assign the commit the one of the four phases under study: i) between 12m before and 6m before (T1), ii) between 6m before and the reference date (T2), iii) between the reference date and 6m after (T3), and iv) between 6m after and 12m after (T4). In addition, the commit will be categorized depending if its message contains a keyword. We have created categories such as "b&r" (bugs and refactorings) for those messages that contain keywords from both activities. If no keyword is found, the commit is assigned to the "Other" category. We have done a manual inspection of the commits of the five projects in order to minimize the number of commits in the "Other"

category, but its presence is inevitable as many commit messages contain just an URL, or offer semantically very poor descriptions (e.g., “preparing the next release”).

The specific keywords we have used for identifying commits that perform i) bug fixing, ii) refactorings and iii) add functionality is given next:

```
bugs = {"bug", "bugs", "fix", "fixed", "fixes", "fixing", "resolves",
"issue", "issues", "bugfix", "bugfixes", "closes", "hotfix", "hotfixes",
"[fixed", "[fixes", "typo", "typos", "correct", "correction",
"incorrect"}

refactorings = {"refactor", "refactoring", "refactors", "refactored",
"improve", "improves", "improving", "improvement", "improvements",
"simplify", "simplifies", "remove", "removed", "removes", "rename",
"renamed", "delete", "deletes", "deprecate", "deprecates", "cleanup",
"cleanups", "rewrite", "rewrites", "modify", "modified", "reworked",
"change", "changed", "changes", "optimize", "optimization",
"optimizations", "deoptimize", "rollback", "reduce", "reducing",
"update", "updated", "formatting", "reformat", "reformatting", "revise",
"revision", "tune", "tuning", "clean", "cleaning", "cleans", "exclude",
"excluding", "replace", "replacing", "adjust", "adjusting", "avoid",
"avoiding", "readded", "readd"}

add = {"add", "adding", "added", "creating", "creation", "created",
"develop", "development", "developed", "new"}
```

Taking into account the snapshots 6 and 12 months before and after the reference date for each project we got the results shown in the next paragraphs. First, we discuss the results for Kafka regarding the bugs (B), refactorings (R), and combinations of B/R including added (A) functionality as Table 15 shows.

If we consider the type of bugs for Kafka (see Table 15), we can see that bugs and refactorings have grown slightly more than commits with new functionality. However, from the findings from RQ1 we have seen that this has not been enough to mitigate the rise of instability.

As we can observe in Fig. 9, the period showing the peak in the graphic refers to the the time where committers removed most of the elements. This fact could be caused by a big refactoring process. We have to remark that this period (around May 2016) was also analyzed in the snapshot considered in RQ1.

Table 15 Bugs, refactorings and added code for Kafka

Kafka	T1 (nov14-may15)	T2 (may15-nov15)	T3 (nov15-may16)	T4 (may16-nov16)
Commits	237	346	957	851
Addition	47	95	193	163
Bugs	36	53	183	147
Refactorings	37	46	193	166
R&B	2	17	59	57
R&B&A	2	10	26	22
Other	113	125	303	306

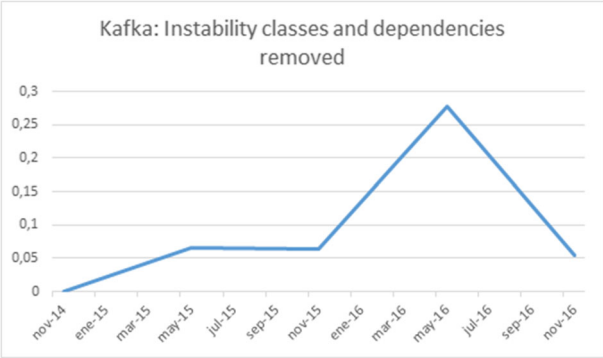

Fig. 9 Instability trends of elements removed in Kafka

Table 16 presents the results for the Jenkins project. A first interesting result is that, although the number of committers has grown during the period under study (we know this because the project fulfilled this as a criterion to be considered), this has not meant an increase in the number of commits, which remains almost stable. Regarding the type of commits, the ones due to addition of new functionality (column "Addition") remain more or less constant as well, but bugs and refactorings shrink. In other words, it seems that the share of effort devoted to adding functionality has grown, while bugs and refactoring has not keeping pace. Again, linking this result with RQ1, we see that instability has increased; again, more refactoring effort should have been devoted in order to maintain the instability measures.

Guava was the other project where we had identified an increase in instability in RQ1. If we have a look at the results of the types of commits in Table 17, we can observe that again commits with additional functionality, bugs and refactorings keep the proportions throughout the timespan under study.

Results for the Dubbo project are presented in Table 18. When answering RQ1 we had found an erratic behavior for this project, as its instability increases sometimes and other times decreases. Interestingly enough we can see that at first there is a phase where commits devoted to add functionality keep pace with bugs and refactorings (see the growth from T1 to T2). However, in T3 and T4 we can observe how committers have performed more actions related to bugs and refactorings that to aggregate new functionality. It looks like these efforts have resulted in Dubbo not having more instability, although because of its erratic behavior it has not been enough to decrease it.

Table 16 Bugs, refactorings and added code for Jenkins

Jenkins	T1 (oct10-apr11)	T2 (apr11-oct11)	T3 (oct11-apr12)	T4 (apr12-oct12)
Commits	1266	1565	1138	1015
Addition	244	319	232	189
Bugs	230	279	196	165
Refactorings	215	228	160	156
R&B	37	38	31	32
R&B&A	10	12	4	5
Other	530	689	515	468

Table 17 Bugs, refactorings and added code for Guava

Guava	T1 (oct11-apr12)	T2 (apr12-oct12)	T3 (oct12-apr13)	T4 (apr13-oct13)
Commits	415	420	469	254
Addition	113	106	140	68
Bugs	36	54	37	26
Refactorings	72	102	126	61
R&B	15	16	27	10
R&B&A	6	4	9	8
Other	173	138	130	81

According these results and to the graphics shown in Fig. 10 where most of the bugs and refactorings happened, both projects exhibit a similar pattern, but the instability values of the classes and dependencies removed in Jenkins are much lower than for Dubbo.

Finally, we can examine Table 19 with the results for PDFBox, the only project where we could see in RQ1 that has mitigated its instability. The results show that the project has devoted during the time period under study much more effort in refactoring than to adding new functionality or bugs, which remain almost constant.

The instability trends caused by the bugs and refactorings in these two projects are shown in Fig. 11. The behavior of these two projects exhibit a decreasing trend with two steps in the case of PDFBox until the curve stabilizes – and one step in Guava, where after a certain period of stability, the instability decreases again. In this case the instability values are easier to compare as the scale of both projects is the same.

6.3 Statistical Results

In a similar vein as in RQ1, we computed the Spearman correlation test and the p-values for the results of RQ2. We show the results in Tables 20, 21, and 22. As we can observe, we found a positive correlation between the instabilities and new functionality in all the projects except for PDFBox while the correlation between the instability values with bugs and refactorings is always positive. Nevertheless, the results are significant only for Kafka and Guava when we add new functionality and for Jenkins in the case of fixing bugs and performing refactorings. This fine-grained analysis of the correlation provides additional insight about the importance of the three activities in certain projects due to the significance of the work of the developers and during the transition between specific releases or in critical milestones.

Table 18 Bugs, refactorings and added code for Dubbo

Dubbo	T1 (dec17-jun18)	T2 (jun18-dec18)	T3 (dec18-jun19)	T4 (jun19-dec19)
Commits	294	469	593	483
Addition	54	103	89	68
Bugs	46	94	133	108
Refactorings	70	108	154	112
R&B	17	38	50	38
R&B&A	5	14	25	7
Other	102	113	142	150

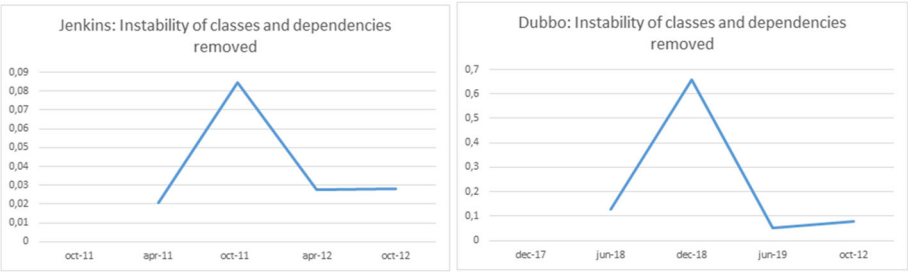

Fig. 10 Instability trends of elements removed in Jenkins and Dubbo

Table 19 Bugs, refactorings and added code for PDFBox

PDFBox	T1 (jan14-jul14)	T2 (jul14-jan15)	T3 (jan15-jul15)	T4 (jul15-jan16)
Commits	827	1206	1450	903
Addition	149	139	134	129
Bugs	97	146	131	92
Refactorings	189	299	460	291
R&B	36	42	39	12
R&B&A	3	5	3	0
Other	353	575	683	379

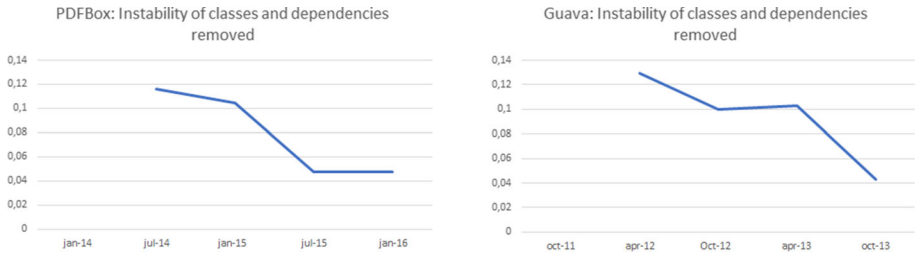

Fig. 11 Instability trends of elements removed in PDFBox and Guava

Table 20 Spearman correlation between instabilities and new functionality

Projects	Spearman correlation	p-value
Kafka	0.999	**1.4e-24**
Jenkins	0.103	0.870
Guava	0.899	**0.0373**
Dubbo	0.499	0.391
PDFBox	-0.300	0.624

The bold numbers indicate the significant values of the statistical results being p-value less than 0.05

Table 21 Spearman correlation between instabilities and bugs

Projects	Spearman correlation	p-value
Kafka	0.700	0.188
Jenkins	0.899	**0.037**
Guava	0.700	0.188
Dubbo	0.499	0.391
PDFBox	0.099	0.873

The bold numbers indicate the significant values of the statistical results being p-value less than 0.05

6.4 Summary and Take-Away Messages

In RQ1 we have seen that projects exhibit different behaviors during the transition phase. While all of them had an increase in functionality, some (Kafka, Jenkins and Guava) showed an increase in instability, while Dubbo and PDFBox seemed to lower it (although the results were only statistically significant for PDFBox). The information of the type of commits from RQ2 has shed some light into this. We have seen that instability seems to be sensitive to the relative amount of refactoring performed. Projects where the effort of refactoring keeps pace (i.e., grows in a similar vein) with the addition of new functionality suffer from an increase of instability. In order to lower instability, the number of refactoring commits has to be significantly larger than the number of commits that add functionality, as we have observed for the Dubbo and especially for the PDFBox projects. Our take-away messages are as follows:

– Like in the results for RQ1, the evolution of instability variations alongside with new functionality added indicates a common trend, so knowing how much functionality is added we can imply the instability varies accordingly until it stabilizes (e.g. PDFBox). However, in some projects like Guava and Kafka, the instability started decreasing abruptly mainly because some reorganization of the new classes impact positively in the instability values. In other cases like Dubbo, the snapshots selected provide different instability values in some peaks during the transition to a bazaar model. Therefore, the selection of different snapshots have also certain influence in the instability values according to the dynamicity of the committers.
– Regarding how instability is affected by bugs and refactorings, we found that in some projects like Kafka, the increasing refactoring operations to fix bugs is not enough to reduce the instability, at least until the last snapshot. We can infer that in this case the project has a significant activity but we need more snapshots to find when it stabilizes again. The opposite happens in Jenkins, as the instability increases while the bugs and refactorings decrease. In this case we can assume that an increment of the instability

Table 22 Spearman correlation between instabilities and refactorings

Projects	Spearman correlation	p-value
Kafka	0.700	0.188
Jenkins	0.899	**0.037**
Guava	0.600	0.285
Dubbo	0.499	0.391
PDFBox	0.099	0.873

The bold numbers indicate the significant values of the statistical results being p-value less than 0.05

is due to the importance of adding new functionality over bugs and refactorings, also because the trend of new functionality happens very abruptly.
- Other projects like Dubbo exhibit a varying behavior with increments and decrements of bugs and refactorings in a similar way as its instability variations. In this case it seems more difficult to predict the instability variations as we need to analyze with more detail the activity of committers for each single period of analysis. Also, although some projects like Jenkins and Dubbo exhibit common behavior patterns, the instability values (in absolute values) of the classes and dependencies removed are lower in Jenkins, which means that certain projects behave more unstable than others in the transition to the bazaar model.
- Additionally, when projects (e.g., PDFBox and Guava) are able to control the instability of the elements removed according to the same scale, it implies that the refactoring operations succeed over other activities. Nevertheless, this fact is sometimes not accomplished by the overall instability values for the same period, because instability clearly decreases in PDFBox as opposed to Guava.
- The use is third-party libraries could increase the instability in some cases but all in all it depends of the topology of the entities involved according to Martin's formula and which timeframe we are investigating. In general, if we have more efferent coupling dependencies to external libraries then the instability will increase but we need to investigate the topology to know if all these efferent dependencies are concentrated in few classes or distributed in several classes, as this will impact on the overall instability of the project.
- Finally, in those cases where the correlation between instability and new functionality, bugs, and refactorings is positive, this can be seen as a good indicator of the importance of some of these tasks.

7 Discussion

In this research we aimed to investigate two related goals. First, how the instability behaves in open-source projects during the transitions from the cathedral to the bazaar models. Second, the impact new functionality and project refactorings play on the instability values.

Regarding the first research question (RQ2), our results found that some projects exhibit an increasing instability trend when the number of new committers make the project transition from the cathedral to bazaar model, which makes sense a a significant amount of functionality increases the complexity of the project. This result was the one we expected, as we intuitively think that when projects transition to the bazaar phase, there is a boost in number of developers and contributors. It makes sense that instability in such a scenario grows. However, in other cases we found the instability behaves differently. For instance, in the Dubbo project it decreases a bit during the transition to the bazaar model and after it increases again. In the case of the PDFBox project the instability behaves completely the opposite to what is expected as it decreased from the cathedral to bazaar mode. It seems that even if a bunch of new functionality is added, its instability decreases.

Answering our second research question (RQ2) offers additional insight into the changes int the value of instability measured in RQ1 for the 5 case studies. By looking at where projects devote effort to (based on the type of commits), we think we can explain the different behaviors found in RQ1. We have observed that the projects where we did not find the expected outcome do a remarkable effort in refactoring. This can be seen from the fact that the amount of commits that we have identified as refactoring grows substantially more than the ones that are due to including new functionality, as can be seen from PDFBox. Interestingly enough,

the projects where we have found a notable increase in their instability values (e.g., Kafka, Jenkins and Guava) do devote also a non-negligible effort in refactoring. Actually, the share of refactoring commits is almost the same at the beginning and the end of the transition phase. However, it seems that this is not enough to maintain the values of instability. To maintain it stable (as in Dubbo) or decrease it (as in PDFBox), refactorings activities should have a higher priority.

We can learn from our research that measuring instability in a precise way is feasible, and with tools that only need to perform a static analysis of the source code. Instability is not only affected by what is added to the project (usually new functionality); activities that are related to maintain a healthy project such as refactoring are of major importance. While this is expected, what we have observed is how much of the refactoring is needed to counteract the rise of instability due to addition of new functionality. We have seen that the efforts required to maintain the instability values constant require projects to increase the effort on refactoring above the effort of introducing new functionality. Our method offers a way to measure it, so that developers and other stakeholders can be aware of where they are, and if the refactoring effort they are putting into the project is enough or should be expanded.

As a consequence, we think that instability metrics should be included in the portfolio of the metrics that are used to evaluate projects that are transitioning from the cathedral to the bazaar phase. Usually, the metrics that are considered currently are the number of contributions (in number of commits), and the number of contributors (in number of committers and other type of participants). Other measures are centered on functionality, with the number of lines as a still widely used one, although number of classes can also be found. We argue that instability metrics offer information on the balance of many of the previous metrics that is key for the healthy evolution of the project, and crucial in stages where projects are transitioning from the cathedral phase, where the development can be considered more under control, to the bazaar phase, where development is decentralized and it is not that easy to control everything.

Different stakeholders may benefit from our work:

- Developers have a way to measure easily their projects in the transition phase, and although they might not have the control they had in the cathedral phase, they are at least aware of if instability is a problem or not in their project.
- Project managers have a metric that allows them to assign effort to specific tasks in the project transitioning from the cathedral to the bazaar. The have now a means to analyze if they are optimally using their resources to have a sustainable project. With our approach they have a way to ensure that the project grows in functionality while maintaining a healthy status in regards to software stability.
- Companies wanting to invest (in monetary terms or with developers) in a rising OSS project can use this information to allocate their investment better, and to avoid risks that are common in this phase.
- Institutions such as the Apache Software Foundation that have programs to nurture projects can use this metrics in their regular portfolio to assess how projects are doing and evaluate if the project is ready for *graduating*.
- Researchers can take our research as a starting point and further deepen in the analysis of the transition phase. We acknowledge that ours is a first, superficial analysis of the impact of instability metrics in software development, in particular when projects undergo severe changes in organization and composition.

8 Related Work

8.1 From the Cathedral to the Bazaar

Since Eric Raymond's (1999) "The Cathedral and the Bazaar" essay was published in 1997 arguing that the way Linux was developed in a more efficient way (decentralized, transparent, with code always available, named the *bazaar*) than other OSS projects such as GNU Emacs or GCC (centralized, not open to external contributions, release based, named the *cathedral*), there have been many publications on how software projects achieve such a *bazaar* state.

It is known that a minor part of all OSS projects reach such a bazaar phase. Already Krishnamurthy (2002) found that only a few projects in SourceForge of the 100 active he sampled in 2002 could be considered as *bazaar* projects. Midha and Palvia (2012) studied how different factors (release frequency, popularity, organization, etc.) affected the chance of a software project being successful (i.e., having a community, so becoming bazaar-drive). Coelho and Valente (2017) point out that even if "developers are creating open source software at speeds never seen before [...], these projects are also facing unprecedented mortality rates". Nowadays it is more difficult to reach the *bazaar* phase, among others because there has been a general drift towards smaller software components that can be combined together easily, forming OSS ecosystems (Franco-Bedoya et al. 2017).

Yet, nurturing a community around an OSS project is the goal of many developers (and companies). To support developer communities in creating and maintaining sustainable OSS projects, non-profit organizations such as the Apache Software Foundation (ASF) run the ASF Incubator (ASFI) (Duenas et al. 2007) where young projects that want to become part of the ASF community receive mentor-like management and guidance to help them eventually become self-sufficient and even top-level projects in ASF. Projects in the ASFI, called podlings, must comply with ASF rules and regulations, including the publication of all engagements and emails. When certain conditions are met, project developers and ASF committees decide whether a podling should graduate, referred to as a "successful" sustainability outcome. Otherwise they withdraw. According to ASFI, an important criterion for a successful transition (*graduate* in the Apache jargon) is the development of an open and diverse performance community. The degree tests whether a project has learned enough and is responsible enough to hold its own as a community. Yin et al. have studied how to forecast if Apache Incubator projects will graduate (Yin et al. 2021), while Ramchandran et al. have developed a tool to "monitor and explore ASFI project sustainability trajectories, including social and technical networks" (Ramchandran et al. 2022).

8.2 Software Instability

Estimating the ripple effect of changes in OO classes has been analyzed by Mansour and Salem (2006), where the authors suggest metrics such as the Number of Children (NOC) and Coupling between objects (CBO) to evaluate the scope of a change in classes. Closer to the architecture level, Diaz2011 evaluate the impact of changes in product line architectures, while Li et al. (2012) classify 23 change impact analysis techniques for different software artifacts. Arvanitou et al. (2015) described a ripple effect formula to predict the effect of changes in classes using coupling metrics. Few works investigated the use of instability metrics in software architecture. Ampatzoglou et al. (2015) suggest an instability metric using probabilistic models to estimate the impact of changes in design patterns. The authors analyze the stability of changes in classes using change proneness measures (i.e., a priori estimation)

and instability measures (i.e., a posteriori measures). An extension of the aforementioned work has been done by Arvanitou et al. (2017), where the authors propose a method for assessing change proneness in classes due to evolving requirements, bug fixing, and ripple effect.

Regarding software instability, Santos et al. (2017) investigate the instability based on afferent and efferent coupling metrics in OSS projects, and performed and statistical analysis of the results observing that 48% of software product had a high instability. In addition, Aversano et al. (2018) analyzed the instability of architecture core components across releases and they defined instability metrics based on the packages that are added, removed, or changed. Salama and Bahsoon (2017) study the architectural stability of self-adaptive systems to achieve stable adaptations, and where instability can be considered an indicator of the sustainability of the system and architecture as well. Although other works, such as Ampatzoglou et al. (2015) and Arvanitou et al. (2017), highlight that instability and change proneness measures a clear effect on the stability of a system as an indicator of its sustainability, only seminal works from Carrillo et al. (2015) and Venters et al. (2018) suggest metrics to estimate the sustainability of architectural decisions.

Sas et al. (2019) investigate in more recent works the evolution of the instability in architectural smells across 524 versions in 14 open-source projects using the Arcan tool. The approach uses a similarity index to measure the percentage of elements that are shared by two sets affected by smell and hence, compute the smell density per component. The smells can be detected using the notion of instability gap as described by Fontana et al. (2016). Also, Hussain et al. (2019) study the historical class stability exploiting change history information as a way to predict unstable classes in 10 open-source projects and based on its correlation with change propagation factors. Salama et al. (2019) provide an updated survey on the notion of stability in software engineering practice which is understood as long-term property for analyzing software evolution. They discuss the characterization and use of stability in software engineering research and the important quality attributes related to it, including sustainability. From the various stability dimensions investigated, the authors highlighted the role of engineering practices for the evaluation of architectural stability, and they summarize the most popular stability metrics. In their findings, they claim that architectural decisions should be seen as planning for stability. Finally, Baig et al. (2019) discuss a similar approach to ours in terms of an analysis of the instability of open-source projects but the goals of the topics investigated are different from our work.

9 Threats to Validity

In this section we discuss the internal, external, and construct validity and how we mitigated the threats according to the guidelines by Yin (2014).

Internal Validity is used to establish a cause and effect relationship between a treatment and the outcome in order to uncover if the results support our claims. Given that this is observational research, we cannot claim that we have found causality; what we have identified is correlation, at most. Anyhow this is a first step to better understand the problem, especially as we provide several angles: instability metrics, increase in functionality, and type of changes performed.

We have taken snapshots of the repository at a given point in time (i.e., a commit) and not releases. We are aware that open-source projects may have different phases in their release cycle (Teixeira 2017) and that taking snapshots may not consider them. We think, however,

that the impact on the results is low for our research purposes, as we are interested more in the overall evolution of the project and this is captured as well with our approach. In addition, and regarding the threshold values regarding the number of committers used to identify the the cathedral and bazaar models, we acknowledge that these numbers can be different for different projects and the selection of different time frames could show different numbers of committers. This is why we had to carefully selected the transitions with 12 months before and after a given period to prove there is a cathedral-bazaar transition.

External Validity refers to how well the outcome of a study can be applied to other settings and to the generalizability of the results. We acknowledge that we only test the approach on five OSS projects, and that our findings can be limited to those projects. All our projects use similar development infrastructure, being hosted in GitHub, which might be not valid for all projects. In addition, due to how we have performed our search, three out of these five projects are developed under the umbrella of the Apache Software Foundation, so they might have more commonalities than projects chosen at random.

Our major challenge when performing this research has been finding suitable projects to be studied that followed the inclusion criteria that we specified in the pre-registered report. It has to be taken into account that in the pre-registered report we proposed threshold values that we thought made sense to ascertain that a project has achieved to be in the bazaar phase starting from a cathedral phase (see 5. Execution plan). As we know that there is no strict rule to specify what is to be in the cathedral and what is to be in the bazaar, we had chosen threshold values that were secure: less than 10 (cathedral) and more than 50 (bazaar) are such values, and are supported by other publications found in the research literature.

When we started looking for Java projects that followed a transition from cathedral (less than 10 contributors) to bazaar (more than 50 contributors) in a short period of time (12 months), we found that this was not as easy as we thought. Section 3.3 reports on our efforts, starting with a general GitHub search, the top 50 Java projects from a popular list, and even doing a "brute force" search in Boa. Even after doing this, we just had 3 projects that fulfilled the inclusion criteria. We had to specifically search for repositories from the Apache project to get to the final number of 5 projects.

All in all, it has not to be forgotten that our goal is not to look for a representative sample of Java projects, but to find out if instability metrics can help us to identify risks. As such, the case study method (Runeson and Host 2009), where a few projects are studied in detail, can be considered adequate. According to Bird and Zimmermann (2012), we refer to the importance of detailed studies of few projects. The authors recall that it is a misconception that this type of studies, on a small number of projects, does not provide to the academic community any value, nor contribute to scientific development. In any case, as we only used five open-source projects we need to evaluate more projects and more possible cathedral-bazaar transitions to confirm our first results. To mitigate this threat we plan to evaluate other OSS projects from different sizes and analyze if we found new patterns that can deviate from the initial results.

Construct Validity indicates the adequacy of the measures we claim and the relation between the theory and our observations. We followed a well-defined research methods (i.e. exploratory study) to measure the results based on a well-known instability metric. We also used standard and popular OSS projects with a certain evolution that could be suitable to be analyzed along a period of time and with enough contributions that makes possible to obtain significant data. Another way that could provide more accurate insight is to check the structure of the software of those snapshots that provide less expected instability values according to Martin's formula. On the other hand, the identification of the type of commits has been done using heuristics. Even if we have used keywords commonly used in the litera-

ture (Rodriguez-Perez et al. 2020), and have done a manual check to minimize the category of "other" commits, this is an error-prone procedure (Herzig et al. 2013).

10 Conclusions

In this study, we have analyzed what information instability metrics can offer to OSS projects, especially when they transition from the cathedral phase, when the project is led in a more centralized by a small number of contributors, to the bazaar phase, when a distributed community of developers drives the project. Therefore, we have used a static analysis tool that offers instability metrics from a set of five Java-based projects, which we have used as case studies. In addition, we have analyzed the type of activity performed in those projects, especially if it is due to the addition of new functionality, the correction of bugs or tasks related to refactoring the source code.

Our initial findings reveal that the projects exhibit increasing or decreasing instability trends following a common behavior patterns of instability values, and validated by a strong correlation between the instability and the number of classes and edges. Only one of the projects examined showed divergences as it exhibited a non-clear trend, but we acknowledge that the number of the projects analyzed is small. Interestingly enough, while three projects (Kafka, Jenkins and Guava) shows a strong correlation between the number of classes and dependencies added to the project and instability, there was one project (PDFBox) where the direction of the correlation was strong but in the opposite direction.

A detailed analysis of the type of commits (adding functionality, correcting bugs and/or performing refactoring) shows that the correlations obtained depend heavily on the type of work performed. We have observed that for projects where the refactoring effort in the transition phase keeps pace (i.e., their relative importance on the total effort is the same, as in Kafka, Jenkins and Guava) with the introduction of new functionality, we obtain increasing values of instability. For PDFBox, the project where correlation goes in the opposite direction (i.e., more functionality has been added but instability is lower), this is because the amount of refactoring work has remarkably increased. Beyond offering evidence that refactoring limits instability, which could be expected, our research shows the relevance of measuring it and the fact that to obtain a balanced situation, where the project can enter the bazaar phase with a healthy and sustainable situation. In short, we have found that instability can be a good metric to alert developers that their project is getting out of hand, and gives them insight into the balance between adding functionality and refactoring.

Developers could use the insights obtained from our research to measure the *health* of their project, and to decide when refactoring should be prioritized. Stakeholders that support bringing projects to the community, as the Apache Software Foundation Incubator initiative does, could benefit from integrating these type of metrics into their portfolio, as it complements well the typical community-related metrics that are used nowadays such as number of contributors and number of commits.

Further research could be devoted to analyze more projects to see if our findings can be generalized, in particular for other programming languages than Java. It would also be desirable to better understand the optimal equilibrium between adding new functionality and performing refactoring tasks. Another research avenue could be to study projects that planned to become bazaar projects, but did not have success. Our selection of case studies has hindered us from selecting such kind of projects (as we intentionally targeted projects that had successfully achieved a community-driven status), but we are aware that learning

from failure can be very enriching. Thus, we think that applying instability metrics to those projects and see if they provide relevant information would be worthwhile to investigate.

One possible reason that explains our difficulties finding projects that fit the inclusion criteria we had specified in our registered report (Valdezate et al. 2022) is that the OSS community is different to how it was 20 years ago. While in its early ages larger projects were sought (in terms of the breadth of objectives and number of contributors), today projects seem to be more specific, smaller in size, having more dependencies to other projects. This could mean that the bazaar (inspired by projects such as Linux, the Apache web server, Mozilla Firefox, OpenOffice.org, among others) may currently not be found in single projects, but as software ecosystems that group many collaborating projects. Only in associations like Apache does it seem that direct collaboration (through many developers) is prioritized over indirect collaboration (through dependencies). This reflection goes beyond the scope of this research, but it should be taken into account for further research.

To conclude, we believe this study advances previous state of the art and provides guidance not only to evaluate the instability of software projects when they transition from the cathedral to the bazaar, or become more popular, but also how some of these trends may serve to identify if a project will have a more sustainable maintenance and evolution in the future.

Acknowledgements The research presented in this paper has been supported in part by the Government of Spain, through project Dependentium (PID2022-139551NB-I00).

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

Data Availability The datasets generated during and/or analysed during the current study are available from the corresponding author on reasonable request.

Declarations

Conflicts of interest The authors declared that they have no conflict of interest.

References

Alenezi M, Khellah F (2015) Architectural stability evolution in open-source systems. In: Proceedings of the The International Conference on Engineering & MIS 2015, ICEMIS'15, New York, NY, USA, 2015. Association for Computing Machinery

Alshayeb M, Li W (2005) An empirical study of system design instability metric and design evolution in an agile software process. J Syst Softw 74(3):269–274

Alshayeb M, Naji M, Elish MO, Al-Ghamdi J (2011) Towards measuring object-oriented class stability. IET Software 5(4):415–424

Alves TL, Hage J, Rademaker P (2011) A comparative study of code query technologies. In: 2011 IEEE 11th international working conference on source code analysis and manipulation. pp 145–154

Ampatzoglou A, Chatzigeorgiou A, Charalampidou S, Avgeriou P (2015) The effect of GoF design patterns on stability: A case study. IEEE Trans Software Eng 41(8):781–802

Arvanitou EM, Ampatzoglou A, Chatzigeorgiou A, Avgeriou P (2015) Introducing a ripple effect measure: a theoretical and empirical validation. In: 2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). IEEE

Arvanitou E-M, Ampatzoglou A, Chatzigeorgiou A, Avgeriou P (2017) A method for assessing class change proneness. In: Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering - EASE 2017. ACM Press

Aversano L, Guarda D, Tortorella M (2018) Analyzing the instability of the core components of software projects. In: Proceedings of the 51st Hawaii International Conference on System Sciences. Hawaii International Conference on System Sciences

Baig JJA, Mahmood S, Alshayeb M, Niazi M (2019) Package-level stability evaluation of object-oriented systems. Inf Softw Technol 116:106172

Benkoczi R, Gaur D, Hossain S, Khan M, Tedlapu AR (2020) Evolutionary hot-spots in software systems. In: Proceedings of the ACM/IEEE 42nd international conference on software engineering: companion proceedings. pp 272–273

Bhowmik T, Niu N, Wang W, Cheng J-RC, Li L, Cao X (2015) Optimal group size for software change tasks: A social information foraging perspective. IEEE Trans Cybernet 46(8):1784–1795

Bird C, and Zimmermann T (2012) Assessing the value of branches with what-if analysis. In: Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering. ACM, pp 45

Borges H, Valente MT (2018) What's in a github star? understanding repository starring practices in a social coding platform. J Syst Softw 146:112–129

Brooks FP Jr (1995) The mythical man-month: essays on software engineering. Pearson Education

Brown A, Wilson G (2011) The Architecture of Open Source Applications: Elegance, Evolution, and a Few Fearless Hacks, volume 1. Lulu. com

Capiluppi A, Michlmayr M (2007) From the cathedral to the bazaar: An empirical study of the lifecycle of volunteer community projects. In: IFIP International Conference on Open Source Systems. Springer, pp 31–44

Carrillo C, Capilla R (2018) Ripple effect to evaluate the impact of changes in architectural design decisions. In: Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings, ECSA 2018, Madrid, Spain, September 24-28, 2018. pp 41:1–41:8

Carrillo C, Capilla R, Zimmermann O, Zdun U (2015) Guidelines and metrics for configurable and sustainable architectural knowledge modelling. In: Proceedings of the 2015 European Conference on Software Architecture Workshops - ECSAW 2015. ACM Press

Chawla MK, Chhabra I (2015) Sqmma: Software quality model for maintainability analysis. In: Proceedings of the 8th Annual ACM India Conference. Association for Computing Machinery, New York, NY, USA, pp 9-17

Coelho J, Valente MT (2017) Why modern open source projects fail. In: Proceedings of the 2017 11th Joint meeting on foundations of software engineering. pp 186–196

Díaz J, Pérez J, Garbajosa J, Wolf AL (2011) Change impact analysis in product-line architectures. Software architecture. Springer, Berlin, Heidelberg, pp 114–129

Dinh-Trong TT, Bieman JM (2005) The freebsd project: A replication case study of open source development. IEEE Trans Software Eng 31(6):481–494

Duenas JC, Cuadrado F, Santillán M, Ruiz JL et al (2007) Apache and eclipse: Comparing open source project incubators. IEEE Softw 24(6):90–98

Dueñas S, Cosentino V, Robles G, Gonzalez-Barahona JM (2018) Perceval: software project data at your will. In: Proceedings of the 40th international conference on software engineering: companion proceeedings. pp 1–4

Dyer R, Nguyen HA, Rajan H, Nguyen TN (2015) Boa: Ultra-large-scale software repository and source-code mining. ACM Trans Softw Eng Methodol 25(1):1–34

Ferreira F, Silva LL, Valente MT (2020) Turnover in open-source projects: The case of core developers. In Proceedings of the XXXIV Brazilian Symposium on Software Engineering. pp 447–456

Fontana FA, Pigazzini I, Roveda R, Tamburri D, Zanoni M, Nitto ED (2017) Arcan: A tool for architectural smells detection. In: 2017 IEEE International Conference on Software Architecture Workshops (ICSAW). pp 282–285

Fontana FA, Pigazzini I, Roveda R, Zanoni M (2016) Automatic detection of instability architectural smells. In: 2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016, Raleigh, NC, USA, October 2-7, 2016. pp 433–437

Franco-Bedoya O, Ameller D, Costal D, Franch X (2017) Open source software ecosystems: A systematic mapping. Inf Softw Technol 91:160–185

Garcia J, Kouroshfar E, Ghorbani N, Malek S (2022) Forecasting architectural decay from evolutionary history. IEEE Trans Software Eng 48(7):2439–2454
Gonzalez-Barahona JM, Robles G, Herraiz I, Ortega F (2014) Studying the laws of software evolution in a long-lived floss project. J Softw Evol Process 26(7):589–612
Gupta A, Suri B, Kumar V, Jain P (2021) Extracting rules for vulnerabilities detection with static metrics using machine learning. Int J Syst Assur Eng Manag 12:65–76
Herzig K, Just S, Zeller A (2013) It's not a bug, it's a feature: how misclassification impacts bug prediction. In: 2013 35th international conference on software engineering (ICSE). IEEE, pp 392–401
Hoegl M (2005) Smaller teams-better teamwork: How to keep project teams small. Bus Horiz 48(3):209–214
Hussain S, Afzal H, Rafiq Mufti M, Imran M, Amjad A, Ahmad B (2019) Mining version history to predict the class instability. PLoS ONE 14(9):1–21
Kim DK (2017) Finding bad code smells with neural network models. Int J Electr Comput Eng 7(6):3613
Koch S (2007) Software evolution in open source projects-a large-scale investigation. J Softw Maint Evol Res Pract 19(6):361–382
Krishnamurthy S (2002) Cave or community?: an empirical examination of 100 mature open source projects. First Monday 7(6)
Laser MS, Medvidovic N, Le DM, Garcia J (2020) ARCADE: an extensible workbench for architecture recovery, change, and decay evaluation. In: Devanbu P, Cohen MB, Zimmermann T (eds) ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020. ACM, pp 1546–1550
Le DM, Carrillo C, Capilla R, Medvidovic N (2016) Relating architectural decay and sustainability of software systems. In: 2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA). IEEE
Li W, Etzkorn L, Davis C, Talburt J (2000) An empirical study of object-oriented system evolution. Inf Softw Technol 42(6):373–381
Li B, Sun X, Leung H, Zhang S (2012) A survey of code-based change impact analysis techniques. Softw Test Verif Reliab 23(8):613–646
Lin B, Robles G, Serebrenik A (2017) Developer turnover in global, industrial open source projects: Insights from applying survival analysis. In: 2017 IEEE 12th International Conference on Global Software Engineering (ICGSE). IEEE, pp 66–75
Malhotra R, Bansal A, Jajoria S (2016) An automated tool for generating change report from open-source software. In: 2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI). IEEE, pp 1576–1582
Mansour N, Salem H (2006) Ripple effect in object oriented programs. J Comp Methods Sci Eng 6(5,6 Supplement 1):23–32
Martin R (1994) OO Design Quality Metrics - An Analysis of Dependencies. In: Workshop pragmatic and theoretical directions in object-oriented software metrics. OOPSLA'94
McConnell S (2006) Software estimation: demystifying the black art. Microsoft Press
Midha V, Palvia P (2012) Factors affecting the success of open source software. J Syst Softw 85(4):895–905
Mockus A, Fielding RT, Herbsleb JD (2002) Two case studies of open source software development: Apache and mozilla. ACM Trans Softw Eng Methodol 11(3):309–346
Moore S, Armstrong P, McDonald T, Yampolskiy M (2016) Vulnerability analysis of desktop 3d printer software. In: 2016 Resilience Week (RWS). IEEE, pp 46–51
Munaiah N, Kroh S, Cabrey C, Nagappan M (2017) Curating github for engineered software projects. Empir Softw Eng 22(6):3219–3253
Ralph P (2021) Acm sigsoft empirical standards released
Ramchandran A, Yin L, Filkov V (2022) Exploring apache incubator project trajectories with apex. In: Proceedings of the 19th international conference on mining software repositories. pp 333–337
Ratiu D, Ducasse S, îrba TG, Marinescu R (2004) Using history information to improve design flaws detection. In: 8th European Conference on Software Maintenance and Reengineering (CSMR 2004), 24-26 March 2004, Tampere, Finland, Proceedings. IEEE Computer Society, pp 223–232
Raymond ES (2001) The cathedral and the bazaar - musings on Linux and open source by an accidental revoltionary (rev. ed.). O'Reilly
Raymond E (1999) The cathedral and the bazaar. Knowl Technol Policy 12(3):23–49
Robles G and Gonzalez-Barahona JM (2006) Contributor turnover in libre software projects. In: Open Source Systems: IFIP Working Group 2.13 Foundation on Open Source Software, June 8–10, 2006, Como, Italy 2. Springer, pp 273–286
Robles G, Merelo JJ, Gonzalez-Barahona JM (2005) Self-organized development in libre software: a model based on the stigmergy concept. In 6th International Workshop on Software Process Simulation and Modeling - ProSim'05, held in St. Louis, USA

Rodríguez D, Sicilia M, García E, Harrison R (2012) Empirical findings on team size and productivity in software development. J Syst Softw 85(3):562–570
Rodríguez-Pérez G, Robles G, Serebrenik A, Zaidman A, Germán DM, Gonzalez-Barahona JM (2020) How bugs are born: a model to identify how bugs are introduced in software components. Empir Softw Eng 25:1294–1340
Runeson P, Höst M (2009) Guidelines for conducting and reporting case study research in software engineering. Empir Softw Eng 14:131–164
Salama M, Bahsoon R (2017) Analysing and modelling runtime architectural stability for self-adaptive software. J Syst Softw 133:95–112
Salama M, Bahsoon R, Lago P (2019) Stability in software engineering: Survey of the state-of-the-art and research directions. IEEE Trans Softw Eng 100:100
Santos D, de Resende AMP, Lima EC, Freire AP (2017) Software instability analysis based on afferent and efferent coupling measures. J. Softw. 12(1):19–34
Sas D, Avgeriou P, Arcelli Fontana F (2019) Investigating instability architectural smells evolution: an exploratory case study. In: 35th International Conference on Software Maintenance and Evolution. IEEE
Senyard A, Michlmayr M (2004) How to have a successful free software project. In 11th Asia-Pacific software engineering conference. IEEE 84–91
Tan X, Zhou M, Fitzgerald B (2020) Scaling open source communities: an empirical study of the Linux kernel. In: 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE). IEEE, pp 1222–1234
Teixeira J (2017) Release early, release often and release on time. An empirical case study of release management. In: Open Source Systems: Towards Robust Practices: 13th IFIP WG 2.13 International Conference, OSS 2017, Buenos Aires, Argentina, May 22-23, 2017, Proceedings 13. Springer International Publishing, pp 167–181
Teixeira J, Mian S, Hytti (2016). Cooperation among competitors in the open-source arena: The case of openstack. In 2016 International Conference on Information Systems (ICIS 2016), held in Dublin, Ireland
Threm D, Yu L, Ramaswamy S, Sudarsan SD (2015) Using normalized compression distance to measure the evolutionary stability of software systems. In: 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE). IEEE, pp 112–120
Valdezate A, Capilla R, Robles G, Salamanca V (2022) Can instability variations warn developers when open-source projects boost? CoRR, arXiv:2204.05209
Venters CC, Capilla R, Betz S, Penzenstadler B, Crick T, Crouch S, Nakagawa EY, Becker C, Carrillo C (2018) Software sustainability: Research and practice from a software architecture viewpoint. J Syst Softw 138:174–188
Wermelinger M, Yu Y, Lozano A, Capiluppi A (2011) Assessing architectural evolution: a case study. Empir Softw Eng 16(5):623–666
Yang N, Ferreira I, Serebrenik A, Adams B (2022) Why do projects join the apache software foundation? In Proceedings of the 2022 ACM/IEEE 44th international conference on software engineering: software engineering in society. pp 161–171
Yin RK (2014) Case Study Research Design and Methods (5th ed.). Sage
Yin L, Chen Z, Xuan Q, Filkov V (2021) Sustainability forecasting for apache incubator projects. In: Proceedings of the 29th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering. pp 1056–1067
Zhang Y, Zhou M, Mockus A, Jin Z (2019) Companies' participation in oss development-an empirical study of openstack. IEEE Trans Softw Eng 47(10):2242–2259
Zhang B, Becker M, Patzke T, Sierszecki K, Savolainen JE (2013) Variability evolution and erosion in industrial product lines: a case study. In: Proceedings of the 17th international software product line conference. pp 168–177
Zhou M, Chen Q, Mockus A, Wu F (2017) On the scalability of linux kernel maintainers' work. In Proceedings of the 2017 11th joint meeting on foundations of software engineering. pp 27–37

Rafael Capilla is a Full Professor at Rey Juan Carlos University of Madrid, Spain. His major research interest focuses on architecture knowledge, product line engineering, technical debt, software sustainability and Industry 4.0. He has been a visiting researcher and professor in several universities in Europe and South America. Rafael is IEEE senior member. Contact him at: rafael.capilla@urjc.es

Victor Salamanca is a senior software engineer for more than 15 years at Santander bank. Victor's main role is project manager, and he oversees technology projects for Santander. His research interests focus on software quality, continuous quality evaluation methods, and Industry 4.0. He is a PhD candidate at Rey Juan Carlos University of Madrid, Spain.

Alejandro Valdezate received a Ph.D. degree in computer science (2022), a bachelor's degree in computer science and the M.Sc. degree from Rey Juan Carlos University of Madrid, Spain. Alejandro is a regular reviewer of computer journals and is a Professor in Computer Science at Interantional University of La Rioja (UNIR), Spain. He has more than 20 years of a professional experience in several Spanish software companies in the areas of DevOps, software testing/QA,and cloud services. His research interests include product line engineering and dynamic variability solutions.

Gregorio Robles is a Full Professor at the Universidad Rey Juan Carlos, Madrid, Spain. He mainly does research in the following two fields: a) mining software repositories (socio-technical issues such as community metrics, software evolution, and development effort estimation of F/OSS); and b) computational thinking (with evaluation tools such as Dr.Scratch).