Journal of Software Engineering Research and Development, 2023, 11:9, doi: 10.5753/jserd.2023.967

Software Architectural Practices: Influences on the Open Source Ecosystem Health

Simone da Silva Amorim [**Federal Institute of Bahia** | ***simone.amorim@ifba.edu.br***]
John D. McGregor [**Clemson University** | ***johnmc@clemson.edu***]
Eduardo Santana de Almeida [**Federal University of Bahia** | ***esa@rise.com.br***]
Christina von Flach Garcia Chavez [**Federal University of Bahia** | ***flach@ufba.br***]

Abstract. The health state of a software ecosystem has been determined by its capacity for growth and longevity. Three health indicators represent a healthy software ecosystem: robustness, productivity, and niche creation. Studies focusing on understanding the causes and processes of the health state of ecosystems have used these indicators largely. Researchers have intensified studies to understand how to achieve a good health state. Despite the growing number of studies, there is little knowledge about influences and actions to achieve health, and more specifically, studies that consider the effects of the software architecture on the ecosystem. This article presents a study exploring seven open source ecosystems within different domains to describe the influence of architectural practices on the software ecosystem health in terms of their motivations and effects. Our main goal was to understand how the software architecture and related practices can contribute to a healthy ecosystem. We conducted a netnography-based study to gather practices used to create and maintain the software architecture of these ecosystems. Our study brings evidence that architectural practices play a critical role in the achievement of ecosystems' health. We found fifty practices that have influenced different aspects of health indicators. We highlight the importance of five influential factors – business goals, experience, requirements, resources, and time-to-market – for motivating the adoption of such practices. These factors may also contribute to understanding different strategies used to achieve a good health state. Moreover, we proposed a novel health indicator, trustworthiness, that accounts for the normal operation of a healthy software ecosystem.

Keywords: *software ecosystems, ecosystem health, software practices, netnographic study*

1 Introduction

Over the years, the development of large-scale software has been problematic and overpriced. Most software companies spend a lot of time fixing problems, and there is not enough time to release the necessary features. Releasing software components suffers delay constantly, consequently delivering value to the customer has been a complex challenge. In trying to solve these problems, companies are using a range of techniques to manage the complexity of software development. Bosch and Bosch-Sijtsema introduced trends guiding the modern large-scale software development (Bosch and Bosch-Sijtsema, 2010). One of these trends is to provide a software platform and open its boundaries to the community around, sharing work and challenges. This community develops relevant solution elements to satisfy customer needs, composing a software ecosystem.

Software ecosystems are platforms that allow developers to build new capabilities, providing value based on the exchange between businesses and people. The benefits of platform technologies permit sharing decisions, risks, and profits, causing significant growth in collaboration and platform-minded reasoning (Liu, 2017). Several organizations follow the ecosystem strategy and have kept their success for several years, such as Hadoop, Amazon, and Apple (Satell, 2016). For instance, Apple rapidly gained the smartphone market introduced in 2007, when this ecosystem sold 270,000 iPhones in the first 30 hours which it was available (West and Mace, 2010). Currently, Apple continues to be the market leader in the fourth quarter of 2020 with its iPhone sales, when it grew almost 15%, and it was the first time revenue passed the $100 billion mark, $65.6 billion of that was from iPhone sales (Silverman, 2021). However, there are some cases of failure, like Nokia and BlackBerry. Nokia had a dominant position in the market in the early 2000s but failed by adopting several strategies in building an ecosystem around their products causing its downfall in 2011 when happened the shift from Symbian to the Windows phone (Bouwman et al., 2014). Another example of the downfall was the BlackBerry. By 2006, it was a market leader providing services such Blackberry Messenger(BBM), Email and QWERTY keyboard. However, from 2009 until 2013, blackberry started to decline, and in 2017 Blackberry finally decided to end the hardware development of mobile devices (Mittal, 2019). Understanding the reasons for the failures and successes of these ecosystems can address the mechanism of governance more efficiently to achieve a healthy ecosystem. A metaphor defined by Iansiti and Levien describes a healthy ecosystem with *"the growing and continuity of the software ecosystem remaining variable and productive over time"* (Iansiti and Levien, 2002). Having awareness of the health status of a software ecosystem is relevant to support decision-making in different areas such as business processes, software design, and social interactions. Stakeholders can decide to enter or leave the ecosystem, or to keep their participation in it. By assessing the health of software ecosystems, Iansiti and Levien proposed three health indicators: robustness, productivity, and niche creation (Iansiti and Levien, 2002).

The health evaluation of a software ecosystem is a complex activity and faces hard challenges. Several factors may influence the achievement of possible health states, including the software architecture. Health evaluations are often performed using the set of indicators defined by Iansiti and Levien (2002), however, there has been little research on the relationship between software architecture and the health state of a software ecosystem, mainly with regard to software architecture practices (da Silva Amorim et al., 2017). To better understand this research topic, we decided to investigate the software architectural practices more deeply to know the used practices in the ecosystem scenarios, what factors influence their use and determine their adoption, and how these practices can influence the health indicators. By knowing factors of adoption for a practice, governance mechanisms can be created to define the proper adoption of a practice, and its consequent influence on the health of the ecosystem. Moreover, defining how many architectural practices influence the health of software ecosystems will allow us to figure out the strength and weaknesses of software development in this environment. Based on proper information, guidelines for success can be created to drive efforts for other organizations. This paper addresses the problem by presenting *"a netnography-based study of the architectural practices and their influences on the health of open source ecosystems"*. Netnography is an adaptation of the ethnography methodology that considers new conditions of interactions intermediated by computers on the internet (Kozinets, 2009). High-level descriptive findings explain the motivations and effects of architectural practices on the ecosystem health.

Our findings were derived using a nethnography-based methodology adapted to the context of software ecosystems that explore the relationships around the architectural practices used. They allowed us to develop substantive results based on data that includes explanations of how things are related to each other. Our results describe the experience of the authors based on a systematic analysis of such practices. We observed that none of the studied ecosystems adopted all the identified practices. Besides, some of them developed singular practices in accordance with their needs. We argue in our discussion that software architecture has a key role in software health. So far, there is no research considering this influence. From the architectural practices, we can analyze possible effects on the health indicators. Based on the awareness of the influence from practices, we highlight the importance to focus not only on metrics already suggested by other studies (da Silva Amorim et al., 2017), but also on the investigation of proper architectural practices to evaluate the ecosystem health.

The rest of this paper is structured as follows. Section 2 provides some theoretical background on software ecosystems, architecture, and health. Section 3 introduces related work in accordance with our study, and section 4 presents the research methodology. Section 5 describes the influence factors on the architectural practices and health indicators. Section 6 discusses our findings and the relationship between practices and health indicators, and section 7 presents some threats and limitations of the study. Finally, section 8 presents our conclusions.

2 Software Ecosystems, Architecture, and Health

This section provides some background on software ecosystems, software architecture, and software ecosystem health.

2.1 Software Ecosystems

Bosch and Bosch-Sijtsema define software ecosystems as *"a software platform, a set of internal and external developers and a community of domain experts in service to a community of users that compose relevant solution elements to satisfy their needs"* (Bosch and Bosch-Sijtsema, 2010). This idea is based on external developers building applications on top of a platform, extending their products to attend specific needs. By opening the boundaries of the platform, the organization can increase its consumer base faster. In addition, the organization allows third-party to build components to address specific segments of the market which it cannot provide in a determined amount of time. Therefore, the ecosystem strategy contributes to organizations accelerating innovation and at the same time sharing the cost of this innovation (Bosch, 2009).

Adopting an ecosystem strategy is not a trivial task. In this scenario, new dependencies are created connecting third-party applications on the top of the platform. From this moment, software evolution should be coordinated, involving all stakeholders. Internal and external teams should be synchronized to furnish the interests of all group participants. Interfaces between the platform and products should be developed in collaboration, including all developers affected by the changes. Also, external developers should validate new releases of the platform to reduce breaks in the components launched in this context (Bosch and Bosch-Sijtsema, 2010).

Software ecosystem characteristics are commonly organized in three-dimensional views (Campbell and Ahmed, 2010; dos Santos and Werner, 2011). These views consider characteristics identifying aspects of business, social, and architectural scenarios. Based on our studies (da Silva Amorim et al., 2017), we provide our characterization of software ecosystems composed by business, community, and technical views: (i) Business views define policies to guide efforts to achieve sustainable growth through proper communication, attracting and retaining developers; (ii) Community views lead to challenges as coordinating work and making decisions, and the delay in performing some tasks acting in conjunction with third-parties; (iii) Technical views encompass software platforms and mechanisms for building software considering primarily the technology, methods, and tools.

Furthermore, there is an important classification for software ecosystems, considering their openness degree. According to Manikas and Hansen, a software ecosystem can be proprietary (commercial), Free or Open Source Software (FOSS), or hybrid. The openness degree is related to the level of access to the platform resources. Proprietary ecosystems control access to the source code and artifacts. Information is protected, so developers should have some kind of permission to engage in the ecosystem. On the other hand, in FOSS ecosystems, developers have other motivations besides the financial return. For instance, they may want to get new knowledge or improve their skills (Manikas and Hansen, 2013). They are often opened to any participant willing to join. Our work is focused on the scenario of open source ecosystems.

2.2 Software Ecosystem Architecture

According to Pelliccione, *"software architecture plays a key role in managing software ecosystems and their evolution"*. Indeed, dependencies among the platform and applications working on top of that generate a set of architectural issues that should be considered during software development. The software architecture will express the business goals of stakeholders through technical mechanisms. Especially, the architecture of the platform will define the boundaries of the evolution of the ecosystem and how and how many developers can work to extend their features. Regarding the large number of software connections found in software ecosystems, architectural decisions have a key role, since wrong decisions can cause serious harm for the balance of the ecosystems. These decisions should consider internal and external influences attending different types of business goals. Also, quality attributes affect the actions of developers that should adapt their applications to work well in sync with the core platform (Pelliccione, 2014).

In brief, software architecture has a critical role in software ecosystems. Some researchers study its impact and challenges to manage the architecture correctly in this scenario (Bosch, 2010; Pelliccione, 2014). However, there are few studies addressing the impact of architectural practices on the health of software ecosystems.

2.3 Software Ecosystem Health

Iansiti and Levien argued that a healthy ecosystem provides *"durably growing opportunities for its members and for those who depend on it"*. They pointed out that in an ecosystem, there are a lot of connections among their members. Besides, the health of individual organizations or their products depends on the health of the whole ecosystem. Business decisions and strategies of operation influence directly the health of the ecosystem. The well-being, longevity, and performance of the ecosystem depend on the choices of its members. In this work, the authors also proposed a framework for assessing different strategies adopted by participants of an ecosystem. This approach involves the result of interactions among their members. For this, they defined three indicators for the ecosystem health based on biological ecosystems: robustness, productivity, and niche creation (Iansiti and Levien, 2002).

Robustness expresses the capability of the ecosystem to survive to crises and disruptions. Productivity denotes the ability to transform inputs to new functions or products at a low cost. Last, niche creation consists of the ability of providing new competences aggregating innovation to the ecosystem (Iansiti and Levien, 2002). The indicators proposed by Iansiti and Levien provide the basis for the majority of studies conducted for assessing the health of software ecosystems (da Silva Amorim et al., 2017). However, no theories support the health concepts applied by now.

Our research aims to understand the impacts of architectural practices on ecosystem health. We argue that "the way" software architecture is designed and maintained influences the operation of related software components. Consequently, architectural practices may interfere with the performance and longevity of the ecosystem, affecting directly the ecosystem health. In particular, we aim to improve our knowledge on the influence of architectural practices used during software architecture design and evolution on the ecosystem health. In previous work, we coined the term *"architectural health of software ecosystem"* to represent the *weight* of architectural practices on ecosystem health (Amorim et al., 2017).

3 Related Work

Recently, challenges and gaps related to the health of software ecosystems have attracted the attention of the research community. Several studies have focused on how to define and measure the health by means of indicators (da Silva Amorim et al., 2017). However, none of these studies were concerned with understanding the health achievement from different perspectives, and figuring out the influence of architectural practices on it. In the following paragraphs, we outline studies similar to ours concerning the discovery of processes and practices on open source ecosystems (Jensen and Scacchi, 2004; Bogart et al., 2021; Jansen, 2020), the experience of conduction of qualitative studies on open source communities (Sigfridsson and Sheehan, 2011), the aspects that can influence the health of the ecosystem Dijkers et al. (2018); Avelino et al. (2019); Charleux and Viseur (2019), and approaches to evaluating the health of ecosystems using metrics and some related processes (Franco-Bedoya et al., 2015; Wnuk et al., 2014; Liao et al., 2019; Goggins et al., 2021).

Jensen and Scacchi (2004) explored techniques for discovering development processes from openly available open source software community. They presented a new approach to identify software processes through mining information from development repositories. The use of techniques such as text analysis, link analysis, and the usage and update of the repositories patterns allowed us to understand the processes used by the community. In another work, Jensen and Scacchi (2005) introduced an approach to recognize processes in open source projects, basically by searching for information on the internet. In addition, Bogart et al. (2021) conducted a mix of methods such as survey, repository mining, and document analysis across 18 ecosystems to discover values and practices of breaking changes, and understand how happens the planning, management, and coordination of these breaking changes inside open source ecosystems. Besides, Jansen (2020) proposed a software ecosystem governance maturity model (SEG-M^2), describing governance practices for ecosystem coordinators. He collected the practices for the maturity model from literature studies, also performing techniques of snowballing forward and backward. Following, he tested the practices in six case studies at four companies to validate and improve his model. All of these authors applied different methodologies to discover different practices with distinct goals. Two authors used repository mining to collect open source practices, and understand how diverse process occurs in the ecosystem. The third one collected practices focused on general ecosystem governance to create a maturity model. Our work used another methodology to gather specific practices related to creation and maintenance of software architecture. By using the netnography-based approach, we extracted practices from the literal texts transcription available on websites, also analyzing the context in which the transcription is included.

Sigfridsson and Sheehan (2011) used qualitative methodologies to study principles and practices used by free and open source software (FOSS) communities. This study contributed to relevant issues about the application of qualitative research as virtual ethnography on open source communities. Findings covered the potential problems of applying qualitative methodologies and highlighted the importance of maintaining an active relationship with the core community group. The authors also performed a case study with the PyPy community to provide examples on the challenges faced by FOSS communities. Despite the similarity to our study, our approach included only the participant observation, without engagement in the community and application of questionnaires. Besides, the case study with PyPy spent an extensive period of time, and we used a condensed period of time for each ecosystem. Moreover, we focused only on practices related to create and maintain the software architecture. However, this study provided us with some insights to understand the dynamics and development of an open source ecosystem, in particular, supporting issues about the qualitative approach.

Concerning the health of open source ecosystems, different aspects have been raised by researchers in the last years. These aspects can influence the health of the ecosystem through different forms. Dijkers et al. (2018) explored the effects of the software ecosystem health on the financial performance of the open source companies. They conducted a case study on two open source companies, Cloudera and Hortonworks by looking at the companies individually and after comparing the companies. The results showed that productivity did not influence the relationship between financial performance and ecosystem health. The robustness has a middle influence on this relation, and the niche creation is the main contributor to this relationship. In another study, Avelino et al. (2019) investigated the abandonment and survival of open source projects. In this study, they aimed to discover the frequency of project abandonment and survival, the differences between abandoned and surviving projects, and the motivation and difficulties faced when assuming an abandoned project. Both studies pointed important aspects that can influence on the ecosystem health, however, the practices related to financial performance and abandonment of the project are not directly addressed. Closer to the practices dimension, Charleux and Viseur (2019) studied the managerial decisions, exploring their impacts and community composition on the health of open source projects. They conducted a longitudinal single case study and a qualitative study based on forty-six complementary interviews with open source community members to identify key managerial changes impacting on the community activity. They concluded that the health depends on the business model and governance. By comparing these studies with our work, we perceived that aspects such as financial performance, abandonment of the community, and managerial decisions influence the ecosystem health, and although it is apparently not related to software architecture, we captured some practices, connected with these aspects, that can also influence the health indicators.

Last, regarding the evaluation of ecosystem health, Liao et al. (2019) created an approach to measure the health of the GtiHub ecosystem. They proposed new health indicators to define the structure and resilience of the health of the GitHub ecosystem. Also, they proposed a health prediction method. This study is an example of an analysis quantitative of health and did not consider the influence of practices on the health state as our investigation. Following, we found two studies performing measurement of the health through activities and/or processes (da Silva Amorim et al., 2017). Franco-Bedoya et al. (2015) introduced a model to measure the quality of open source ecosystems. Based on a literature review, they collected several metrics used to measure the health of an ecosystem. After, they analyzed relationships among quality characteristics that can be assessed by these metrics and proposed the QuESo quality model, composed of quality characteristics and measures. Basically, the proposed model considered health as a kind of quality attribute and extracted a set of values to all model areas. In spite of QuESo model to be a large model, it did not address practices and health indicators defined by Iansiti and Levien (2002). QuESo defines quality characteristics and measures, but our work considers practices in a detailed point of view to influence different types of health indicators. The other model was proposed by Wnuk et al. (2014). They conducted an evaluation of the governance model proposed by Jansen et al. (2013) and Jansen and Cusumano (2013). A hardware-dependent ecosystem called Axis was evaluated, considering processes and practices preserving and improving the health of software ecosystems. They analyzed governance activities to gather the degree in which some activities were performed to support the health indicators (productivity, robustness, and niche creation). Governance activities in diverse areas compose this model, influencing the ecosystem health through their indicators. They operate with a set of general practices composing these governance activities. Our work is focused only on practices related to software architecture design and evolution. Finally, Goggins et al. (2021) described the work performed by the Linux Foundation's Community Health Analytics in Open Source Software (CHAOSS) project over four years to understand how to achieve open source ecosystem health. The main strategy adopted by the group was to define metrics to provide a full understanding of the open source project health over time. This group also provided tools to work with these metrics, trying to discover how healthy and sustainable is an ecosystem community. In spite of our study has similar objectives on how to understand the open source ecosystem health, we adopted a different strategy. We focused specifically on architectural practices and their influence on health indicators.

All these studies guided us to investigate and extract information from open source communities and the relationship between practices and ecosystem health. However, there is a lack of approaches and evidence to support these connections, especially, in the software architecture setting. Hence, this work is an initial effort towards understanding these relationships.

4 Methodology

Kozinets (2009) argues a qualitative approach is useful to explore and understand a context, and the choice of the research method must match the nature and scope of the questions. He also claims qualitative techniques help to map new terrain in a constantly evolving on the internet environment. In this environment, netnography is an appropriate approach to study virtual communities that manifest important social interactions virtually. Given the nature of our research questions, aiming to understand the universe of architectural practices in open source ecosystems and their relationships with the health of these ecosystems, we chose the netnography using an observational approach in an environment with ample information available. As a result, we could understand the behavior of the community, as well as its internal structure, to answer our research questions.

We conducted a qualitative study on seven open source ecosystems to approach our research questions. We decided to use mixed empirical methodologies, a netnography-based approach for data collection and a grounded theory-inspired approach for data analysis (Kozinets, 2009; Stol et al., 2016). The netnography-based study adapts common ethnographic procedures for observing the research object in the physical world to be used in the virtual universe mediated by computers. By conducting the netnography-based study, we followed the guidelines proposed by Kozinets (2009) and adapted the steps from the general protocol of the ethnography for our study: study planning, data collection and interpretation, a guarantee of ethical standards, and research representation.

Regarding the netnography-based approach, we performed the following steps: (i) 1st step - we defined our research question aiming to identifying what practices were adopted by each ecosystem and their influences, as well as open source ecosystems as the focus communities of our research; (ii) 2nd step - we chose the seven ecosystems described in Section 4.2; (iii) 3rd step - we joined to the community, but we did not engage as a member, our participation was observational to monitor the dynamics of the work and collect data; (iv) 4th step - we performed data analysis and interpreted the results using a grounded theory-inspired approach; (v) 5th step - we reported the research results in Section 5. **Figure 1** shows the steps defined by Kozinets (2009) and used to conduct our netnography-based study.

Kozinets (2009) introduced the netnography method as an observational and participatory approach, in which the researcher should be immersed for an extended time in a community or culture. The researcher should conduct several online interactions through online interviews, engagement in the community, and observation. Our approach is an adaptation of the method proposed by Kozinets (2009). We named the approach netnography-based because we only performed the observational part without interacting directly with members of the communities. Also, we could not do an immersion for an extended time into the seven ecosystems.

Concerning ethical standards, Kozinets (2009) argues the netnographer participatory should follow a short protocol to identify himself and ask permission to work with the community since he is engaged in the community. In our case, when the netnography is only observational, we had not needed to ask permission to the community. All information collected was published in public space and the access was free. In addition, we respected all copyright, and we cited and recognized data from each community when published. Also, we did not publish the names of members or pseudonyms to identify individuals in particular. Our research was conducted by characterizing data related to participants and the community, avoiding damage to community members, as well as following the guidelines defined by the netnography approach.

Kozinets (2009) also suggested two types of data analysis for netnography studies: analytical methods based on coding and hermeneutic interpretation. We chose to apply the first one because it supported us in handling the whole volume of data collected during our study. The grounded theory method generates a theory that considers the context, conditions, strategies, and results from data. This method is used in qualitative research, inductive paradigm, to develop a theory from situations in the real-world (Stol et al., 2016). However, our goal was not to create a theory but to provide an understanding of the influence that architectural practices exert on the health of open source ecosystems. In addition, we aimed to gather useful contributions that provide new foundations to consolidate such influences. Therefore, we decided to borrow some elements from the grounded theory method to restrict coding techniques (data analysis, constructing codes and categories, constant comparative method, writing memos), characterizing a grounded theory-inspired approach following the guidelines for coding from Charmaz (2006); Saldaña (2009); Stol et al. (2016). Charmaz (2006) also states data should be analyzed to emerge a theory, not from preconceived deduced hypotheses. In our case, we applied grounded theory techniques to capture the connection between architectural practices and existing health indicators. The concepts of ecosystem health are preconceived, but their possible existing connections have been not investigated yet.

Finally, by applying the netnography study, as Kozinets (2009) states, the researcher is compelled to make assumptions about cultural meanings that he does not fully understand. He offers a purely descriptive analysis of the content found online, when the researcher is not a participant in the community. In our study, some practices were already defined explicitly on the webpages or there was data allowed to deduce an adoption of an existing practice. Concerning the second approach, Charmaz (2006) advocates that grounded theory uses an abductive inference method since it considers the possible theoretical explanations for the data, and then form hypotheses to search for possible explanations until it finds the most probable explanation. We applied the grounded theory-inspired approach to uncover how connections between software architectures and ecosystem health happen, applying the reasoning about experience for making theoretical conjectures jointly with their verification through additional experiences. The use of grounded theory-inspired is to discover possible explanations for these connections, aimed to find the most probable explanation. Through coding techniques, the approach allowed, not to generate a theory, but to clarify relationships and influences existing between two different conceptual worlds.

4.1 Research Questions

The goal of this study is to identify and understand possible practices used to create and maintain the software architecture of software ecosystems. By elucidating potential reasons for making architects adopt specific practices, and figuring out if any architectural practice might influence the open source ecosystem health, we defined the following research questions:

RQ1. *What are the factors that can influence the adoption of architectural practices in the design of open source ecosystems?*
RQ2. *How can architectural practices influence the health of the open source ecosystem?*

As described previously, the software architecture has a crucial role in the development and evolution of a software ecosystem. Problems with the platform can cause serious damage to the ecosystem. Therefore, the investigation of factors that influence the practices adopted by architects, and the effects of these practices on the ecosystem health, contribute to the definition of ecosystem strategies. Findings can guide decisions and concerns about the software architecture that help keep the whole ecosystem balanced.

4.2 Research Context

Our study analyzed seven open source ecosystems: GitLab, Jenkins, KDE, MapServer, Node.js, Open edX, and WordPress. In accordance to the classification proposed by Manikas (Manikas and Hansen, 2013), GitLab, Open edX, and WordPress are hybrid ecosystems that adopt the open source strategy concerning software platform development. In addition, all the ecosystems fit to the definition of software ecosystems provided by Bosch and Bosch-Sijtsema (Bosch and Bosch-Sijtsema, 2010). They have a community with internal and external members working to create relevant solutions upon a software platform. The criteria used to choose these ecosystems were their degree of openness and the availability of documentation on their websites. This enabled us to access documents, diagrams, artifacts, and code. Besides, ecosystems with diverse sizes and domains were considered to allow a large range for the generalization of practices adopted in the open source context. In addition, we observed the working environment of the community. This way, we could identify practices adopted by them to capture the dynamics in their entirety.

- **GitLab**[1] started as a git-based software repository manager. It was created in Ukraine by Dmitriy Zaporozhets in 2011. Nowadays, their functions were extended, and it supports all stages of the DevOps lifecycle for product, development, quality assurance, security, and operations of a software project. The code is mainly written in Ruby. GitLab has a different characteristic regarding its license because it has two software: GitLab CE which is open source and GitLab EE which is closed source. The company proprietary of the GitLab EE manages the open source project. We consider our work the GitLab CE community, which follows the behavior of open source ecosystems.
- **Jenkins**[2] is an automation server that helps to automate the non-human part of the software development process. It provides automation for several tasks related to building, testing, and delivering or deploying software, implementing continuous integration, and contributing to implement continuous delivery. The initial release was launched in February 2011 and came from a project originally named Hudson. After a dispute with Oracle, which had forked the Hudson project, Jenkins was a name chosen for the project through an election in the community. The code is mainly written in Java. Nowadays, Jenkins provides various infrastructure tasks and an extensive library of over one thousand and three hundred plugins.
- **KDE**[3] provides a platform to easily build new applications upon, not to mention the advanced graphical desktop and set of applications for communication, work, education, and entertainment. KDE was founded by Matthias Ettrich in 1996. It has a strong community spread by several countries sharing experiences and contributing to strengthening one of the largest active open source ecosystem communities. The code is written mostly in C++ into a mature codebase. The platform KDE Frameworks allows building all kinds of applications upon. Presently, KDE has the ambition of providing a reliable monopoly-free computer solution.
- **MapServer**[4] is a platform for publishing spatial data and interactive mapping applications to the web. It was created at the University of Minnesota in 1994. Written in language C, MapServer renders geographic data and allows the image of maps to work on the internet, providing a spatial context for these maps. This ecosystem is supported by several organizations that manage funds for the adoption of the open geospatial technology.
- **Node.js**[5] provides a JavaScript run-time environment to perform JavaScript code outside of a browser. It allows for building dynamic web page content before the page is sent to the final browser. Applying an event-driven non-blocking I/O model, Node.js has a lightweight for applications running across distributed devices. The initial release was in 2009, and it is written in C++, JavaScript, and Assembly.
- **Open edX**[6] is the open source platform for massively scalable learning. As a provider of massive open online courses (MOOC), the platform produces weekly learn-

[1]https://about.gitlab.com

[2]https://jenkins.io
[3]https://kde.org
[4]https://mapserver.org
[5]https://nodejs.org
[6]https://open.edx.org

Table 1. Open Source Ecosystems Studied

Name	Foundation	Age	Klocs	Contributors
GitLab	GitLab Inc.	8 years	752,409	2,530
Jenkins	Continuous Delivery Foundation	8 years	1,031,859	2,132
KDE	KDE Foundation	22 years	57,899,444	5,539
MapServer	OSGeo Foundation	25 years	413,852	147
Node.js	Node.js Foundation	9 years	6,340,475	2,871
Open edX	edX Inc.	7 years	1,114,609	727
WordPress	WordPress Foundation	16 years	560,703	541

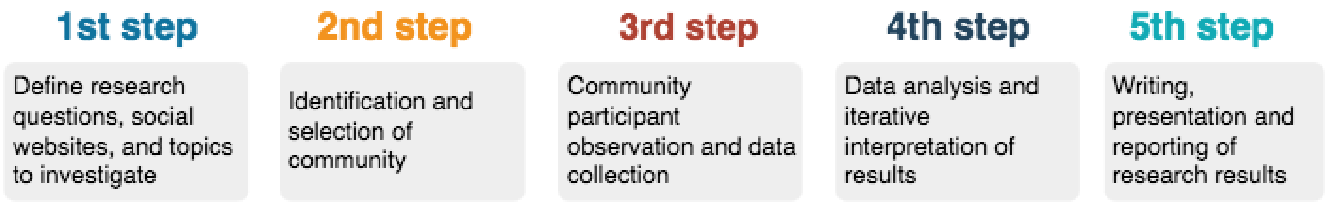

1st step
Define research questions, social websites, and topics to investigate
2nd step
Identification and selection of community
3rd step
Community participant observation and data collection
4th step
Data analysis and iterative interpretation of results
5th step
Writing, presentation and reporting of research results

Figure 1. Steps to perform the netnography methodology (Kozinets, 2009)

ing sequences of courses. These courses include tutorial videos, online discussion forums, and textbooks. Open edX was created by The Massachusetts Institute of Technology and Harvard University in May 2012, and it is mostly written in Python.

- **WordPress**[7] is an online publishing platform that supports users, even those without a technical background, to quickly create blogs, apps, or websites. They keep the site free, but also offer some paid plans with tools to improve the user experience. WordPress is built on PHP and MySQL. It was created in 2003 by Mike Little and Matt Mullenweg.

Table 1 presents more information about the studied open source ecosystems in accordance with Open Hub[8].

4.3 Data Collection

Conducting data collection from open source ecosystem, we have extracted data from different online sources. The set of communication tools found in open source ecosystems such as forums, newgroups, blogs, gitHub pages, and wikis, as well as some external websites connected to the ecosystem such stack overflow, reddit, bugzilla, and others provides relevant information for researchers outside the community context. It is possible to observe behaviors, rules, and values that guide the community steps. The communication leaves traces that are easily observable, recorded, and copied. Information can be widely captured and recorded. In this environment, the processes of accessing and analyzing data are facilitated (Kozinets, 2009). Although open source ecosystems make available several dynamic communication channels such as internet relay chats (IRCs), we kept our focus on extracting data from widely published information about work in the community on webpages reached through links provided by the ecosystem home pages. We did not consider analyzing conversations in chat format to analyze official practices released by the community board. For each ecosystem, we started the research from the initial page (Home page) and navigated to several weblinks.

Since the practices were identified, we classified them in a set of seven software architecture key areas introduced by our previous work (Amorim et al., 2017). Each practice should fit into one of these areas. They represent the architectural design decisions through a division in logical areas. Architects should focus on these key areas to exercise their activities such as architectural knowledge, external management, choice of technology, resources management, design-making, quality management, and change management. Each key area is classified in one of the three software ecosystem views: community, technical, or business. The key areas are classified according to the objective of the ecosystem views. In this way, the practices of each area fit by affinity to the objective of each ecosystem view in which it is allocated. For example, the key area named architectural knowledge that encompasses practices of knowledge management in the community is allocated to the community view and so on. **Table 2** presents the mapping of the key areas by ecosystems views.

Table 2. Mapping of key areas by views (Amorim et al., 2017)

View	Key Areas
Community	Architectural Knowledge External Management
Business	Choice of Technology Resources Management
Technical	Design-Making Quality Management Change Management

- *Architectural Knowledge*. Practices related to this key area support tasks to manage and share architectural knowledge with the community.
- *External Management*. External Management encompasses practices keeping the diversity of contributions of the community and support the management of ar-

[7]http://wordpress.org
[8]https://www.openhub.net/

chitectural interfaces used by third-party developers.

- *Choice of Technology*. This key area is related to the choice of the best technologies to build the systems.
- *Resources Management*. This key area comprises practices used to supporting architectural process.
- *Design-Making*. This key area encompasses practices related to taking technical decisions.
- *Quality Management*. This key area is responsible for controlling all practices satisfying quality criteria previously defined.
- *Change Management*. Change Management controls all practices used to regulate and keep the balance into the architecture facing all the implemented changes.

Based on these seven areas, we defined fourteen *search topics* to facilitate the search of the practices on the websites. These *search topics* aimed to cover common topics of software development and that could be included within a key area. In addition, a *search topic* can cover more than one key area. For example, Business organization could have practices in several key areas.

Table 3 shows these *search topics*.

When we identified a candidate practice that could fit into a search topic, we registered information such as ecosystem foundation, date, url, description, type of website, and the topic of activity of this practice. These records concentrate data about the practice found on the websites. The *description* is exactly the clipping of the text as found on the website. For instance, regarding the MapServer ecosystem, we collected the description *"PSC management responsibilities: setting the overall development road map and project infrastructure (e.g. GitHub, CVS/SVN, Trac/Bugzilla, hosting options, etc...)"*.

Table 3. Search topics of the netnography-based study

ID	Search Topics	ID	Search Topics
1	Business organization	8	Release launching
2	Communication	9	Resources
3	Coding	10	Security
4	Documentation	11	Taking decisions
5	Financial	12	Training
6	Meetings	13	Translation
7	Quality	14	Tests

Data extraction was focused on the documentation and guidelines on each web page. We also analyzed information about activities to create and manage the software architecture found in code repositories. However, we did not conduct a code analysis itself to identify practices.

Navigation always started on the initial web page of the ecosystem. For each search topic, we searched for pages with related information to the current search topic, guiding the navigation on the ecosystem website. We read the entire page in search of some information related to the current search topic. Finding some related information, we analyzed it to see if any practice could be extracted. Then, we went through the links on the page to search for more information related to the current search topic. This search process through the pages

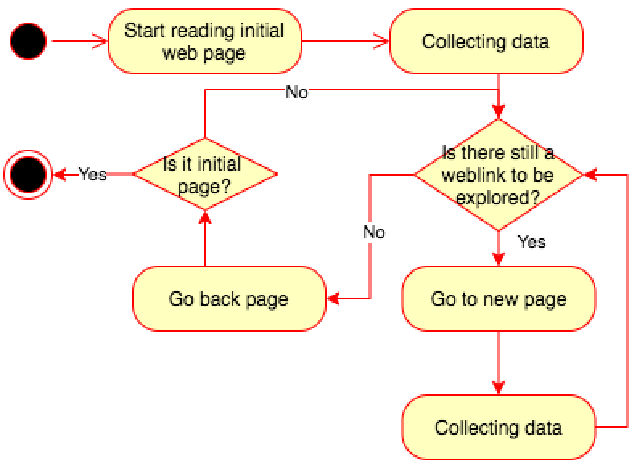


Figure 2. Steps to collect data

continued until the page no longer contains related information and no further links were found on the page related to the search topic. In the majority of cases, we arrived at pages where there were no more links and no way to go to another page. Besides, there were a few cases, where we arrived at cycles clicking by links that sent us to go back to the initial page. During each search process, we focused our data collection only on information related to the *search topic* conducted at the moment to capture architectural practices related to the topic. **Figure 2** presents the steps performed in the data collection process. We did not record the different levels that we went through in the search for practices, since each ecosystem has a different infrastructure for building and organizing the pages on the website. Besides, the level of navigation between pages varied according to each current search topic in the same ecosystem.

Regarding the registration process, Kozinets (2009) argues smaller or more limited investigations of online communities and cultures may employ manual coding, categorization and classification, as well as hermeneutic interpretive analysis, to gain insights. Besides, a semiautomatic (manual and computer-assisted) method can be used to organize different levels of coding and abstraction using a spreadsheet tool. Our research was restricted to identifying and collecting architectural practices and their context. Although we had collected a reasonable amount of data, we were able to manage data using this semiautomatic method. Therefore, we used a general spreadsheet tool to support data collection and data analysis processes, instead of using sophisticated software packages.

At the end of this process, a set of architectural practices were identified. The selection process aimed at identifying all practices related to software architecture in some way, based on the researcher's background. Moreover, we included practices that are not directly related to the technical view, but which are also relevant practices for the business and the community views. For example, the practice *"(P33) Define a financial board to manage the financial resources"* influences the software architecture when the financial board team provides resources to support the architecture such as hardware, software, developers, and it also defines market guidelines that need to be met at any given time. It is closely connected to the business area, but also ensures the provision of technical operation.

4.4 Data Analysis

From the netnography-based study, we started the process to discover the architectural practices adopted in the ecosystems. So, the first step of the data analysis was to define a common text for practices found in the seven open source ecosystems. Based on the *description* of each practice, we defined a *practice specific* and a *practice catalogued*. The *practice specific* describes the practice in a more abstract level, and *practice catalogued* describes the practice in a formal way suitable for all the observed ecosystems. For example, regarding the MapServer ecosystem, we collected a description *"PSC management responsibilities: setting the overall development road map and project infrastructure (e.g. GitHub, CVS/SVN, Trac/Bugzilla, hosting options, etc...)"*. So, the practice specific was defined as *"PSC committee is responsible to define technology"*, and the practice catalogued was *"The project leaders define specific technology that impacts into the work of project community"*. By analyzing the practice specific, we checked if the practice fit into any existing practice catalogued that could be attributed to it, otherwise, this practice originated a new practice catalogued. This process was conducted for all practices found in all ecosystems. As a result, we catalogued fifty architectural practices.

Following data analysis, we started to analyze all practices and information about their context to answer the research questions. According to Saldaña (2009) coding is the initial step before a powerful analysis and interpretation for reporting. He states that the quality of the code used is essential to gather important information from the research history. In addition, qualitative codes allow the establishment of patterns and facilitate the development of categories from their connections. Coding is a way to organize things systematically to infer some classification (Saldaña, 2009). Moreover, research should be critical about data asking questions to identify relevant practices (Charmaz, 2006). In this context, we adapted questions from (Charmaz, 2006) to evaluate the practices: (i) What are they doing?; (ii) What are they saying?; (iii) Who is doing?; (iv) Why are they doing?; and (v) When are they doing it?

During data collection, we addressed our search to answer these questions and registered information for each practice identified in the process. For example, the practice (P34) in WordPress ecosystem had the following answers: (i) What are they doing? - *"Provide financial resources to support meetings face-to-face"*; (ii) What are they saying? - *"Companies that sponsor WordPress community events support the WordPress open source project by helping our volunteer-organized, local events provide free or low-cost access for attendees"*; (iii) Who is doing it? - *"global community sponsors"*; (iv) Why are they doing? - *"They believe that a casual, non-commercial, and educational event permit discussing WordPress issues face-to-face easily and strengthens the community"*; and (v) When do they do it? - *"during the WordCamp, an annual conference for local WordPress communities"*. A similar practice was also observed in the other 6 ecosystems (see appendix A) and the answers to these questions were recorded by each ecosystem.

By conducting the data analysis in our grounded theory-inspired of the answers jointly with other data collected, we performed the followings steps for coding: (a) Identifying and labeling data using a *code* reflecting its meaning; (b) Performing a constant comparison searching for patterns that point to *concepts*; (c) Grouping similar concepts in a high-level abstraction called *categories*. This process occurs until no new conceptual relationships emerge for categories; (d) Writing *memos* to describe ideas and relationships among codes, concepts, and categories; (e) Developing an understanding of the studied phenomenon, in our case, the factors that motivate the adoption of the practices and the influence of the practices on the health of open source ecosystems.

The steps described previously present some key components of the grounded theory (Charmaz, 2006; Coleman and O'Connor, 2007):

- **Codes.**They are words used to provide meaning to the data. They summarize and reflect the experience described.
- **Concepts.** They are ideas derived from a set of codes and organized in a high-level abstraction.
- **Categories.** They are a classification that explains ideas or processes gathered from data expressing common patterns in various codes.
- **Memos.** They are notes describing thoughts, capturing comparisons and connections from data. They also delineate directions and issues that should be considered.

Figure 3 presents our research analysis processes for the grounded theory-inspired approach. Data collection and analysis processes were conducted by one researcher. Other researchers subsequently analyzed the results found and the written article, requesting clarification and reviews when necessary. They had complete access to the entire database and information about the process. The needed adjustments were done in the review process. As Seaman (1999) pointed out, *"any proposition that the researcher synthesizes must be clearly and strongly supported by the data"*. This way, we constructed a set of propositions about factors that influence the adoption of architectural practices based on evidence collected on the studied ecosystems. All factors synthesized were inferred using abductive logic, and categorized and labeled with their properties Stol et al. (2016).

Regarding the influence from architectural practices on the health indicators, the process required an additional literature review to support the findings. Previously, Jansen (2020) stated the relationship between practices and health indicators, as well as its effects of one on the other, are unknown. Currently, determining these relationships and effects still constitute a considerable scientific challenge. Hence, looking for a solution, we observed that Stol et al. (2016) presented that the literature could support the process of grounded theory, using concepts and improving theoretical sensitivity as additional data sources. In addition, Charmaz (2006) also suggests using a literature review to support the work of analysis. So, we consulted concepts, metrics, and arguments of the health indicators introduced by Iansiti and Levien (2002) to make explicit and rational connections between our concepts inferred from data and the health indicators presented in this earlier study and developed insights to answer our research questions, allowing us to make claims from our grounded theory-inspired

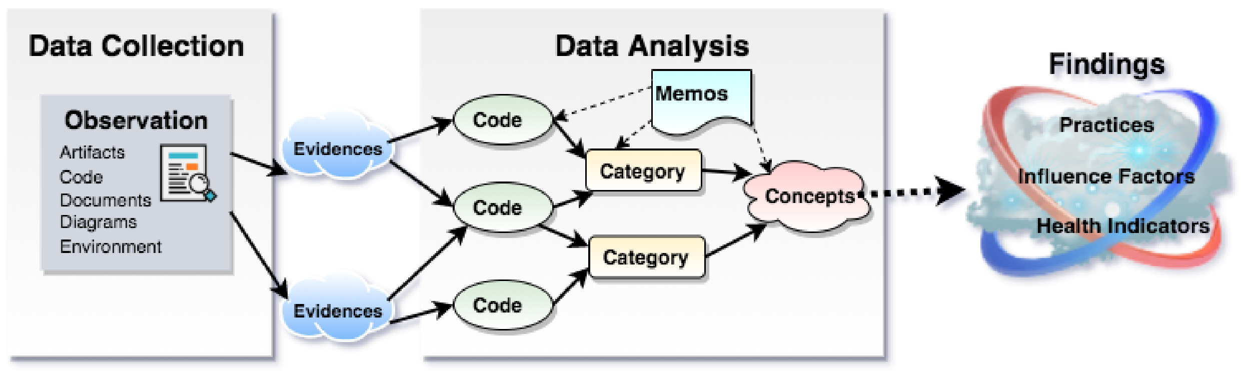


Figure 3. Research analysis process

approach. As a result, from the codes, categories, and concepts that emerged during data analysis, and supported by concepts from literature, we derived ideas and constructed relationships between architectural practices and health indicators, which enhanced the understanding of reasons for adoption and influences on the health indicators. In the same way, we used the literature with concepts introduced by Bass et al. (2012) to support the analysis of codes, categories, and concepts to find the factors that influence the adoption of architectural practices. **Table 4** presents the codes that emerged from the study. The codebook is available at http://doi.org/10.6084/m9.figshare.23657856. **Table 5** presents the categories emerged from the study.

Table 4. Codes emerged from study

Codes		
Documentation	Reuse	Changes
Sharing Knowledge	Novelties	Compatibility
Design Decisions	Efficacy	Obsoleteness
Problems	Patterns	Marketing
Security	Knowledge	Meeting
Automated	Translate	Money

Table 5. Categories emerged from study

Categories
Budgeting
Design-Making
Innovation
Knowledge Management
Quality
Standardization

Figure 4 presents a short example of how the data analysis in the coding process was conducted. For instance, the practice specific "Provide an official channel to publish all changes to the community" was observed in the KDE and MapServer ecosystems. The practice catalogued that can fit in all ecosystems had been defined as *"(P43) Publish widely the architectural changes for the community"*. In addition, we also collected additional information about the context of these practices, such as "core changes in MapServer can affect existing applications" and "KDE provides the KDE.News as the official news channel". Furthermore, from the data analysis process emerged the following codes: Documentation, Sharing Knowledge, and Design Decisions. Besides, the category Knowledge Management also emerged, jointly with the concept *"Everybody must be aware of changes in the architecture that will impact in their work"*. Based on this concept, our analysis emerged, reasoning about what factors can influence the adoption of the practice. For factors, the reasoning was *"the knowledge of the architects should be shared with the community to guide the work of developers"* represented by the factor Experience, and *"Everybody must know about changes to adapt their applications for the new scenarios and avoid break of operation"* represented by Business Goal. For influences on the healthy indicators, the reasoning was *"Communications of critical changes improve the interactions among organization and third-party that have applications influenced by these changes"* represented by Trustworthiness, and *"Communicate everyone about changes in the architecture avoid breaks of applications considering lack of information"* represented by Robustness. **Table 6** presents graphic symbols that are used on the graphic schemes of the examples in this study. These symbols were defined by authors to improve the graphic representation.

Table 6. Semantics of shapes applied on the graphic scheme

Symbol	Description
☁	Concepts
▢	Category
⬭	Codes
▢	Practice
▭	Health Indicator
▭	Influence Factor
▭	Texts (Reasoning)
⇢	Connections from Data
→	Connections from Inferences

5 Findings

This section presents the findings of our study, including the architectural practices identified (Section 5.1), organized

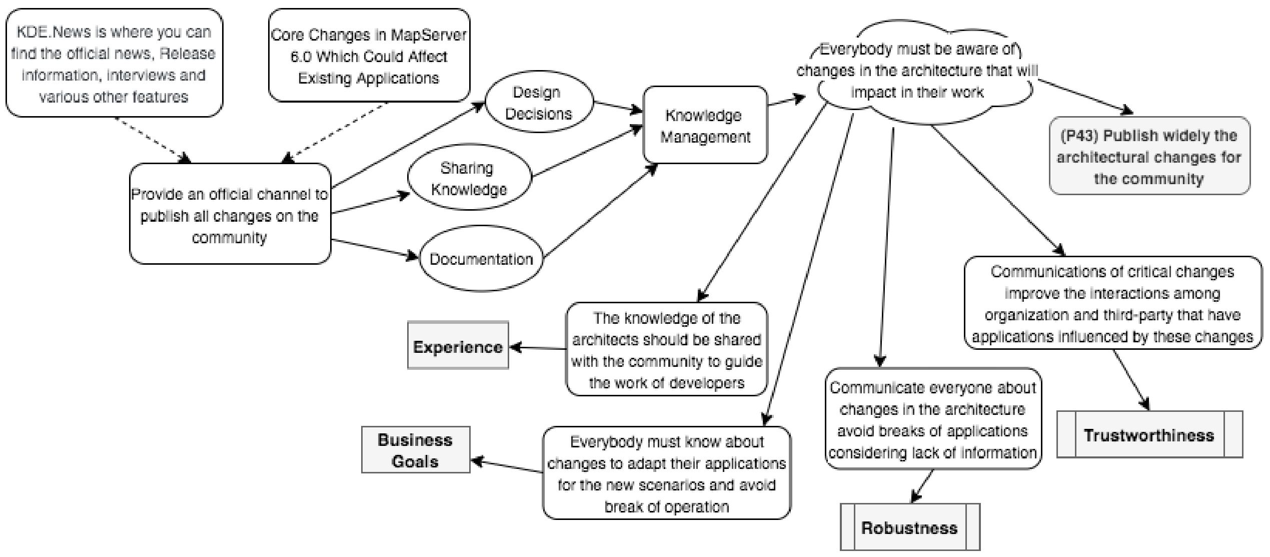

KDE.News is where you can find the official news, Release information, interviews and various other features
Core Changes in MapServer 6.0 Which Could Affect Existing Applications
Provide an official channel to publish all changes on the community
Design Decisions
Sharing Knowledge
Documentation
Knowledge Management
Everybody must be aware of changes in the architecture that will impact in their work
(P43) Publish widely the architectural changes for the community
Communications of critical changes improve the interactions among organization and third-party that have applications influenced by these changes
The knowledge of the architects should be shared with the community to guide the work of developers
Experience
Communicate everyone about changes in the architecture avoid breaks of applications considering lack of information
Trustworthiness
Business Goals
Everybody must know about changes to adapt their applications for the new scenarios and avoid break of operation
Robustness

Figure 4. Emergence of the influences from underlying concepts

according to factors that have an influence on their adoption (Section 5.2) and the rationale behind their influence on ecosystem health and its indicators (Section 5.3).

5.1 Architectural Practices

Fifty architectural practices used by the studied open source ecosystems were identified during our study. **Table 7** presents all practices captured in our study by the nethnography-based approach. Following, the **Table 8** presents a categorization for all architectural practices with respect to key area and ecosystem view. In addition, **Table 9** introduces only a small subset of architectural practices (P1-P7) identified for each ecosystem, where the ecosystem that has the practice checked, means that it was found in its environment by the researchers. For instance, the practice *(P7) Provide a newcomer-specific page or portal guiding their first steps, including development information* has been adopted by the seven ecosystems. P7, along with other six practices (P10, P13, P18, P27, P35, P39) were adopted by all ecosystems. Five practices were used by one ecosystem only (P46, P47, P48, P46, and P50). **Appendix A** shows the complete set of practices and ecosystems where they were found.

5.2 Factors that influence the adoption of architectural practices

During architecture design, there are many influences guiding/forcing the software architecture towards some direction (Bass et al., 2012). These influences are diverse and depend on the environment in which the architecture will operate. To understand these factors of adoption for a practice allow us to create governance mechanisms to define whether a practice should be adopted, influencing the final result on the health of the ecosystem. Some influence factors are related to requirements, technical environment, and experience of the architects. For them, software architecture is constrained by a large variety of sources. Some influences can be implicit and others explicit, however, it is difficult to capture all properties required by the architecture. Finally, there are gaps that can cause conflicts among the goals of the software architecture (Bass et al., 2012).

Regarding this reality in the ecosystem scenario, architects should also consider demands from third-party and be aware that external business goals also suffer the impact of changes in the platform. In addition, they must take application response time into account when making necessary corrections by releasing software versions. As the cost of innovation and development are shared with the community, business strategies, and development resources should be considered together. In our study, we found five relevant factors that we understand performing influences in our set of fifty practices: Business Goals, Experience, Requirements, Resources, and Time-to-market. **Table 10** shows each factor with the practices that are influenced by them.

5.2.1 Business Goals

Business goals refer to strategic goals that the software ecosystem aims to accomplish. These goals determine positions to be achieved and drive specific tasks and deadlines to conduct to these positions (Bass et al., 2012). Business goals are very important factors to guide all activities in the ecosystem. Effective goals will impact directly on the success or failure of the whole ecosystem. In order to build a successful architecture, the architect should understand his competitors, software products, and strategies. Moreover, she should know the key factors in the business environment that affect the progress of the organization (Bredemeyer et al., 2000). In the ecosystem, business goals express the desires of internal and third-party developers. Architects should balance the goals of third-party that behave as collaborators and at the same time competitors among themselves. There are different concerns that should be aligned to promote the success of the ecosystem.

Our research identified eighteen practices that may have been directly influenced by business goals. For instance:

- A business goal such as "provide the market needs and keep the fidelity of customers" motivates the practice

Table 7. Architectural practices found in the netnography-based approach

Id	Architectural Practices	Id	Architectural Practices
P1	Create personal blogs and/or wikis to inform about the development and architectural issues	P26	Do online meetings in a timezone adequate for most of the community
P2	During code review, provide feedback information about architecture, good practices to code, doing refactoring and show the best way to solve problem	P27	Provide several online meetings to discuss architectural problems by IRC, email
P3	Document APIs constantly	P28	Create partnerships with third-parties to solve problems of the core and their interfaces
P4	Provide internal or third-party mentoring programs to train newcomers	P29	Setting a code of conduct to avoid mistreatment among members
P5	Provide code recommendations, defining a standard in the community	P30	Provide a message template for newcomers to use to interact with the community
P6	Keep a register of meetings available to community to know all decisions of the meeting	P31	The organization board defines some technologies that should be used by the whole community as tools for testing, communication, coding review, bugging manager, and navigation
P7	Provide a newcomer-specific page or portal guiding their first steps, including development information	P32	The organization board provides hardware and software resources to be used by the community
P8	Identify and dismiss outdated information on websites	P33	Define a financial board to manage the financial resources
P9	Provide generation of (semi-)automated documentation filtered to up-to-date information relevant to newcomers	P34	Provide financial resources to support meetings face-to-face
P10	Answer questions on the mailing list quickly	P35	Provide meetings (sprints) face-to-face to accelerate the development of critical issues and solve development problems with interdependent modules
P11	Create a detailed step-by-step tutorial linking information about common problems and possible solutions	P36	Define minimal quality criteria requirements to add an application to the ecosystem (documentation, automatic tests, dependence restrictions)
P12	Provide updated official documentation about code's organizational structure, and how the components, modules, classes, and packages are related to each other	P37	Define a team to test performance and behavior of the application
P13	Support the participation of the ecosystem on aggregators' sites such as stack overflow, reddit, hacker news, and so on	P38	Use some tools to compute some quality metrics
P14	Provide a dictionary to newcomers to facilitate their learning of the technical jargon, acronyms of the community	P39	Use automatic tests to gather problems with the code recently added
P15	Provide video-classes or tutorials about introducing the ecosystem, installing technologies, configuring the development environment, and using the dependencies with other ecosystems	P40	Provide an automatic process to launch release of applications
P16	Provide a manifesto explaining requirements of an application belonging to the ecosystem	P41	Divide the parts of the software in layers defining restrictions for managing dependencies among the layers
P17	Provide video-conference sessions with questions and answers about relevant topics	P42	Discuss with the community about critical changes into architecture that will impact in the applications
P18	Provide information about the translation process (how to participate and tools used)	P43	Publish widely the architectural changes for the community
P19	Provide documentation about translation rules that developers must follow to prepare code for other languages	P44	Build the architecture based in plug-ins to facilitate the coupling of applications
P20	Keep the backward compatibility for a medium or long time to allow the community update their software	P45	Provide guidelines with a set of steps to be followed by developers how to add code to repository
P21	Provide different levels of security access for the parts of the ecosystems in accordance with the degree of commitment and tasks in the ecosystem	P46	Provide a virtual machine with pre-configured build environments, web-based IDEs, or a container management tool
P22	Point newcomers to easy tasks filtered by difficulty, skills needed, and topics	P47	Inform newcomers about technical background required. Identify which specific technologies they need to know or they should learn to achieve their goal of contributing to the ecosystem
P23	Use tools to publish known cyber security vulnerabilities. For example the Common Vulnerabilities and Exposures (CVE)	P48	Tag all tasks in accordance with degree of difficulty (easy, medium, difficult)
P24	Keep a team to manager the security problems registered	P49	Provide a group for gardening to care the global state of ecosystem
P25	Preference to use the English language written to avoid misunderstanding	P50	Keep the list of tasks updated informing about who is working on the solution

Table 8. Categorization of the practices by views and key areas

View	Key Areas	Practices
Community	Architectural Knowledge	P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P45
	External Management	P20, P21, P22, P23, P24, P25, P26, P27, P28, P29, P30, P46, P47, P48, P50
Business	Choice of Technology	P31
	Resources Management	P32, P33, P34
Technical	Design	P35
	Quality Management	P36, P37, P38, P39, P40, P49
	Change Management	P41, P42, P43, P44

(P36) Define minimal quality criteria requirements to add an application to the ecosystem (documentation, automated tests, dependence restrictions). Defining quality rules for applications will provide a good quality service to support such business goals.

- A business goal such as "attract and retain developers and customers in the ecosystem" induces the practice *(P44) Build the architecture based on plug-ins to facilitate the coupling of applications*. It contributes to protecting the core from unwanted changes avoiding damaging the code and at the same time allowing the integration of several external applications easily.

5.2.2 Experience

Experience describes the effect of the knowledge acquired by the software architects considering their practices. Design decisions are guided by the acquired background through the experience of success and/or failure (Bass et al., 2012). Architects are already prepared to take decisions considering generic knowledge such as patterns and guidelines. However, they also consider past architectural decisions in the architectural design of other systems. Over time, they create best practices based on specific projects. So the challenge is to adapt and use these past decisions for new projects with generic knowledge (Weinreich and Groher, 2016). Bringing these challenges to the ecosystem setting, in a plural environment, many architectural decisions should be taken in conjunction with the community. We observed that architectural knowledge comprises the experience of various architects. The architectural decisions should also consider the impact on third-party applications and share the knowledge with the community, ensuring ecosystem surveillance.

We have identified that the experience of architects influenced directly twenty-one practices. For instance:

- The architectural practice *(P2) During code review, provide feedback information about architecture, good practices of coding, do refactor and show the best way to solve problems* shares the experience of the architects during code reviews. This practice is very common and important to prepare newcomers and guarantee architectural knowledge sharing. Furthermore, architectural knowledge is important to support and define tools and environmental issues.
- The practice *(P31) The organization board defines some technologies that should be used by the whole community as tools for testing, communication, coding review, bugging manager, and navigation* is also related to experience. The experience of architects should support the decision about the best tools to work within the context of the software ecosystem.

5.2.3 Requirements

Requirements are the basis for building the software architecture. In open source ecosystems, requirements are discussed with the community; virtual or physical meetings are used to decide what is important to be launched into the next version. Besides, sharing the cost of innovation, external members can develop new features apart from the community. These features can be added to the platform in the future. The software architecture must address requirements, and architectural practices are adopted to allow the fulfillment of the requirement.

In our research, eight practices related to this factor were found. So, the adoption of these practices is encouraged to contribute to the achievement of goals defined by the requirements. For instance:

- *(P3) Document APIs constantly* is used to reduce the impact of changes for third-party. This is because, to meet the requirements, some changes to the platform may be necessary, including into the API. The API should be documented constantly, allowing updating applications on top of the platform.
- The practice *(P16) Provide a manifesto explaining requirements of an application belonging to the ecosystem* defines criteria to applications be engaged into the ecosystem platform. As a result, all projects of the ecosystem already start with a set of predefined requirements.
- *(P49) Provide a group for gardening to care the global state of ecosystem* is a gardening practice adopted by the KDE ecosystem to provide a team to care for important bugs, find stale review boards, and ping people to review them. The idea is to care about the general state and keep alive the projects. So that, gardening activities ensure that many requirements with problems or stopped activity can be accomplished.

5.2.4 Resources

Resources describe the elements used to accomplish tasks. These resources encompass software, hardware, people, environment, money, and so on (Bass et al., 2012). The architect also determines some dimensions of the software environment, including the infrastructure. Choosing the right tools helps to quickly achieve architectural goals. This choice is based on a set of aspects such as the understanding of the domain, business environment, costs, integration with other

Table 9. A subset of architectural practices

Id	Architectural Practices	GitLab	Jenkins	KDE	Map-Server	Node.js	Open edX	Word-Press
P1	Create personal blogs and/or wikis to inform about the development and architectural issues	✔		✔	✔	✔	✔	✔
P2	During code review, provide feedback information about architecture, good practices to code, doing refactoring and show the best way to solve problem	✔		✔	✔	✔	✔	✔
P3	Document APIs constantly	✔		✔				✔
P4	Provide internal or third-party mentoring programs to train newcomers	✔	✔	✔	✔			✔
P5	Provide code recommendations, defining a standard in the community	✔		✔	✔		✔	✔
P6	Keep a register of meetings available to community to know all decisions of the meeting	✔		✔	✔	✔	✔	✔
P7	Provide a newcomer-specific page or portal guiding their first steps, including development information	✔	✔	✔	✔	✔	✔	✔

Table 10. Influence Factors with Practices

Influence Factors	Practices
Business Goals	P4, P7, P13, P14, P16, P17, P18, P20, P25, P28, P29, P36, P38, P43, P44, P46, P47, P49
Experience	P1, P2, P4, P5, P9, P11, P12, P15, P17, P19, P20, P22, P27, P31, P35, P37, P41, P42, P43, P45, P50
Requirements	P3, P16, P19, P20, P27, P36, P37, P49
Resources	P2, P4, P6, P7, P13, P15, P17, P21, P23, P24, P25, P28, P31, P32, P33, P34, P35, P38
Time-to-market	P1, P3, P5, P6, P7, P8, P9, P10, P11, P14, P20, P22, P24, P26, P27, P46, P48, P30, P31, P37, P38, P39, P40, P50

systems, and so on (Jade, 2019). The environment of ecosystems provides a rich set of features regarding the diversity of members, and at the same time, some resources may be scarce. For example:

- *(P32) The organization board provides hardware and software resources to be used by the community* means that the infrastructure is provided by the organization. However, some resources cannot be provided. Due to the large range of possible configurations for a system, some configurations are difficult or expensive to be offered and tested.
- The practice *(P28) Create partnerships with third-party to solve problems of the core and their interfaces* allows partners to provide the support needed to work with some scarce resources. This way, the community continues developing innovation and aggregating value to different features.
- Another important practice is *(P34) Provide financial resources to support meetings face-to-face*. Face-to-face meetings promote knowledge sharing and strength social interactions. In many cases, these events can help to solve important issues that are hampering or harming the ecosystem. Therefore, the money of the community should be applied to the benefit of the community itself, increasing members' interaction.

5.2.5 Time-to-market

Time-to-market refers to a continuous period in that the team should build and deliver the software. This time is expressed through deadlines to conclude tasks. The time pressure in development and delivery can cause technical debt. The lack of rigor, insufficient tests, and no time for proper design or careful reflection can accumulate massive amounts of debt rapidly (Philippe Kruchten and Ozkaya, 2012). On the other hand, market forces demand cost reductions and release cycles, depending on the business segments. So, reducing time-to-market can bring competitive advantages. The software architecture can contribute to reducing the time-to-market, establishing some strategies such as the use of existing assets and common architectural frameworks. Also, it can optimize the integration and mechanisms of generation of the architecture (Garlan and Perry, 1995). For open source ecosystems, time-to-market depends on the business scenario and community goals. The needs of the internal and external community are important variables to be considered. Platform and third-party objectives must converge to allow the growth of the ecosystem. This way, the time-to-market should be defined in sync with stakeholders' demands.

We found several practices contributing to reducing time-to-market:

- The practice *(P7) provide a newcomer-specific page or portal guiding their first steps, including development information,* contributes to reducing the time-to-market because this page informs the community rules, decision-making processes, tutorials, and documentation of the code and software architecture. This facilitates a faster engagement of a member in the development of the software. So, he could evolve and contribute more quickly to features that will be released in the next versions. The faster the newcomer knows the architecture, the faster it can make use of existing assets and common architectural frameworks, contributing to reducing the time-to-market.

- *(P20) Keep the backward compatibility for a medium or long time to allow the community updates their software* provides time to developers get used to a new environment and change their applications. At the same time, it allows for launching releases and following market needs.

5.3 Health Indicators

The idea of ecosystem health and its measurement, as introduced by Iansiti and Levien (2002), requires the use of three health indicators: Robustness, Productivity, and Niche Creation. Analyzing fifty practices, our study found concepts demonstrating the influence of architectural practices on the health of the seven open source ecosystems. In addition, our study also identified other types of influences not covered by the existing three health indicators. The literature in the ecosystem area has suggested the use of new indicators, some studies defend not all aspects of health are covered by existing indicators. Campbell and Ahmed (2011) uses the indicators from Iansiti and Levien (2002), but they also suggested creating more one indicator, Internal Characteristic, to fully assess the health. Hyrynsalmi et al. (2018) defended the improvement of health indicators, suggesting the use of indicators for specific domains of the ecosystem.

On top of that, some practices together with information about their contexts suggest actions to keep the "normal functioning" of the ecosystem. For instance, "*(P28) Create a partnership with third-party to solve problems of the core and their interfaces*" signalizes that the ecosystem should create partnerships with other organizations to solve problems in the core platform, create new features, and share the cost with third-party. This is one of the basic goals of an ecosystem. Also, "*(P23) Use tools to publish known cybersecurity vulnerabilities. For example, the Common Vulnerabilities and Exposures (CVE®)*" shows that ecosystem members register vulnerabilities and their possible solutions to ensure the community is aware of security issues, contributing to protecting the ecosystem to do not anything unexpected. The compilation of these data allowed us to suggest another health indicator to represent the healthiness of the ecosystems, filling this gap. So, we proposed the **Trustworthiness** indicator that accounts for the normal operation of the ecosystem. **Table 11** shows each indicator and the practices that influence them. Following, we describe the influence of the architectural practice on each health indicator.

5.3.1 Robustness

In the context of software ecosystems, robustness refers to the capacity of the ecosystem to survive crises and disruptions. In a robust ecosystem, the connection between members and technologies remains in front of a collapse. The ecosystem has the capacity to adapt to new situations without harming its core. In addition, most of its active resources can be used for this new phase (Iansiti and Levien, 2002). We observed twenty-one practices influencing this indicator. By adopting these practices, the ecosystem prevents or mitigates the effects of problems, reinforcing the robustness. For example, the practice *"(P44) Build the architecture based in plugins to facilitate the coupling of applications"* contributes to isolating the core from external applications. This way, the core trend suffers less with the impact of changes in the applications upon. The core platform is more protected from breaking and can remain unaffected in crisis.

Table 11. Health Indicators with Practices

Health Indicators	Practices
Robustness	P2, P3, P5, P8, P9, P10, P18, P19, P20, P23, P24, P28, P33, P35, P36, P37, P38, P42, P43, P44, P49
Productivity	P1, P2, P3, P5, P6, P7, P8, P9, P10, P11, P12, P14, P15, P17, P19, P22, P25, P26, P27, P28, P30, P34, P35, P37, P38, P39, P40, P41, P50
Niche Creation	P1, P2, P3, P4, P14, P16, P18, P25, P28, P35
Trustworthiness	P1, P2, P3, P4, P5, P6, P7, P9, P10, P11, P12, P13, P14, P16, P17, P18, P19, P20, P21, P22, P23, P24, P25, P26, P27, P28, P29, P31, P32, P33, P38, P42, P43, P44, P45, P46, P47, P48, P50

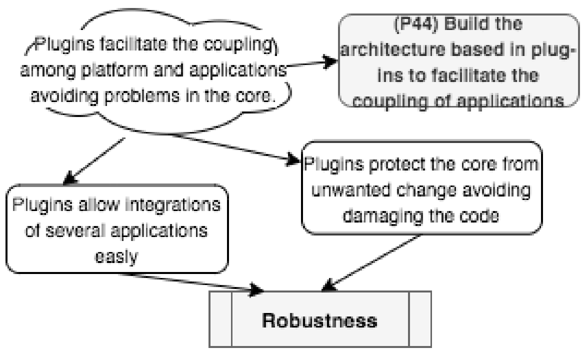
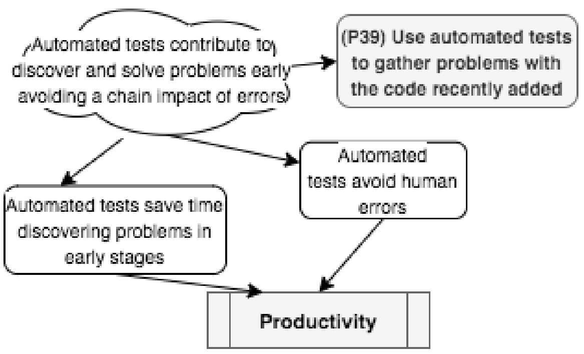

Plugins facilitate the coupling among platform and applications avoiding problems in the core.
(P44) Build the architecture based in plug-ins to facilitate the coupling of applications
Plugins protect the core from unwanted change avoiding damaging the code
Plugins allow integrations of several applications easly
Robustness

Figure 5. Influences on the Robustness from underlying concepts

Figure 5 illustrates an example of how the influence on the robustness emerged from the underlying concepts. Regarding another practice, "*(P23) Use tools to publish known cybersecurity vulnerabilities. For example, the Common Vulnerabilities and Exposures (CVE®)*", we realized that publishing known vulnerabilities allows developers to be aware of the danger. Also, they can protect their applications and/or help to fix the vulnerability. Knowing security problems help to lead and recover in front of security collapses. Another example is that "*(P38) Use some tools to compute some quality metrics*" contributes to the robustness when existing tools collect quality metrics to identify problems in advance and avoid crises.

5.3.2 Productivity

Productivity refers to the ability of the ecosystem to transform inputs into outputs efficiently. The productive ecosystem works efficiently, reducing progressively its cost. Besides, it also creates new production techniques, and delivers them for all members to improve the productivity of the whole community(Iansiti and Levien, 2002). The impact of practices on this indicator is clear. Some practices influence directly the form of how the work is done. This way, our research can observe and deduce the reasoning of the influences.

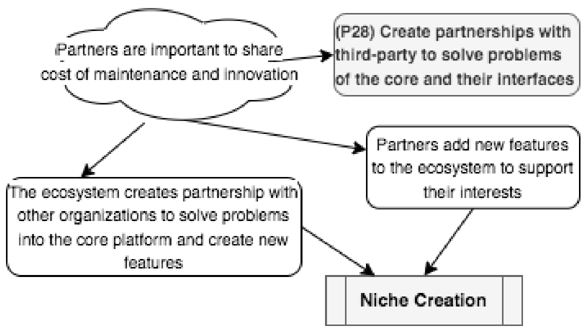

Automated tests contribute to discover and solve problems early avoiding a chain impact of errors
(P39) Use automated tests to gather problems with the code recently added
Automated tests save time discovering problems in early stages
Automated tests avoid human errors
Productivity
Figure 6. Influences on the Productivity from underlying concepts

For instance, practices such *"(P35) Provide meetings (sprints) face-to-face to accelerate the development of critical issues and solve development problems with interdependent modules"* and *"(P2) During code review, provide feedback information about architecture, good practices of coding, do refactor and show the best way to solve the problems"* intend to reduce the time for exchanging information among members in the face-to-face meeting. Also, they can train inexperienced members and keep the focus on the development of code, instead of losing time trying to learn things alone. Moreover, practices such as "*(P39) Use automated tests to gather problems with the code recently added*" and "*(P40) Provide an automated process to launch releases of applications*" use automation for improving tests and release launching, reducing the time of these development tasks. They also avoid human errors. Automated tests contribute to discovering errors early, and automated release launching contributes to reducing launching problems, ensuring that all files are included in the final package.

Figure 6 illustrates how the influence on the productivity emerged from the underlying concepts.

5.3.3 Niche Creation

This indicator refers to the capacity to create opportunities, add new functions, and carry out innovation in the ecosystem. The niche creation is represented by the number of new features and technologies created by the ecosystem. A good level of niche creation is expressed by new business scenarios and technologies or ideas that aggregate value for the ecosystem. The diversity creates value, contributing to the inovation of the ecosystem (Iansiti and Levien, 2002). From the practices adopted, several business opportunities can be generated easier and naturally.


Partners are important to share cost of maintenance and innovation
(P28) Create partnerships with third-party to solve problems of the core and their interfaces
Partners add new features to the ecosystem to support their interests
The ecosystem creates partnership with other organizations to solve problems into the core platform and create new features
Niche Creation
Figure 7. Influences on the Niche Creation from underlying concepts

The practice "*(P18) Provide information about the translation process (how to participate and tools used)*" contributes to the expansion of the ecosystem to new markets. The translation for other languages can create opportunities for new applications supporting new needs in other markets. In addition, the practice "*(P28) Create partnerships with third-party to solve problems of the core and their interfaces*" conduces third-party to add new features to the ecosystem to support their interests. These new features will aggregate innovation, and they have the cost-shared with the community. **Figure 7** shows a part of the influence on the niche creation which emerged from the underlying concepts. Another example is that "*(P4) Provide internal or third-party mentoring programs to train newcomers*" facilitates the appearance of new ideas through the reception of new developers.

5.3.4 Trustworthiness

We propose *trustworthiness* as a novel health indicator to address the need to express the ordinary functioning of the ecosystem when its behavior is as expected, without unwanted surprises. We characterize the *trustworthiness indicator* as *the likelihood of an ecosystem to work as expected and doing nothing beyond what is supposed to*. This indicator expresses the level of accomplishment for the following tasks: (i) facilitating interactions among organizations and third-party; (ii) increasing the attractiveness for new users/developers; (iii) sharing the maintenance with ecosystem partners; (iv) sharing cost of innovation; and (v) incorporating in the platform features developed by third-party. In addition, the ecosystem should not have some security issues such as the presence of faults, catastrophic consequences, unauthorized disclosure of information, and unauthorized access. These characteristics ensure that the ecosystem does not do anything it should not do. As well as, it should do a correct service for a given duration time in a reasonable response time. Taking all these into account, we believe that an ecosystem presents an operation as expected, deserving of trust. **Figure 8** shows the features of this new health indicator.

The term *"trustworthiness"* has been used with different meanings by some authors. First, Deljoo *et al.* introduced a computational trust model where they provided mechanisms for estimating trustworthiness and assessing trust. This model can support taking decisions to establish future relationships. Their concept of trustworthiness involves taking the risk to use the system no matter what, without monitoring or controlling the environment. This concept makes mem-

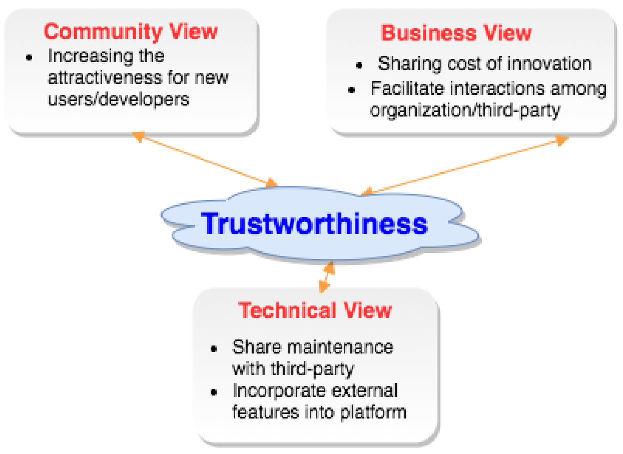

Community View
Increasing the attractiveness for new users/developers
Business View
Sharing cost of innovation
Facilitate interactions among organization/third-party
Trustworthiness
Technical View
Share maintenance with third-party
Incorporate external features into platform

Figure 8. Features of Trustworthiness

bers vulnerable to the actions of other members based on the expectation that specific actions will be taken (Deljoo et al., 2018).

Second, the trustworthiness for open source ecosystems was studied by del Bianco *et al*. They proposed to define a notion of trustworthiness for software products and artifacts in open source systems (OSS). Besides, they identified some factors that influence this notion. The idea is to support the choice of OSS products that can provide user needs. This model identifies strengths and weak points to help to improve the quality of the OSS products choice (del Bianco et al., 2009). Next, Becker *et al.* discussed the terminology for the term trustworthiness in software systems, presenting its definition and characteristics (Becker et al., 2006). For them, the trustworthiness of a system happens when a system operates as expected, despite disruptions or errors in the environment. In this study, trustworthiness is composed of other characteristics such as security, reliability, privacy, safety, and survivability. This concept is very comprehensive, including several features, and is very similar to the robustness.

Lastly, regarding ecosystem health, Franco-Bedoya *et al.* introduce trustworthiness as *"the ability to establish a trusted partnership of shared responsibility in building an overall open source ecosystem"* (Franco-Bedoya et al., 2015). In their model, trustworthiness is a sub-characteristic of the characteristic of resource health. Related to financial health, trustworthiness is represented by operational financial measures. The trusted partnership should create value for end products.

Based on these concepts, we suggested trustworthiness as a new health indicator. The definition of software ecosystems was analyzed and considered the trustworthiness of the whole ecosystem, not only for their products but including all relationships in the ecosystem scenario. The trustworthiness encompasses the normal operation of the ecosystem. Applying the definition of Bosch to software ecosystems and their characteristics, we figured out a normal behavior for an ecosystem (Bosch, 2009). **Figure 9** shows part of the influence on trustworthiness that emerged from the underlying concepts.

In our study, during data analysis process with the GT-inspired, we also investigated how the practices adopted by the ecosystem influence this novel health indicator. We found thirty-nine practices that contribute in some way to the

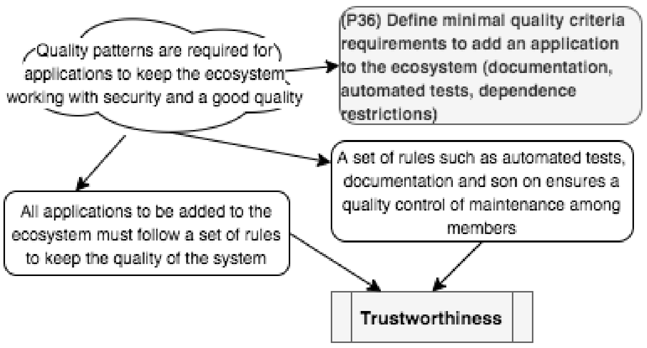

Quality patterns are required for applications to keep the ecosystem working with security and a good quality
(P36) Define minimal quality criteria requirements to add an application to the ecosystem (documentation, automated tests, dependence restrictions)
A set of rules such as automated tests, documentation and son on ensures a quality control of maintenance among members
All applications to be added to the ecosystem must follow a set of rules to keep the quality of the system
Trustworthiness

Figure 9. Emergence of the indicator Trustworthiness from underlying concepts

ecosystem working as expected. For example, the practice "*(P7) Provide a newcomer-specific page or portal guiding their first steps, including development information*" helps to increase the attractiveness for newcomers. This is because the portal facilitates the first steps for newcomers, preventing them from abandoning the attempt to become involved in the community. Another practice "*(P12) Provide updated official documentation about code's organizational structure, and how the components, modules, classes, and packages are related to each other*" makes it easy to share maintenance with third-party due to the documentation that provides knowledge about the code infrastructure. The practice "*(P28) Create partnerships with third-party to solve problems of the core and their interfaces*" facilitates interactions among organizations and third-party, as well as sharing the cost of innovation due to the partners to aggregate valuable features to the core. In addition, the practice "*(P36) Define minimal quality criteria requirements to add an application to the ecosystem (documentation, automated tests, dependence restrictions)*" provides a set of rules such as automated tests, documentation and so on to ensure the quality of the components that will be incorporated into the ecosystem platform.

6 Discussion

Our initial effort was purely exploratory, aiming to know *if* and *how* the architectural practices could influence the ecosystem health. To answer our research questions, first, we investigated *which* practices were used in the design of open source ecosystems. The netnography-based approach allowed us to collect several practices in their context of use. The grounded theory-inspired approach helped us to identify the motivations for adopting these practices and the influences of their adoption. The investigation led us to a set of concepts about reasons for using such practices. The findings of the study include a number of relevant factors and indicators for understanding the relationship between software architecture and the health of open source ecosystems. **Figure 10** summarizes our findings.

Our findings reinforce previous statements about the influence factors found in the literature. (Bass et al., 2012) described some factors that influence the architecture. However, we organized these influence factors considering architectural practices and related characteristics. In addition, with respect to health indicators, our work also reinforces the in-

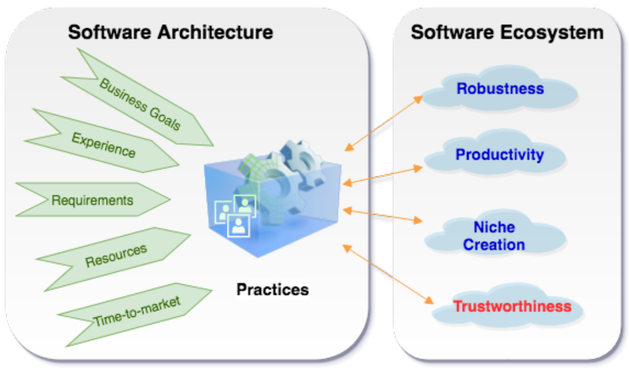


Figure 10. Summary of Findings

dicators introduced by (Iansiti and Levien, 2002). In addition, this research suggests a new health indicator: *Trustworthiness*. In our analysis, we highlighted that some practices contributed to improving and keeping the normal operation of the ecosystems in accordance with what is expected. So, we reframed this characteristic as a new health indicator. A software ecosystem needs to provide trustworthiness for the community members to achieve a good health state. However, we observed a thin line between the trustworthiness concept of (Becker et al., 2006) and our proposal. They also consider trustworthiness as robustness. For us, they are different concepts. Robustness is the ability of the ecosystem to survive disruptions, but trustworthiness considers its operation in normal conditions, representing what is expected from the ecosystem in normal scenarios.

Trustworthiness can also be questioned, considering the business behaviors of the organization boards. In some cases, the board defines rules that are not in total agreement with the whole community. As a result, third-parties cannot trust totally in the ecosystem, but they continue engaging in the ecosystem aiming for some profit. For us, if you are engaged in an ecosystem, you are subordinated to rules determined by the organization. Although you know that your judgment can sometimes disagree with them. You know what to expect from them, and you trust them to do nothing against you. You believe your application will not be harmed. Despite the rules, you will still make a profit. Trustworthiness is exactly that, believing that it plays its role as an ecosystem, without harming it.

Regarding data analysis, we also observed different influence factors can affect the same practice. One factor can determine various practices driving the ecosystem to achieve its goals. At the same time, we realize that a practice impacts different health indicators. This is because the consequences of a practice spread its effects throughout the ecosystem.

In our study, not all fifty practices are adopted by all ecosystems. In fact, none of the ecosystems has adopted the entire set of practices; however, each practice is carried out in at least one ecosystem. In addition, we also found some interesting and unique practices that were outside the scope of this study. They were found only in one ecosystem, however, they also could be adopted by others. For example, the GitLab ecosystem establishes a "Hall of Fame" where for every release, team members elect a community contributor as the MVP (most valuable person) of the release. This member receives the prestigious golden fork. This practice could increase the attractiveness of new developers. Also, in another practice, the GitLab provides a log automatic for changes registering all changes done. This way, everyone can be aware of the changes and analyze the impact of these changes. In addition, the Open edX establishes a contract for contributors about intellectual property rights. This way, it protects itself of future problems with property rights. Moreover, they have one practice to provide accessibility guidelines to ensure that any user interfaces are usable by everyone, regardless of any physical limitations. In summary, some ecosystems develop different practices according to their particular needs. However, our research focused on practices that we perceive to influence software architecture. These unique practices observed during the phase of netnography were not considered in our study.

The overall direction of the results showed trends that could be helpful to learn about how to build a good healthy state. Knowing the motivations and influences of the practices on the health will contribute to understanding mechanisms to make a healthy ecosystem. The relevance of our results using such architectural practices to figure out the health state encompasses a crucial part of the ecosystem: the software architecture. As mentioned previously, the architecture plays a key role in supporting the entire ecosystem. We are aware that ecosystem health is constructed based on a large set of elements; however, our research only focuses on the architectural component. The software architecture is built based on a set of factors that determine how the practices will be performed. In turn, the results show evidence that the practices adopted affect all products in the ecosystem, and their management, prosperity, and longevity.

According to Jacobson, practices can be easily disseminated and used many times to produce some results. They provide a picture of a specific aspect of software development, describing outcomes, and how to achieve them (Jacobson et al., 2007). In addition, they can be analyzed individually to clarify part of the impact from their use. This way, understanding the context and mechanisms for the adoption of practices could signalize influences on the health state. This approach opens up ways to investigate different health scenarios and guide choice according to ecosystem strategies.

Our interpretation considers that the architectural practices used directly impact the health status of the ecosystem. However, further studies are needed to explain *how* the practices can be used to improve the health of the ecosystem. By now, it was not possible to extract a quantitative value for the level of influence of each practice, just as we did not know the implications of interactions between practices, because a practice can also harm another practice instead of helping to improve health. The results of this study create a positive perspective to find forms to measure influences and also construct an approach to guide the ecosystem governance.

7 Threats of Validity

There are some threats to the validity of the study. In order to reduce these threats, we describe briefly some mitigation strategies for them:

- **Context:** Findings in this study is applicable only in the context of open source ecosystems. Many practices in commercial ecosystems are different practices in open source ecosystems. By expanding the study for all ecosystem types, all practices should be included in the study.

- **Generalization:** The fifty practices came from seven open source ecosystems with different domains and sizes. Despite mature ecosystems concentrate the majority of practices, we believe that a range of practices is also used for other ecosystems. Even different ecosystems have common characteristics and, therefore, for which the findings are relevant.
- **Observation:** Due to the methodology used, we could not cover all practices used for the whole community. The lack of some practice in our collected data does not mean that this practice is not used by the community. It can happen in a situation where the researchers could not find evidence of the adoption of this practice. Besides, all practices collected cannot represent all aspects of the architecture, as there is no ground truth based on architectural practices. The choice was done based on the background of the researchers. Moreover, we cannot guarantee that community members or even the governance of the ecosystem are following the practices published on their website. We make assumptions that they really do what they publish on the internet. In addition, we found some drawbacks during the data collection. First, we read the web page describing a practice, but the artifact was not found. For example, Node.js ecosystem presents that they have a standard code, but we did not find it. On the other hand, the inverse situation occurred, when the practice is performed, but we did not find its documentation. For example, some ecosystems do a translation of the system; however, there is no material to guide a translation. Moreover, a practice is performed with different levels of details by different ecosystems. For example, Node.js is very simple describing the translation process, but KDE has a comprehensive and detailed translation process and uses several supporting tools in this process. The translation process at Node.js depends on the language group. For example, the Portuguese language has a basic description, but the Spanish language provides a large description. Another threat was the challenge of identifying discussions about critical changes in software architecture, if it is not explicitly published. We only found changes shared with the community. We performed efforts to mitigate information not found searching data in several links on the ecosystem website. Following, essentially, netnography is performed by a single researcher who engages in an online community, with a participatory role, mediated by computers to gather research data. In our study, only a single researcher conducted data collection and analysis. When the nethnographer does not have a participatory role immersed in the community, Kozinets (2009) explained that he is compelled to make assumptions about meanings that he does not fully understand. This is a weakness of the approach. To mitigate misinterpretation of some practices, the nethnographer visited several pages in the community to clarify doubts. Also, during the writing process review, some questions could be solved by the authors. Lastly, the research was focused on identifying evidence of practices adopted and their influences. It was not the scope of the study to identify any practice should-have but not-have in accordance with our experience.
- **Data Analysis:** The scope of this study was to identify practices and their influences. We could not identify the comprehensive reasons for different systems to adopt a different set of architectural practices. The occurrence of each practice on each project was recorded by data collection. However, we analyzed the influences of the practice in a general way for all ecosystems without considering a particular influence for each ecosystem. Moreover, we could not measure the level of influence of the practice on the health of the ecosystems. In addition, one researcher conducted the data collection and analysis process. Consequently, some bias can have been introduced by the researcher originating from her own ideas about some practice, situation, or ecosystem behavior. Also, the researcher can have some inclination to follow preconceived ideas about the ecosystem area, resulting in discovering the limited scope of information and/or omitting other important data concerned with architectural practices and the ecosystem health. In trying to mitigate the ill effects of the bias, the researcher was extra careful in the processes of data collection, analysis, and interpretation, reviewing their conclusions. Also, the conclusions in the written article were analyzed and reviewed by other researchers. They had available to them the complete dataset of information.
- **Time:** A netnographic study is performed over a long-time and prolonged involvement with the community. Our activities focused on the observation of the practices for a short time. Due to the short observation period, we may not have found or understood some practices used by the community. To avoid misunderstandings, we searched several web pages to confirm the adoption of a practice.

8 Conclusion

A healthy ecosystem works for achieving and maintaining success. Usually, its health state is represented by health indicators. These indicators are influenced by practices adopted by the ecosystem members, more specifically, architectural practices used to create and maintain the software architecture of ecosystems. This study aimed at understanding the software architectural practices universe in the ecosystem scenarios, the factors that can influence their adoption, and how these architectural practices influence the open source ecosystem health.

Our experiences can help other researchers to understand influences of the architectural practices on the health of open source ecosystems. In particular, the main contribution of this work was the identification and discussion of five factors influencing the adoption of architectural practices, as well as the knowledge about the influences of four health indicators on the ecosystem health. We proposed a novel health indicator – *trustworthiness* – to represent a health dimension that has not been considered in previous studies. The *degree of trustworthiness* of the whole ecosystem indicates how much the ecosystem operates as expected regarding the ordinary behavior, including all ecosystem views. This can signalize a good accomplishment of the ecosystem activities.

This work is an initial step towards a health evaluation approach considering the influence of architectural practices on the ecosystem health. Future work includes the replication of this study for proprietary software ecosystems, rather than open source environments, and will highlight similarities and differences in results in the scope of that environment. In addition, we could establish a standard for the most common practices used in software ecosystems. Besides, we could clarify the explicit benefits of adopting the best practices. Further investigation is also needed to determine whether these findings could be applied to build an approach to measure ecosystem health. We plan to identify qualitative and quantitative forms of understanding the weight of the software architecture on ecosystem health.

Acknowledgements

This study was partially supported by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Grant 001.

References

Amorim, S., McGregor, J., Almeida, E., and Chavez, C. (2017). Software ecosystems′ architectural health: Another view. In *Proc. of the 5th ICSE Int. Workshop on Software Engineering for SoS and 11th Workshop on Distributed Software Development, Software Ecosystems and SoS*, SESoS/WDES '17, pages 66–69.

Avelino, G., Constantinou, E., Valente, M. T., and Serebrenik, A. (2019). On the abandonment and survival of open source projects: An empirical investigation. In *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '19, pages 1–12.

Bass, L., Clements, P. C., and Kazman, R. (2012). *Software Architecture in Practice*. Addison-Wesley Professional, third edition edition.

Becker, S., Hasselbring, W., Paul, A., Boskovic, M., Koziolek, H., Ploski, J., Dhama, A., Lipskoch, H., Rohr, M., Winteler, D., Giesecke, S., Meyer, R., Swaminathan, M., Happe, J., Muhle, M., and Warns, T. (2006). Trustworthy software systems: A discussion of basic concepts and terminology. *SIGSOFT Softw. Eng. Notes*, 31(6):1–18.

Bogart, C., Kästner, C., Herbsleb, J., and Thung, F. (2021). When and how to make breaking changes: Policies and practices in 18 open source so ware ecosystems. *ACM Transactions on Software Engineering and Methodology*, 30(4).

Bosch, J. (2009). From software product lines to software ecosystems. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 111–119.

Bosch, J. (2010). Architecture challenges for software ecosystems. In *Proceedings of the Fourth European Conference on Software Architecture*, ECSA '10, pages 93–95.

Bosch, J. and Bosch-Sijtsema, P. (2010). From integration to composition: On the impact of software product lines, global development and ecosystems. *The Journal of Systems and Software*, 83:67–76.

Bouwman, H., Carlsson, C., Carlsson, J., Nikou, S., Sell, A., and Walden, P. (2014). How nokia failed to nail the smartphone market. In *Proceedings of the 25th European Regional Conference of the International Telecommunications Society*, ITS '14, pages 1–18.

Bredemeyer, D., Malan, R., and Consulting, B. (2000). The role of the architect.

Campbell, P. R. J. and Ahmed, F. (2010). A three-dimensional view of software ecosystems. In *Proceedings of the Fourth European Conference on Software Architecture*, ECSA '10, pages 81–84.

Campbell, P. R. J. and Ahmed, F. (2011). An assessment of mobile os-centric ecosystems. *Journal of Theoretical and Applied Electronic Commerce Research*, 6:50–62.

Charleux, A. and Viseur, R. (2019). Exploring impacts of managerial decisions and community composition on the open source projects' health. In *Proceedings of the 2nd International Workshop on Software Health*, SoHeal '19, pages 1–8.

Charmaz, K. (2006). *Constructing Grounded Theory A Practical Guide through Qualitative Analysis*. SAGE Publications.

Coleman, G. and O'Connor, R. (2007). Using grounded theory to understand software process improvement: A study of irish software product companies. *Information and Software Technology*, 49:654–667.

da Silva Amorim, S., Neto, F. S. S., McGregor, J. D., de Almeida, E. S., and von Flach Garcia Chavez, C. (2017). How has the health of software ecosystems been evaluated? a systematic review. In *Proceedings of the 31st Brazilian Symposium on Software Engineering*, SBES '17.

del Bianco, V., Lavazza, L., Morasca, S., and Taibi, D. (2009). Quality of open source software: The qualipso trustworthiness model. In *Proceedings of the IFIP International Conference on Open Source Systems*, OSS '09, pages 199–212.

Deljoo, A., van Engers, T., Gommans, L., and de Laat, C. (2018). The impact of competence and benevolence in a computational model of trust. In *Proceedings of the IFIP International Conference on Trust Management*, IFIPTM '18, pages 45–57.

Dijkers, J., Sincic, R., Wasankhasit, N., and Jansen, S. (2018). Exploring the effect of software ecosystem health on the financial performance of the open source companies. In *Proceedings of the 1st International Workshop on Software Health*, SoHeal '18, pages 48–55.

dos Santos, R. P. and Werner, C. (2011). Treating business dimension in software ecosystems. In *Proceedings of the International Conference on Management of Emergent Digital EcoSystems*, MEDES '11, pages 197–201.

Franco-Bedoya, O., Ameller, D., Costal, D., and Franch, X.

(2015). Measuring the Quality of Open Source Software Ecosystems Using QuESo. In *Proc. of the 10th Int. Conference on Software Technologies (ICSOFT)*, pages 39–62.

Garlan, D. and Perry, D. E. (1995). Introduction to the special issue on software architecture. *IEEE Transaction on Software Engineering*, 21(4):269–274.

Goggins, S., Lumbard, K., and Germonprez, M. (2021). Open source community health: Analytical metrics and their corresponding narratives. In *Proceedings of the 4th International Workshop on Software Health in Projects, Ecosystems and Communities (SoHeal)*, SoHeal '21.

Hyrynsalmi, S., Ruohonen, J., and Seppänen, M. (2018). Healthy until otherwise proven: Some proposals for renewing research of software ecosystem health. In *Proceedings of the 41st International Workshop on Software Health (SoHeal)*, SoHeal '18, pages 18–24.

Iansiti, M. and Levien, R. (2002). Keystones and Dominators: Framing Operating and Technology Strategy in a Business Ecosystem. *Harvard Business School*, 3(61).

Jacobson, I., Ng, P. W., and Spence, I. (2007). Enough of processes-lets do practices. *Journal of Object Technology*, 6(6):41–66.

Jade, V. (2019). Software architecture tools. IASA Global. Accessed: 2019-05-12.

Jansen, S. (2020). A focus area maturity model for software ecosystem governance. *Information and Software Technology*, 118.

Jansen, S., Cusumano, M., and Brinkkemper, S. (2013). *Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry*. Edward Elgar Publishers.

Jansen, S. and Cusumano, M. A. (2013). Defining software ecosystems: A survey of software platforms and business network governance. In Jansen, S., Brinkkemper, S., and Cusumano, M., editors, *Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry*, chapter 1, pages 13–28. Edward Elgar Publishing.

Jensen, C. and Scacchi, W. (2004). Data mining for software process discovery in open source software development communities. In *Proceedings of the 1st International Workshop on Mining Software Repositories*, MSR '04, pages 96–100.

Jensen, C. and Scacchi, W. (2005). Experiences in discovering, modeling, and reenacting open source software development processes. In *Proceedings of the International Software Process Workshop*, SPW '05, pages 449–462.

Kozinets, R. (2009). *Netnography: Doing Ethnographic Research Online*. SAGE Publications.

Liao, Z., Yi, M., Wang, Y., Liu, S., Liu, H., Zhang, Y., and Zhou, Y. (2019). Healthy or not: A way to predict ecosystem health in github. *Symmetry*, 11(2).

Liu, D. (2017). The art of building platforms. Forbes. Accessed: 2018-11-16.

Manikas, K. and Hansen, K. M. (2013). Software ecosystems - a systematic literature review. *Journal of Systems and Software*, 86:1294–1306.

Mittal, R. (2019). Blackberry ltd. marketing downfall in mobile handset industry. Technical report, University of Roehampton - Eu Business School.

Pelliccione, P. (2014). Open architectures and software evolution: the case of software ecosystems. In *Proceedings of the 23rd Australian Software Engineering Conference*, ASWEC '14, pages 66–69.

Philippe Kruchten, R. L. N. and Ozkaya, I. (2012). Technical debt: From metaphor to theory and practice. *IEEE Software*, 29:18–21.

Saldaña, J. (2009). *The Coding Manual for Qualitative Researchers*. SAGE Publications.

Satell, G. (2016). Platforms are eating the world. Forbes. Accessed: 2018-11-16.

Seaman, C. B. (1999). Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering*, 25(4):557–572.

Sigfridsson, A. and Sheehan, A. (2011). On qualitative methodologies and dispersed communities: Reflections on the process of investigating an open source community. *Information and Software Technology*, 53:981–993.

Silverman, D. (2021). Apple back on top: iphone is the best-selling smartphone globally in q4 2020. Forbes. Accessed: 2021-03-24.

Stol, K.-J., Ralph, P., and Fitzgerald, B. (2016). Grounded theory in software engineering research: A critical review and guidelines. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 120–131.

Weinreich, R. and Groher, I. (2016). The architect's role in practice: From decision maker to knowledge manager? *IEEE Software*, 33:63–69.

West, J. and Mace, M. (2010). Browsing as the killer app: Explaining the rapid success of apple's iphone. *Telecommunications Policy*, 34:270–286.

Wnuk, K., Manikas, K., Runeson, P., Lantz, M., Weijden, O., and Munir, H. (2014). Evaluating the Governance Model of Hardware-Dependent Software Ecosystems – A Case Study of the Axis Ecosystem. In *Proc. of the 4th International Conference on Software Business (ICSOB)*, pages 212–226.

A Architectural Practices

We identified fifty architectural practices during our study. Table 12 presents the architectural practices and, for each ecosystem, the adopted practices.

Table 12. Architectural practices

Id	Architectural Practices	GitLab	Jenkins	KDE	MapServer	Node.js	Open edX	WordPress
P1	Create personal blogs and/or wikis to inform about the development and architectural issues	✓		✓	✓	✓	✓	✓
P2	During code review, provide feedback information about architecture, good practices to code, doing refactoring and show the best way to solve problem	✓		✓	✓	✓	✓	✓
P3	Document APIs constantly	✓		✓				✓
P4	Provide internal or third-party mentoring programs to train newcomers	✓	✓	✓	✓			✓
P5	Provide code recommendations, defining a standard in the community	✓		✓	✓		✓	✓
P6	Keep a register of meetings available to community to know all decisions of the meeting	✓		✓	✓	✓	✓	✓
P7	Provide a newcomer-specific page or portal guiding their first steps, including development information	✓	✓	✓	✓	✓	✓	✓
P8	Identify and dismiss outdated information on websites	✓			✓			
P9	Provide generation of (semi-)automated documentation filtered to up-to-date information relevant to newcomers	✓		✓	✓	✓	✓	✓
P10	Answer questions on the mailing list quickly	✓	✓	✓	✓	✓	✓	✓
P11	Create a detailed step-by-step tutorial linking information about common problems and possible solutions	✓		✓	✓		✓	
P12	Provide updated official documentation about code's organizational structure, and how the components, modules, classes, and packages are related to each other	✓		✓	✓	✓	✓	✓
P13	Support the participation of the ecosystem on aggregators' sites such as stack overflow, reddit, hacker news, and so on	✓	✓	✓	✓	✓	✓	✓
P14	Provide a dictionary to newcomers to facilitate their learning of the technical jargon, acronyms of the community						✓	✓
P15	Provide video-classes or tutorials about introducing the ecosystem, installing technologies, configuring the development environment, and using the dependencies with other ecosystems	✓		✓	✓	✓	✓	✓
P16	Provide a manifesto explaining requirements of an application belonging to the ecosystem	✓		✓				
P17	Provide video-conference sessions with questions and answers about relevant topics	✓		✓				
P18	Provide information about the translation process (how to participate and tools used)	✓	✓	✓	✓	✓	✓	✓
P19	Provide documentation about translation rules that developers must follow to prepare code for other languages	✓	✓	✓		✓	✓	✓
P20	Keep the backward compatibility for a medium or long time to allow the community update their software	✓		✓				
P21	Provide different levels of security access for the parts of the ecosystems in accordance with the degree of commitment and tasks in the ecosystem	✓		✓				
P22	Point newcomers to easy tasks filtered by difficulty, skills needed, and topics	✓		✓		✓	✓	✓
P23	Use tools to publish known cyber security vulnerabilities. For example the Common Vulnerabilities and Exposures (CVE)	✓		✓	✓	✓		✓
P24	Keep a team to manager the security problems registered	✓		✓		✓	✓	✓
P25	Preference to use the English language written to avoid misunderstanding	✓						✓
P26	Do online meetings in a timezone adequate for most of the community	✓				✓		✓
P27	Provide several online meetings to discuss architectural problems by IRC, email	✓	✓	✓	✓	✓	✓	✓
P28	Create partnerships with third-parties to solve problems of the core and their interfaces			✓		✓	✓	
P29	Setting a code of conduct to avoid mistreatment among members	✓		✓		✓		✓

Continued on next page

Table 12 – *Continued from previous page*

Id	Architectural Practices	GitLab	Jenkins	KDE	MapServer	Node.js	Open edX	WordPress
P30	Provide a message template for newcomers to use to interact with the community	✓		✓		✓	✓	✓
P31	The organization board defines some technologies that should be used by the whole community as tools for testing, communication, coding review, bugging manager, and navigation	✓		✓	✓	✓	✓	✓
P32	The organization board provide hardware and software resources to be used by the community			✓				✓
P33	Define a financial board to manage the financial resources	✓		✓	✓	✓	✓	✓
P34	Provide financial resources to support meetings face-to-face	✓	✓	✓	✓	✓		✓
P35	Provide meetings (sprints) face-to-face to accelerate the development of critical issues and solve development problems with interdependent modules	✓	✓	✓	✓	✓	✓	✓
P36	Define minimal quality criteria requirements to add an application to the ecosystem (documentation, automatic tests, dependence restrictions)	✓		✓			✓	
P37	Define a team to test performance and behavior of the application			✓			✓	✓
P38	Use some tools to compute some quality metrics	✓		✓			✓	
P39	Use automatic tests to gather problems with the code recently added	✓	✓	✓	✓	✓	✓	✓
P40	Provide an automatic process to launch release of applications	✓		✓		✓	✓	
P41	Divide the parts of the software in layers, defining restrictions for managing dependencies among the layers	✓		✓				
P42	Discuss with the community about critical changes into architecture that will impact in the applications	✓		✓			✓	✓
P43	Publish widely the architectural changes for the community	✓		✓	✓		✓	✓
P44	Build the architecture based in plug-ins to facilitate the coupling of applications	✓	✓	✓			✓	✓
P45	Provide guidelines with a set of steps to be followed by developers how to add code to repository	✓		✓				
P46	Provide a virtual machine with pre-configured build environments, web-based IDEs, or a container management tool	✓						
P47	Inform newcomers about technical background required. Identify which specific technologies they need to know or they should learn to achieve their goal of contributing to the ecosystem						✓	
P48	Tag all tasks in accordance with degree of difficulty (easy, medium, difficult)	✓						
P49	Provide a group for gardening to care the global state of ecosystem			✓				
P50	Keep the list of tasks updated, informing about who is working on the solution							✓