System Requirements-OSS Components: Matching and Mismatch Resolution Practices – An Empirical Study

Claudia Ayala, Anh Nguyen-Duc, Xavier Franch, Martin Höst, Reidar Conradi, Daniela Cruzes, Muhammad Ali Babar.

Abstract—

Context: Developing systems by integrating Open Source Software (OSS) is increasingly gaining importance in the software industry. Although the literature claims that this approach highly impacts Requirements Engineering (RE) practices, there is a lack of empirical evidence to demonstrate this statement.

Objective: To explore and understand problems and challenges of current system requirement–OSS component matching and mismatches resolution practices in software development projects that integrate one or more OSS components into their software products.

Method: Semi-structured in-depth interviews with 25 respondents that have performed RE activities in software development projects that integrate OSS components in 25 different software development companies in Spain, Norway, Sweden, and Denmark.

Results: The study uncovers 15 observations regarding system requirements-OSS components matching and mismatch resolution practices used in industrial projects that integrate OSS components. The assessed projects focused mainly on pre-release stages of software applications that integrate OSS components in an opportunistic way. The results also provide details of a set of previously unexplored scenarios when solving system requirement–OSS component mismatches; and clarify some challenges and related problems. For instance, although licensing issues and the potential changes in OSS components by their corresponding communities and/or changes in system requirements have been greatly discussed in the RE literature as problems for OSS component integration, they did not appear to be relevant in our assessed projects. Instead, practitioners highlighted the problem of getting suitable OSS component documentation/information.

Keywords— Open Source Software, OSS, Requirements Engineering, Empirical Study, Qualitative Study, Survey.

1 Introduction

The genesis of Requirements Engineering (RE) research in the mid-1970s was motivated by practitioners who noticed the urgent need for a disciplined consideration of requirements in software projects that had grown large and unmanageable [74]. Much of RE research since then has focused on artifacts that maintain the intellectual discipline. However, the environment in which RE has been practiced is continuously changing and new challenges are continuously emerging [25], [42]. One of these challenges stems from the dominating trend of integrating third-party software components to build complex software applications [47].

Third-party components, also known as Off-The-Shelf (OTS) components, refer to pieces of software that other software projects can reuse and integrate into the software products that they produce. OTS components are frequently categorized into OSS (Open Source Software) and COTS (Commercial-Off-The-Shelf) components. COTS components are licensed and distributed by a commercial vendor who retains the source code and rights over the software. In contrast, OSS components are freely available on the Internet and are openly and collaboratively developed and evolved by OSS communities. OSS components are "owned" by their corresponding communities under diverse kinds of OSS licenses [70]. The licensing schemas that are inherent to OSS have been considered a major legal aspect regarding the use of OSS components [79].

In the last decades, the research community has proposed a plethora of approaches aimed at supporting the integration of OTS components into software systems [49]. From the software development process point of view, it has been suggested that companies must adapt their software development processes in response to integrating OTS; otherwise, they might fail to accommodate many challenges of using them [15], [95]. From the RE point of view, the implications of integrating OTS components have been stated as major ones [48], [72]. The main reason is that integrating OTS components changes the focus of the development-centric approach assumed in traditional software development by a solution- or feature-driven approach that is mainly driven by the component features and availability [15]. This leads to two important interrelated processes related to RE:

the requirements-component matching and requirements-component mismatch resolution, aimed to reach a successful component integration [47].

The requirements-component matching process refers to the activities performed to find components that cover the system requirements. It typically involves searching, evaluating and deciding components. The literature commonly refers to these activities as component selection; however, it focuses mostly on the evaluation and decision-making phases, setting aside the searching activity [7]. Requirement-component mismatch resolution refers to the activities aimed to find and solve mismatches (i.e., a misalignment between the behaviour of a given component and a functional or non-functional requirement [60]) between the system requirements and the components. For instance, after selecting a component to be integrated, the software development team might find that some characteristics of the chosen component do not fill well to the current system requirements. Therefore, some decisions need to be taken to solve such misalignments.

The early literature on this topic focused mainly on the requirement-component matching processes related to OTS components (e.g., [2], [12], [55], [60]) without clarifying the potential practical differences that might exist among integrating COTS or OSS (if any). In addition, there is a lack of evidence on how the industry deals with requirement-component mismatches [60].

Studying the requirement-component matching and mismatch resolution approaches followed in industrial projects that integrate OSS components is currently a relevant need for principally two reasons:

First, the integration of OSS is currently playing a crucial role in the entire software development industry. OSS components are integrated in almost 85% of commercial software [34] and this percentage is increasing. It is not surprising to read that *"OSS can be a major enabler of productivity and savings; IT organizations that are mature in OSS-based development methods have the potential to be 5 to 10 times more productive and responsive than those that do not"* [34]. The OSS phenomenon has matured to the point where the collaborative development model often associated with OSS communities has inspired software companies to evolve their existing development processes and to collaborate both internally and across company borders to form complex OSS ecosystems [39], [84], [90], [91]. In addition, the general lack of license fees in OSS had contributed to shifting the software industry's traditional license-based business model towards other sustainable business models [82] and adoption ways [38]. The success and popularity of the OSS phenomenon have encouraged the Software Engineering (SE) research community to investigate how OSS communities produce software systems without overtly following traditional software development practices [16], [61], [67], [81], [82], [86]. In the RE arena, one of the first works aimed at understanding RE practices in OSS projects was reported by Scacchi in 2002 [80]. Since then, other studies have reported some RE practices in the context of OSS communities that develop OSS components (e.g., [1], [66], [71]). However, these efforts have focused on understanding the internal development/requirement processes in OSS communities and neglected exploring the needs of industrial projects that integrate OSS components [42].

Second, organizations that integrate OSS components (especially those organizations that extend and possibly modify them) face additional challenges than those organizations that integrate COTS components. This is because integrating OSS components into other software systems is endangered by the volatile nature of both OSS components and system requirements. It means that potential mismatches between system requirements and OSS components might occur at different stages of software development and maintenance [38], therefore the integration of OSS components increases the dependency of the integrator on the OSS communities [7]. Furthermore, the volatile nature of system requirements and the open evolution of OSS components [47] could make the selected component's features differ from the system requirements in post-selection phases. These mismatches between OSS components and system requirements are unavoidable and need to be resolved during the project's lifetime [60]. This might impose relevant challenges to current RE practices since the inherent characteristics of OSS communities [25] makes them a special stakeholder that must be assessed when selecting OSS components and possibly during the project's lifetime [53]. It is because OSS communities pose uncertain evolution factors (not previously agreed by contractual means as in the case of integrating COTS components) that should be considered further to ensure the desired lifetime of the resulting application [19]. Furthermore, the fact that OSS components' code and OSS communities are freely available on the Internet might lead to diverse ways of solving potential mismatches that would probably not exist when using COTS. These aspects about the integration of OSS components have not yet been further explored [25] or supported by industrial evidence fully dedicated to OSS instead of OTS [48], [60].

Thus, this research aims to explore and describe relevant industrial requirement-component matching and mismatch resolution practices and the influence of OSS components on RE activities from the perspective of practitioners that integrate OSS components into their systems. This paper presents the results of an empirical study that includes data collected through in-depth interviews with 25 respondents who had actively participated in projects that integrate OSS components (hereafter OBSD -OSS-Based Software Development-) in 25 software-intensive organizations in Spain, Norway, Sweden, and Denmark. It extends a preliminary, mostly quantitative analysis of 15 interviews from 15 companies that was presented at an international conference [65].

The results of this study are expected to describe problems and challenges of current system requirement–OSS component matching and mismatches resolution approaches in order to shed some light to help mature the RE arena: researchers and practitioners may benefit from the evidence-based findings of this study in order to better understand the practical challenges of RE when integrating OSS components and to properly align their efforts when confronting these challenges. Specifically, researchers can use this evidence to identify and align new research questions, generate and test hypotheses, and interpret the results of such tests. Similarly, practitioners and diverse actors that are related to the OSS arena (e.g., OSS communities, component intermediaries, and providers of services around OSS components) can use the findings reported in this paper to identify and understand other RE practices and needs and to envisage strategic actions for improvement.

The remainder of this paper is structured as follows. In Section 2, we provide a brief background of RE approaches that deal with component integration as well as previous empirical related work. In Section 3, we explain the objectives of this research. Section 4 discusses the methodological approach followed to reach the objectives of the research. Section 5 presents and discusses the empirical results. Section 6 discusses threats to validity. Section 7 concludes, highlights the implications of the results, and states intentions for future work.

2 Background and related work

Traditional RE has generally been described as the elicitation of stakeholders' needs, the analysis and specification of the acquired knowledge into non-conflicting requirements, and the validation of these requirements [87]. However, the integration of OTS components is usually characterized by a constant, iterative trade-off among user requirements, system architecture, and component availability, which leads to additional activities in order to find suitable components (i.e., component searching, evaluation, and decision making) [15]. This leads to requirement-component matching and mistmatch resolution processes, that are overlapping activities that occur in different phases of software development in order to perform a successful integration [15], [42], [75].

Sections 2.1 and 2.2 provide an overview of the different proposals for dealing with requirement-component matching and mismatch resolution approaches, respectively. Section 2.3 provides an overview of current industrial evidence related to the integration of OTS components.

2.1 Requirement-component matching

Early component selection (i.e., component matching) proposals focused mainly on COTS components; however, the increasing adoption of OSS has shifted this focus to OSS components [89]. Several proposals and large-scale research projects specifically focus on OSS selection particularities. Some examples of these initiatives are: the Open Source Maturity Model –OSMM– [35]; Open Business Readiness Rating –OpenBRR– [69]; and the Qualification and Selection of Open Source software –QSOS– [77]. Besides suggesting a number of new evaluation criteria that reflect the nature of OSS components, these proposals share the same fundamental selection principles as those for COTS components. For instance, specific evaluation criteria for OSS components are further explored by the QualOSS Model Framework [19], the QualiPSo model of OSS trustworthiness [27], and EFFORT (Evaluation Framework for Free/Open source projecTs) [6]. Some works also focus on OSS licensing aspects that are stated to have an impact on the success or failure of OBSD, for instance, Sen et al. [79], [83], and [4]. However, although major legal aspects of using OSS components and related strategies for mitigating risks have been discussed, few follow-up studies have been performed to examine how the OSS licensing issues are managed in practice [56], [78].

In general, existing proposals for selecting COTS and OSS components range from suggesting sets of evaluation criteria, legal issues, and changes to the software development processes to proposing novel technologies emerging from other areas such as decision support systems, method engineering, strategic contracting and procurement, simulation, and formal reasoning. However, there is empirical evidence indicating that most of these methods have been scarcely adopted by industry [7], [50]. Furthermore, existing studies mainly focus on OTS, limiting the understanding of potential practical differences among integrating OSS and COTS –if any-

(see Section 2.3 for a summary of current evidence).

Comprehensive surveys about the extensive work done regarding OTS component selection can be found in [10], [43], [49], [54], [58], [59].

2.2 Requirement-component mismatch resolution

The inherent volatility of requirements has been recognized since RE appeared as a discipline [42]. Requirements change in accordance with several factors such as laws (much like physical laws that prescribe physical phenomena) or changes in stakeholder requirements [42]. In OBSD, the volatility of requirements is claimed to be even more dramatic because of the changing nature of OSS components [7]. On the one hand, OSS components already have built-in capabilities (sometimes even unknown capabilities, not all of which are required or even desired) and these capabilities must be assessed to understand which system needs cannot be satisfied by a single component (or collection of components) [47]. On the other hand, OSS components undergo changes in their capabilities as their OSS communities release new versions of the components, and sometimes the communities terminate the support for their components' older versions. Thus, all of these potential changes on the system requirement and the OSS components not only introduce challenges to the integration of the components [13], but they might also affect the way requirements are elicited, specified, assessed, and managed [42].

A plethora of theoretical proposals have been suggested to deal with requirement-component mistmatches. Researchers have suggested that requirement-component mismatches appear from the very beginning, when determining the extent to which system requirements can be satisfied by a software component's features [64], [75], when it takes a long time for external support [2], or when there is a need to adapt to new changes in system requirements [51].

Some authors propose that requirement-component mismatches could be solved by modifying or adapting the selected component to fit the system requirements [2], [51], [60]. Maiden and Ncube [55] suggested that the process of solving requirements mismatches should be iterative, starting from a customer's initial wish-list and the components available in the marketplace. Then, the mismatches would progressively force requirements negotiation and candidate filtering until the final component is selected. Other proposals have suggested goal-oriented approaches for considering mismatches at the business level and then define goal matching as the conceptual framework for resolving them or for promoting call-for-tender processes [2], [63].

Other works propose specific strategies to handle requirement-component mismatches when integrating OSS components. For instance, Li et al [50] recommend a close and long-term relationship with the corresponding OSS communities in order to solve potential mismatches. According to this, OSS component integrators should not only download software from the OSS community, but they should also participate and collaborate with OSS communities by suggesting new requirements, making modifications to the existing ones, or uploading local modifications [24]. This relationship between integrators and the OSS community is supposed to benefit both the OSS communities and the users [38]. Another suggested strategy to deal with requirement–OSS component mismatches in the maintenance phase is to build an internal OSS reuse repository, which includes the source code, documentation, and previous users' feedback about OSS components [62]. However, some researchers regard having such an internal comprehensive repository of OSS components as being unrealistic [28].

Despite the existence of all of these proposals and suggested strategies, a fundamental problem is that there is a lack of evidence from the software industry that supports them [20]. No evidence exists on how the industry solves requirement-component mismatches in OBSD [25], [42].

2.3 Body of evidence

This subsection summarizes representative studies that offer industrial evidence about integration practices for OTS components. Such representative studies were taken from a previous summary of evidence about OTS components selection published by Ayala et al. [7] and were complemented with current literature reviews [10] to collect more recent studies presenting industrial evidence. However, no additional studies presented industrial evidence related to RE were found, as also stated by Daneva et al. [25]. In this paper, we took such a list of studies previously surveyed in [7] to extract evidence related to RE practices especially related to matching and/or mismatch resolution approaches, as summarized in Table 1.

Table 1. Summary of existing evidence related to RE practices in OBSD

Paper	Research Agenda	Findings related to RE practices in OBSD
Torchiano and Morisio, 2004 [88]	An interview study of third-party component usage in IT companies in 2002	▪ **EV1.** OSS is often used as closed source.
		▪ **EV2.** Integration problems result from lack of compliance with standards; architectural mismatches constitute a secondary issue.
		▪ **EV3.** Custom code mainly provides additional functionality.
		▪ **EV4.** Integrators seldom use formal selection.
		▪ **EV5**. Architecture is more important than requirements for product selection.*
		▪ **EV6.** Integrators tend to influence the supplier on product evolution whenever possible
Li, Conradi, Slyngstad, Torchiano, Morisio and Bunse, 2008 [51] Li, Conradi, Bunse, Torchiano, Slyngstad and Morisio, 2009 [50]	Series of empirical studies focused on process improvement and risk management in the development of systems that integrate third-party components (from 2003 to 2005)	▪ **EV7.** Companies use traditional processes enriched with specific activities to integrate components. ▪ **EV8.** Integrators select components informally. They rarely use formal selection procedures. ▪ **EV9.** There is no specific phase of the development process in which integrators choose components. ▪ **EV10**. Components only rarely have a negative effect on the overall system's quality. ▪ **EV11**. Integrators usually use OSS components in the same way as commercial components (i.e., without modification). ▪ **EV12**.Although problems with components are rare, the cost of locating and debugging defects is substantial. ▪ **EV13**.The relationship with the component provider involves much more than defect fixing during the maintenance stage. ▪ **EV14.** Involving clients in component decisions is rare and sometimes unfeasible. ▪ **EV15**. Knowledge that goes beyond components' functional features must be managed.
Chen, Li, Ma, Conradi, Ji and Liu, 2008 [20]	A web-based survey on software development practices using OSS in the Chinese software industry in 2007	▪ **EV16.** No formal methods were used to find and decide components. Familiarity was mainly used for evaluating and deciding components. ▪ **EV17.** Chinese integrators ranked requirements compliance as the most important criteria to compare OSS components, rather than architecture compliance. Technical support from the OSS community and licensing issues were regarded as the least important criteria to evaluate OSS. ▪ **EV18**. Regarding licenses, most respondents did not understand OSS licensing terms very well and stated having only partly read OSS licensing terms. On the other hand, twenty-one percent of the respondents had never encountered OSS license-related troubles, while the remaining respondents rarely encountered such problems. ▪ **EV19.** Few respondents stated having contributed to the OSS community due to limited time and personnel resources. Other ways of participating in the OSS community, such as providing feedback and reporting bugs or proposing new features and trial implementations of these features, were considered more cost-effective for such respondents.
Land, Sundmark, Lüders, Krasteva, and Causevic, 2009 [49]	A web-based survey about how software reuse is performed in practice	▪ **EV21.** Integrators evaluate components insufficiently and use test cases and prototyping for evaluation.
Ayala, Hauge, Conradi, Franch and Li, 2011 [7]	An interview study with industrial practitioners to investigate third-party component selection practices	▪ **EV22.** The use of informal procedures to search for, evaluate and choose components was the most popular way of selecting components. ▪ **EV23.** Previous experience with and criticality of the component in the whole system were the most influential factors leading the way companies selected components. ▪ **EV24.** OTS components are mostly selected at early stages of software development. ▪ **EV25.** OTS component decisions are mainly taken by the development team. ▪ **EV26.** Integrators informally share their knowledge and experience to select components. ▪ **EV27** Integrators typically use Google as a search engine to identify new components and information about them. ▪ **EV28** Integrators hardly ever use repositories to identify components. ▪ **EV29** Hiring specialized companies to select components was used as a risk-reduction strategy. ▪ **EV30.** There seems to be a potential market niche for component selection support. ▪ **EV31.** The list of evaluation criteria used to select components is neither formally established nor documented.

It is important to note that the goals of these surveyed studies were not really focused on exploring RE practices, but also on other topics related to OTS components such as component usage [88], software process improvement and risk management [51], [50], software reuse [49] or component selection practices [7]. The only study that specifically focused on software development practices using OSS is [20], but it was performed in the context of the Chinese software industry and the authors remark that their results might not apply to the Western software industry. Although these studies provide some findings that could be related to the requirement-component matching and mismatch resolution practices, their goals were not really related to these practices, hence limited information about their context is provided. As a result, further research should be done to understand the rationale and context of these practices. In addition, all these studies (except one [20]) do not clarify whether their results apply for both COTS and OSS.

Anyway, the availability of the evidence provided by these studies help us to understand the importance of approaching some open issues in order to compare, contextualize and/or better understand our results. Below we provide a brief explanation of the existing studies highligthing the evidences related to the requirement-component matching and mismatch resolution practices, as well as the impact of component integration on the software development process. Further details of the studies and their assessment should be consulted at [7] or their corresponding publication.

Torchiano and Morisio [88] performed a qualitative study on the use of OTS components in seven IT companies in 2002. The study identified six theses on third-party component usage. These theses were somewhat related to requirement-component matching and resolution approaches and are stated as EV1-EV6 in Table 1. From 2003 to 2005, Li et al., [50] performed a series of empirical studies aimed at testing and clarifying the theses stated by Torchiano and Morisio regarding the integration of OTS components [88] with focus on software process improvement and risk-management issues. Nine of the ten facts that summarized the conclusions from Li et al. are related to requirement-component matching and resolution approaches and are stated as EV7-EV15 in Table 1.

From August 2005 to November 2006, Chen et al. [20] performed a study to investigate the major challenges facing the Chinese software industry using OSS components. Their conclusions included evidence denoted as EV16-EV19 in Table 1. It is relevant to mention that this study is the only one that focused on OSS components and does not consider COTS components. Furthermore, it was restricted to Chinese industry, so the same authors declared that there may be significant variations with respect to industry in Western regions.

Land et al. [49] carried out a web-based survey to gather information about how software reuse was performed in practice. Given the general nature of this survey, the findings regarding practices related to RE were limited to a single observation, which is denoted as EV21 in Table 1. Finally, Ayala et al. [7] performed a survey based on semi-structured interviews with 23 employees from 20 different software intensive companies that integrate OTS components and obtained evidence related to the component selection processes. It is important to note that this was the only study that focused on OTS selection practices, so related evidence was gathered and the results are stated as EV22-EV31 in Table 1.

Despite the existence of these studies, the requirement-component matching and mismatch resolution practices in industrial OBSD is still unclear. Some researchers assume that system requirements are losing their importance when integrating OTS components because the stakeholders typically try to adapt to what is already available; other researchers assume that system requirements provide selection/matching criteria and drivers for product change [1], [33], [47], [75]. However, none of these positions has been explored further nor confirmed in industrial practice neither for OTS nor OSS components [37].

3. Research goals

As stated above, although existing research has helped us understand several factors of OBSD, the requirement-component matching and mismatch resolution approaches in industrial OBSD has not been sufficiently explored [25], [42]. A fundamental problem is that there is little empirical evidence showing how industry deals with the evolution of OSS components and the volatility of system requirements in OBSD projects. A better understanding of this phenomenon is required to foster the necessary alignment and synergy between research and industry in order to develop effective RE related solutions [42], [48], [75].

The general objective of our research is:

To explore and describe the practices, problems and challenges of current system requirements-OSS components matching and mismatches resolution approaches in software development projects that integrate one or more OSS components into their software products.

This general objective has been broken down into three specific research questions that provide a focus for our empirical investigation:

RQ1. *How is the requirement-component matching process conducted in industrial OBSD projects?*

In order to undertand industrial requirement-component matching practices, our goal was to gather information about decisions and processes performed by practitioners to search, evaluate and decide OSS components. RQ1 was specially designed to gather richer qualitative information than previous industrial studies summarized in Table 1, to better understand: a) the potential influence of OSS components on RE aspects of requirement-component matching processes and b) the potential particularities of integrating OSS instead of OTS. We focus mainly on: 1) the software development stages where OSS components are decided and the rationale behind the need of integrating them at these stages; 2) how the search, evaluation and decision-making activities are done. In addition, as some RE literature has suggested, without any available industrial evidence, that architecture plays a relevant role in the requirement-component matching processes [42], [85], we also investigate about the influence of OSS components on the definition of the system architecture.

RQ2. *How is the requirement-component mismatch resolution process conducted in industrial OBSD projects and what factors influence it?*

In order to undertand the practices related to solving potential requirement-component mismatches in industrial OBSD projects, our goal was to gather information about decisions, strategies, and processes faced by practitioners to solve functional and non-functional mismatches. In addition, we inquiry about the factors that influence the way requirement-component mismatches are solved. To our knowledge, this is the first industrial study that addresses requirements-component mismatch resolution approaches in OBSD.

RQ3. *How OSS component integration impacts on RE and the software development process and what are the main problems of integrating OSS components?*

The focus of this reseach question is twofold. First, it focuses on exploring the influence of OSS components on OBSD and RE related activities. Second, it inquires about main problems related to the integration of OSS components. We paid special attention to the following: 1) the potential change of the system requirements and/or the OSS components during OBSD since the literature has emphasized it as being a relevant source of mismatches [13], [47]; and 2) licensing issues since they have been described as being a relevant challenge of OBSD [85]. To our knowledge, no previous industrial research has focused on these aspects.

Fig. 1 shows the mapping of the research questions with RE activities in OBSD projects.

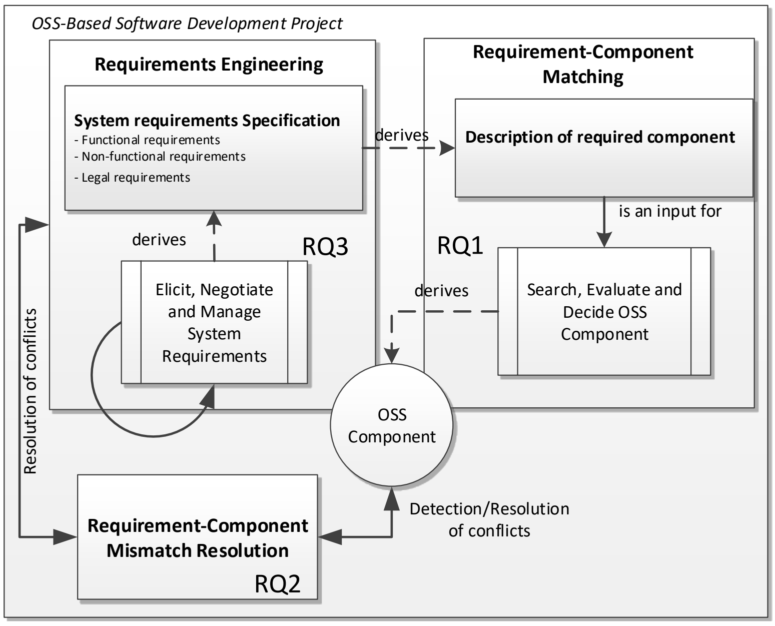
OSS-Based Software Development Project
Requirements Engineering
System requirements Specification
- Functional requirements
- Non-functional requirements
- Legal requirements
derives
Elicit, Negotiate and Manage System Requirements
RQ3
Requirement-Component Matching
Description of required component
is an input for
derives
RQ1
Search, Evaluate and Decide OSS Component
derives
OSS Component
Resolution of conflicts
Detection/Resolution of conflicts
Requirement-Component Mismatch Resolution
RQ2

Fig. 1 Mapping of the research questions

4 Research methodology

The maturation, acceptance, and adoption of good SE ideas depend on many factors, one of them being the availability of evidence [32], [46]. In order to gather and understand evidence for answering our research questions, we devised a qualitative approach. The goal of qualitative research is to investigate and understand phenomena within their real-life context [73]. A qualitative research approach is useful when the purpose is to explore an area of interest, and when the goal is to improve understanding of the phenomena [73], [76]. Qualitative studies have been claimed to be necessary to complement quantitative ones, given that qualitative knowledge is an essential prerequisite for the generation and testing of hypotheses and for interpreting the results of such tests [76].

In line with the non-deterministic nature of the requirement-component matching and mismatch resolution approaches in OBSD [57] (i.e., contextual project circumstances might vary a decision related to the matching or mismatch resolution approach) and with the exploratory nature of our research questions; we decided to survey several OBSD projects. We based our study on semi-structured interviews, as suggested by [96], to explore several situations and to capture rich qualitative information from each of the projects' contexts.

4.1 Protocol and Research Team

At the very beginning of this research, as suggested by Robson [73] and Runeson and Höst [76], we designed a research study protocol to register and update our research questions, procedures, instruments, decisions, and deviations, as it is usual in any qualitative research. All of the researchers involved participated in the development and revision of the study protocol. The main team was made up of researchers from four countries: Spain, Norway, Sweden, and Denmark. Given this multinational composition, this protocol was essential since there were several country-related aspects (e.g., cultural issues, diverse languages, and regulations) that we needed to agree upon. Once we had formulated the research questions, we devised the most feasible methodological approach for the research team to answer them. We formed national subteams to gather data from each country based on the agreed protocol and instruments and, we held several Skype meetings whenever they were required.

4.2 Sampling

The target population of this study was practitioners that integrate one or more OSS components into their software products. To obtain the sampling population, we contacted companies from our industrial collaboration network. We did not constrain any domain or sector, the only requirement for companies to participate was that they had previously integrated OSS components into a released software product. We contacted companies by phone and/or email and asked them to participate. Once they agreed to participate, we asked them to select a suitable respondent; we asked that person to choose a project to be used as a unit of study.

We conducted two rounds of interviews. To conduct the first round, we contacted 64 companies and 15 of them agreed to participate. We presented our preliminary quantitative results from the first round of interviews in an international conference [65]. We received useful feedback from the reviewers and attendees to the conference mainly related to the need of gathering more data to be able to balance the impact of some possible confounding factors in our preliminary results. Thus, we decided to perform another set of interviews for consolidating our observations, which are presented in this paper. Please see the Threats to validity Section for further details about the goals of each round of interviews and the related confounding factors we wanted to balance.

To conduct the second round of interviews, we sent emails to those companies that did not reply during the first round, and to 24 other companies that we got through our indirect collaboration network. In this way, we achieved the participation of 10 more companies, making a total of 25 companies which helped us to consolidate the results presented in this paper.

4.3 Procedure and Instruments

Due to the potential richness and diversity of data that could be collected, we considered in-depth semi-structured interviews to be the most suitable approach for data collection for the objectives of this study. We designed an interview guide together with explicit guidelines on how to proceed with the interview so that each national subteam followed the same procedure for gathering data from the sampling population.

Semi-structured interviews helped us to ensure that common information on predetermined areas was collected from all the studied projects, and they allowed us to probe deeper when required since follow-up questions were possible when deemed necessary. We chose interviews mainly because RE practices and requirements-related concepts are understood, named, and treated very differently from project to project. For this reason, it was important to us to promote discussions and clarifications when eliciting the data, making it possible to elaborate on what we were looking for and compensating for differences in understanding, culture, and terminology.

The interview guide had five sections with both closed- and open-ended questions. The first section of this guide contained closed questions aimed at gathering as much contextual information as possible about the participating companies, respondents, and projects beforehand in order to understand potential sources of variability [30]. This information was very useful in helping us to better prepare the subsequent four parts of the interview that mainly contained open-ended questions. The interview guide is provided in Appendix 1.

We discuss the particularities of the procedures followed to apply each interview guide's questions in the context of the results of each RQ.

In order to process the data gathered from the different national subteams (in different languages), we designed transcript format guidelines for reporting the relevant native statements in English. In this way, the entire research team could assess and discuss all of the data since we all use English for work. The English report transcript guidelines used by the subteams are provided in Appendix 2.

4.4 Data Collection

The interview guide was emailed to each of the respondents one week before the interview, to allow them to prepare their information before the interview session. Each of the respondents was requested to choose a suitable project for the interview and to fill in the first part of the interview guide (i.e., information about the companies, respondents and projects). Most of the respondents sent us back the requested information some days before their respective interview session; only four of the respondents sent us back the guide the same day as the interview. Having the information in advance allowed the subteams to better prepare for the interviews. The interviews were conducted mostly face-to-face by one or two researchers from the subteams that acted as interviewers (only five of the interviews were held by phone or Skype). Most of the interviews were done in the local language of the respondents. Each interview lasted from 40 to 70 minutes and was audio-recorded and prepared for analysis through the manual transcription of the audio recorders into a predefined English report transcript template. The English report transcripts varied in length from 13 to 21 pages.

4.5 Data Analysis Procedure

We performed the same data analysis procedure for the two rounds of interviews as the two rounds used the same instruments.

We analyzed the data from the interviews using the qualitative data analysis tool NVivo [68] which has functionality for organizing and structuring qualitative data such as interview data from each respondent. We used the respondents' answers stated in the English transcript reports, and individual notes taken by the interviewer(s) during the interviews. The approach followed for open questions was a tailored thematic analysis as suggested by Cruzes et al. [22] for case-study synthesis. It consisted of the following steps:

1) extracting data from the original interviews to the English report transcripts
2) grouping the data into fundamental groups based on the questions of the interview guide
3) identifying and coding interesting concepts and findings from each group
4) translating codes into themes
5) discussing the codes and themes and linking relevant themes together.

Step 1 was performed by each subteam using the English report transcript template and guidelines (provided in Appendix 2).

Step 2 was performed by four members of the research team using the data gathered in the English report transcripts from each subteam.

Step 3 was performed individually by two members of the research team. Each researcher assessed the answers from each respondent for each open question of the interview guide. For each question, there were at least 25 answers corresponding to each respondent. Based on all answers to each question, each researcher identified interesting concepts and findings. Each researcher identified around 3-12 codes for each question.

The resulting individual codes generated by the two researchers for each question were discussed among them in order to reconcile their visions and generate consensuated themes for each question in Step 4. Each question ended up having from 2 to 9 themes. It is worth to mention that theinformation gathered from the second round of interviews did not add new themes, but it helped us to consolidate and enrich our understanding of the themes identified in the first round of interviews.

The aim of Step 5 was to discuss the codes and themes for each question identified in previous steps with the rest of the team in order to ensure the correct interpretation of each theme and the evidence that supports it.

For instance, for processing the information coming from the question "1.14 Did you have previous experience with OSS-based development before joining the project?" we used the procedure described above. Hence, based on the answers of the respondents it was decided to categorize respondents'answers into 3 themes: a) Extensive experience: those that explicitly said that they have "significant/substantial experience" or mentioned more than four projects where they played a crucial role. b) Medium experience: those that explicitly said that they have “some experience” or mentioned at least 2 previous projects. c) Limited Experience: those that explicitly said that their experience was scarce or limited but had at least participated in a previous project. In most of the cases, the respondents explicitly mentioned how they considered their experience (i.e., extensive/medium/limited).

In the context of our research it was particularly important to search for possible associations among the data in order to realize those associations than could show some impact of OSS integration on RE and the software development process. This activity was supported by the use of the software tool Weka [92] to generate and visualize clusters that helped us to better relate and interpret our qualitative data. Cluster analysis is an explorative analysis that tries to identify structures within the data in order to identify characteristics and homogenous groups of cases [29]. We used the simplest cluster analysis algorithm, named *simple k-means,* to identify groups that were subsequently assessed and discussed to confirm their meaningfulness. In addition to the cluster analysis, we also generated frequencies of codes just as an indicator of popular and unpopular practices in our sample. In this way, we achieved a broader understanding of the practices in each project. Consequently, our discussions led us to split, modify, discard, or add themes to ensure that all answers and their contexts were well-represented. We tried to be thorough with the codes and themes in order to include as much detail provided by the respondents as possible.

Using Weka to conduct cluster analysis helped us to relate sets of data to observe potential meaningful relations. The procedure we used was simply to relate two sets of data that we thought could be related and to test the generation of n clusters over such sets of data. Given the small set of instances we had, we usually tested from 2 (that is the minimum number of clusters) to 6 clusters until we found a cluster’s number that showed potentially meaningful relations. In most of the cases, when we related the sets of data, we did not observe any meaningful relation in the generated clusters, but in some cases, it helped us to identify likely meaningful associations. This was particularly useful to find associations among the data. The cluster tables show how diverse attributes (stated in rows and columns) come together to form clusters. The number value in each cell represents the average of the attribute in the cluster formed by the intersection of the attribute in the row and the attribute in the column. Thus, each cluster shows a type of behavior from which we can begin to draw conclusions.

For instance, using Weka, we related the data gathered about the percentage of experience of the teams with OSS with the level of detail used to specify requirements from each project, and got some potential insights with the generation of 3 clusters (as discussed in the context of Table 13). Although we used Weka in a basic way, it helped us to visualize in a more effective way some possibly relations that were then discussed with the team in order to realize their suitability.

4.6 Context of the Studied Projects

4.6.1 General characteristics of respondents, companies and projects

Table 2 and Table 3 summarize the main characteristics of the respondents, participating companies and projects, respectively.

Table 2 Characteristics of the respondents

ID	Highest Education Degree	Job Position	Previous OSS Experience	Country
A	MSc in Computer Science	Systems Engineer	Extensive	NO
B	MSc in Computer Science	President	Extensive	NO
C	MSc in Computer Science	CEO	Extensive	NO
D	PhD in Engineering	Software Leader	Extensive	NO
E	BSc in Computer Science	Programmer	Medium	NO
F	BSc in Information Systems	Senior Consultant and OSS Leader	Extensive	NO
G	MSc in Computer Science	Managing Consultant	Extensive	NO
H	MSc in Electronics Engineering	Chief Engineer	Extensive	NO
I	MSc in Computer Science	Programmer	Extensive	NO
J	MSc in Computer Science	Chief Engineer	Extensive	ES
K	BSc in Computer Science	Chief Engineer	Medium	ES
L	BSc in Computer Science and MBA	Project Leader	-	ES
M	BSc in Computer Science	Product Selling Team	Extensive	ES
N	BSc in Computer Science	Functional Analyst	Limited	ES
O	MSc in Computer Science	Research Assistant	-	ES
P	BSc in Computer Science	CTO	Extensive	ES

Q	PhD in Artificial Intelligence	CTO - Technical Director	Medium	ES
R	BSc in Computer Science	Software Architect	Medium	ES
S	MSc in Computer Science	Software Developer	Limited	SE
T	MSc in Computer Science	R&D Director	Medium	SE
U	Data Analyst	CEO	Extensive	DK
V	High School Diploma	Technical Leader	Medium	DK
W	BSc in IT	System Developer	Extensive	DK
X	BSc in IT	IT System Specialist	Limited	DK
Y	MSc in Economics	Managing Partner / Owner	Extensive	DK

(-) Not declared. NO: Norway, ES: Spain, SE: Sweden, DK: Denmark

Table 3 Characteristics of Participating Companies and Projects

ID	Nb. employ ees	Project staff size (number of people)	% staff with previous experience in OBSD	Some OSS used in the Project	% OSS proportion	Total project effort (person/month)	Main functionality	Main application sector of the project
A	170	20-25	30%	JBPM, Jetty, Spring, LogBack, Maven	90%	>2000	Messaging system conforming to NATO standards	Public Sector
B	*	4	50%	Impact, LPng	10%	480	*	ICT
C	3	2	100%	SolR, Xapian Twisted, NLTK.	80%	12	Search platform on top of various systems	ICT
D	350	18	25%	Linux Kernel, MD5 Checksum	*	*	*	ICT
E	500	2	50%	PDfLib, OpenPyExcel	77%	18	Document acquisition and report mining of semi-structured documents.	Public Sector
F	*	200	60%	Flex Framework, Batch part of Spring	75%	*	System to assess pension rights and calculate payments	Public sector
G	230	4	100%	WideShot, CryptoPP, ParseXs, Weaks	10%	36	System to sign multiple documents for procuring financial products	Bank
H	190	20	100%	JBOSS, OpenSummer, USD	66%	1000	System for administrative tasks	Public sector
I	6	1.5	66%	Python, SOAP, Django	90%	3	Content management system to manage real estate issues	Real state brokers
J	4	3	100%	Sun grid engine, cluster FS, Linux Debian, Ganglia	90%	30	Computing cluster system	Public Sector
K	100	3	100%	Apache, MySQL, PHP, FFTP tools	5%	7.5	Architectural improvement of existing web systems	ICT
L	*	5	100%	Mantis, Ant, Apache	80-90%	72	Management of academic tasks	Public sector
M	150	6	100%	Hibernate Libraries, Spring, Acegy, Jasper Reports, DOJO, Apache, Quark	20%	157	Management of curricula of academic employees	Public sector
N	30	7	14%	Jenkins, Cucumber, Mercurial	10%	84	Corporative social web	ICT
O	15	3	100%	Joomla	50%	56	Improvement of the visualization aspects of a web system	ICT
P	5	2.5	67%	Zope, Plone	99%	6	Management of a cultural agenda	Public sector
Q	14	3	100%	Varnish, Engine egg	80%	9	OSS Plugins to access infrastucture services	ICT
R	500	25	80%	Jasper Reports, Junit, Jmeter, MediaWiki, OpenCSV	30%	900	Integral management of water provision and treatment	Public Sector
S	2	2	100%	Eclipse, MySQL, RXTX, Palcom	60%	36	System for management of medical equipment	Medical
T	6000	250	50%	Android kernel	50%	1000	Mobile phone platform	ICT
U	11	2	100%	Speed -Typo3CMS, FPDF, Apache, Stability	40%	20	Front-end and back-end for electricity selling company	Public Sector

ID	Nb. employees	Project staff size (number of people)	% staff with previous experience in OBSD	Some OSS used in the Project	% OSS proportion	Total project effort (person/month)	Main functionality	Main application sector of the project
V	2500	4	100%	Mongo DB – binary serialization	100%	8	Data access to media metadata and binary assets for on-demand video/audio services	ICT
W	4	10	50%	Apache, MySQL, PHP Suite	100%	24	A movie database	ICT
X	1	1	100%	Stability – Ubuntu Enterpise Cloud (UEC) & Eucalyptus, NappIt, pfSense, FreeBSD based firewall	100%	6	Private cloud computing in an academic environment	Public Sector
Y	7	1	0	Zope, Plone, Apache, MySQL, R, Ubuntu	100%	3	Clinical database system for use in all regional hospitals	Medical

*Respondent did not know the answer or asked to keep this information confidential.

The resulting set of participating companies varied in size from 1 to 6000 employees; most of them (20 out of 25) were relatively small (i.e., less than 500 employees).

The respondents occupied different positions in their respective companies and had actively participated in RE related processes in at least the project that they based their answers on. All but one of the respondents had an education background related to computer science or information systems engineering (that respondent had a background in economics). Eleven of the respondents had a Master's degree, 10 had a Bachelor's degree, 2 had a PhD, and 2 had an undergraduate/high school degree. Most of the respondents (fourteen out of twenty-five) had extensive experience in OBSD projects, 6 respondents said "medium" experience, 3 had limited experience, and 2 did not answer the question.

The portion of the whole system covered by the OSS components varied from 5% to 100%: 13 projects ranged from 70% to 100%; 4 projects ranged from 50% to 69%; 7 projects ranged from 10% to 49%; 1 project based only 5% of the whole system on OSS components. In general, it can be observed that most of the assessed projects made intensive integration of OSS; in seventeen out of twenty-five cases, OSS components covered more than 50% of the system requirements.

The percentage of project members that had previous experience with OSS for each analyzed project ranged from 0 to 100%: Thirteen out of twenty-five projects had 100% of their project staff with 100% OSS experience; eight out of twenty-five projects had a percentage higher than 50%; and four out of twenty-five projects had between 0% and 30% of their staff with previous experience in OSS components integration. Thus, it can be observed that most project members had previous OBSD related experience.

The set of projects considered as units of study used a variety of OSS components that ranged from libraries to more complex frameworks and solutions. Eleven of the projects were related to the public sector such as electricity management, defense communications, water treatment, and education, while 14 projects were from other non-public ICT-related sectors. With regard to size, the total effort spent on the analyzed projects involved from one to 150 people (ranging from 1 to 2000 person/month).

It is important to note that even if it was not intentional, the majority of our sample projects developed web information system applications and did not cover critical domains such as real-time or life-critical requirements.

4.6.2 Software development processes

Since the software development process used could influence the RE practices, we asked the respondents to describe such processes in order to better understand their contexts. Table 4 summarizes the results.

Table 4 Software development processes used in the projects

Software development process	**Respondents**	**Total**
Agile	C, D, F, G, H, I, J, K, N, P, Q, T, U, V, W, X, Y	17
Iterative/Incremental	A, E, M, O, R, S	6
Waterfall	L	1
No answer	B	1

Most of the respondents (seventeen out of twenty-five) used agile-based processes. This wide adoption of agile methodologies in our assessed projects seems to be in line with previous observations made in the RE literature indicating that agile RE practices are gaining attention in industry [25].

4.6.3 Main source of requirements approval

Given our intention of further assessing RE issues, we considered important to describe the main sources of the system requirements approval for each of the projects in order to realize the degree of flexibility that the requirements could have. The respondents classified their system requirements as being approved mainly by internal or external stakeholders

Internal requirements sources implied that the system requirements were approved mainly from the software development team or internal departments of the company (i.e., requirements were market-driven [11]). Some of the participanting companies that stated to have internal requirement sources had specialized departments to harvest and manage important requirements of their software products. Even though these departments were part of the organization, they were usually seen as "customers" by the development teams, mainly because these departments were considered sources of requirements. For instance, projects K and N stated having special marketing departments that played an important role in the elicitation and specification of their system requirements. In addition, project N also had a usability department to support human-computer interaction aspects of their products. Finally, project F also involved a legal department since the main functionality of the project was related to legal aspects.

External requirements sources refer to requirements that are approved by an external stakeholder, usually a client who paid for the project (i.e., customer-driven requirements [11]). All but one of the respondents declared the client as the main source of their requirements (that project (L) mentioned not just the client but also an external consulting company as the external approval source of their requirements).

Table 5 presents the types of requirements sources and their corresponding respondents. It can be observed that the sampling set of projects was quite balanced regarding internal and external requirements sources.

Table 5 Types of requirements sources of approval

Types of requirements sources of approval	**Respondents**	**Total**	**Percentage**
External requirements sources of approval	A, G, H, I, J, L, M, P, S, T, U, V, W, Y	14	56%
Internal requirements sources of approval	B, C, D, E, F, K, N, O, Q, R, X	11	44%

4.6.4 Notation and level of detail of requirements

With regard to notation, most of the respondents declared using more than one approach to specify their requirements. All the approaches were based on the use of natural language. The majority of respondents stated using Free Text (FT): *"We did not use any specific notation, we just stated some basic definitions in spreadsheets, in natural language"* (Y). Others mentioned that they structured text sentences by using some kind of template, which we called Structured Text (ST): *"We used a template that was based on the ISO 9126 international standard"* (M). Other respondents detailed the use of Use Cases (UC): *"We paid attention to the use cases to understand the functionalities"* (B); or Flow Diagrams (FD): *"We stated some requirements by means of state diagrams mixed with structured text to understand the flow of actions"* (A). Test Cases (TC) and Mockups were also mentioned, for instance: *"We usually focus on test cases as we know the client will check them"* (U); *"We used screen-like pictures with annotations, I mean mock-ups to understand the required functionality"* (N). Table 6 summarizes the results. Please note that some respondents declared more than one notation.

Table 6 Notations used to specify requirements

Notations	**Respondents**
Free Text (FT)	A, E, F, H, I, J, L, O, P, Q, S, U, X, Y
Structured Text (ST)	A, B, C, D, G, M, W
Use Cases (UC)	B, C, G, R, V
Flow Diagrams (FD)	A, K, O
Test Cases (TC)	B, C, E, Q, U
*Mockups (M)	N
No answer	T

*By mockups we mean those artefacts such as screen-shoot like pictures (without any implemented functionality) that are aimed to show a specific behaviour of the intended system.

In order to gather information about how system requirements were specified, we asked the respondents to characterize their system requirements according to five categories: very fine-grained, detailed, medium, coarsely grained, very sketchy. We gave them some previously agreed upon examples to calibrate this scale (e.g., for very fine-grained requirements, we mentioned those requirements that are carefully stated together with explicit metrics, such as the ones used for very critical missions at NASA). All respondents provided further explanations of the level of detail used, for instance: *"We had very fine-grained descriptions because they were provided by our client"*(J); *"In this case, we used detailed descriptions because we were considering the possibility of selling the resulting product, so we wanted to have comprehensive requirements documents"*(R); *"We used medium-detailed descriptions, we tried to follow the ISO 9126 quality standard as our client requested it"* (M); *"We certainly do not provide much detail to the requirements, we coarsely stated requirements just to help the team to understand what we needed to do"* (O); *"We worked without specifications. We developed fast and small prototypes in each sprint and we made several modifications as desired by the client. So, we had very sketchy requirements descriptions"* (Y).

The explanations given by the respondents when we asked them to characterize their system requirements according to the five categories detailed above, helped us to better rely on the calibration of the provided scale as we did not experience any situation where the scale and explanation provided by the respondents differed. Table 7 summarizes the results.

Table 7 Level of detail used to specify requirements

Level of detail of requirements	Respondents
Very fine-grained	J, T
Detailed	B, P, R
Medium	A, K, L, M, N, U, W
Coarsely grained	C, D, E, F, G, H, I, O, V
Very sketchy	Q, S, X,Y

5 Results and discussion

The following subsections present and discuss the results related to each research question. We state some observations followed by a discussion that is related to their derivation. We present the results in a narrative form in order to provide detailed evidence that can help to promote further understanding by the reader when confronted with other situations that are not considered by our research questions. In some cases, we summarize our results by cluster tables or tables with frequencies of responses. In this way, we aim to provide insights into popular issues in our sample or potentially meaningful relations rather than quantitative facts (i.e., the numbers presented in the results are not aimed to be representative percentages/proportions).

5.1 Requirement-component matching processes (RQ1)

In order to analyze and understand the practices and characteristics of the requirement-component matching process, Section 5.1.1 presents the results about the software development stages where requirement-component matching processes are done and the rationale behind them. Section 5.1.2 presents the results related to OSS components search, evaluation and decision-making activities. The aspects of RQ1 presented in the subsection 5.1.1 and 5.1.2 were especially designed to gather richer qualitative information than previous related studies [7], [20], [49], [50], [88]. Therefore, the interview guide's questions corresponding to these aspects were carefully designed based on the questions and approaches from these previous studies. We are aware that the goals, contexts and approaches followed to conduct these previous studies are different than this study. Therefore, in some cases, the relationship of these previous studies with the results obtained from this study could be argued. However, we have taken care to present and discuss the results, relationships and differences among the studies with caution.

Section 5.1.3 presents the results about the influence of architecturally significant requirements on the requirement-component matching process that has not been previously approached by any other study.

Finally, Section 5.1.4 presents a summary of the relation of the RQ1 results with the previous results presented in Table 1.

5.1.1 Stages when requirement-component matching is performed

In order to inquiry when in the development process was the requirement-component matching done, we allowed the respondents to state more than one stage because several components were usually integrated in the projects. We suggested the following stages: requirements elicitation and analysis, design, and coding. Even though we let them add any other stage if they considered it necessary to describe their practices, none of the respondents mentioned any other stage. It is important to highlight that the respondents were not involved in post-release stages and this may be the reason why they did not mention maintenance. Table 8 summarizes the answers.

Table 8 Stages where requirement-component matching was performed

Software development stage	Respondents	Total
Requirements elicitation and analysis	A, B, C, D, G, H, I, K, L, O, P, Q, S, U, X, Y	16
Design	A, B, C, D, F, G, H, J, L, M, T, V	12
Coding	A, B, C, E, F, G, H, N, O, R, W	11

Fig. 2 shows the relationship between the stages where requirement-component matching was done and the main requirements approval stakeholder. The figure suggests that projects that need external requirements approval are more likely to perform requirement-component matching processes at early stages of OBSD. Observations regarding these facts are stated below.

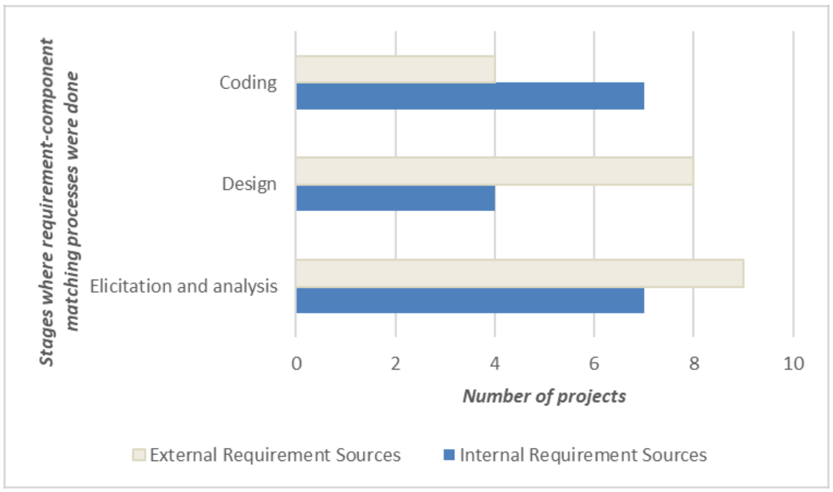

Fig. 2 Stages where requirements-component matching processes were done in relation to the source of requirements' approval

Observation 1: Requirement-component matching processes are done at different stages of OBSD projects

Researchers have claimed that components should be selected early in the development process so that potential problems can be identified as soon as possible [47] [64]. However, from the results in Table 8, it can be observed that there is not a highly dominant stage where requirement-component matching processes were done in OBSD projects. This result is in line with previous evidence related to OTS such as EV9 from Li et al. [50], but it contradicts EV24 from Ayala et al. [7] that found that most of the projects selected components at early stages. The qualitative information about project contexts helps us to understand and find explanations for this potential contradiction. Ayala et al. focused mostly on projects that selected components at very early stages (i.e., in the contractual phase) mainly because of customer contractual obligations [7]. This suggests that most of the assessed projects by Ayala et al. [7], seemed to need external requirements approval. From Fig. 2, our results suggest that OBSD projects that need external requirements approval show a slightly higher likelihood of performing requirement-component matching processes at early requirements stages. We think that this might explain why EV24 from Ayala et al. [7] contradicts EV9 from Li et al. [50]. This is due to the fact that Li et al. [50] studied a more balanced set of projects that included projects that need both external and internal requirements approval sources, as we have done in this study. Therefore, similarly to Li et al. [50], our results suggest that requirement-component matching processes are done in diverse software development stages regardless of whether or not they refer to COTS or OSS.

Observation 2: OSS components that cover coarse-grained functionalities are usually selected in early requirements stages, while fine-grained OSS components are usually selected in late development stages

For understanding the rationale behind performing the requirement-component matching processes in diverse stages, we asked the respondents for the main reasons to select OSS components at these mentioned stages.

From the answers of our respondents, we observed that one main reason for selecting OSS components at the requirements elicitation and analysis stage was an explicit requirement from the client or the development team to use an OSS solution or a specific OSS component. Requesting OSS components at the design and coding stage seemed to be a decision taken mostly by the technical team in order to facilitate the software development process. Representative quotes are: *"The OSS components were previously defined by the application. So they were a kind of pre-requisite since the very begining"* (K); *"I suggested these OSS components to the client as the only option before we started the project"* (Y); *"Using this OSS component was a pre-requisite as the organization strategically wanted to use it in order to leverage our experience afterwards"* (O); *"The customer wanted an OSS based solution for price, speed and stability, so in the analysis stage we tried to search and find OSS components that covered these client requirements"* (U); *"We as a supplier proposed the application platform and architecture as part of our bid, and from that we selected what we considered the most appropriate technical components"* (H); *"At the design stage we realized that there was a component that could save us a lot of future integration effort, so the team decided to go for it"* (F); "*In the coding stage, we sometimes realized that there were some OSS components covering a small functionality, so we tried them and use them to easy our work"* (E).

Furthermore, from associating the OSS components used and the stages where they were requested and selected, we observed that those components that were requested and selected at the coding stage seemed to be considerably smaller (in terms of covered functionality) than other components requested in earlier stages. Some representative quotes also support our observation: *"Yes, the OSS components that we select at the coding stage use to cover small functionalities"* (E)*; "The major functional components are usually described in early development stages"*(A). This suggests that OSS components that are usually selected at the coding stage mostly reuse OSS components in a pragmatic and opportunistic way. Opportunistic reuse is a need based sourcing of software components, without a prior reuse plan [41]. It means that the components are reused in a pragmatic way, just to cover (some) system requirements, without a previously established component reuse plan.

5.1.2 How are the OSS components search, evaluation and decision-making activities done?

To inquiry about the processes used to search, evaluate and decide upon OSS components, we provided the respondents with a set of potential alternatives that we obtained from previous studies on component selection [7], [20], [49], [50], [88] (see Question 2.2 in Appendix 1). However, we encouraged the respondents to comment on these alternatives and explain and describe their processes. In this way, we were able to compare not just previous results, but also to gather further details that helped us to better understand and formulate the resulting themes/categories that describe the respondents' practices. The results are summarized in Table 9. Further comments of these results are below.

Table 9 Approaches for searching, evaluating and deciding upon OSS components

Aproaches followed for searching and evaluating OSS components	**Category**	**Respondents**	**Grand total**
Search Stage	Use of domain-specific web portals	A, B, C, E, I, J, K, M, Q, R, S, T, U	13
	Use of general search engines	B, C, E, G, H, I, M, O, Q, R, U	11
Evaluation Stage	No search, but previous experience with the component	A, B, D, F, G, H, I,K, N, O, P, Q, R, S, T, U, V, W, Y	19
	Consulted with colleagues	A, B, C, F, G, H, J, M, N, Q, R, Y	12
	No search, it was imposed by the client	L	1
Resources used during the Search and Evaluation Stage	Peer review literature	B, C, E, H, M, O, T	7
	Seminars, workshops	B, C, H, I, T	5
	Consulted the customer (internal or external)	F, G, L, R	4
	Grey literature	B, E, I, T	4
	Hire an expert company	M, N	2
	Used a documented method	-	0

Observation 3: Previous own's experience or colleagues' experience with the component mainly drive OSS component searching and evaluation.

Although several respondents in Table 9 stated using domain-specific portals such as sourceforge.org or general search engines such as Google to search for OSS components and information about them, most of these respondents also commented that the searching phase was mostly influenced by previous experience and previous awareness of the component (this aligns with EV23 from [7] that focuses on OTS). This means that integrators usually use to search on domain-specific portals or general search engines, but they in advance had some preliminary ideas about the component they were looking for because they or their colleagues have previously used the component. Furthermore, Table 9 shows that several respondents stated that in some cases, they do not even use any portal/search engine for looking for components but fully rely on their colleagues or their own previous experience to decide the component to be used. For instance,: *"Well, I did not properly search on the Internet. I had been using these components from before, so I was pretty sure that they could help us to cover the functionalities"* (A); *"I used to ask some colleagues to have an idea of which OSS components were worthwhile for my task"* (Q). Another case, stated that the search phase was limited to confirming that the component requested by the client was worthwhile; *"In this case, it was 80% imposed by the client, so our team of architects just confirmed that we could use these components without problems"* (L).

Observation 4: No documented method was mentioned to be used for selecting OSS component but resources such as literature, seminars, workshops or hiring specialized companies to perform OSS selection were mentioned as valuable.

As shown in Table 9, none of the assessed projects used any documented method or procedure to search and evaluate OSS components. This is in line with previous evidence related to OTS and OSS (see EV4, EV8, EV16, EV21, and EV22 from Table 1. However, our results provide additional data to understand such evidence; e.g.the respondents emphasized that the way components were searched for and evaluated depended on the importance of the OSS component in the final product: *"For minor personal-use tools, we can find something that solves a problem and use it without a very serious process. For more significant components (i.e., those that are part of the final product), we typically do a survey to find various alternatives (we usually include both OSS and COTS). Then a review board composed of senior engineers must approve the inclusion of the components"* (A); *"In cases when we know the component very well, we base our evaluation on our experience; but in other cases, we investigate the alternatives and make some practical tests to see which one best satisfies our needs"* (R).

Other resources that were mentioned to be used to support searching, evaluating and deciding OSS components, were: peer review literature, seminars and workshops, and grey literature (apart from the website of the OSS community). In addition, two of the participating organizations hired experts to select and integrate critical OSS components as a risk reduction strategy. This result is is line with EV29, which comes from a study focused on OTS in 2011 [7]. It could suggest that hiring expert companies to perform OSS selection is still a valued industrial resource used to support the component selection.

5.1.3 Influence of architecturally significant requirements and OSS components on requirement-component matching

The literature suggests the emergent role of software architecture as a critical stabilizing force to moderate, constrain, and enable suitable requirement-component integration and evolution [42], [88]. However, as far as we know, there is no evidence on the role of OSS components and architecturally significant requirements in the requirement-component matching processes. Thus, we asked the respondents to state what was decided first: the application platform and architecture or the major OSS components. Table 10 summarizes the answers. Details of our observations regarding this are stated below.

Table 10 The influence of major OSS components on system architecture

Categories	Respondents	Total
Platform and Architecture were decided first before considering OSS components	D, H, J, K, L, M, N, R, T, U, V, W, X	13
Major OSS components were decided first before considering architectural issues	A, C, E, F, G, I, O, P, Q, S, Y	11
No answer	B	1

Observation 5: OSS components might influence architectural aspects of OBSD in the requirement-component matching process, but this potential influence greatly depends on the consolidation and popularity of the OSS component and its application domain.

Thirteen out of 25 respondents stated that the platform and architecture were defined first regardless of the OSS components to be used. Some details of their answers are: *"The architecture was decided first as it was constrained by the system that already existed"*(K). *"We first decided the architecture, and then we identified three or four candidate components for each part of the architecture"* (J). However, 11 out of 25 respondents stated that OSS components led or influenced architectural decisions. Representative examples of their responses are: *"We already used this content management system (CMS). We have previously sold some other systems based on this CMS, so we mastered it. Thus, so much of the architecture of this project was predetermined from that because we aimed to capitalize our knowledge gained in previous projects*" (I); *"Some OSS are as defacto standards, so you better adapt to them"* (O); *"I would say it was both core OSS components and the requirements we had that led us to define the architecture. In fact, we first decided the OSS components, then we did a test and then some adjustments to the architecture were done underway"*. One respondent did not provide details for this question since this process was run by another team of software architects.

Although the number of respondents that stated the influence of OSS components on architecturally significant requirements was quite similar to those that did not state such influence, our qualitative data helps us to better understand and contextualize a potential explanation for these results.

When we assessed our results using Weka, we found that all projects that were architecturally influenced by OSS components referred to web information system applications, while all projects that did not referred to web information systems belonged to the category that did not state any OSS influence on architectural aspects. Thus, one possible explanation of the OSS influence on architectural aspects could be related to their application domain. The web information system domain has evolved in the last few years and consolidated OSS frameworks have emerged that help integrators to better confront integration issues. Currently, many OSS frameworks provide common architecture patterns and infrastructures for web application development. Thus, the use of these existing frameworks might explain the relevance of following OSS architectural recommendations to avoid integration problems.

5.1.4 Summary of findings related to RQ1 and their relation to previous evidence

As discussed above in the context of each RQ1's aspect, the results from this study help to better understand several aspects of requirement–OSS component matching and are specifically addressed to OSS. Table 11 summarizes the relation of the results from this study with previous evidence presented in Table 1. It also highligths new evidence raised by this study.

Table 11 Summary of RQ1 findings and their relation to previous evidence from Table 1

	Observation	**Related to previous evidence about**		**Confirms**	**New evidence**
		OTS	**OSS**		
RQ1	1-Requirement-component matching processes are done at different stages of OBSD projects	EV9 EV24		EV9	
	2-OSS components that cover coarse-grained functionalities are usually selected in early requirements stages, while fine-grained OSS components are usually selected in late development stages				√
	3-Previous own's experience or colleagues' experience with the component mainly drive OSS component searching and evaluation.	EV23 EV27		EV23 EV27	
	4-No documented method was mentioned to be used for selecting OSS component but resources such as literature, seminars, workshops or hiring specialized companies to perform OSS selection were mentioned as valuable.	EV4, EV8, EV21, EV22 EV29	EV16,	EV4, EV8, EV16 EV21, EV22 EV29	
	5-OSS components might influence architectural aspects of OBSD in the requirement-component matching process, but this potential influence greatly depends on the consolidation and popularity of the OSS component and its application domain.				√

Table 11 shows that observations 1, 3 and 4 collected from this study are in line with previous studies related to component integration, regardless of whether or not they focus on OTS or OSS. Notice that in comparison with the previous evidence related to OTS or OSS, our results provide more qualitative information to understand the contex of requirement–OSS component matching. Thus, we suggest that no fundamental changes exist in requirement-component matching processes applied when integrating OTS or OSS.

Although some years have passed since these previous studies were performed, it seems that the situation has not changed much regarding the requirement-component matching process as still most of the OSS components were mainly selected on the basis of previous experience without any documented method (as stated in observations 3 and 4). While the value of experience is important, considering it to be the most influential factor for selecting components is actually hampering the full exploitation of the potential benefits of OSS components availability.

In addition, we found two new observations (observations number 2 and number 5) that provide evidence about requirement–OSS component matching that have not yet been stated by other researchers and contribute to the understanding of the industrial practices.

5.2 Requirement–OSS component mismatch resolution processes (RQ2)

To inquire about functional requirement-component mismatches, we asked the respondents explicitly about it. We assessed their answers and devised a set of scenarios that describe their approaches. Section 5.2.1 summarizes these scenarios. Section 5.2.2 presents the results related to requirement-component mismatches originated by non-functional requirements. Section 5.2.3 highlights the factors that influenced the requirement-component mismatch resolution processes followed in the assessed projects.

5.2.1 Requirement-component mistmatch resolution scenarios

As our aim was to develop a set of scenarios aimed at describing the main characteristics of the approaches followed to solve conflicts between a system requirement (*r*) and an OSS component (*c*), we proceeded as follows to elicit this information: First, we asked the respondents "What did you do when functional requirement-component mismatches occurred in this project?" (See question 3.1 from the Interview guide). Then, in order to guide their answers towards the description of their different approaches followed, we further inquired "What do you usually do when the decision is that the OSS component or System requirement should be affected to solve the mismatch?". As a result, they described the most usual action they do in these two scenarios. Furthermore, in order to ensure that we covered all the mismatch resolution approaches followed, we added: "Did you have any other approach to solve the potential mismatches?" and some of the respondents added the case where neither the OSS component nor the system requirements were affected. Proceeding in this way, we ended up with details about the most usual ways to solve mismatches in the scenarios presented in Fig. 3 (i.e., OSS component (c) is affected; System requirement (r) is affected; (c) and (r) are not affected).

Fig. 3 summarizes the different situations and their conflict resolution approaches.

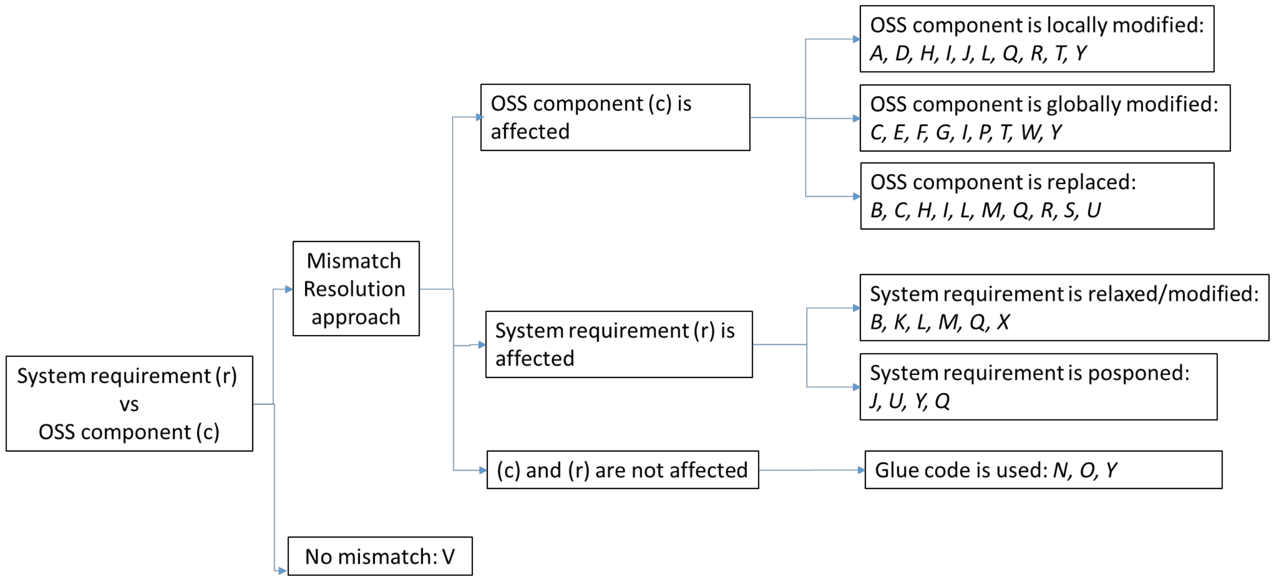
OSS component is locally modified: A, D, H, I, J, L, Q, R, T, Y
OSS component (c) is affected
OSS component is globally modified: C, E, F, G, I, P, T, W, Y
OSS component is replaced: B, C, H, I, L, M, Q, R, S, U
Mismatch Resolution approach
System requirement is relaxed/modified: B, K, L, M, Q, X
System requirement (r) is affected
System requirement (r) vs OSS component (c)
System requirement is posponed: J, U, Y, Q
(c) and (r) are not affected
Glue code is used: N, O, Y
No mismatch: V

Fig. 3 Scenarios of matching and conflict resolution between system requirements vs. OSS components

5.2.1.1 No mismatch

Twenty-four out of twenty-five respondents reported situations where mismatches occurred, and only one respondent stated the situation where no mismatches were experienced. When we asked him about this, the respondent stated *"We did not have mismatches because the client wanted this specific OSS component, so his/her requirements were actually in line with it"* (V). In this respect, we emphasize that some system requirements seem to be greatly influenced by the OSS components from their very origin, so no mismatches are likely to appear.

5.2.1.2 Mismatch resolution approaches

The resolution approaches reported by the remaining twenty-four respondents were grouped on the basis of the issue affected (i.e., the OSS component or the functional requirement).

a) The OSS component (c) is affected

Most of the respondents experienced this situation and commented that modifying the OSS component to accommodate it to the system requirements was the most common resolution approach in their projects: *"For us, there is no case of giving up on the system requirements. System requirements are usually a first priority, so we usually adapt the OSS component to fit them"* (C). Some variations with respect to this strategy were the following:

- *The OSS is locally modified*: This scenario emphasizes that changes to the OSS component are implemented at the project level in order to adapt it to the system requirements, but they are not shared to the OSS community. This practice is reported by some respondents as being dependent on the importance or volume of the modification (e.g., *"Yes, we have modified the OSS component, but we do this in very rare cases and just for minimal things"* (R)). Although several respondents reported this behaviour, some of them also highlighted that it was a risky practice that should be done with caution to prevent losing the synchrony with future OSS component updates (e.g., *"We have certainly modified OSS code, even if we are aware that it would bring some future update problems. So we just do it in fully justified situations"* (L)). Other respondents stated that they actually avoid this strategy as company policy: *"No, we have not touched any OSS code. Our policy is never to change OSS code locally…"* (P)).
- *The OSS is globally modified*. This scenario emphasizes that changes to the OSS component to adapt it to the system requirement are done in the form of patches or plug-in components that are shared with the OSS community. This strategy aims to foster the synchrony with the mainstream OSS project. However, some respondents mentioned some notable considerations. 1) Sending OSS changes back to the OSS community is not a trivial task. As one of the respondents stated: *"We tried to contribute some changes we did in the OSS code, but we gave up when we realized that all code was linked with our specific things and it would take many months to isolate it in order to share it with the community"* (A). 2) An informed decision about the need for keeping synchrony with the mainstream OSS project should be taken. Some representative responses are: *"We should analyse if we really need to be in synchrony with the OSS project; if it is not necessary for us, we don't"* (I), *"We selectively decide if it is worth doing the patch. Our team usually votes to decide"* (Y). 3) It must be determined whether or not the changes are sound for the OSS community: *"We usually consult with the community whether they are interested in the changes. If they are nor interested, it makes no sense for us to contribute our changes back because they will not be incorporated"* (P).
- *The OSS component is discarded*: this strategy requires that the potential mistmatch be solved by replacing the OSS component with another component (either OSS, COTS, or bespoke). This strategy was usually adopted to avoid the drawbacks coming from extensive local modification of OSS code to adapt it to the system requirements: *"If we realize that the OSS component has several inconveniences [mismatches], we use to replace it"* (U); *"If there is an important requirement that we cannot fulfill with the OSS component, then we switch the component"* (C).

b) The System Requirement (*r*) is affected

In some situations, the respondent decided to adapt the requirement to fit the OSS component. The main motivation behind this situation was usually to ensure that the system followed up-to-date market requirements. The most common justification for this was that *"These OSS are as defacto standards"* (M); therefore, the respondents influenced the system requirement to fit the OSS component. One of the respondents commented: *"System requirements can be bent a bit. I do not see black and white situations"* (U). Two scenarios were identified to modify the requirements:

- *The system requirement is relaxed/modified*. In this strategy, the system requirement is slightly modified to fit the OSS component. We observed two cases. In the first case, there is (commonly) an external audit of the system requirement, which is usually done by the customer. The respondents recognized having influenced the customers to convince them of the suitability of relaxing/modifying the original system requirement: *"If the OSS component does not perfectly match the system requirements, we influenced our customer with things like 'This OSS component does not fully cover all your requirements, but it is like a standard and offers other things that might be of value for you'"* (M). In the second case, the requirements were flexible enough to be straightforwardly adapted to the OSS features by the respondents' team: "*[Our] requirements were quite vague, so we could easily accommodate them to the OSS"* (L).

- *The system requirement is postponed*. In this strategy, instead of modifying the system requirement, it postponed to fit the OSS component. This strategy was used mainly in situations where the system requirement was not critical and could be postponed for future versions of the project. All of the respondents that declared this situation (except one), mentioned reaching an agreement with the customer: *"We decided to use [the OSS component], so we postponed some requirements and agreed with the client to incorporate them in future versions"* (J); *"It is a soft process, we talk with the customer to postpone things, if the customer does not agree, we usually choose another component"* (U). The only case that did not mention a customer, just stated: *"Postponing system requirements is our last option to solve mismatches. We usually try to put forward all our system requirements and to find the way of doing it"* (Q).

c) Neither the OSS component nor the System requirement are affected

Few cases (three out of twenty-five) reported that the system requirements and OSS components were not affected because mismatches were solved by adding glue code or additional plugins to complete the functionality that was not covered by the OSS component, for instance, *"We studied other plugins to complete the functionality of the OSS component"* (N); *"We just implemented the missing functionality"* (O).

To our knowledge, these scenarios represent the first industrial evidence of approaches being used to solve RE mistmatches in the integration of OSS components in OBSD. It is important to note that given the fact that none of the respondents stated having participated in post-development stages such as maintenance, the scenarios presented above reflect the matching and resolution approaches that occur in the pre-release development stages (i.e., from requirements elicitation and analysis to the first release of the OBSD product). Therefore, mismatches that arise after the first release are still unexplored. Our general observation regarding the approaches followed in these scenarios is as follows:

Observation 6: Modifying OSS components (either locally, globally or replacing the component) was the most common way of solving functional requirement-component mismatches. Modifying system requirements (either relaxing or postponing system requirements) was done in fewer cases. Adding glue code was the least used approach.

The scenarios devised based on the answers of the respondents show that the most common way to solve potential requirement–OSS component conflicts was to affect the OSS component. In this case, the respondents could proceed to modify the OSS component either locally or globally or to replace the OSS component. The second most popular approach in our sample to solve conflicts was to affect the system requirements. In this case, the main approaches followed were: to relax/modify system requirements or even to postpone the requirements. Finally, in few cases, the conflicts were solved by adding glue code.

Our results also show that the case of not finding any mismatch between the system requirement and OSS component was rare as just one respondent stated this scenario.

The literature has suggested two types of requirement-component mismatches: (1) component mismatches, which are due to an excess or shortage of component features with respect to system requirements; and (2) architectural mismatches, which arise when integrating components that do not fit well together [60]. In all of the assessed projects, the requirement-component mismatches detailed in the scenarios raised by the participants where due to an excess or shortage of OSS component features with respect to system requirements. We believe that this is related to the previous observation that suggest that important architectural issues are considered as early as in the requirement-component matching process; even before these relevant architectural mismatches occur.

Furthermore, our data shows that, in several cases, functional system requirements were affected (either relaxed, modified, or postponed) to adapt them to OSS components.

Although some previous evidence related to OTS components usage stated that some component integration projects used custom code mainly to provide additional functionality [88] (see EV3 from Table 3), such previous evidence did not clarify if this apply to COTS and/or OSS and did not provide any context to understand this result. On the contrary, the set of scenarios that resulted in this study, try to extensively describe the context of each of the requirement–OSS component mismatch situations. Compared to other previous works [88], [51], [50] which say that integrators usually used OSS components without modifications (see EV1 and EV11 from Table 1), our results show quite a different setting. While system requirements still seem to be of higher priority than OSS components when solving conflicts; most of the projects carried out local and/or global modifications to the OSS components, and adding glue code was just mentioned by a few respondents in our study. This suggests that even if some respondents are still reluctant to modifiy OSS code, the ease of integration/modification of OSS components is valued positively by integrators. Therefore, integrators are more willing to modify OSS either locally or globally, but they also recognize some of the implications of this decision, as stated in the description of the corresponding scenarios.

In some extreme cases, OSS components are being used as a basis to define system requirements so that no mismatches are likely to occur between system requirements and OSS components in pre-release stages. We note that, in these cases, the integrators justified this decision based on the fact that the OSS component is recognized as a de facto standard that offers further opportunities and competitive advantages.

5.2.2 Role of OSS components on potential mismatches with non-functional requirements.

To inquiry about mismatches with respect to non-functional requirements, we proceeded differently as we did for inquiring about functional requirements. On the one hand, non-functional requirements are normally system characteristics that are often verified in the later software development stages, when the modules are integrated and tested. On the other hand, we were aware that non-functional requirements are often not well-described and poorly understood [11], [26], [37]. Hence mismatches between non-functional requirements and components are hard to investigate and assess. Consequently, instead of asking the respondents about the mismatches between non-functional requirements and components, we asked which and how non-functional requirements were fulfilled by using OSS components (See question 3.2 of the interview guide).

To better interpret the respondents' answers, we coded each non-functional requirement mentioned by the respondents and registered all comments related to it. Similarly to Ameller et al. [3], we experienced some problems in consolidating the answers because the respondents used different terms to define the same non-functional requirements (e.g., some respondents used the terms efficiency and performance synonymously). Therefore, we used the ISO/IEC 9126-1 quality standard [40] as the unifying framework.

In addition, after processing all answers we realized that respondents mentioned positive and negative aspects related to the role of OSS components on their non-functional requirements. Thus, we had themes with positive and/or negative aspects. This of course denotes the contextual nature of the non-functional requirements and OSS components used in the projects studied [7].

Table 12 shows an excerpt of positive and negative comments about non-functional requirements explicitly mentioned in the interviews. The frequencies shown in the table are just to provide insight into the popularity of certain aspects mentioned by the respondents.

Table 12 Influence of OSS components on non-functional system requirements

Aspect	**ISO 9126-1 Definition**	**Pos**	**Example of positive comment**	**Neg**	**Example of negative comment**
Stability	The capability of the software product to avoid unexpected effects from modifications of the software.	11	"It is quite stable. We are happy with that" (A)	1	"We experienced a problem with stability: after a while we could not use [the OSS component]. We had to change the code, otherwise we could not continue running our system." (T)
Time To market	Not explicitly approached by ISO 9126-1	11	"It definitely contributed to time to market as we used OSS components that keep us from developing complex functionalities ourselves" (D)	1	"It was actually the opposite. We experienced delays and over-costs because we abused the use of OSS" (L)
Cost	Not explicitly approached by ISO 9126-1	9	"The cost issue was very well achieved, since we were able to implement a lot of features without developing them ourselves or without having to pay license fees to commercial vendors" (D)	2	"We exceeded a bit the cost because of some migration problems)"(R)

Aspect	ISO 9126-1 Definition	Pos	Example of positive comment	Neg	Example of negative comment
Maintainability	The capability of the software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications.	7	"We could change things and it was very easy to understand, so we could easily create reliable software" (E)	1	"For us, maintenance was an issue as we did not have expertise in several of the OSS components used" (L)
Reliability	The capability of the software product to maintain a specified level of performance when used under specified conditions.	7	"We always use very reliable components (with solid communities behind), so we did not experienced reliability problems" (C)	2	"We experienced both reliable and unreliable components. Non-reliable components have a lot of bugs …" (B)
Efficiency	The capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions.	6	"We do not have any complaints with performance. The performance of these OSS was pretty good"(B)	4	"We experienced performance problems. However, this cannot be just attributed to the OSS component but to the amount of data, and the load and the hardware issues…" (F)
Security	The capability of the software product to protect information and data so that unauthorised persons or systems cannot read or modify them and authorised persons or systems are not denied access to them.	2	"Security was a very important aspect of the project, and OSS was not really an issue to worry regarding security" (J)	0	-
Usability	The capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions.	3	"The component had a great level of usability, as our client wanted" (P)	1	"User interface was bad, as it was a very small project" (C)
Learnability	The capability of the software product to enable the user to learn its application	1	"The OSS components used were very well documented and easy to learn" (A)	1	"The learning curve was so high and it ended up as a disaster project" (L)

During the interviews, we guided the respondents to assess whether the aspect in question was really related to the OSS components or to other project characteristics (e.g., "*Everything was fine. We have cost problems, but they are not related to OSS but to the scope of the project*" (M)). In a few cases, some negative aspects were justified as being related to the specific OSS components, but not totally caused by them (e.g., see the negative comment for efficiency). Our finding is summarized in the following observation.

Observation 7: OSS components appear to rarely have a negative effect on non-functional requirements

Our results show that most of the respondents consider the OSS influence on stability, time-to-market, cost, maintainability, reliability, efficiency, usability, security and learnability to be positive. Fig. 4 shows graphically the frequency of positive and negative comments about non-functional factors.

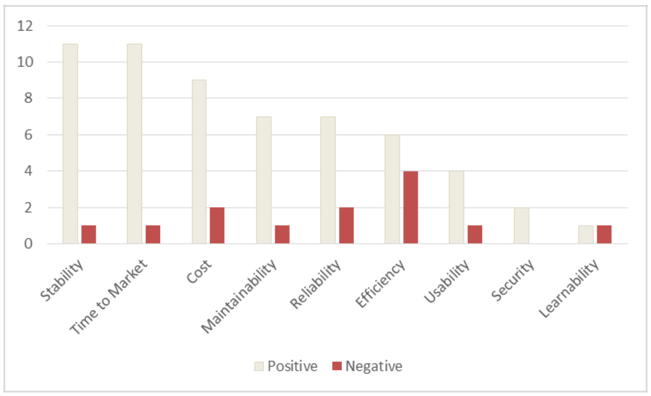

Fig. 4 Positive and negative influence of non-functional factors

Interestingly enough, we found evidence of two relevant aspects mentioned by the respondents that were not considered in the ISO/IEC 9126-1 quality standard: time-to-market and cost. These aspects had been highlighted by Carvallo et al. [17] as non-technical aspects that play an important role in COTS-based systems development. This result supports previous results from Bonaccorsi and Rossi [14] which found that firms are motivated to be involved with OSS for various reasons: it allows them to reduce time-to-market and cost, which lead them to innovate; "many eyes" assist them in software development; the quality and reliability of OSS; and the ideological fight for free software (at the bottom of the list).

It is worth mentioning that another study performed in 2009 [50] also found that OSS components appear to rarely have a negative effect on the non-functional requirements since most of the comments from the responses were also positive opinions (see EV10 from Table 1). However, the impact of this positive result might be influenced by the success and popularity of the OSS phenomenon [82] and the eventual threat of personal bias from the respondents (as discussed in the Threats to validity section). Therefore, we claim that further research is needed to investigate the potential impact of external factors on the possitive perceptions of OSS in OBSD.

5.2.3 Factors that influence requirement-component mismatch resolution processes

To understand the factors that influenced the way practitioners solved conflicts between their system requirements and the OSS components, we explicitly requested the participants to explain the factors that influenced the approach followed to solve conflicts. Our results are as follow:

Observation 8: System requirement coverage and ease of integration of OSS component are the main factors in determining which requirement-component mismatch resolution approach to use, but non-technical factors such as the value added by the OSS component to the organization's business are also relevant.

All of the respondents explicitly highlighted the coverage of the system requirements and the ease of integration of the OSS components as the primary factors to consider when assessing how to solve conflicts. However, other interesting business related aspects that influenced their decisions also emerged, for instance: *"First, the preliminary study of the components and their matching to our requirements. Second, we value the time and costs associated with integrating the OSS component to our project. I should say that we mostly use OSS components that are like de facto standards. In this way, it is easier to convince your client by saying that the component is a very well-known component and that there is no problem to find documentation..."* (M). *"We value the OSS components and our requirements more or less the same. But if we see the possibility of using OSS that can bring value to our products, we try to use them. For instance, the possibility of participating in the community and sharing the maintenance"* (T). *"The thing is that our customers are increasingly accepting or even requesting the use of OSS, so we are also influenced by that. It gives you a competitive advantage"* (H).

5.3 Influence of OSS on OBSD and RE activities and associated problems (RQ3)

In order to analyze and understand the influence of OSS on OBSD and RE practices and the problems originated by the integration of OSS components, we explicitly asked the respondents about it. Section 5.3.1 summarizes our results and observations regarding the influence of OSS components while Section 5.3.2 focuses on the problems.

5.3.1 Influence of OSS components on OBSD and RE practices

We first asked to the respondents about the influence of integrating OSS components in their software development practices (it corresponds to question 5.2 from the Interview guide); then we explored some potential relations among the gathered data with Weka [93] and got some relevant observations based on such potential relations. Our results are as follows:

Observation 9: Software development processes seem not to be very much influenced by the integration of OSS components

None of our respondents mentioned significant influences in their software development processes because of the integration of OSS components (except the activities related to requirement-component matching and mismatch resolution processes that were performed regardless of their usual software development processes). In several cases, the respondents mentioned deciding on the development process before they even started to think about using OSS components. Although the literature claims that companies must extensively adapt and modify their current software development processes in response to the use of third-party components [15], [95], our results seem to be in line with previous evidence related to OTS components that indicates that companies use traditional processes enriched with specific activities for requirement-component matching and mismatch resolution [50] (see EV7).

Furthermore, in line with a previous survey made in the RE literature about the method and the level of detail used to specify requirements in industry [25], most of our assessed projects used medium and coarsely-detailed requirements descriptions using natural language and free text. It suggest that there are not relevant differences in the way requirements are specified (regarding notation and level of detail) in our studied projects in comparison to common requirements specification industrial practices.

Thus, it seems that regardless of the nature of the OTS components (i.e, COTS or OSS) the integration of COTS or OSS does not significantly influence the whole software development process.

Observation 10: The experience of the software development team with OSS seems to marginally influence system requirements specification.

Using cluster analysis, we related the percentage of experience of the teams with OSS with the level of detail and the notation used to specify requirements (shown in Table 13 and Table 14 respectively). These tables show 3 clusters of percentage of experience of the teams with OSS (0-33%, 34-66% and 67-100% respectively) and the percentages of respondents in each cluster that mentioned the level of detail or notation stated in the corresponding table. The percentage 73.68% corresponding to the column Full Data in Table 13 and 14 represents the average experience with OSS of the 25 respondents.

Table 13 Cluster analysis of percentage of experience of the team with OSS vs. the level of detail of requirements

	Full Data	**Clusters Percentage of Experience With OSS**		
	73.68%	0%-33%	34%-66%	67%-100%
Number of respondents	(25)	(4)	(6)	(15)
LevelOfDetail				
very sketchy	0.16	0.25	0	0.20
coarsely	0.36	0.25	0.50	0.33
medium	0.28	0.50	0.17	0.27
detailed	0.12	0	0.17	0.13
very fine-grained	0.08	0	0.17	0.06

Table 14 Cluster analysis of percentage of experience of the team with OSS vs. the notation of requirements

	Full Data	**Clusters Percentage of Experience With OSS**		
	73.68%	0%-33%	34%-66%	67%-100%
Number of respondents	(25)	(4)	(6)	(15)
Number of valid answers	(35)	(6)	(8)	(20)
Notation				
Free text	0.4	0.33	0.38	0.45
Structured text	0.2	0.33	0.25	0.10
Use cases	0.14	0	0.13	0.20
Flow/state diagrams	0.09	0.17	0	0.10
Mockups	0.03	0.17	0	0
Test cases	0.14	0	0.25	0.15

In the case of Table 13, the available data referring to the level of detail used to specify requirements (see Table 7) shows that the participants were allowed to provide just a single answer, therefore, each cell in Table 13 represents the percentage of participants' responses to the cluster.

However, in the case of Table 14, the available data referring to the notations used to specify requirements (see Table 6) shows that the participants were allowed to provide more than one answer. Therefore, the computation of the percentages in Table 14 was calculated based on the number of valid responses provided by the participants instead of the number of participants belonging to the cluster. Thus, the cluster representing 0-33% of experience of the team with OSS considered 6 valid responses from the 4 participants belonging to that cluster. The cluster representing 34-66% of experience, considered 8 valid responses from the 6 participants belonging to this cluster. Please note that respondent T corresponding to this cluster did not answer the question and it was considered as a non-valid answer. Finally, the cluster representing 67-100% of experience, considered 20 valid responses from the 15 participants belonging to that cluster.

As a result, the number value in each cell represents the normalized average of the attribute in the cluster formed by the intersection of the attribute in the row and the attribute in the column.

As stated in Section 4.6.4, in line with previous surveys done in the RE literature, the majority of the respondents in this study used mainly sketchy, coarsely-grained and medium requirements descriptions with free text and structured text as notations.

The results from Table 13 and 14 might suggest that less experienced teams (i.e., those in the cluster 0-33% of experience with OSS) avoid detailed and very fine-grained requirements descriptions and more prescribed notations such as use cases and test cases. However, the few number of respondents in this cluster does not properly support this observation. Therefore, the influence that we could observe from the experience of the teams with OSS and the level of detail and/or notation to specify requirements was marginal. Further studies are needed to provide more substantiated observations in this respect.

Observation 11: Some OSS components seem to highly influence requirement-component matching and mismatch resolution processes because they: a) are considered de facto standards in some domains, b) are already mastered by the development team, or c) adhere to consolidated architecture patterns or frameworks from its domain.

Our qualitative data led us to uncover some aspects that have not been previously stated by any other study. When assessing the rationale behind selecting OSS components in diverse software development stages, we realized that using OSS components was already a requirement from the client or their internal business and/or technical teams (usually stated in the requirements elicitation and analysis stage). This also coincides with the observation that the mean of the percentage of the OSS proportion in projects that need external requirement approval sources is higher than in projects that need internal requirement approval sources. It suggests that projects that need external requirements approval sources tend to integrate OSS components more than projects that need internal requirement approval sources.

Furthermore, from the data gathered in the scenarios of mismatch resolution and additional comments from the respondents, we observed that 10 out of 25 projects were influenced by the OSS components to affect their system requirements to solve conflicts. In addition, one project did not report any mismatch because the system requirements were actually in line with the OSS component features, because the client requested such OSS component. All this together, provides evidence on the influence of OSS on the system requirements of the projects and gives insight into the increasing influence of OSS on the software industry.

Additional comments given by some respondents also support these observations. Representative quotes are: *"I have experienced from a few other projects I have participated in, where the customers defined their requirements based on the OSS, so their own OSS was used as a basis to define the system requirements"* (U). *"The number of clients that explicitly request to use OSS components is increasing, so this is something that, of course, affects our company as we know that we should build and consolidate our expertise in using OSS solutions to gain a competitive advantage"* (O).

Based on the comments of the respondents, we identified some OSS facts that seemed to influence the requirement-component matching and mismatch resolution processes in the assessed projects:

a) Some OSS are considered to be "de facto" standards in some domains, so integrators try to adapt to them.

b) OSS are already mastered by the development team, so integrators try to capitalize on their experience.

c) Some OSS application domains have consolidated architecture patterns or standardized development frameworks (such as web applications), and integrators try to use them to better confront integration issues.

Observation 12: Contributing to OSS communities to deal with requirement-component mismatches is gaining industrial importance.

From the uncovered scenarios of requirement-component mismatch resolution presented above, we observed that 19 out of 25 projects used to modify OSS components (either locally or globally) to solve potential conflicts. Regardless of the extent of the modification (locally or globally), it intrinsically entails a potential relationship with the OSS community, as the literature suggest that the decision to modify OSS components increase the likelihood to envisage a potential OSS community involvement [38]. Furthermore, some respondents commented that their organizations were explicitly moving towards other ways of adopting OSS that involve community involvement, namely, participating in OSS communities and providing OSS products, based on the classification provided by Hauge et al. [38]. Participating in OSS communities refers to the involvement of the organization in the software development tasks of existing OSS communities, although without having decisive control over the OSS products or the community. Providing OSS products refers to the development and release of software products under OSS licenses, in addition to pushing forward a collaborative community. The projects corresponding to organization P and Q explicitly mentioned moving towards participating in OSS communities and providing OSS products, respectively. Previous evidence from [20] (See EV19 from Table 1) found that holding a relationship with OSS communities was rare in the context of the Chinese software industry. However, our results suggest that relating to OSS communities is gaining industrial importance. Of course, the comparison of our result with the previous evidence EV19 from [20] is endangered by the different contexts approached and the years that have passed among the studies. EV19 refers to the Chinese software industry in 2007 and our study approached organizations from European countries. Therefore, further research is needed to investigate the reason behind the slight change in trend.

5.3.2 Problems that affect OBSD and RE activities

In order to inquire about problems related to the OSS components integration, we first asked about two aspects that seemed to cause relevant problems in OBSD as stated in the RE literature: the changes in OSS components and system requirements, and licensing issues (these aspects correspond to questions 4.1 and 4.2 from the Interview guide). Observations are detailed below.

Observation 13: Component changes made by the OSS community seldom affect OBSD during pre-release stages.

Since the literature highlighted that changes to OSS components made by the OSS community as well as the changes in system requirements during the software lifecycle are a source of relevant requirement-component mismatch problems [47], we asked the respondents to comment on them. Surprisingly, only two out of twenty five respondents declared having been affected by OSS community changes in the OSS components, and none of them mentioned relevant changes in their system requirements: *"It was between a pair of releases of [....], the OSS community changed something and so we made transformations to keep our system working...And that's a powerful incentive for us to make our changes available so that the community takes care of changes in the APIs and such things."* (T) *"I had to find the solution myself"*(X).

Since the involvement of the respondents in the assessed projects focused on the pre-release development stages, our results suggest that integrators have certain control over the changes in their system requirements in the development stages. Therefore, no mismatches were mentioned; however, changes in the OSS components by their corresponding communities are unavoidable, making it very important to make an informed decision when selecting OSS components in order to avoid potential conflicts.

We did not observe many requirement-component matching and mismatching problems due to changes in OSS components and/or system requirements as stated in the literature [47]. This could be because these potential problems are mostly caused by changes in post-release stages that were not explored in this study.

Observation 14: Licensing issues are rare in opportunistic reuse of OSS components. Avoiding the use of viral licenses was the most typical way to deal with licenses.

To gather evidence on the role of licenses, we specifically asked the respondents about their experience with licensing aspects. All of them agreed that they carefully reviewed the licences and respected them. "*We do actively choose OSS components and have not experienced any problems with the license. We respect the licenses and have control of the license we use"* (A). Most respondents also emphasized that they actually avoid using GPL licenses *"if a commercial company plans to use OSS components, it usually has to choose some kind* *of permissive license – of course not GPL"* (C). *"A lot of GPL components cannot be used because the license says that if you integrate it into your product you have to ship the entire source code of your customer. For us, it is quite simple, if the component is GPL, we just discard it… there are usually other options available."* (G)

With regard to the common perception that OSS licenses are a complex issue to deal with, one experienced respondent commented: *"What seems to complicate the license environment is that the major software vendors try to give a bad reputation to OSS, fostering the idea that using OSS might `violate´and cause viral infection to your software, so people that are not familiar with it are scared"* (G). Another stated factor that seems to contribute to the perception of a complex OSS licensing environment was the huge diversity of available licenses [70]. *"The problem is that many developers invent their own licenses. There are more than a couple of hundred licenses. It should be enough with GLP, LGPL, and BSD. It would be a lot easier to use OSS components if you didn't have to read the terms of the agreement every time"* (G).

Although the literature highlights the critical nature of the role of licenses in OBSD [4], [85], our results show that none of the respondents mentioned relevant problems related to licenses. Instead, they simplify their licensing tasks by mastering the kind of licenses they can use and just carefully avoid GPL licenses as a risk mitigation strategy. This result agrees with that from Chen et al. [20] since they also found that most of their respondents did not face relevant licensing problems.

Our qualitative data allowed us to observe the contexts of the projects and suggests a potential explanation for this limited involvement in licensing. Even though a couple of the projects stated having the goal of further community involvement (i.e., P and Q), none of the assessed projects at the time of the study had any further community engament or further business model involvement. Therefore, the reuse of OSS components in most of the projects that we assessed seemed to be opportunistic and based on previous experience. Thus, as a possible explanation for our results we suggest that the main concern of OSS integrators that do not have a former component reuse plan (i.e., those that perform pragmatic and opportunistic reuse) is to avoid the use of viral licenses. However, projects that strategically decide further OSS community involvement [38] would probably face other licensing concerns to avoid exposing their intellectual property rights while taking advantage of other community involvement benefits.

Observation 15: From the RE perspective, the most relevant problem in OBSD was related to OSS component information/documentation

We asked the respondents to state any relevant problems they faced in their RE activities in OBSD. Table 15 summarizes the categories of their answers.

Table 15 Problems related to RE practices in OBSD

Category	Respondents	Total
No relevant problems encountered	A, B, E, G, K, M, Q, R, T, U, V, W, Y	17
YES, we experienced problems getting OSS component relevant documentation	C, D, F, H, I, L, S, X	8

It was interesting to see that most of the respondents did not claim to have had relevant problems, while others explicitly mentioned problems related to documentation. Seventeen respondents agreed there were some issues, but they were actually not relevant enough to be considered a problem from the requirements perspective. A representative quote from these respondents is: *"No problems. Large and mature OSS components are usually good quality and well documented, either in documentation or in information available on the Internet. There may be issues with finding information on how to use components on less popular platforms (e.g. Solaris)"* (A). Eight respondents stated having experienced some problems. The evaluation of their answers led us to observe that all of these problems referred to the availability of technical documentation of the OSS components (since it was not available or updated in some cases). This problem hampered their understanding of the OSS component and made their learning curve more difficult. Some representative quotes from their answers are: *"Yes, documentation was usually scarce, especially with respect to internals. "* (C). *"The Python language and the Django framework have excellent documentation, but many of the smaller libraries we used did not have good documentation"* (I). *"Yes, the information about the component was probably not properly updated."* (S)

Thus, our results suggest that from the RE perspective, the most relevant problems faced by the assessed projects were related to OSS components documentation/information. This agrees with a challenge of using OSS stated by Stol and Ali Babar [85] and with a claim made by Rubython and Maiden [75]: *"Although methods and tools now exist, specifying requirements, understanding OTS components, and aligning requirements to package features remain difficult tasks [...]. The first [reason] is the [improper] information that suppliers often* *provide about their [OTS] components..."*

The respondents stated that potential problems depended on the OSS component used. Small components and/or less popular ones tended to lack of a proper documentation. More guidance about how to use this information needs to be provided.

6 Threats to Validity

Like any other empirical study in Software Engineering, our study also faces some validity threats that have been discussed in terms of construct, internal, and external validity, as suggested by Robson [73] and Wohlin et al. [94]. We also report the strategies used to deal with these potential validity threats to our study's findings.

6.1 Construct validity

Construct validity refers to the issues that affect our ability to reflect the constructs under study using approapriate instruments. To reinforce this aspect, we carried out rigorous planning of the study and to establish a rigorous protocol to be strictly followed by all of the researchers involved in this study, as suggested by Runeson and Höst [76].

A semi-structured interview guide was developed as the main data collection instrument. The guide was originally designed in English by the whole team and then translated to the native local languages of most of the respondents (except Denmark and Sweden). In the two cases where the guide was not translated, the respondents did not have any problem understanding the guide in English, and most of the interviews were held in the native local language of the respondents. We piloted the interview guide with three practitioners in each country by each national subteam in order to ensure its calibration, principally with respect to the use of suitable vocabulary that the respondents were familiar with. In addition, we detected some wording issues in some questions; therefore, we tried to improve these questions by rephrasing or alerting the interviewers that special attention should be given to them and that follow-up questions could be expected.

We also designed a template report and guidelines in English to unify and report the data gathered from the different subteams.

6.2 Internal validity

Internal validity refers to the issues that affect our ability to conclude causal effects. It is important to emphasize that our study was not aimed to infer causality but also to raise observations that should be further validated. In order to raise as consolidated observations as possible, we made relevant decisions.

Regarding the units of study: The units of study were OBSD projects selected by the respondents from the sampling organizations. We are aware that information collected from practitioners through interviews tends to be subjective perceptions that might be precise or vague [73]. To encourage the precision of their answers and to mitigate the vagueness of their opinions, we asked them to chose the project in advance and to fill out the details in the Part 1 of the interview guide (personal, company, and project information). It is true that the respondents may have chosen the most successful projects to base their answers on. To reduce this effect, we explained them that our study was not focused on analyzing "wrong practices" but on knowing "how it is done in industrial settings". Focusing each interview on a single project allow us to further inquire and analyze specific contexts. This strategy provided us with better qualitative information as it helped us to better focus, understand and discuss the rationale and context of each project.

The semi-structured nature of our guide allowed us to extensively explore the views and experiences of each of the respondents during the interview, and in some cases through follow-up questions. In all of the cases, we also had the opportunity to contact the respondents when we really needed clarification. We contacted them to seek clarification in only two specific cases. All this enhanced the value of our analysis and observations since it allowed for a better understanding of the rationale behind certain RE practices and decisions. We also included specific mitigation actions for evaluation apprehension by ensuring the confidentiality and aggregation of the answers, so the respondents could freely share their real perceptions.

Regarding potential confounding factors: We acknowledge that several factors that were not explicitly requested or approached in our interview guide could influence RE practices. For instance: the specific characteristics of OSS components used in the projects, organizational processes and policies, the influence of the clients and/or the software development team on the software development practices used, cultural issues related to the different countries of the studied projects, etc. These could clearly be potentially confounding factors since each of these situations has its own peculiarities. Thus, we tried to provide as much contextual information as possible when reporting the results of the study in order to understand the settings.

In addition, we first ran the study over a sampling of 15 projects from 15 different companies. After some discussions of the gathered data, we decided to perform another set of interviews to be able to balance the impact of some possible confounding factors, thus consolidating our observations. We did our best to mitigate the effects of some potential confounfing factors with the second round of interviews, so we invited a wider set of companies covering different sectors. The set of main confounding factors that we aimed to improve with the second round of interviews was:

a) Unbalanced set of projects regarding internal/external requirements sources. Most of the projects assessed in the first round of interviews used external requirement sources (10 out of 15 projects). We thought that this could affect our results as the source of requirements might influence the way requirements are managed and the role of OSS. Fortunately, our final set of sampling projects (after the second round of interviews) resulted quite balanced with respect to internal/external requirements sources.

b) OSS components covered a limited portion of the whole systems. In the first round of interviews, we realized that the mean of the percentage of OSS proportion used in the projects was 60%. So, we though that to better observe the influence of OSS in the projects, we could try to increase such percentage. In our second round of interviews, the resulting projects increased such percentage to 90%.

c) Limited set of the domains covered. Since the first round of interviews, we realized that our sampling projects were mainly dominated by the development of web system applications. Even if it was not intentional, our second round of interviews was also dominated by this domain. Given our limitations to get more companies to extend our sampling, we decided to contextualize our results in this domain.

We considered these improvements as acceptable mainly giving the difficulty in getting industrial participation and remarking that we did not have control over the projects chosen by the participating companies.

Regarding data analysis: We recorded and transcribed all of the interviews in different languages using predefined English report transcript guidelines in order to provide a better understanding and assessment of the data gathered by the entire research team. The analysis of the data was done in English. Since we are aware that translating the original answers from each respondent to English could increase the risks of biasing the answers with the researchers' interpretation, we paid special attention to ensure that all of the subteams properly followed the detailed procedures of the English report transcript guidelines. In addition, we made clear to the researchers in charge of initially identifying codes and themes, that any doubt regarding the translation of the respondents' answers should be clarified by contacting to the corresponding interviewer. However, we did not experience such situation.

Another fact that greatly reinforced the internal validity of our observations and conclusions was the use of the software tools (NVivo and Weka) for organizing the data and assessing potential relations from the extensive qualitative data collected respectively. NVivo helped us to organize the data collected and Weka allowed us to assess and visualize the data by means of clustering analysis in order to uncover meaningful relations that were not easily obtained manually. The identified clusters helped us to promote discussions and to make interesting observations based on the trends we saw within the groups. Even though in some fields there may be some restrictions on the use of statistical tools (such as cluster analysis) depending on the type and amount of data, Hand [36] concluded that these restrictions are more important in contexts that pertain to model fitting and hypotheses testing than in those that pertain to the generation of models or hypotheses. Based on Hand's observations, the use of statistical operations over small sets of data is legitimate in the initial search for potentially interesting relationships [36], such as our research study. Therefore, we consider that the application of cluster analysis to our data was useful in the assessment of our extensive qualitative data.

The findings of this study are neither causal nor inferential; rather the findings are descriptive of the industrial practices. We provide our raw data in [8], so the interested reader can corroborate our observations, further assess the data, or create new clusters at their convenience. A useful tutorial for using Weka is available at [93], which can be used to analyze our data.

6.3 External validity

It is important to highlight that qualitative studies such like ours rarely attempt to make generalizations. Instead, they are more concerned with characterizing, explaining, and understanding the phenomena in the studied contexts [73]. For a qualitative study the concepts of external validity changes, Lincoln and Guba [52] substituted reliability and validity with the parallel concept of "trustworthiness" consisting of four aspects: credibility, transferability, dependability, and confirmability; with credibility as an analog to internal validity, and transferability as an analog to external validity. Transferability refers to the degree to which the results of the qualitative research can be generalized or transferred to other contexts depending upon the degree of similarity between different contexts. That is why it is important that researchers provide sufficient descriptive data about the context of a study to make the assessments of contextual similarities possible. We have provided a detailed description of the context in which the data was collected. We assert that the contextual information for this study provided in this article is sufficient for researchers and practitioners to better understand and contextualize the results. Moreover, in Section 4.6 we have described the contexts of the studied projects and we have highlighted some of the relevant characteristics of most of the assessed projects that could have an impact when interpreting the results:

a) They were mostly web aplications and did not cover domains such as real-time or life-critical requirements.

b) They focus on pre-release stages (i.e., post-release stages such as maintenance were not represented in the projects studied).

c) Most OSS components were reused in an opportunistic way (i.e., without a previously established component reuse plan) as only in two cases, the respondents explicitly considered a long-term involvement with the OSS community to further reuse the component.

Consequently, our results could be more relevant for projects that are similar to the ones we studied. We emphasize that the results presented here should not be taken as assertions but rather as potential sources of forming hypotheses that need to be validated further and should be considered with caution in their corresponding contexts.

7. Conclusions and future work

This paper presents the results of an empirical study that examines RE aspects when integrating OSS components in practice. The data was collected through in-depth interviews with 25 respondents that had performed RE activities in software development projects that integrate OSS components in different software development companies in Spain, Norway, Sweden, and Denmark.

The answers given in our in-depth interviews have helped us to uncover 15 observations and a set of previously unexplored scenarios when solving system requirement–OSS component matching and mismatch resolution approaches in industrial OBSD projects (which focus mainly on pre-release stages of system applications that integrate OSS components in an opportunistic way). The observations presented in this study provide new evidence that has not yet been observed by researchers and help to confirm or further understand previous evidence.

Table 16 summarizes the research questions, their associated findings and the extent to which the findings represent new evidence, not previously reported in other studies.

Table 16 Summary of findings and their relation to previous evidence from Table 1

	Observation	**Complement and/or help to understand existing evidence**	**New evidence about OSS**
RQ1	1-Requirement-component matching processes are done at different stages of OBSD projects	√	
	2-OSS components that cover coarse-grained functionalities are usually selected in early requirements stages, while fine-grained OSS components are usually selected in late development stages		√
	3-Previous own's experience or colleagues' experience with the component mainly drive OSS component searching and evaluation.	√	
	4-No documented method was mentioned to be used for selecting OSS component but resources such as literature, seminars, workshops or hiring specialized companies to perform OSS selection were mentioned as valuable.	√	

	Observation	Complement and/or help to understand existing evidence	New evidence about OSS
	5-OSS components might influence architectural aspects of OBSD in the requirement-component matching process, but this potential influence greatly depends on the consolidation and popularity of the OSS component and its application domain.		√
RQ2	Set of 6 scenarios that describe requirement-component mismatch resolution approaches.		√
	6-Modifying OSS components (either locally, globally or replacing the component) was the most common way of solving functional requirement-component mismatches. Modifying system requirements (either relaxing or postponing system requirements) was done in fewer cases. Adding glue code was the least used approach.		√
	7-OSS components appear to rarely have a negative effect on non-functional requirements.		√
	8-System requirement coverage and ease of integration of OSS component are the main factors in determining which requirement-component mismatch resolution approach to use, but non-technical factors such as the value added by the OSS component to the organization's business are also relevant.		√
RQ3	9- Software development processes seem not to be very much influenced by the integration of OSS components.		√
	10-The experience of the software development teams with OSS seems to marginally influence requirements specification.		√
	11-Some OSS components seem to highly influence requirement-component matching and mismatch resolution processes because they: a) are considered de facto standards in some domains, b) are already mastered by the development team, or c) adhere to consolidated architecture patterns or frameworks from its domain.		√
	12-Contributing to OSS communities to deal with requirement-component mismatches is gaining industrial importance.		√
	13-Component changes made by the OSS community seldom affect OBSD during pre-release stages.		√
	14-Licensing issues are rare in opportunistic reuse of OSS components. Avoiding the use of viral licenses was the most typical way to deal with licenses.		√
	15-From the RE perspective, the most relevant problem in OBSD was related to OSS component information/documentation		√

Our results highligth issues that have received considerable attention from researchers but not from a particular sector of the software industry (such as the one we assessed), and vice versa. For instance, although licensing issues and the potential changes in OSS components by their corresponding communities and/or changes in system requirements have been greatly discussed in the RE literature and are considered relevant challenges [85], they did not appear to be relevant in the context of our assessed projects. Instead, the respondents highlighted the problem of getting suitable OSS component documentation/information. Some potential explanations for all the findings have been discussed in the paper. Also, we have compared our results with previous empirical work that provides evidence about OBSD (see Table 1 for a summary), our qualitative data helped us to confirm and understand some potential contradictions among them.

The provided evidence may provide a broader understanding of industrial RE practices when integrating OSS components and also have positive implications for research and practice.

• For SE researchers (especially those interested in OSS or RE), our results may encourage them to find solutions taking into account the factors that are actually used in industrial practice and to identify new research challenges and aspects that have been overlooked by the literature. We also advise researchers to try to focus their research on aspects of real industrial relevance and to consider actual industrial practices in order to promote the industrial adoption of their research proposals.

• For software-intensive organizations that integrate OSS components, the results presented in this paper may help to increase their awareness of the implication of some OSS components in their whole RE processes. This would enable them to consider diverse engagement strategies with the OSS communities [53].

• For OSS communities that act as component providers, our results may increase their awareness of how OSS components are selected and matched in OBSD and how the mismatch resolution processes and the factors that influence them are handled in industrial projects. This will help them to better address their product improvements and marketing strategies.

To conclude, we would like to emphasize that while our findings cannot be generalizable and should be further validated, they can contribute to the maturity of SE since they provide insights of problems that really affect the daily practice in industrial projects. We hope that our study might motivate researchers and practitioners to envisage more effective actions to improve the state of the practice of RE in OBSD, and thereby contribute to optimal management of the potential risks and rewards of using OSS components. Specifically, we hope to motivate researchers to perform more qualitative research that allows for a better understanding of real world practices and their evolution.

We view this study as an initial step in directing our future research in this area. We have focused this paper on detailing our methodological approach and describing our results. We aim to further studying the relations of our results with potentially existing theories that help us to identify the most valuable hypothesis that could be approached by further empirical studies. We are especially interested in to focus on a more specific domain in order to consolidate and improve our results.

Acknowledgments

We would like to thank all of the people who participated in piloting an early version of the interview guide and the interview participants who took time from their workdays to participate in our interviews. This work has been partially supported by the Spanish project ref. TIN2016-79269-R.

References

[1] T.A. Alspaugh, W. Scacchi: Ongoing Software Development without Classical Requirements. RE 2013: 165-174.

[2] C.F. Alves, A. Finkelstein: Investigating Conflicts in Cots Decision-Making. International Journal of Software Engineering and Knowledge Engineering 13(5): 473-493 (2003).

[3] D. Ameller, C.P. Ayala, J. Cabot, X. Franch: Non-functional Requirements in Architectural Decision Making. IEEE Software 30(2): 61-67 (2013).

[4] S. Aminat, A. Selamat, S.Sahibudin, Open Source Integration into Business Strategies: A Review. In Communications of the IBIMA, 2(17): 122-128 (2008).

[5] W.B. Anderson: COTS Selection and Adoption in a Small Business Environment: How Do You Downsize the Process? ICCBSS 2004: 216.

[6] L. Aversano, M. Tortorella: Quality Evaluation of FLOSS projects: Application to ERP systems. Information and Software Technology, 55, : 1260-1276 (2013).

[7] C.P. Ayala, Ø. Hauge, R. Conradi, X. Franch, J. Li: Selection of Third-Party Software in Off-The-Shelf-based Software Development - An interview Study with Industrial Practitioners. Journal of Systems and Software 84(4): 620-637 (2011).

[8] Raw data from this study is available at: http://www.essi.upc.edu/~cayala/DatosForWEKA2.csv.

[9] N. Ayewah, D. Hovemeyer, J. Morgenthaler, J. Penix, W. Pugh: Using Static Analysis to find Bugs. IEEE Software 25(5): 22-29 (2008).

[10] D. Badampudi, C. Wohlin, K. Petersen: Software Component Decision-Making: In-house, OSS, COTS or Outsourcing - A Systematic Literature Review. Journal of Systems and Software 121: 105-124 (2016).

[11] R. Berntsson-Svensson, T. Gorschek, B. Regnell, R. Torkar, A. Shahrokni, R. Feldt: Quality Requirements in Industrial Practice - An Extended Interview Study at Eleven Companies. IEEE Transactions on Software Engineering, 38(4): 923-935 (2012).

[12] B.W. Boehm: Requirements That Handle IKIWISI, COTS, and Rapid Change. Computer, 33(7): 99-102 (2000).

[13] B.W. Boehm, J. Bhuta: Balancing Opportunities and Risks in Component-Based Software Development. IEEE Software 25(6): 56-63 (2008).

[14] A. Bonaccorsi, C. Rossi: Comparing Motivations of Individual Programmers and Firms to take part in the Open Source Movement. Knowledge Technology Policy, 18(4): 40–64 (2006).

[15] L. Brownsword, T. Oberndorf, C.A. Sledge: Developing New Processes for COTS-Based Systems. IEEE Software, 17(4): 48-55 (2000).

[16] E. Capra, C. Francalanci, F. Merlo: An Empirical Study on the Relationship between Software Design Quality, Development Effort and Governance in Open Source Projects. IEEE Transactions on Software Engineering 34(6): 765-782 (2008).

[17] J.P. Carvallo, X. Franch, C. Quer: Managing Non-technical Requirements in COTS Components Selection. . RE 2006: 323-326.

[18] J.P. Carvallo, X. Franch, C. Quer: Determining Criteria for Selecting Software Components: Lessons Learned. IEEE Software, 24(3), 84-94 (2007).

[19] M. Ciokolwski, M. Soto: Towards a Comprehensive Approach for Assessing Open Source Projects. IWSM/Metrikon/Mensura 2008. : 316-330.

[20] W. Chen, J. Li, J. Ma, R. Conradi, J. Ji, C. Liu:: An Empirical Study on Software Development with Open Source Components in the Chinese Software Industry. Software Process: Improvement and Practice 13(1), 89-100 (2008).

[21] D. Cruz, T. Wieland, A. Ziegler:. Evaluation Criteria for Free/Open Source Software Products Based on Project Analysis. Software Process: Improvement and Practice, 11(2), 107-122 (2006)

[22] D.S. Cruzes, T. Dybå, P. Runeson, M. Höst: Case Studies Synthesis: a Thematic, Cross-case, and Narrative Synthesis Worked Example. Empirical Software Engineering Journal, 20(6): 1634–1665 (2015).

[23] D.S. Cruzes, T. Dybå: Research Synthesis in Software Engineering: A Tertiary Study. Information & Software Technology 53(5): 440-455 (2011).

[24] K. Crowston, K. Wei, J. Howison, A. Wiggins: Free/Libre Open-Source Software Development: What We Know and What We Do Not Know. ACM Computing Surveys, 44(2): 7 (2012).

[25] M. Daneva, D. Damian, A. marchetto, O. Pastor: Empirical Research Methodologies and Studies in Requirements Engineering: How Far Did We Come? Journal of Systems and Software 95: 1-9 (2014).

[26] M. Daneva, A. Herrmann, L. Buglione: Coping with Quality Requirements in Large, Contract-Based Projects. IEEE Software 32(6): 84-91 (2015).

[27] V. Del Bianco, L. Lavazza, S. Morasca, D. Taibi: Quality of Open Source Software: The QualiPSo Trustworthiness Model. OSS 2009: 199-212.

[28] H. Dagdeviren, R. Juric, T.A. Kassana: An Exploratory Study for Effective COTS and OSS Product Marketing. ITI 2005: 644–649.

[29] W.R. Dillon, M. Goldstein. Multivariate Analysis Methods and Applications. John Wiley & Sons (1984).

[30] T. Dybå, D.I.K. Sjøberg, D.S. Cruzes: What Works for Whom, Where, When, and Why?: On the Role of Context in Empirical Software Engineering. ESEM 2012: 19-28.

[31] T.Dybå: Contextualizing Empirical Evidence. IEEE Software 30(1): 81-83 (2013)

[32] H. Erdogmus: How important is evidence, really? IEEE Software. 27(3): 2-5 (2010)

[33] X. Franch:Do We Need Requirements in COTS-Based Software Development? ICCBSS 2004:11-12.

[34] M. Driver: Hype Cycle for Open-Source Software. Technical Report, Gartner (2013).

[35] B. Golden: Succeeding with Open Source, Addison-Wesley Professional (2004).

[36] D.J. Hand "Statistics and Theory of Measurements", J. Royal Statistical Soc.159 (3):445-492 (1996).

[37] S.Hansen and K.Lyytinen: Challenges in Contemporary Requirements Practice. HICSS 2010: 1-11.

[38] Ø. Hauge, C. P. Ayala, R. Conradi: Adoption of Open Source Software in Software-Intensive Organizations - A Systematic Literature Review. Information & Software Technology 52(11): 1133-1154 (2010).

[39] R. Hoving, G. Slot, S. Jansen: Python: Characteristics Identification of a Free Open Source Software Ecosystem. DEST 2013:13-18.

[40] International Organization for Standarization. ISO Standard 9126: Software Engineering – Product Quality, part 1. International Organization for Standarization, 2001.

[41] S. Jansen, S. Brinkkemper, I, Hunink, C. Demir: Pragmatic and Opportunistic Reuse in Two Innovative start-up Companies. IEEE Software, Special Issue on Opportunistic Software Systems Development. 25(6), 42-49 (2008).

[42] M. Jarke, P. Loucopoulos, K. Lyytinen, J. Mylopoulos, W. N. Robinson: The Brave New World of Design Requirements. Information. Systems 36(7): 992-1008 (2011)

[43] A. S. Jadhav, R. M. Sonar: Evaluating and Selecting Software Packages: A Review. Information and Software Technology 51(3):555-563 (2009)

[44] R.W. Jensen: Lessons Learned From Another Failed Software Contract. CrossTalk, The Journal of Defense Software Engineering, September (2003).

[45] L. Karlsson, Å. G. Dahlstedt, B. Regnell, J. N. O. Dag, A. Persson: Requirements Engineering Challenges in Market-Driven Software Development - An Interview Study with Practitioners. Information & Software Technology 49(6): 588-604 (2007).

[46] B. A. Kitchenham, T. Dyba, M. Jorgensen: Evidence-Based Software Engineering.ICSE 2004:273-281.

[47] R.J. Kohl: Requirements Engineering Changes for COTS-Intensive Systems. IEEE Software 22(4): 63-64 (2005).

[48] D.S. Kusumo, M. Staples, L. Zhu, H. Zhang, and R. Jeffery: Risks of Off-The-Shelf-based Software Acquisition and Development: A Systematic Mapping Study and A Survey. EASE 2012: 233-242.

[49] R. Land, D. Sundmark, F. Lüders, I. Krasteva, A. Causevic: Reuse with Software Components - A Survey of Industrial State of Practice. ICSR 2009:150-159.

[50] J. Li, R. Conradi, C. Bunse, M. Torchiano, O.P.N. Slyngstad, M. Morisio: Development with Off-The-Shelf Components: 10 Facts. IEEE Software. 26(2), 80–87 (2009).

[51] J. Li, R. Conradi, O.P.N. Slyngstad, M. Torchiano, M. Morisio, C. Bunse:A State-of-the-Practice Survey of Risk Management in Development with Off-the-Shelf Software Components. IEEE Transactions on Software Engineering, 34(2), 271-286 (2008).

[52] Y. S. Lincoln and E. G. Guba. Naturalistic inquiry. Beverly Hills, CA: SAGE Publications, Inc.(1985).

[53] L. López, D. Costal, C. Ayala, X. Franch, M. Annosi, R. Glott, C. Haaland: Adoption of OSS components: a Goal-Oriented Approach. In Data & Knowledge Engineering 99:17-38 (2015).

[54] S. Mahmood, R. Lai, Y.S. Kim: Survey of Component-Based Software Development. IET Software 1 (2):57-66 (2007).

[55] N. A. M. Maiden, C.Ncube: Acquiring COTS Software Selection Requirements. IEEE Software 15(2): 46-56 (1998).

[56] M. Majchrowski, J.C. Deprez: An Operational Approach for Selecting Open Source Components in a Software Development Project. EuroSPI 2008:176-188.

[57] D. Méndez-Fernández, S. Wagner, K. Lochmann, A. Baumann, H. de Carne: Field Study on Requirements Engineering: Investigation of Artefacts, Project Parameters, and Execution Strategies. Information & Software Technology 54(2): 162-178 (2012).

[58] J. Merilinna, M. Matinlassi:State of the Art and Practice of Open-Source Component Integration. EUROMICRO 2006: 170-177.

[59] A. Mohamed, G. Ruhe, A. Eberlein:COTS Selection: Past, Present and Future. ECBS 2007:103-114.

[60] A. Mohamed, G. Ruhe, A. Eberlein: Mismatch Handling for COTS Selection: a Case Study. Journal of Software Maintenance 23(3): 145-178 (2011).

[61] A. Mockus, R.T. Fielding, and J.D. Herbsleb: Two Case Studies of Open Source Software Development: Apache and Mozilla. ACM Transaction on Software Enginieering and Methodology 11(3): 309-346 (2002).

[62] S. Morad, T. Kuflik: Conventional and Open Source Software reuse at Orbotech – an Industrial Experience. SWSTE 2005:110-117.

[63] M. Morisio, (Ed.):Reuse of Off-The-Shelf Components. ICSR2006)

[64] C.Ncube., P. Oberndorf, A.W. Kark: Opportunistic Software Systems Development: Making Systems from What's Available. IEEE Software, 25(6), 38-41 (2008).

[65] A.D. Nguyen , D.S. Cruzes, R.Conradi, M. Höst, X. Franch, C. P. Ayala: Collaborative Resolution of Requirements Mismatches When Adopting Open Source Components. REFSQ 2012: 77-93.

[66] J. Noll: Requirements Acquisition in Open Source Development: Firefox 2.0. OSS 2008: 69-79.

[67] J. Noll, S. Beecham, and D. Seichter: A Qualitative Study of Open Source Software Development: The Open EMR Project. ESEM 2011: 30-39.

[68] Nvivo tool, further information: http://www.qsrinternational.com/product

[69] OpenBRR. Business Readiness Rating for Open Source a Proposed Open Standard to Facilitate Assessment and Adoption of Open Source Software. http://www.openbrr.org/wiki/images/d/da/BRR_whitepaper _2005RFC1.pdf, Request for Comments (2005).

[70] Open Source Initiative: https://opensource.org

[71] B. Paech, B. Reuschenbach: Open Source Requirements Engineering. RE 2006: 252-259.

[72] V. Perrone: A Wish List for Requirements Engineering for COTS-Based Information Systems. ICCBSS 2004: 146-158.

[73] C. Robson:Real World Research: A Resource for Social Scientists and Practitioner-researchers. Second Edition. Blackwell Publishers Inc. (2002).

[74] D. T. Ross and K. E. Schoman Jr.: Structured Analysis for Requirements Definition. IEEE Transaction on Software Engineering 3(1): 6-15(1977).

[75] A. Rubython,N. Maiden:The Effect of Variability Modeling on Requirements Satisfaction for the Configuration and Implementation of Off-The-Shelf Software Packages. RE 2014:394-401.

[76] P. Runeson, M. Höst: Guidelines for Conducting and Reporting Case Study Research in Software Engineering. Empirical Software Engineering 14(2): 131-164 (2009).

[77] R. Semeteys, O. Pilot, L. Baudrillard, G. Le Bouder, W. Pinkhardt:Method for Qualification and Selection of Open Source software (QSOS) version 1.6, Technical report, Atos Origin (2006).

[78] R. Sen, S. S. Singh, S. Borle: Open Source Software Success: Measures and Analysis. Decision Support Systems 52(2): 364-372 (2012).

[79] R. Sen, C. Subramaniam, M. L. Nelson: Open Source Software Licenses: Strong-Copyleft, Non-Copyleft, or Somewhere in Between? Decision Support Systems 52(1): 199-206 (2011).

[80] W. Scacchi: Understanding the Requirements for Developing Open Source Software Systems. IEE Proceedings - Software 149(1): 24-39 (2002).

[81] W. Scacchi, J. Feller, B.Fitzgerald, S. A. Hissam, K. Lakhani: Understanding Free/Open Source Software Development Processes. Software Process: Improvement and Practice 11(2): 95-105 (2006).

[82] R. Schuwer, M. van Genuchten, L. Hatton, On the Impact of Being Open. IEEE Software, 32 (5):81-83 (2015).

[83] K.J. Stewart, A.P. Ammeter, L.M. Maruping: Impact of License Choice and Organizational Sponsorship on Success in Open Source Software Development Projects. Information System Research 17 (2):126-144 (2006).

[84] D. Spinellis, V. Giannikas: Organizational Adoption of Open Source Software. Journal of Systems and Software 85(3): 666-682 (2012).

[85] K. Stol, M. Ali Babar: Challenges in Using Open Source Software in Product Development: a Review of the Literature. Workshop on Emerging Trends in FLOSS Research and Development 2010:17–22.

[86] K.-J. Stol, B. Fitzgerald: Inner Source- Adopting Open Source Development Practices in Organizations: A Tutorial. IEEE Software 32(4): 60-67 (2015).

[87] Software Engineering Body of Knowledge international standard ISO/IEC TR 19759:2005.

[88] M. Torchiano, M. Morisio: Overlooked Aspects of COTS-Based Development. IEEE Software, 21(2), 88-93 (2004).

[89] T. Vale, I.Crnkovic, E.Santana de Almeida, P. A. da Mota Silveira Neto, Y. Cerqueira Cavalcanti, S. Romero de Lemos Meira: Twenty-eight years of Component-Based Software Engineering. Journal of Systems and Software 111: 128-148 (2016).

[90] K. Ven, J. Verelst, H. Mannaert: Should You Adopt Open Source Software? IEEE Software 25(3): 54-59 (2008).

[91] J. H. Wesselius: The Bazaar Inside the Cathedral: Business Models for Internal Markets. IEEE Software 25(3): 60-66 (2008).

[92] Data Mining Software in Java, ver 3. Available at: http://www.cs.waikato.ac.nz/ml/weka/

[93] IBM developerWorks tutorial: http://www.ibm.com/developerworks/library/os-weka2/

[94] C. Wohlin, P. Runeson, M. Host, M.C. Ohlsson, B. Regnell, A. Wesslen: Experimentation in Software Engineering. Springer (2012).

[95] Y. Yang, J. Bhuta, B. W. Boehm, D. N. Port: Value-Based Processes for COTS-Based Applications. IEEE Software 22(4): 54-62 (2005).
[96] R. K. Yin: Case Study Research. Design and Methods. Second edition. Thousand Oaks: Sage (1994).

Claudia Ayala is a Researcher and Lecturer at the Universitat Politècnica de Catalunya (UPC). She received her PhD degree in Informatics from UPC. She was a postdoctoral ERCIM research fellow at the Norwegian University of Science and Technology (NTNU), Norway. Her current research interests include empirical software engineering, open source software engineering, requirements engineering and software quality. She serves as reviewer of several journals and is a member of several program committees in the area.

Anh Nguyen-Duc is a post-doctoral researcher at the Norwegian University of Science and Technology (NTNU). He received his PhD from NTNU. He is currently a postdoctoral fellow at NTNU.

Xavier Franch is Full Professor at the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain. He received his BSc and PhD in Informatics from the UPC. His main research interests include requirements engineering, open source software engineering, software evolution and dynamic adaptation. He currently belongs to the Editorial Boards of journals as IST, JSS, REJ, IET Software and IJCIS and has been Program Co-chair of international conferences as RE, CAiSE, ICSOC and REFSQ.

Martin Höst is a Professor in Software Engineering at Lund University, Sweden. He received an M.Sc. degree from Lund University in 1992 and a Ph.D. degree in Software Engineering from the same university in 1999. His main research interests include software process improvement, software quality, risk analysis, and empirical software engineering. The research is mainly conducted through empirical methods such as case studies, controlled experiments, and surveys. He has published more than 70 articles in international journals and proceedings from conferences and workshops.

Reidar Conradi received the MS and PhD degrees from the Norwegian University of Science and Technology (NTNU), Trondheim, in 1970 and 1976, respectively. He has been at NTNU since 1975. He is now an Emeritus Professor of the Department of Computer and Information Science (IDI), NTNU. He was a visiting scientist at the Fraunhofer Center for Experimental Software Engineering in Maryland and at the Politecnico di Milano in 1999 and 2000. His current research interests lie in software quality, software process improvement, version models, software evolution, open source software, and associated empirical studies.

Daniela Cruzes is a researcher at SINTEF ICT since 2013. Previously, Dr. Cruzes worked as adjunct associate professor at the Norwegian University of Science and Technology (NTNU). She has also worked as a post doctor researcher at NTNU and researcher fellow at the University of Maryland and Fraunhofer Center for Experimental Software Engineering-Maryland. She received the Dr. Ing. degree in Computer and Electrical Engineering from the University of Campinas – UNICAMP in Brazil in 2007. Her research interests are empirical software engineering, research methods and theory development, systematic reviews, software maintenance, global software development and software security. She serves as reviewer of several journals and is a member of several program committees in the area.

Muhammad Ali Babar is Professor of Software Engineering with the School of Computer Science of the University of Adelaide, Australia. He also holds an academic position with the Software and Systems Section (SSS) at the IT University of Copenhagen, Denmark. He obtained a Ph.D. in Computer Science and Engineering from the school of computer science and engineering of Univeristy of New South Wales. His main research interest includes areas of software architecture evaluation, architectural knowledge management, and process improvement using empirical methods.

APPENDIX 1

SEMI-STRUCTURED INTERVIEW-GUIDE

Part 1.A: Background Questions on Company and Person *(Completed by the respondent before the meeting)*

1.1. Current date (dd.mm.yyyy):
1.2. Your company/local business unit:
1.3. Company URL (only used internally):
1.4. Number of company employees, part-time and full-time:

Personal information:
1.5. First name and last name:
1.6. Email address (only used internally):
1.7. Phone number (only used internally):
1.8. Age (only used internally):
1.9. Degree of highest completed education, and in what area:
1.10. Current job position
1.11. Describe your experience with software development and with OSS (tasks, products, duration etc.)

Part 1.B: Background Questions on Project and System *(Completed by the respondent before the meeting)*

The **study object** for this survey is a **software development project,** with at least **one release** of the corresponding **product,** and with reuse of **one or more OSS components**. If you have experience with several such projects, please **select** the one that you are **most familiar with and motivated to discuss.**

1.12. What was the mean annual staff-size of the project (both full- and part-time employees)?
1.13. What part of the staff had previous experience with OSS-based development?
1.14. Did you have previous experience with OSS-based development before joining the project?
1.15. What was the total effort of the project?
1.16. What was (roughly) the starting time of the project?
1.17. Which part of the system do the OSS Components occupy?
1.18. What was the time of the first complete delivery of the project?
1.19. What were the major application domain(s) of the system?
1.20. Please provide a brief description of the chosen project and its main functionalities and architectural environment.
1.21. What was the overall, software development process/environment of the project?
1.22. Where did the requirements come from? [Interviewer should provide the following alternatives: there is an external client who paid for the project and "supervise" the requirements / there is no a specific client that paid for the project but it is the own company who "supervise" the requirements]
1.23. How were the functional requirements described with regard to level of detail? [Interviewer should provide the following alternatives: very sketchy/ coarsely-grained/ medium/ detailed/ very fine-grained/]
1.24. Which notation was used to describe the requirements?

Part 2. Details of requirement-component matchin process

2.1. In which lifecycle phases were these OSS Components searched, evaluated and decided?
2.2. How was the search process and the evaluation for these OSS Components done?
[Interviewer should provide the following alternatives to promote discussion]
   a) No real search/evaluation needed, since I have successfully used this component before.
   b) Consulted with my job colleagues/engineers, or with similar people elsewhere.
   c) Consulted with the customer about company policies or preferences.
   d) Read/heard about it in"peer-review" literature: scientific conferences, journals, books etc.
   e) Read/heard about it in"grey" literature: reports, ads, bulletin boards etc.
   f) Heard about it in connection with trade fairs, seminars, workshops, courses etc.
   g) Searched for it in general portals (http://Sourceforge.net) or in domain-specific ones.
   h) Searched for it using general or specialized search engines: google, google code etc.
   i) Used a formal method named ________to search for and assess candidate components.
   j) Used a systematic/documented method named:___________________.
   k) Some other way (please explain):

2.3. What was decided first – application platform and architecture vs. major components

Part 3: Details of processes to solve potential requirement-component mismatches

3.1. What did you do when the functional requirement-component mismatches ocurred?
[Interviewer should inquiry about the following:
OSS component is affected/ System requirement is affected / Any other approach
and let the respondent provide details and examples of the most usual approach followed in each of these cases]

3.2. Which and how the major non-functional requirements were achieved by using OSS components?

Part 4: Problems Experienced

4.1. Did changes in OSS Components by the community or changes in your Requirements create any problems?
4.2. Did you experience problems with licensing issues?
4.3. Did you experience other problems to understand and integrate major OSS components?

Part 5: General summary and reflections

5.1. Which factors influenced your prioritization of Requirements vs. OSS Components?
5.2. What is the influence of integrating OSS components in your software development practices?
5.3. Other comments?

APPENDIX 2

ENGLISH REPORT TRANSCRIPT GUIDELINES

Here are the format requirements for the transcript of the audio interviews. Please read these very carefully. Although they are detailed, they represent the information required to make your transcript fully useable for other researchers. This information is vital for the long-term effectiveness of our survey. Begin work on the transcript early. It is an essential and time-consuming task that must be completed as soon as possible after each interview.

1) The transcripts will be typed directly on each interview form in the language that it was conducted in the interview. In the analysis phase, we will determine how to conduct the analysis with the different languages.
2) Include the following information on the first page of the word document containing the interview:

Process:	Record details of the process followed in the interview (e.g. we completed first the Part 1 and 2 and we then interviewed him by phone).
Interview Time:	X minutes
Interview Date:	dd/mm/year
Interviewee:	XXXX
Interviewers:	XXX and XXXX
Media:	Phone/Face to face/Skype
Language:	English
Comments:	Please provide as much detail as possible

3) Use the blue color to highlight the transcripts. Please see the example transcription file in the companion material.
4) Try to put the transcription related to each question in the appropriate place, if possible, so that it will be easier to perform the analysis. If there are conversations that do not follow the question order, just leave the block of transcription, in the part where you are doing the interview. We will try to link to the question later on the analysis.
5) The speakers should be identified. On the first page, respondents and interviewers should be identified in parentheses following their name. Indicate what abbreviations will be used to identify the various speakers on the transcript. For example, you can use initials for speakers present, e.g., “MD” for Marion Doe.
6) Indicate any non-verbal responses in parentheses such as (narrator weeping), (laughter), (narrator very agitated) and so forth. However, do not reproduce irrelevant sounds such as “Ahhh, let me see…” Do not, correct the interviewee’s grammar or syntax. Faithfully transcribe slang expressions, exclamations (“Gosh!”) and fragmentary sentences. Do not use quotation marks unless the speaker is quoting someone else or reading from a document.
7) The explanatory remarks you add for clarity should be in [square brackets], e.g., [inaudible in 00:19:15]. Then, we will know that there were some inaudible parts at that specific time in the audio.
8) Be sure to proofread your transcript from the tapes once it is completed to ensure accuracy.
9) If at all possible, give the respondent an opportunity to review the transcript and make corrections.