The Journal of Systems and Software 207 (2024) 111877

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Eclipse Open SmartCLIDE: An end-to-end framework for facilitating service reuse in cloud development

Nikolaos Nikolaidis [a], Elvira-Maria Arvanitou [a], Christina Volioti [a], Theodore Maikantis [a], Apostolos Ampatzoglou [a,*], Daniel Feitosa [b], Alexander Chatzigeorgiou [a], Phillipe Krief [c]

[a] Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece
[b] Department of Computer Science, University of Groningen, Groningen, the Netherlands
[c] Eclipse Research Labs, Eclipse Foundation, Germany

ARTICLE INFO

Keywords:
Reuse
Service-based development
Cloud development
Platform

ABSTRACT

Service-Oriented Architectures (SOA) have become a standard for developing software applications, including but not limited to cloud-based ones and enterprise systems. When using SOA, software engineers organize the desired functionality into self-contained and independent services that are invoked through end-points (with API calls). The use of this emerging technology has changed drastically the way that software reuse is performed, in the sense that a "*service*" is a "*code chunk*" that is reusable (preferably in a black-box manner), but in many (especially "*in-house*") cases, white-box reuse is also meaningful. To confront the reuse challenges opened-up by the rise of SOA, in the SmartCLIDE project[1] we have developed a framework (a methodology and a platform) to aid software engineers in systematic and more efficient (in terms of time, quality, defects, and process) reuse of services, when developing SOA-based cloud applications. In this work, we: (a) present the SmartCLIDE methodology and the Eclipse Open SmartCLIDE platform; and (b) evaluate the usefulness of the framework, in terms of relevance, usability, and obtained benefits. The results of the study have confirmed the relevance and rigor of the framework, unveiled some limitations, and pointed to interesting future work directions, but also provided some actionable implications for researchers and practitioners.

1. Introduction

With the advent of open-source software (OSS) and the continuous adoption of open practices, software reuse has become widely popular, due to the enormous amount of freely and openly available software assets (e.g., code, components, or services) (Wang et al., 2008). Reuse of software assets is the process of using already available solutions to construct new software or enhance an existing one with new functionalities; thus, forgoing (or at least trying to minimize) the from-scratch development process (Krueger, 1992). Reuse is expected to bring important benefits to software development, especially with respect to time to market and quality (i.e., fewer bugs, improvement of product and development process) (Baldassarre et al., 2005). In literature, there are two mainstream processes to reuse: systematic reuse, e.g., through product lines, model-driven engineering, etc. (Brinkkemper et al., 2008); and opportunistic reuse, e.g., by searching development forums like StackOverflow for pieces of code (Digkas et al., 2019), or OSS repositories for classes, libraries, or products (Capiluppi et al., 2011). Focusing on OSS, as a "*code reuser*", a developer must perform two major tasks:

- ***Identify Reusable Assets***. In this step, the reuser must identify a piece of source code (e.g., method, service, class, set of classes, complete project, etc.) that implements the functionality that s/he wants to reuse. This task is a very difficult one, in the sense that: (a) the available number of reusable assets is vast and usually not well documented, and (b) there is a lack of platforms, acting as engines for accessing OSS repositories—especially in the domain of service-based development.
- ***Adapt the Assets to the Target System***. In this step (after the reusable asset has been identified—and if modification is required), the reuser must adapt the source code of the asset as extracted from the source system (in case of white-box reuse) to fit the architecture of the target system. Such an adaptation requires that the asset is well-structured

* Corresponding author.
E-mail address: a.ampatzoglou@uom.edu.gr (A. Ampatzoglou).

[1] An EU-funded Research and Innovation project with a duration of 40 months, supported by a consortium of 11 research, innovation, and industrial partners.

https://doi.org/10.1016/j.jss.2023.111877
Received 4 March 2023; Received in revised form 8 September 2023; Accepted 11 October 2023
Available online 12 October 2023
0164-1212/© 2023 Elsevier Inc. All rights reserved.

and maintainable. The assessment of the maintainability of a service is a non-trivial task since the current state-of-practice lacks dedicated methods and tools.

By considering the rise of the '*everything as a service (XaaS)*' model, as well as the current advancements in cloud computing, software development rapidly moves towards developing and deploying all software assets as services (Turner et al., 2003). This emerging change has raised several challenges in software development (Xu et al., 2018; Yang, 2012) and yields for specialized solutions. For example, according to Lewis et al. (2010) the research agenda on service-based software development (published by the SEI) prompts for the introduction of development methodologies that foster service reuse to reduce development costs (***need-1***). Currently in the literature there are various methods for service reuse (see Section 2.2), but to the best of our knowledge there is no end-to-end framework for facilitating service reuse in practice, helping developer through rigor and industrially relevant solutions (***need-2***). Additionally, by considering that the general field of service-based development has produced some very well-established solutions, such as Docker for deployment, Jenkins for CI/CD, and Kubernetes for Orchestration, any provided solution shall not re-invent the wheel, but try to integrate all these proven tools. However, an additional aim of such endeavors must be to hide the complexity and the need for configuring isolated tools, to reduce the levels of required knowledge and development time (***need-3***).

In this paper, we introduce the SmartCLIDE framework that promotes systematic reuse, while developing service-based software solutions for the cloud. The framework comprises of the SmartCLIDE development methodology and the Eclipse Open SmartCLIDE platform (i.e., an Eclipse Research Labs open-source project, available under a EPL 2.0 license) that is developed for enabling the adoption of the methodology in practice. The proposed methodology can be "*sketched*" as follows:

1 [SERVICE COMPOSITION] the software engineer, first specifies a set of services (***candidates for reuse***) that will be composed to build the final system;
2 [SERVICE DISCOVERY] then, attempts to discover existing services that implement the desired functionality, from in-house and OSS sources (***identification of reusable assets***) to be reused;
3 [SERVICE CREATION] if not reused, the software engineer develops services of optimal reusability and adaptability (***develop for reuse***) and places them into a reusable assets' repository.

The SmartCLIDE methodology contributes towards alleviating ***need-1***. While instantiating the methodology, we have selected various existing technologies to be integrated into the Eclipse Open SmartCLIDE platform (related to ***need-2***) that provides a unified interface that hides the complexity of managing various independent tools (related to ***need-3***). SmartCLIDE[2] has been implemented as part of an EU-funded research project, whose main outcome is the proposed framework (methodology and platform). We note that the way of instantiating the individual steps of the SmartCLIDE methodology, have been already published during the project—validating their research rigor. Therefore, we do not repeat them in this work to focus exclusively on novel and unpublished material. Thus, in this work, we empirically validate the SmartCLIDE framework in terms of: (a) industrial relevance; (b) usability, and (c) benefits that it can bring to SOA-based cloud development companies.

The rest of the paper is organized as follows: In Section 2, we present related work on service-oriented software development (Section 2.1), and service reuse (Section 2.2). In Section 3, we introduce the SmartCLIDE methodology, starting from the requirements for its construction (Section 3.1) and the details of its instantiation (Section 3.2). In Section 4, we present the case study design, whose results we report and discuss in Section 5. In Section 6, we present the limitations / threats to validity, whereas in Section 7, we conclude the paper.

[2] https://smartclide.eu/

2. Related work

2.1. Service-oriented software development

Service-Oriented Software Development was proposed in the 1990s as a new design paradigm to decouple service-side applications and improve the reuse of components (Singh and Peddoju, 2017). Microservices inherited the principles and concepts of the service-oriented architecture (SOA) style and structures a multiple service-based application into a set of loosely coupled software services (De Lauretis, 2019). Moreover, microservices not only allow for the decomposition of software components but also allow the organization of software development companies into small, autonomous teams that can implement various programming languages and technologies while working independently. So, in a way microservices can be considered a subtype of SOA (De Lauretis, 2019). There has been a lot of work on SOA and microservices regarding their benefits and weaknesses, as well as the comparison and migration from one to the other. The SOA represents a new way of developing systems, which promotes a shift from writing software to assembling and integrating services (Mahmood, 2007). Some of the main benefits are: (a) Loosely coupled applications and location transparency; (b) Enhanced reuse of existing assets and applications; (c) Process-centric architecture; (d) Parallel and independent development; and (e) Better scalability (Mahmood, 2007; Johnson, 2004; Djogic et al., 2018). All these benefits are also valid for microservices; Raj et al. (2022) extended the list with business benefits: (a) Focus on business requirements; (b) Quick evolution; (c) Organizational alignment; (d) Reduced costs; (e) Reduced Time-to-market.

To siege these benefits, the organization must be willing to face some costs, but with the right project or development team, it can be worth the selection. The challenges are pointed out by a plethora of papers (Mahmood, 2007; Raj et al., 2022; Djogic et al., 2018; Liu et al., 2020) and some of them can be performance issues, debugging, and data consistency. Dai et al. (2020) pointed out that each service needs to use the communication interface between its services to perform transaction operations. Therefore, network communication is a negative factor affecting the performance of microservices. Moreover, (Zhou et al., 2018) pointed out that the current debugging of microservices systems depends largely on the developer's experience with the system and similar failure cases, and mainly relies on manual methods to check logs. Having said that, each architecture has its benefits and disadvantages, so Liu et al. (2020), also provided a comparison of traditional monolithic architecture, service-oriented architecture (SOA), and microservices architecture. Also, a lot of studies have been done on the aspects of migration from one to the other, along with strategies and lessons learned. Regarding the migration strategies, first Raj et al. (2022), demonstrated a5-stepp migration strategy, and performed it in a standard case study application. Secondly, De Lauretis (2019) defined a migration strategy, from a Monolithic Architecture to a Microservices Architecture, to take advantage of several benefits offered by microservices architecture, such as scalability and maintainability, and more. This strategy rotates around the business functionalities concept, and it is composed of five phases: (a) Function analysis, (b) Business functionalities identification, (c) Business functionalities analysis, (d) Business functionalities assignment, and (e) Microservices creation.

2.2. Software reuse in SOA

With respect to supporting the reuse of services in SOA, there is related work that takes on some of the problems also addressed by the SmartCLIDE methodology, e.g., service discovery, composition, or creation. Blal et al. (2017) tackled the area of model-driven service

specification, proposing a method that generates SOA services from the specification of business processes expressed in BPMN, aiming to bridge the gap between business processes and SOA-based applications that support them. SmartCLIDE also employs BPMN, however, it also exploits this approach to explore service composition by extracting textual representations that can be used to discover services. Elqortobi et al. (2018) proposed an architecture for the dynamic composition of web services guided by a live testing technique. This approach aims to optimize resources and improve the efficiency of automating and integrating business processes based on SOA. Masood et al. (2018) also mentioned the importance of smooth selection, configuration, and composition of existing services to deal with runtime changes or the evolution of end-user requirements in service-based systems. SmartCLIDE also aims to support service composition, however, it does not focus on the runtime, but rather on the design time, i.e., it supports the composition of services during the design of a business process. Chen et al. (2023) proposed a keyword-driven service recommendation approach, which employs a deep reinforced Steiner tree search (K-DRSTS) on a service-keyword correlation graph (SKCG). Similarly, Bianchini et al. (2018) developed a discovery and recommendation technique, considering factors such as developers' social networks and experience in web application development. Finally, Zhao et al. (2022) addressed the issue of the lack of attention to semantic and syntactic information in web service classification and, in response, proposed the use of a relation-aware graph attention network. In contrast, SmartCLIDE provides a tiered solution, considering in-house, domain-related, and domain-agnostic services, which boosts the finding capabilities of SmartCLIDE multi-model search discovery approach. Finally, when focusing on microservices, noteworthy is the ecosystem that Ericsson developed to systematically practice large-scale reuse of microservices in a cloud-native context. Usman et al. (2022) discusses in detail how various ecosystem aspects facilitated the development and reuse of microservices across Ericsson products, along with some lessons learnt. The Ericson ecosystem (ADP) shares several similarities to SmartCLIDE, especially in terms of motivation (reuse through a Marketplace or Repository) and development paradigm, but also differences: e.g., among others, not offering an IDE for Service Creation.

2.3. Novelty and contributions

The main novel contributions of this work can be summarized as follows: First it describes the end-to-end SmartCLIDE development methodology, helping organizations to systematically apply reuse, when developing service-based software applications. The integration of the specific steps of the methodology, under a high-level working guide, can help organizations in understanding the bigger picture and the context through which each step is applied. Second, we strongly believe that to aid organizations in applying the methodology, there is a need for tool support. The development of Eclipse Open SmartCLIDE is an important contribution to the community, since this advances state-of-practice, enabling the developers to easily use all the developed approaches. The fact that portion of this platform is reused, is expected, since we do not want to re-invent the wheel in all steps: e.g., we do not need a new Docker, or a new Jenkins, but we believe that it is an important improvement to integrate it together with one of the most popular IDEs (Eclipse) and use a variety of tools through a common interface. Finally, this work provides evidence that the end-to-end solution is valuable to the practitioners, since it explores its industrial relevance, the benefits that it brings, and assesses its usability. We note that this step is completely novel, in the sense that this is the first evaluation on the platform and the proposed development methodology.

3. Introduction of the smartclide framework

3.1. Industrial requirements for smartclide framework

The requirements elicitation process started in mid-2020 and was continued as an on-going process for almost one year. The pilot cases of SmartCLIDE provided the basis for the derived industrial requirements, which have been prioritized by the industrial pilot partners and have been categorized to facilitate the detailed specifications of technologies, methods, and tools that needed to be developed within the project. Further inputs have been collected from external industrial software developers, via an online survey to verify, validate and indicate the potential scope of requirements coming from the pilot partners in the project.[3] The industrial pilot cases and broader survey results have been analyzed by the R&D partners, the associated technical challenges, and technological approaches for addressing the industrial requirements are identified for each of the major development environment components. The resulting requirements from the industrial pilot cases and those specified for each of the development environment technology components provide the baseline requirements that have driven the development of the SmartCLIDE framework.

The first two steps of the process (i.e., pilot case analysis and categorization) have led to a set of 105 functional requirements[4] that have been classified into the following key high-level requirements: (a) Services Creation; (b) Services Discovery; (c) Services Composition; (d) Services Testing; (e) Services Deployment; (f) Usability and Visualization Requirements; (g) Code Repository; (h) Quality Assurance; and (i) AI support. The elicited requirements have been tagged as "*MUST*", "*SHOULD*", and "*COULD*", by all pilot providers, so that an initial prioritization to take place. The industrial survey that followed was conducted with 25 practitioners, with different software development roles, representing 18 different companies from different regions of the EU. Among these companies, 50% were SMEs mainly in the range of 10–150 employees, with the other 50% being from large organizations with 500+ employees. Based on the pilot partners' requirements prioritization and the results of the online survey, in Table 1, we present the key features that we have implemented in SmartCLIDE framework. We remind the reader that for most of these features, an individual validation on their accuracy and research rigor has already been performed in previous studies (see Appendix A). The implementation of each feature in Eclipse Open SmartCLIDE is presented through a walkthrough in Appendix B.

3.2. SmartCLIDE end-to-end methodology for developing SOA applications

The proposed solution aims to boost the reuse of services, at three levels (ordered by priority): (a) ***in-house reuse***—reusing services that have been developed and deployed internally; (b) ***domain-specific reuse***—reusing services that are released as OSS, and the source and target systems belong to the same application domain (e.g., games, business applications, etc.); and (c) ***domain-agnostic reuse***—reusing

[3] Due to an NDA with the industrial partners of the consortium, the deliverable describing the requirements of SmartCLIDE has been characterized by EU as confidential, in the sense that the pilot applications were presented in rich details. Therefore, a link is not available for the full reporting. In this section, we describe only the necessary information to understand the rest of the document.

[4] Examples of such requirements are: "*SmartCLIDE is able to search resources and services of an existing system through its REST API*", "SmartCLIDE provides workflow mechanisms to easily decompose problems into smaller pieces", "*SmartCLIDE provides support for integrating an online coding IDE and a BPMN Editor for code formatting and syntax error highlighting and integration with external services*", etc.

Table 1
SmartCLIDE features.

Feature	Description
Service composition	The user must be able to design SOA-based systems in a drag and drop manner to enable novice users to compose workflows, from deployed services or microservices. The composition of services and their visualization must rely on well-defined and -known standards or notations (e.g., UML, BPMN, etc.). In terms of reuse, this feature is the key for functional decomposition and the initial stage for identifying candidates for reuse.
Workflow summarization	To enable the understanding of the functionality of an already designed workflow, even from stakeholders not involved in the original development the system should provide an AI-based mechanism that will textually summarize the functionality of the workflow. In terms of reuse, this feature can provide textual information on the needed functionality, to feed the service discovery feature (see below) more accurately.
Workflow quality assessment	The workflow (composition of services) represents the system-level design, i.e., which services and how work together so as to provide the needed functionality. Therefore, quality assessment at the level of the workflow must be facilitated. The platform must be able to assess the security and the maintainability of the workflow. In terms of reuse, it is of paramount importance of being able to monitor (in a high-level way) the quality of the target system, to accept reuse suggestions or decline them to develop from scratch a more high-quality level components' solution (if not satisfactory).
Service discovery	The solution must be able to support queries to external and internal records to identify services that can serve a given functionality. This feature is the heart of reuse in the SmartCLIDE solution. The solution must be able to retrieve, store, classify, and recall services from: (a) open-source software; and (b) in-house repositories. These services must be recalled by functional relevance, when requested by the workflow composition feature.
Service creation	The solution must provide a cloud-based IDE for developing services if they cannot be reused. The IDE must support (initially the Java language), must provide support for managing a code repository, and seamlessly initialize a CI/CD process. The solution must be easy to use and hide the technical complexity of the tools.
Design patterns selection assistant	When developing services from scratch, the solution should support the decision-making of the software engineer in terms of object-oriented design. The system should provide some AI-based decision-support for instantiating GoF patterns, i.e., the most-known pattern catalogue. Other types of patterns, such as architectural patterns and security patterns will also be considered, but no instantiation mechanisms (i.e., code generation) will be offered.
Test generation	The solution must be able to provide AI-based support for testing. The platform must generate some (at least the basic) test cases for a service developed from scratch. The tests generated are in the form of Junit code.
Service quality assessment	The solution should be able to assess the quality of a software service. This service can either be a service developed from scratch or a service that has been discovered and cloned for modification in the Eclipse Theia project workspace (i.e., a local code repository). Similarly, at the workflow level, the solution must assess security & maintainability. Since this feature is relevant only for the case of white-box reuse, reusability will also be assessed.
Deployment	The solution must be able to easily deploy and monitor both services and workflows. The deployment must rely on well-known and existing solutions, similarly to service composition (technology reuse).

OSS services from a different application domain. We clarify that domain-specific reuse is promoted in SmartCLIDE, since reuse is more efficient when performed within the same application domain (Snook et al., 2004).

To achieve this goal, in SmartCLIDE, we have tailored the REACT reuse process (Lampropoulos et al., 2018) to fit the service-based software development paradigm, as illustrated in Fig. 1—in the discussion, with bold and italic fonts we denote where each feature presented in table is placed.

[PHASE-1: REUSE CONCEPTION] We consider that after analyzing the requirements, the engineer designs the solution through BPMN modeling (i.e., theoretically solves the problem by composing services—***service composition***).

[PHASE-2: REUSABLE ASSET IDENTIFICATION] The services in this model correspond to the candidates for reuse, which are then attempted to be identified (Alizadehsani. et al., 2022) (***service discovery***) in the SmartCLIDE repository (in house, domain-specific, domain-agnostic services are queried).

[PHASE-3: REUSABLE ASSET ADAPTATION] If the service discovery is not successful, the flow goes to ***service creation***, i.e., the developers would need to build their own solution from scratch. This process is repeated until the reuse candidates list is exhausted. During the SmartCLIDE project, we have brought several enhancements to the process, by supporting the developer with various additional tools (such as a Polyzoidou et al. (2023) ***design patterns selection,*** Maikantis et al. (2021) ***test case generation, automated deployment,*** Kotsikoris et al. (2022) ***workflow summarization***, Nikolaidis et al. (2022) quality assurance, etc.) that can accelerate development / improve the process, leading to a product with fewer bugs and better structural quality.

The full list of the developed features is presented in Appendix A, as well as the way that we have instantiated them. We remind that the goal of the upcoming validation is not to assess how the features are implemented, but if the way they are implemented is ready to apply in practice (i.e., relevant, useful, and easy to use).

4. Empirical validation

4.1. Objectives & research questions

The main target of the SmartCLIDE project was the provision of a platform that is relevant (i.e., applicable, and useful) to the SOA-based cloud application development industry, as well as usable, that would foster a reuse culture. These targets are the basis for the empirical validation presented in this study. According to the goal of the study, we have derived three research questions (RQ):

- ***RQ$_1$***: Is the SmartCLIDE framework industrially relevant?

Through this research question, we first explore if the framework is

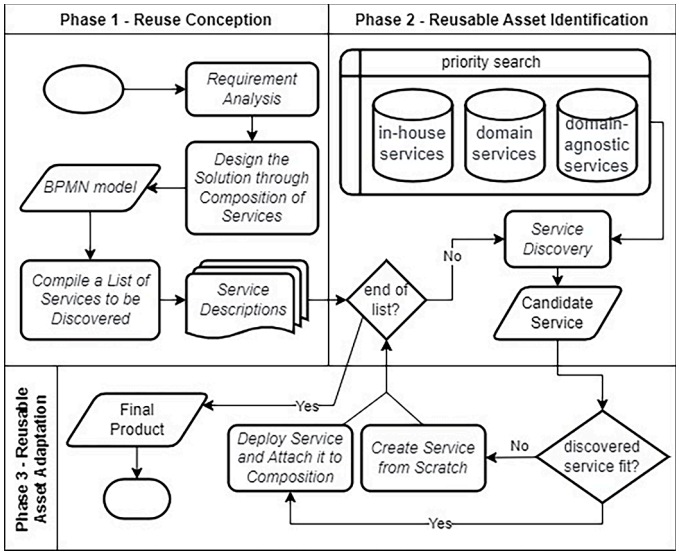


Fig. 1. SmartCLIDE methodology for the reuse of software services.

appealing to industrial stakeholders and therefore if they would find it useful to adopt in the future. In this research question, we are interested in understanding if some specific features of SmartCLIDE are more useful than others. We note that the research rigor of SmartCLIDE approaches has already been evaluated in previous studies—see Section 2.2. The answer to this question can unveil interesting research directions for the cloud and SOA communities, the Research & Development team of SmartCLIDE, and is interesting to participants in the sense that they can get a hint on which features are more applicable, based on the perception of their colleagues.

- ***RQ*$_2$**: To what extent can the use of the SmartCLIDE platform bring benefits in SOA-based cloud application development industries?

The motivation for this research question is two-fold: (a) to explore if the SmartCLIDE solution can "*persuade*" practitioners to employ reuse into their cloud development projects. From the analysis one can also identify the features that they need to promote inside their company to build a reuse culture; (b) to explore the benefits that the solution can bring to industries, by facilitating reuse. Such benefits could be time saving, increase in the quality of the solution; less buggy code; or process improvement. The answer to this research question can be useful for researchers and practitioners, in the sense that practitioners can identify the features of the platform that they can employ to achieve cost reduction and better quality, whereas researchers can get insights on the effectiveness of various techniques; thus, better focusing their future research endeavours.

- **RQ$_3$:** What is the usability of the Eclipse Open SmartCLIDE platform?

Apart from being relevant and useful in practice, for a research prototype to be industry-ready, a key factor is to be usable. Through this research question, we focus on the usability of the SmartCLIDE solution, assessing its ease of use, learning curve etc. We note that for this research question the evaluation is system-wide, since the usability cannot substantially differentiate among features. The outcome of this research question is of paramount importance to the R&D team of SmartCLIDE for improvement suggestions, as well as the interested practitioners since it guarantees to some extent the end-users' experience.

4.2. Case selection and units of analysis

This study is an embedded multiple case study, in the sense that it involves multiple units of analysis within the multiple cases. As units of analysis, we consider software practitioners that participate in the study; whereas as cases the companies that these practitioners work for. The context of the study is the service-based software development since all cases employ this development paradigm. The companies are anonymized due to an NDA, but in Table 2, we list the country and the number of the units of analysis that they have contributed. In total the study was comprised of 20 units of analysis. The main source of participants was the three (3) pilot providers of the project, but to avoid bias, we have also included developers from various companies in Greece. In this respect, they are considered as experts in the field, and their ability to judge industrial relevance was safeguarded. The process for reaching this sample was initiated by first contacting their companies, who provided us with as many engineers as possible, based on availability and time constraints. Then, after having an initial pool of participants we selected individual engineers that work on SOA; with the aim to keep a balance between developers and other roles, as well as between junior and senior engineers (see Fig. 2). This balance was a goal of unit selection to avoid bias and boost generalizability. We note that from the sample of subjects, we have removed practitioners that have not used SmartCLIDE in their working routine, during the 15-days trial (e.g., because in the end, they did not have time for this).

Table 2
Participating companies demographics.

Country	Participants	Size
Luxemburg	6	Large enterprise
Spain	3	Small-medium enterprise
Germany	4	Small-medium enterprise
Greece	7	Various

We note that all the involved companies (or isolated practitioners) have used the platform for a 15-day period before the evaluation (December 2022). C1 used the platform for developing a pensioning system, C2 for compiling IoT applications from existing services, and C3 for developing a system that manages cloud infrastructure on customer demand. The independent evaluators were employed by companies developing services for casino games, banks, and retail websites.

4.3. Data collection

To answer the research questions and given the fact that the SmartCLIDE platform has been recently released (thereof participants did not have a prior experience with it), we have performed the data collection, upon a 15-day trial. We note that this period could have not been expanded further, due to the limited budget of participating industries for the purpose of this study. In the beginning of this period, the participants got acquainted to the platform through a video tutorial and were then asked to involve the platform in their development routines (using the source-code of their industrial projects) in the way that they perceive as most beneficial. Upon the completion of the trial period, we proceeded to data collection through a 1-day workshop. Data collection was comprised of two methods (2 surveys sessions and 1 focus group) aiming to achieve method triangulation for all research questions. A mapping between the research questions and data collection methods is presented in Table 3. Below, we discuss in detail each data collection method, and how it was applied for the purpose of our study. As part of the workshop, the participants were asked to complete the task, described in Appendix B (Platform Walkthrough), to be reminded with the basic platform capabilities, as well as to be more accurate in their answers in the questionnaires, focus group, and usability assessment.

Survey-1: The first survey was aiming to collect data for answering RQ$_1$ and RQ$_2$. Each participant was provided with an online questionnaire,[5] focusing on the industrial relevance of the SmartCLIDE platform, its ability to foster a reuse culture, and its contribution to cost reductions or quality improvement. To build the survey instrument, we developed a 9-section questionnaire (one for each feature), and 3 questions per section (one for each targeted impact—apart from usability), tailored to match the context of each feature, as shown below. The responses were provided on a 5-point Likert scale.

> Do you consider <<FEATURE>> as relevant (i.e., applicable, and useful) for your company?
> Do you believe that <<FEATURE>> can boost the extent of reusing services?
> Do you think that <<FEATURE>> can contribute to time saving, increase in the quality of the solution, less buggy code, or process improvement?

Survey-2: This survey aimed at evaluating the usability of the SmartCLIDE platform. Similarly, to before, we used an online questionnaire[6] to get the responses of the participants. The questionnaire was structured based on the System Usability Scale (SUS) (Brooke, 1996), which is one of the most well-known instruments for usability assessment. Thus, the participants have been given the following statements to

[5] https://forms.gle/qh8JHgcFEKBmqgA39.
[6] https://forms.gle/jWr3qDYQHUzZxjKu5.

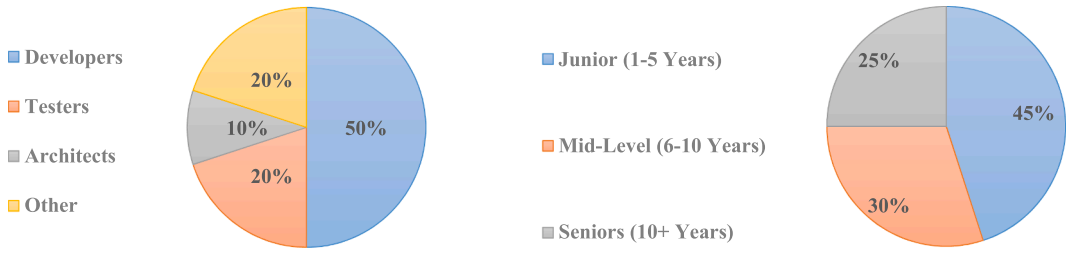


Fig. 2. Sample demographics.

Table 3
Mapping of data collection methods to research questions.

	Survey-1	Survey-2	Focus group
RQ_1	X		X
RQ_2	X		X
RQ_3		X	X

indicate their level of agreement, in a 5-point Likert scale.

I think that I would like to use this system frequently
I thought this system was too inconsistent
I found the system unnecessarily complex
I felt very confident using the system
I thought the system was easy to use
I found the system very cumbersome to use
I think I would need the support of a technical person to be able to use this system
I would imagine that most people would learn to use this system very quickly
I found the various functions in this system were well integrated
I needed to learn a lot of things before I could get going with this system

Focus-Group: As a final means of data collection, we have performed 4 industrial focus groups (one for each company). During the planning of the focus group (Kontio et al., 2004), we defined the goals: "*to discuss: (a) applicability / usefulness of the platform; (b) the benefits that can be obtained; and (c) usability issues*". Regarding the design, each focus group was intended to last for 45′ (with each company—3 h in total)[7]; and was conducted using a tele-conferencing platform. The first 2 blocks were intended to last for 12 min, whereas the last one was for approximately 10 min. While conducting the focus group the discussion was focused on the topics outlined below. We note that in many cases, when there was an agreement among the participants, we asked the remaining participants to only provide complementary or contradictory claims. The size of each focus group was between 3 and 8 participants, which falls within the limit of effective focus group design, based on Kontio et al. (2004).

Block 1: Industrial Relevance Assessment
- Can you explain to us the tasks that you have performed over the last period, using the SmartCLIDE features?
- Have you found these features useful in practice, and why?
Block 2: Benefits of using SmartCLIDE Framework
- Do you think that by using the SmartCLIDE features, you are more probable to reuse existing services?
- Can the use of SmartCLIDE reduce the time that you need for developing a cloud application? Why?
- Can the use of SmartCLIDE increase the quality of the final product? Why?
- Can the use of SmartCLIDE prevent the introduction of bugs in the final product? Why?
(*continued on next column*)

(*continued*)

- Can the use of SmartCLIDE improve your process for developing a cloud application? Why?
- Do you see other benefits?
Block 3: Usability Assessment
How do you perceive the usability of the SmartCLIDE platform in terms of:
- Effectiveness (i.e., the accuracy and completeness with which users achieve specified goals)?
To what extent did you manage to complete the intended tasks?
How did you experience the navigability in the tool?
- Efficiency (i.e., the resources expended compared to the achieved goals)?
How much time did you spend per task?
Was it more or less than the expected time needed based on your experience using the current process?
- Satisfaction (the comfort and acceptability of use)?
How confident did you feel while using the system?
Would you like to use this system frequently?
How complex would you characterize the system?
To what extent would you need the support to be able to use the system?
How easy would you consider learning how to use the tool?
What kind of background would you consider as mandatory before you get going with the system?
How would you describe the experience in terms of the reactions of the system to possible stimuli?
- Have you faced any other usability issues?

At the end of this process, we thanked the participants for their time, we explained the next steps (i.e., transcribe, analyze, report the results), and we asked them if they want to receive the results by mail. We note that the transcriptions of the focus group, as well as the data obtained from questionnaires, have not been made available, due to confidentiality reasons, and the signed NDA. The data collection instrument has been piloted with software engineers, as part of a MSc course on Service-Based Software Development. The group of graduate students involved in the piloting phase was not overlapping with the participants presented in Table 2. Upon piloting, based on the received feedback, as well as our experiences, we have finalized the data collection instrument, as presented above.

4.4. Data analysis

To validate the SmartCLIDE platform, we have used quantitative analysis for providing a synthesized overview of the achieved impacts, and qualitative analysis for the interpretation of the results. To synthesize qualitative and quantitative findings, we have relied on the guidelines provided by Seaman (1999). On the one hand, to obtain quantitative results, we use the data obtained from the two surveys. To aggregate the ordinal values of the Likert-scale we have summed the scores assigned by all participants to a specific question. For presentation purposes, we used bar charts to visualize the sum score of responses for all participants. The maximum value in the y-axis for each bar would be 100 points (20 respondents * maximum Likert-scale rating), for RQ_1 and RQ_2. The closer the bar to the maximum, the higher the grade that the participants have assigned to the feature, for each evaluation criterion (relevance and benefits). For usability, we provided the total SUS score, along with the most common scales for interpretation, in terms of acceptance, adjective, and grade. The way that the usability score is

[7] The focus groups with most of the companies lasted for approximately 1 hour, due to the input that we received from the participants. Especially, regarding Blocks 1-3, the average discussion time was 15-20 minutes, depending on the company.

achieved through the SUS instrument is described by Brooke (1996). Next, we briefly describe the process: The participants' scores for each question (sometimes 5 is best, for others 5 is the worst, based on the nature of the question) are converted to a number, added together, and then multiplied by 2.5 to convert the original scores (of 0–40) to a range from 0 to 100. Though the scores are limited to an [0, 100] range, the score is not a percentage and should be considered only in terms of its percentile ranking. Based on the literature, SUS scores higher than 68 are considered above average and anything lower than 68 is below average (Brooke, 1996). On top of this, the analysis of SUS results can be performed at specific questions' level, to spot specific points that might yield improvement (e.g., question 7 aims on the self-explanatory power of the UI).

On the other hand, to obtain the qualitative assessments, we use the focus group data, which we have analysed based on the Qualitative Content Analysis (QCA) technique (Elo and Kyngäs, 2008), which is a research method for the subjective interpretation of the content of text data through the systematic classification process of coding and identifying themes or patterns. This process involved open coding, creating categories, and abstraction. To identify the codes to report, we used the Open-Card Sorting (Spencer, 2009) approach. Initially, we transcribed the audio file from the focus group and analysed it along with the notes we kept during its execution. Then a lexical analysis took place: in particular, we counted word frequency, and then searched for synonyms, and removed irrelevant words. Then we coded the dataset, i.e., categorized all pieces of text that were relevant to a particular theme of interest, and we grouped together similar codes, creating higher-level categories. The categories were created during the analysis process by both the first and the third author and were discussed and grouped together through an iterative process in several meetings of all authors. The reporting is performed by using codes (frequency table) and participants' quotes. Based on Seaman (1999) qualitative studies can support quantitative findings by counting the number of cases in which certain keywords occur and then comparing the counts of different keywords or comparing the set of cases containing the keyword to those that do not.

5. Findings and interpretation

In this section, we present the results of this study, organized by research question. In the narrative, we present: (a) codes with capital letters; (b) the features of the platform in bold fonts; (c) participant quotes in italics; and (d) in capital letters and with italic fonts, we denote the scales of the Likert questionnaires. As the basic means for interpreting the results, we have preferred not to provide our own subjective interpretations, but to support any claims with quotes of practitioners. The lessons learnt and implications to researchers and practitioners (i.e., our own interpretations) are discussed in Section 6.1.

5.1. Industrial relevance

In Fig. 3, we present the findings regarding the industrial relevance of the nine (9) features of the Eclipse Open SmartCLIDE platform that we have evaluated. We present a bar chart with the sum of the score of each feature in the Likert scale question on industrial relevance. Each feature got five (5) points for "*VERY RELEVANT*" and one (1) point for "*NOT RELEVANT AT ALL*". Given the fact that we had 20 respondents, the maximum score that a feature could reach would be 100, if all respondents have evaluated as "*VERY RELEVANT*"; thus, the vertical axis showcases the proximity of each service to the 100 points.

Given this interpretation, we can deduce that all features have been positively assessed, especially considering that the MODE values for ***Service Discovery, Workflow Summarization, Service Quality Assessment***, and ***Design Pattern Selection Assistant*** were "*VERY RELEVANT*". For the rest of the features, the MODE value was "*RELEVANT*", and no feature scored lower than that. For the special case of ***Design Pattern Selection Assistant***, and the controversy of the results by using SUM and MODE, we can assume that some evaluators were very positive, whereas others were neutral or slightly negative; however, the majority provided a positive vote. To interpret and complement the findings with a qualitative analysis, we discuss the main findings by processing the outcomes of the focus group. First, we need to note that the features that have been tested by the participants before the start of the workshop were: ***Service Creation*** (75%), ***Service Composition*** (40%), and ***Service Discovery*** (25%). For the rest of the features, the participants have not performed

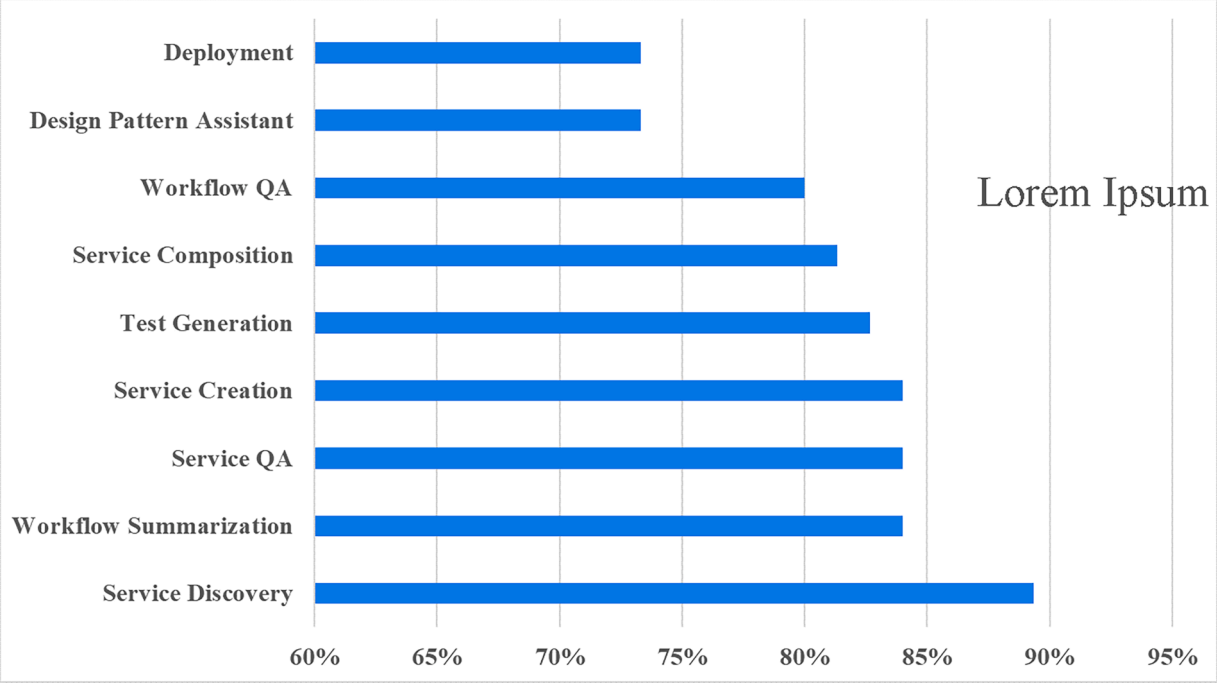


Fig. 3. SmartCLIDE features industrial relevance.

any related task beforehand. However, this finding does not seem correlated to their perception on industrial relevance.

The main arguments that the participants have used to champion the relevance of the Eclipse Open SmartCLIDE are discussed below. The most important aspect (brought up by 80% of the participants in the focus group) was the ALL IN ONE ability of SmartCLIDE, i.e., the ability to handle everything in a single platform, without the need to change among tools and environments. As vividly explained by some stakeholders "*SmartCLIDE enables you to have a variety of components ready-to-use in the workflows and overall has everything in one place*", or "*SmartCLIDE saves time programming and executing scenarios, because you can create services in the IDE integrated, if needed you can re-discover services which you don't need to create again…*". Furthermore, 60% of the participants, mentioned TIME SAVING as an important competitive advantage, obtained through reusing services, through speeding up the design process (e.g., through the Design Pattern Selection Assistant), and having everything in the same place. Additionally, it seems that an important percentage of the participants (40%) considered as important for the adoption of the platform the "*extra*" functionalities that the platform brings (e.g., ***Design Patterns Assistant, Service Quality Assurance***), beyond the must have ones (***Service Composition, Service Discovery***, and ***Service Creation***). Finally, a small percentage (15%) of the participants emphasised on the usefulness and industrial relevance implied by RUNTIME MONITORING ("*…have a clear image of what is executed when running a workflow and it's easy to find where something has failed to execute, also you can see whenever you want the values of the input / output parameters at the time of execution. This saves much time because you might not need to search in logs…*") and provision of ABSTRACTIONS ("*… at development stage such as abstractions on data transformations or processing; at the testing stage, mechanisms to visualize flow and status of artefacts to automatically test the expected behavior; at the deployment stage, abstractions of physical and virtual resources…*"). However, the participants have complained on some missing features. The most important missing feature, underlined by 20% of the participants, was the lack of integration with EXTERNAL REPOSITORIES which is characterised as a very handy option.

5.2. Obtained benefits

In Fig. 4, we provide a quantitative birds-eye view of the perceived benefits that each feature can bring to SOA-based development. The method of representation is the donut-in-a-pie plot; an approach that can nest several indicators for the same feature, in the same plot. In our case, we have developed one pie (per feature), including five (5) donuts—one for every aspect that is being evaluated in a Likert scale (Survey #2). Similarly to RQ_1, a full donut would have been attributed to a feature that scores as "*CERTAINLY*" by all 20 participants (100%) in the corresponding question on the anticipated type of improvement.

Fig. 4 can be interpreted in two ways: (a) within-pie comparisons—to identify the quality aspects for which the feature is deemed as more beneficial; and (b) cross-pie comparisons—to identify which feature is considered as the most beneficial, with respect to a quality aspect of interest. For instance, by focusing on ***Service Discovery***, we can observe that it is considered as most beneficial in terms of BOOSTING REUSE and TIME SAVING. Reading the figure, in the alternative way, we can deduce that if a team is interested in IMPROVING PRODUCT QUALITY, they should invest on ***Test Case Generation*** and ***Service Quality Assessment***.

Regarding the BOOSTING REUSE opportunities offered by SmartCLIDE, 90% of the respondents were positive during the focus group. By far, the most mentioned feature regarding reuse was ***Service Discovery***, followed by ***Service Composition***. It goes without saying that Service Discovery lies in the heart of the reuse mechanism of SmartCLIDE, and this has been acknowledged by all participants. Even the ones that were not seeing substantial benefits in reuse, have agreed that Service Discovery (if properly implemented) can drive reuse. On the other hand, it seems that Service Composition has been promoted to an important feature, since it is the starting point for reuse: i.e., you first need to decompose a system to functionalities properly, and then, you can start searching for reusable components. Cumulatively, (for all features) BOOSTING REUSE has not proven to be the strongest selling point of the Eclipse Open SmartCLIDE platform, with the only exception of Service Discovery, which scored very high (70% of the respondents ranked the feature as "*CERTAINLY*" in the possibility to boost reuse).

In terms of TIME SAVING 100% of the respondents were positive in the focus group discussions (example mini-quotes: "*…with the services discovery module you can find services you might not need to create again…*", "*…automatic workflows help to reduce development time…*", "*…because the deployment of the project happens with one click…*", and "*…it facilitates service reuse and removes friction in service configuration…*"). The same observation can be made through the Likert-scale assessment, since ***Service Creation, Service Discovery, Service Composition***, and ***Deployment*** scored more than 90%, and obtained a MODE value of "*CERTAINLY*" in the possibility to reduce development time question. The Service Creation feature has been acknowledged to hide the complexity of configuration; Service Discovery has been credited for saving time, due to reuse; Service Composition saves time since it offers a drag and drop solution for composing services; whereas Deployment (especially Workflow Deployment) can be fully automated through the platform.

With respect to IMPROVING PRODUCT QUALITY the discussions during the focus group were not that enthusiastic, since 70% were obviously positive ("*…if the quality assessment tools work as expected, the result should be a product of better quality than before…*", "*…the use of software design patterns is facilitated, and the code is tested by auto-generated unit tests. Also, the whole workflow is demonstrated well, giving the chance for evaluation and improvement…*"); whereas others were more conservative (*"…not sure. More complex scenario is necessary to be confirmed about that…*"). Through the quantitative analysis, ***Service QA, Workflow QA***, and ***Test Generation*** have been promoted to the most important features to safeguard product quality—with approximately 40% of the respondents being "*CERTAIN*" for improving product quality with these features. At this point, it deserves to be noted that this question despite targeting product quality (and having a different question for bugs), in the end attracted tools to safeguard both the structural, as well as the functional views of product quality. By comparing the results for service-level and workflow-level quality assessment, the results pointed to the service-level as the most important one: probably due to the largest presence in the study setup of developers, compared to architects or managers (who would be interested in more high-level quality viewpoints).

Regarding LESS BUGGY CODE, even though a larger majority of participants were positive (80%) the average score was the lowest among all envisioned benefits, for the complete group of features (70%, MODE: 4). Even the statements of the participants in the focus group were not as confident as the other envisioned benefits (*"…not sure based on the scenario that we tested…*", "*…if used in conjunction with other tools…*"). Nevertheless, numerous positive remarks have been made, especially concerning ***Test Generation*** (87%) and ***Workflow QA*** (83%). The main ways to get benefits in terms of bug reduction have been acknowledged to be: (a) test automation; (b) increase of test coverage; (c) transparency in workflows execution; and (d) the improved internal quality that can help writing more bug-free code.

Finally, PROCESS IMPROVEMENT has been unveiled to be the second most important benefit of using SmartCLIDE (80% score, and a MEAN value of "*CERTAINLY*" in the possibility to improve the development process). Although this was not originally the main advancement that we were targeting in the project, we cannot claim that this finding was surprising, since the methodology and the toolchain are certainly standardizing the development process, hiding many of the steps, reducing development time, and lowering the need for configuration and multiple tools (as suggested by the participants of the study). Another very interesting observation for this potential benefit is that all features

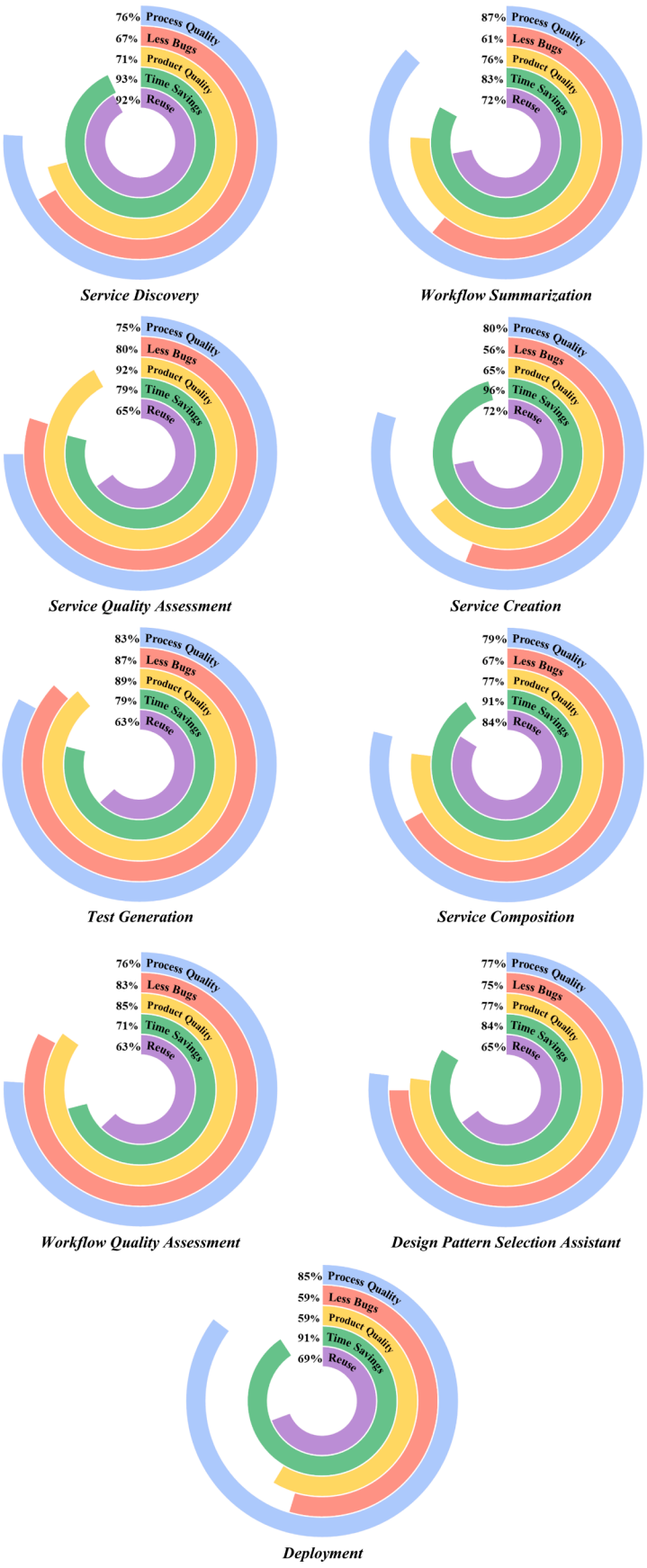


Fig. 4. SmartCLIDE features obtained benefits.

contributed towards the quality score, showcasing a standard deviation of 4% (when the next lower SD is 8% and the largest 12%)—also no feature scored lower than 75% in this criterion. ***Workflow Summarization*** stood out, probably because it can help in understanding the composition of services (i.e., the backbone of the SOA-based development), enabling communication horizontally through the project phases and people.

5.3. Usability assessment

By assessing the usability of the system, we obtained an average SUS score of 74, which is considered as ACCEPTABLE, and can be interpreted, as presented in Fig. 5. By analysing the answers to specific questions (note that in the right part of Fig. 5, we worked with the positive meaning of the question—i.e., many stars declare user satisfaction), we have concluded that the most important usability weaknesses were: "*User Confidence*", "*Integration*", and "*Background Knowledge*". While in the focus group, we further dug into the specific aspects of usability and discussed tentative issues with the participants. In the open discussion, the participants shared with us that they were not very confident, and that they would require written documentation to follow the task flow. The participants felt that there were many things to learn for the first time, but probably the amount of new information needed would decrease after 2–3 weeks of regularly using the system. However, despite the usability issues, all participants managed to finalize the task at hand, given the guidance of the video, and all declared that they have spent less time than expected, for completing it. To explore the level of agreement among the raters, we have performed the non-parametric test Kendall W, which unveils the level of agreement in the rating of all participants, cumulatively for all the SUS questions. The outcome of the test (W: 0.61 with $p<0.01$) suggest a strong agreement among the respondents of the SUS questionnaire.

6. Discussion

6.1. Lessons learnt: implications to researchers and practitioners

The main lessons-learnt from this user study can be decomposed to two parts: (a) the weak and strong points of the framework that can lead to interesting implications to researchers; and (b) the perceived benefits of the SmartCLIDE framework per feature—that can lead to implications to practitioners.

Implications to Researchers: Based on the findings of the study, we can claim that SmartCLIDE has brough some important advancements to the state-of-practice on SOA-based application development, but has also left some problem as partially solved, yielding for additional research opportunities. First, through the evaluation we can claim that the motivation to develop an end-to-end methodology / platform that would combine all steps of the process in a single place, was a successful decision. Thus, we advise researchers of similar future endeavors to integrate their solutions in existing workspaces (e.g., IDEs) to gain similar benefits as we did from the SmartCLIDE adoption into the Eclipse ecosystem (Che, Theia, etc.). Additionally, we can consider that the goal of shrinking development time and effort in SOA-based systems was successful, and the produced approaches can be employed in future research. Finally, we advise researchers to also evaluate possible improvements in the software development process, and not only focus on the core aspects of quality, such as bugs and structure.

On the other hand, the evaluation has also unveiled several limitations that as a consortium we retain as future work, but also point to useful implications to other researchers. First, we need to acknowledge that the proposed end-to-end framework has only partially achieved the goal of improving reuse. More specifically, the framework boosted reuse through Service Discovery and Service Composition; however, we cannot claim that the side tools had showcased a similar success. In that sense, we believe that future research can focus on proposing novel approaches for enhancing reuse: e.g., AI-based suggestions for service composition, proposal of next steps of a workflow, based on similarity, etc. Moreover, the design pattern selection assistant can be promoted to a mechanism for reusing design rationale in the domain of SOA-based development. Furthermore, the solutions offered by the platform did not manage to persuade the developers on the reduction of bugs, apart from Test Case Generation. However, we believe that Service Creation can yield important benefits in this aspect, in the sense that it aids in preventing configuration mishaps. In that sense, we believe that a longitudinal study that explores the effect of automated Service Creation would be beneficial. Finally, in terms of future work, we believe that despite the positive evaluation of quality assessment, further research can be performed both at the service, as well as the workflow level.

Implications to Practitioners: Based on the evaluation of the individual features, we can advise practitioners to use the following features of SmartCLIDE, based on their quality goals. We note that although each feature can be useful in multiple aspects, in Fig. 6, we present only the most striking benefits of each feature. Therefore, we can advise practitioners to use Service Quality Assessment and Workflow Quality Assessment, only if they want to impose a quality assurance process. Similarly, the Test Generation feature is considered relevant, and we recommend its use when product and process improvement are of interest. On the other hand, regarding time savings the most core features have been identified as the most relevant: Service Creation, Service Composition, Design Patterns Selection Assistant, Workflow Summarization, Service Discovery, and Deployment. Out of these features, some are having as side benefits reuse or process improvement. We note that these implications can be transferred outside SmartCLIDE in the sense

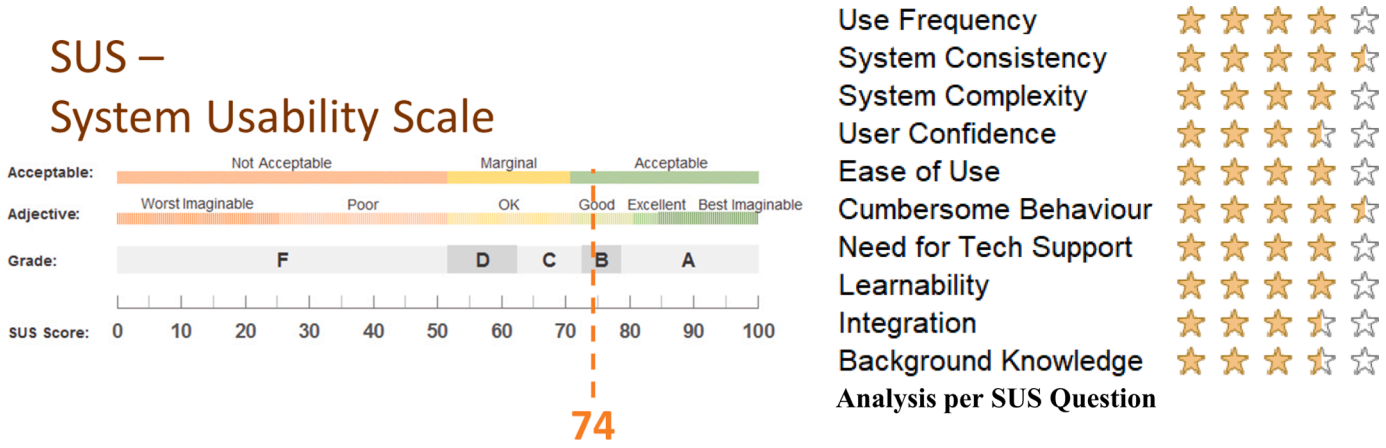


Fig. 5. Eclipse Open SmartCLIDE platform usability assessment.

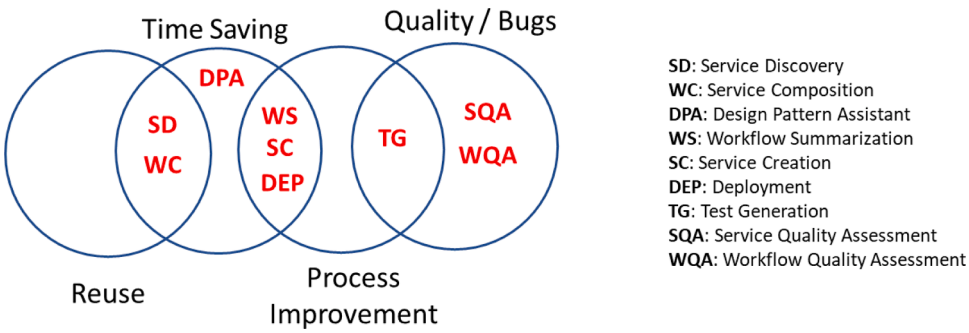


Fig. 6. Features evaluation for practitioners.

that alternative tools that can be used for similar purposes can have similar results. For example, finding service directly through the *ProgrammableWeb* interface can also bring benefits in terms of time saving. However, we need to note that the empirical evidence that we bring correspond only to the SmartCLIDE implementation.

6.2. Threats to validity

Construct Validity. The first threat to construct validity may stem from partial knowledge of platform capabilities to provide an informed assessment. To mitigate this threat, we conducted a 1-day workshop on the platform and asked the participants to use the SmartCLIDE for a period of 15-days before performing the evaluation. Another threat is that practitioners may misinterpret the questions in both Survey #1 and #2. We mitigated this threat by using a structured questionnaire. In Survey #1 (for RQ_1 and RQ_2), the questions were precise and asked for the information we intended to learn about each feature. That said, we acknowledge this decision allows for framing bias, priming the participants to provide a positive evaluation. We mitigated this by explicitly informing participants about framing bias and asking them to reflect on their answers. We also informed them that a perceivably negative evaluation is also valuable to the project, so they should not feel compelled to provide an opinion they disagree with. In Survey #2, we used a well-established survey instrument for usability (SUS) and provided the same clarification on framing bias and priming. We also sought to mitigate mentioned threats via the focus group, which allowed us to revisit the questionnaire topics framed from a different perspective and investigate the output of the questionnaires in depth.

External Validity. The generalizability of our validation is threatened by the sample size of the study. On the one hand, a larger sample could provide a more representative portion of the population of interest. To mitigate this threat, we ensured the representatives of the main industrial stakeholders. Also, to the best of our knowledge, these companies and application domains, although not encompassing all industries, do represent common cases that exploit SOA. That said, to complete these, we also involved practitioners working on other application domains (e.g., games, banks, and retail websites). Additionally, we note that our analysis is purely qualitative, and is not threatened from the sample size, since statistical hypothesis testing is not performed.

Reliability. The open coding process is subject to bias from the researchers. To mitigate this threat, we have used a systematic approach to coding and provided as much detail of the process as possible. We note that peer review was extensively used in the coding process and for verifying the various data analyses performed for the study. Finally, to avoid a confirmation bias of industrial relevance due to the participation of pilot providers, we also included seven practitioners (35% of the sample) from outside the project.

7. Conclusions

In this paper, we presented the SmartCLIDE framework for facilitating service reuse in cloud development. The framework aims to enable developers to boost the reuse of services at three levels: in-house reuse, domain-specific reuse, and domain-agnostic reuse. This goal was achieved by developing six main features and integrating them into a single platform, which is an extension of Eclipse Che and the Eclipse Theia IDE. The features are: (a) service composition; (b) workflow summarization; (c) service discovery; (d) service creation and deployment; (e) design patterns selection assistant; and (f) workflow quality assessment and service quality assessment. Although some features have been validated independently in previous studies, this study aimed to validate the framework end-to-end in terms of industrial relevance, benefits, and usability. To this end, we conducted an embedded multiple-case study with 20 practitioners. Participants were trained to use SmartCLIDE and worked with the platform for 15 days. After that, the practitioners answered two surveys and participated in a focus group to evaluate SmartCLIDE and its features. The results of the study showed that the framework is effective and efficient in facilitating service reuse in cloud development. The framework is currently being used in the three companies that also provided pilots to the project and is publicly available on the project website.

CRediT authorship contribution statement

Nikolaos Nikolaidis: Methodology, Software, Data curation, Formal analysis, Writing – original draft. **Elvira-Maria Arvanitou:** Methodology, Validation, Writing – original draft. **Christina Volioti:** Data curation, Validation, Writing – original draft. **Theodore Maikantis:** Methodology, Software, Data curation, Formal analysis, Writing – original draft. **Apostolos Ampatzoglou:** Conceptualization, Methodology, Validation, Writing – original draft, Supervision. **Daniel Feitosa:** Validation, Writing – original draft, Supervision. **Alexander Chatzigeorgiou:** Conceptualization, Validation, Writing – original draft, Supervision. **Phillipe Krief:** Software, Validation, Supervision.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data that has been used is confidential.

Acknowledgments

Work reported in this paper has received funding from the European

Union's Horizon 2020 research and innovation programme under grant agreement no 871177 (project: SmartCLIDE).

Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.jss.2023.111877.

References

Alizadehsani, Z., et al., 2022. Service classification through machine learning: aiding in the efficient identification of reusable assets in cloud application development. In: 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA'22), September. Gran Canaria, Spain.

Baldassarre, M.T., Bianchi, A., Caivano, D., Visaggio, G., 2005. An industrial case study on reuse oriented development. In: 21st IEEE International Conference on Software Maintenance (ICSM'05), September.

Bianchini, D., De Antonellis, V., Melchiori, M., 2018. Services discovery and recommendation for multi-datasource access: exploiting semantic and social technologies. Stud. Big Data 31, 375–390.

Blal, R., Leshob, A., 2017. A model-driven service specification approach from BPMN models. In: 14th International Conference on E-Business Engineering (ICEBE 2017). November. Shanghai, China.

Brinkkemper, S., Jansen, S., Demir, C., Hunink, I., 2008. Pragmatic and opportunistic reuse in innovative start-up companies. IEEE Softw. 25 (06), 42–49.

Brooke, J., 1996. System Usability Scale (SUS): a quick-and-dirty method of system evaluation user information. In: Jordan, P.W., Thomas, B., Weerdmeester, B.A., McClelland, I. (Eds.), Usability Evaluation in Industry. Taylor & Francis, pp. 189–194.

Capiluppi, A., Boldyreff, C., Stol, K.-J., 2011. Successful reuse of software components: a report from the open source perspective. In: Hissam, S.A., Russo, B., de Mendonça Neto, M.G., Kon, F. (Eds.), Open Source Systems: Grounding Research. Springer Berlin Hei-delberg, Berlin, Heidelberg, pp. 159–176.

Chen, H., Wu, H., Li, J., Wang, X., Zhang, L., 2023. Keyword-driven service recommendation via deep reinforced steiner tree search. IEEE Trans. Ind. Inf. 19 (3), 2930–2941.

Dai, F., Chen, H., Qiang, Z., Liang, Z., Huang, B., Wang, L., 2020. Automatic analysis of complex interactions in microservice systems. Complex. 1–12.

De Lauretis, L., 2019. From monolithic architecture to microservices architecture. In: *International Symposium on Software Reliability Engineering Workshops (ISSREW)*, October. IEEE, Berlin, Germany, pp. 93–96.

Digkas, G., Nikolaidis, N., Ampatzoglou, A., Chatzigeorgiou, A., 2019. Reusing code from stackoverflow: the effect on technical debt. In: 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA'19), August. Kallithea-Chalkidiki, Greece.

Djogic, E., Ribic, S., Donko, D., 2018. Monolithic to microservices redesign of event driven integration platform. In: 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), May. Opatija, Croatia.

Elo, S., Kyngäs, H., 2008. The qualitative content analysis process. J. Adv. Nurs. 62 (1), 107–115.

Elqortobi, M., Bentahar, J., Dssouli, R., 2018. Framework for dynamic web services composition guided by live testing. Lect. Note. Instit. Comput. Sci. 206, 129–139.

Johnson, B., 2004. The benefits of service oriented architecture. Objectsharp Consult.

Kontio, J., Lehtola, L., Bragge, J., 2004. Using the focus group method in software engineering: obtaining practitioner and user experiences. In: International Symposium on Empirical Software Engineering. Redondo Beach, CA, pp. 271–280.

Kotsikoris, P., Chaikalis, T., Ampatzoglou, A., Chatzigeorgiou, A., 2022. Automated summarization of service workflows to facilitate discovery and composition. In: 17th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE'22), April. Online Streaming.

Krueger, C.W., 1992. Software reuse. ACM Comput. Surv. (CSUR) 24 (2), 131–183.

Lampropoulos, A., Ampatzoglou, A., Bibi, S., Chatzigeorgiou, A., Stamelos, I., 2018. React - a process for improving open-source software reuse. In: 11th International Conference on the Quality of Information and Communications Technology (QUATIC'18).

G. Lewis, D. Smith, and K. Kontostathis, A Research Agenda for Service-Oriented Architecture (SOA): Maintenance and Evolution of Service-Oriented Systems, Carnegie Mellon University, Software Engineering Institute's Digital Library. Software Engineering Institute, Technical Note CMU/SEI-2010-TN-003,2010. Available: https://doi.org/10.1184/R1/6571745.v1. [Accessed: 21-Oct-2023].

Liu, G., Huang, B., Liang, Z., Qin, M., Zhou, H., Li, Z., 2020. December. Microservices: architecture, container, and challenges. In: 20th International conference on software quality, reliability and security companion (QRS-C), December. Macau, China.

Mahmood, Z., 2007. Service oriented architecture: potential benefits and challenges. In: 11th WSEAS International Conference on Computers.

Maikantis, T., Chaikalis, T., Ampatzoglou, A., Chatzigeorgiou, A., 2021. SmartCLIDE: shortening the toolchain of SOA-based cloud software development by automating service creation, composition, testing, and deployment. In: 25th Pan-Hellenic Conference on Informatics (PCI'21), November.

Masood, T., Cherifi, C., Moalla, N., 2018. Service recommendation model based on service composition networks monitoring. In: 12th International Conference on Software, Knowledge, Information Management & Applications (SKIMA), December. Phnom Penh, Cambodia.

Nikolaidis, N., Ampatzoglou, A., Chatzigeorgiou, A., Tsekeridou, S., Piperidis, A., 2022. Technical debt in service-oriented software systems. In: 23rd International Conference on Product-Focused Software Process Improvement (PROFES '22), November. Finland. Springer.

Polyzoidou, E., Papagiannaki, E., Nikolaidis, N., Ampatzoglou, A., Mittas, N., Arvanitou, E.M., Chatzigeorgiou, A., Manolis, G., Manganopoulou, E., 2023. SmartCLIDE design pattern assistant: a decision-tree based approach. J. Softw.: Pract. Exper.

Raj, V., Srinivasa Reddy, K., 2022. Best practices and strategy for the migration of service-oriented architecture-based applications to microservices architecture. In: 2nd International Conference on Advances in Computer Engineering and Communication Systems: ICACECS'21, February. Hyderabad, India.

Seaman, C., 1999. Qualitative Methods in Empirical Studies of Software Engineering. IEEE Transact. Softw. Eng. 25 (4), 557–572.

Singh, V., Peddoju, S.K., 2017. Container-based microservice architecture for cloud applications. In: International Conference on Computing, Communication and Automation (ICCCA), May. Greater Noida, India.

Snook, C.F., Butler, M.J., Edmunds, A., Johnson, I., 2004. Rigorous development of reusable, domain-specific components, for complex applications. In: International Workshop on Critical Systems Development with UML (CSDUML'04). Lisbon, Portugal.

Spencer, D., 2009. Card Sorting: Designing Usable Categories, 1st edition. Rosenfeld Media.

Turner, M., Budgen, D., Brereton, P., 2003. Turning software into a service. Comput. (Long Beach Calif.) 36 (10), 38–44.

Usman, M., Badampudi, D., Smith, C., Nayak, H., 2022. An ecosystem for the large-scale reuse of microservices in a cloud-native context. IEEE Softw. 39 (5), 68–75.

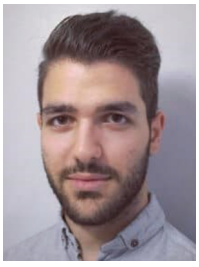
Wang, J., Yu, J., Falcarin, P., Han, Y., Morisio, M., 2008. An approach to domain-specific reuse in service-oriented environments. In: Mei, H. (Ed.), High Confidence Software Reuse in Large Systems. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 221–232.

Xu, X., Motta, G., Tu, Z., Xu, H., Wang, Z., Wang, X., 2018. A new paradigm of software service engineering in big data and big service era. Computing 100, 353–368.

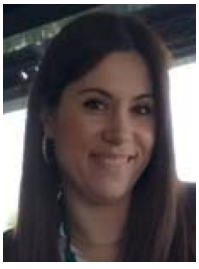
Yang, H., 2012. Software reuse in the emerging cloud computing era. IGI Glob.

Zhao, K., Liu, J., Xu, Z., Liu, X., Xue, L., Xie, Z., Zhou, Y., Wang, X., 2022. Graph4Web: a relation-aware graph attention network for web service classification. J. Syst. Softw. 190.

Zhou, X., Peng, X., Xie, T., Sun, J., Ji, C., Li, W., Ding, D., 2018. Fault analysis and debugging of microservice systems: industrial survey, benchmark system, and empirical study. Transact. Softw. Eng. 47 (2), 243–260.

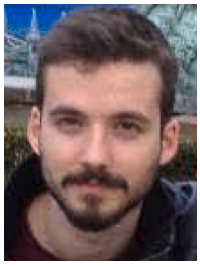


Nikolaos Nikolaidis is an PhD student at the Department of Applied Informatics in the University of Macedonia, Greece. He received a BSc in Applied Informatics from the same university in 2018. He is currently employed as a research associate in the Software Engineering group of University of Macedonia, working on multiple research projects. His-research interests are technical debt management, mining software repositories, and software quality assurance.



Dr. Elvira-Maria Arvanitou is a Post-Doctoral Researcher at the Department of Applied Informatics, in the University of Macedonia, Greece. She holds a PhD degree in Software Engineering from the University of Groningen (Netherlands, 2018), an MSc degree in Information Systems from the Aristotle University of Thessaloniki, Greece (2013), and a BSc degree in Information Technology from the Technological Institute of Thessaloniki, Greece (2011). Her PhD thesis has been awarded as being part of the top-3 ICT-related in Netherlands for 2018. Her research interests include technical debt management, software quality metrics, and software maintainability.

Dr. Christina Volioti is a researcher at the University of Macedonia in the field of Human-Computer Interaction. She received her BSc and MSc in Computer Science with a specialization in Information and Communication Technologies (ICT) in Education from the Aristotle University of Thessaloniki and her Ph.D. in Applied Informatics from the University of Macedonia. She has been a visiting lecturer at the Department of Applied Informatics (University of Macedonia) (from 2016 to 2018) as well as at the Department of Information and Electronic Engineering (International Hellenic University), where she teaches until today. She also works as a SEP Member at Hellenic Open University. She has published her work in international journals and conferences. She serves as a reviewer and co-organizer at international conferences. Finally, she is involved in various national and European research projects.



Theodore Maikantis is an PhD student at the Department of Applied Informatics in the University of Macedonia, Greece. He received a BSc from the International Hellenic University, and an MSc in Applied Informatics from the University of Macedonia. He is currently employed as a research associate in the Software Engineering group of University of Macedonia, working on multiple research projects. His-research interests are technical debt management, mining software repositories, and software quality assurance.

Dr. Apostolos Ampatzoglou is an Assistant Professor in the Department of Applied Informatics in University of Macedonia (Greece), where he carries out research and teaching in the area of software engineering. Before joining University of Macedonia, he was an Assistant Professor in the University of Groningen (Netherlands). He holds a BSc on Information Systems (2003), an MSc on Computer Systems (2005) and a PhD in Software Engineering by the Aristotle University of Thessaloniki (2012). He has published more than 100 articles in international journals and conferences, and is/was involved in over 15 R&D ICT projects, with funding from national and international organizations. His-current research interests are focused on technical debt management, software maintainability, reverse engineering software quality management, open-source software, and software design.

Dr. Daniel Feitosa is an Assistant Professor in the University of Groningen (Netherlands). He holds a B.S. of Computer Science in the Institute of Mathematics and Computational Sciences of the University of São Paulo, with specialization in Software Engineering. He has completed his M.*Sc.* in the Institute of Mathematics and Computational Sciences of the University of São Paulo, within the Software Engineering group (LabES) and Mobile Robotics group (LRM). Finally, Ph.D. in the Department of Computer Science of the University of Groningen, within the Software Engineering and Architecture group (SEARCH). Doctoral dissertation: Applying Patterns in Embedded Systems Design for Managing Quality Attributes and their Trade-offs. He has published numerous articles in international journals and conferences, and is/was involved in various research projects, with funding from national and international organizations. His-current research interests are focused on technical debt management, software maintainability, reverse engineering software quality management, open-source software, and software design.

Dr. Alexander Chatzigeorgiou is a Professor of Software Engineering in the Department of Applied Informatics and Dean of the School of Information Sciences at the University of Macedonia, Thessaloniki, Greece. He received the Diploma in Electrical Engineering and the PhD degree in Computer Science from the Aristotle University of Thessaloniki, Greece, in 1996 and 2000, respectively. From 1997 to 1999 he was with Intracom S.A., Greece, as a software designer. His-research interests include object-oriented design, software maintenance, technical debt and evolution analysis. He has published more than 150 articles in international journals and conferences and participated in a number of European and national research programs. He is a Senior Associate Editor of the Journal of Systems and Software.

Philippe Krief is a project leader, expert in OOD, embedded dev, dev tools and Agile dev process. He received his Ph.D. in CS from Univ. Paris 6 in 1990. He has a passion for Eclipse since its beginning, when he was involved in the early versions of the platform. Before joining the EF staff, he was Sr Architect on Embedded developments for OTI, Collaborative ALM evangelist, and R&D Dev. Mgr for IBM. Since he joined the EF, he is leading research activities. He is involved in the following projects: - Brain-IoT - PDP4E - SmartCLIDE.