

NatVi - A Framework for Agile Software Development, Service-Oriented Architecture and Quality Assurance

James Taylor Faria Chaves(✉) and Sergio Antônio Andrade de Freitas(✉)

University of Brasilia, Darcy Ribeiro Campus, Brasôlia, DF, Brazil
jameschaves@gmail.com, sergiofreitas@unb.br

Abstract. This research presents the NatVi Framework for Agile Software Development, Service-Oriented Architecture (SOA), and Quality Assurance (QA). The research took place in a context of profound and rapid changes in business environments that affect the software development environment. Our previous work did a Systematic Literature Review trying to find articles dealing with SOA and Agile and the challenges inherent in this combination of solutions. In order to build the Framework, this work took the shortcomings found in the solutions presented in the papers and further incorporated the necessary QA concept. In this context, the Framework attempts to provide an answer to how to develop software with quality and rapid deliveries in an ever-changing environment, where the traditional forms of software development may not handle it. Background research identified trade-offs among SOA, QA, and Agile, e.g., formality, documentation, and planning. The background research also identified strengths, weaknesses, and gaps in papers that addressed solutions to problems that arise in software development in the presented context. The results of the background research were assessed and exploited in Framework construction. In a single life-cycle, the NatVi Framework combines Agile, SOA, and QA and addresses the values and principles that guide them in 13 phases distributed in four layers. Each phase is carried out by people who perform specific roles, expressed in terms of inputs, tasks, and outputs. As a Framework, it is not exhaustive, and its main concepts may change and adapt to each environment. The Framework validation is underway, and a forthcoming paper will present the results.

Keywords: Service-oriented-architecture · SOA · Agile Software Development · Quality Assurance · QA · Software engineering

1 Introduction

Since the beginning of the 21st century, organizations must respond flexibly, rapidly, and dynamically to the demands of ever-changing customers, market opportunities, and external threats. This context arose with the increase in

© Springer Nature Switzerland AG 2020
O. Gervasi et al. (Eds.): ICCSA 2020, LNCS 12254, pp. 442–458, 2020.
https://doi.org/10.1007/978-3-030-58817-5_33

technology that allowed people and organizations to seek improvements that fit their needs. In this context, traditional forms of software development can present some disadvantages, such as reduced integration, delays in deliveries, and problems in sharing information among stakeholders [25]. These problems can compromise the quality of the process as a whole. Then, how to develop software with quality and rapid deliveries in an ever-changing environment?

This paper presents the NatVi Framework for Agile, Service-Oriented Architecture (SOA), and Quality Assurance (QA). It is a study for a software engineering solution to deal with the presented context, using concepts in terms of architectural design and development methods that are capable of maintaining or improving Quality Assurance (QA).

The Framework was the result of background research, including our previous work, a Systematic Literature Review (SLR) [8] that identified articles dealing with SOA and Agile and the challenges inherent in this combination of solutions. The SLR identified papers that addressed solutions for problems that arise in the presented context, and this work took the shortcomings found in those solutions and incorporated the necessary QA concept.

The Framework construction used the solutions found in the background research. It tried to take advantage of the strength and address the weakness and gaps of those solutions. It also sought to deal with the trade-offs among the central concepts identified in the studies, e.g., degree of formality, documentation, and planning.

The NatVi Framework does not merely combine agile development and SOA. It works as a single life-cycle with the necessary phases to implement services solutions in an Agile way. The NatVi Framework addresses in its four layers and 13 phases the values and principles of Agile and SOA. Likewise, it allows people to execute a well-planned and structured process that continually seeks to QA. The people involved in the development process perform specific roles in each phase, e.g., sponsor, end-user, developer, QA manager, and tester. Each phase of the Framework presents inputs, tasks, and outputs. As a Framework, it is not exhaustive, and its concepts, inputs, tasks, and outputs can change and adapt to each environment.

From the requirements' elicitation to the system monitoring, the Framework seeks to balance the principles and values of the three central concepts, Agile, SOA, and QA, putting them together in a well-constructed process. This balance sometimes seems to break some principles and values in some phases, e.g., initial phases can present more planning and documentation than in a normal Agile process, but, less documentation than an SOA process. This apparent breakdown of concepts is just enough to maintain balance, trying not to lose the essence of the concepts.

The Framework appears to be suitable for environments that experience rapid changes in business needs, reflecting the requirements of software systems. In a governmental setting, the Framework can help adapt systems to changes in the law and the interest of the citizen. In a service industry scenario, it can

help organizations find end-user satisfaction. Nevertheless, all scenarios in which business changes occur quickly can exploit the use of the Framework.

As a theoretical work, the Framework requires validation. To that end, a case study will be carried out and presented in future work. The evolution of QA in the use of the Framework will be evaluated through the monitoring of error metrics in the source code, the involvement of the development team and customer satisfaction with the new development process.

Section 2 provides an overview of Agile, SOA, and QA. Section 3 discusses some previous solutions that helped to build the NatVi Framework. Section 4 presents the NatVi Framework, and Sect. 5 presents the conclusions.

2 Background

This section presents some background about Service-Oriented Architecture (SOA), Agile Software Development, and Quality Assurance (QA). The values and principles introduced are suitable for understanding these concepts and how they are essential in the Framework, but they are not exhaustible.

2.1 Service-Oriented Architecture (SOA)

Building new functionalities and reusing existing systems, all integrated, can be an issue. This context worsens by adding the expectation of a large number of stakeholders with different goals that can change at any given moment. In this context, at the end of the 20th century, the concept of services appeared. A new way of thinking about systems architecture, breaking complex structures into small parts of functionality [4,11,24]. Service-Oriented Architecture (SOA) arose as a solution to handle this context, integrating and uniforming old and new solutions, and simplifying business processes.

SOA is an architectural pattern that presents service providers and service consumers, kept together by elements of Information Technology (IT) [4,5,11]. It aims to enhance the efficiency, agility, and productivity of an enterprise [1]. To achieve these aims, it works in eight services principles [13]: Standardized Service Contract; Service Loose Coupling; Service Abstraction; Service Reusability; Service Autonomy; Service Statelessness; Service Discoverability; and Service Composability.

SOA also can be seen as a system development process [24,29]. However, the complexity of implementing SOA presents some challenges. Its flexibility represents a complex integration of many different resources, which can exhibit a large number of critical interdependencies across many of the resources. This complexity can also bring issues to its deployment, mainly in reliability [5,30]. Stakeholder interests also may be an issue while they can demand conflicting requirements that have to be rightly managed [7]. Performance is also a concern. If the processes present an undetermined number of requests, SOA may generate bottlenecks in peaks of a request [27]. Service providers may present various issues in an SOA environment, e.g., reusability, interoperability, and scalability [6,30].

2.2 Agile Software Development

In the 1990s and early 2000s, many writers were looking for a lightweight method for software development. They wrote articles where they were concerned with problems in the software development processes that were heavy and oriented to vast documentation [7]. These concerns led to Agile Software Development that came in the form of the Agile Manifesto, published in 2001 [2]. According to [36], 'Agile methods are the solution to the problems that can be caused by traditional methodologies'.

This software development method is rapid, flexible, and stakeholders' goals-driven. The Agile manifesto [2] values are as follows: '(1) individuals and interactions over processes and tools, (2) working software over comprehensive documentation, (3) customer collaboration over contract negotiation and (4) responding to changes over following a plan'. 'That is, while there is value in the items on the right, we value the items on the left more'.

This software development method is rapid, flexible, and stakeholders' goals-driven. Agile methods value the integration and collaboration between self-organized and multifunctional teams. It also values informal communication among stakeholders. It also emphasizes tacit knowledge over explicit knowledge and promotes the teams' improvement in the software process [3].

Among the main advantages of Agile are quick responses to changes in requirements, customer satisfaction with the fast and continuous delivery of useful software [18], lower error rates, and a shorter development cycle [36]. It has to be iterative, incremental, cooperative, and adaptable to changes in environments and requirements. To achieve the necessary flexibility, Agile is formal documentation-less and uses informal communication [1,9,14].

These properties of Agile also can be seen as a disadvantage of the method regarding the bureaucratic organization, critical systems, and maintainability [31]. Another disadvantage is the difficulty of assessing the effort required at the beginning of the software development life cycle [18].

Regardless of this disadvantage, Agile Software Development makes use of good programming practices that are considered one of the pillars of quality assurance [15]. Some advocate that agile development is oriented to a higher quality compared to plan-driven development [26].

2.3 Quality Assurance (QA)

High-quality delivery is expected in any software engineering process, agile or otherwise, so Quality Assurance (QA) is itself a necessity. Software quality is one critical criterion used to measure the success of a software development project [22]. QA, for software engineering, means that the requirements need to be met, and the final product needs to be appropriate for the use [32].

QA is a process to guarantee quality in a software development process, which leads to quality in the final product, and it is also a process to evaluate the development itself [16,20,32]. There are many descriptions in the literature of QA [26] and some other concepts that share the same description, such as

Software Assurance [33], Quality Attributes [12], and Software Quality [23]. In a broad concept, QA represents the degree to which a product or process meets established requirements. This definition may refer to the ability to meet functional requirements, but it may also refer to non-functional requirements, i.e., performance, reliability, availability, portability, and maintainability [26].

Hence, Quality Assurance (QA) is at the same time a planned and systematic process to provide adequate confidence that software conforms to requirements [16] and a set of activities designed to evaluate the process by which the software is developed. As a planned and systematic process, it is possible to list from literature some activities that help ensure the quality: demonstration of the software, test-driven development, automated acceptance testing, daily builds with Testing, pair programming, coding standard, refactoring, peer review, defect analysis, defect reporting, unit testing, test automation, continuous integration, testing level, defect prevention and static analysis [17,34]. For each activity, a set of features must be defined according to each environment: guidelines, benefits, processes, best practices, templates, and customization [17].

QA depends upon the degree to which those established requirements accurately represent stakeholder needs, wants, and expectations [20]. It depends on a good plan and documentation. It depends on a good process and a proper evaluation of this process. In other words, the entire process of software engineering is committed to QA [26].

3 Related Work

While Service-Oriented Architecture (SOA) and Quality Assurance (QA) follow more traditional concepts, e.g., comprehensive documentation and up-front design, Agile Software Development strongly refutes these concepts and aims a light way of developing software, mainly valuing people, arising a trade-off. Once Agile Software Development is a trend, the papers presented in this subsection aim to find a solution to this trade-off.

Our previous work, the Systematic Literature Review (SLR) in [8], presents papers that deal with Agile and SOA, and in [15], the authors show papers that deal with Agile and QA. In the preparation of this paper, the researchers did not find an article that deals straightforwardly with the three concepts together, Agile, SOA, and QA.

[28] exhibits a Service-Oriented Framework of Interface Prototype Driven Development that deals rapidly with changes in requirements and improves the quality and efficiency of analysis and design for a data-centered application system.

[35] shows the 'SOA based Model-driven Rapid Development Architecture - SMRDA'. The authors performed a combination of SOA, Model-driven Architecture (MDA), proposed by Object Management Group (OMG), and agile methods. They concluded that this combination is the "main trend of modern software development in enterprise applications. The key of which is modeling services correctly and applying agile development techniques".

In [9], authors present a platform called FraSCAti to be used with their proposed framework, the FASOAD, a Framework for Agile Service-Oriented Architectures Development.

[10] shows the Agile and Service-Oriented Software Development Method (ASOSDeM) that aims to overcome web-based development projects' complexity by dividing it into sub-projects to allow the application of agile methods' practices. It defines how a self-organizing team should execute an SOA agile development project. It describes the concepts that may be used in an SOA project such as 'Artifacts', 'Tasks', and 'Roles'. It addresses: 'development, analysis, architecture elaboration, granularity identification, components assembling, deployment, integration tests, and business processes assembling'.

From [15], is highlighted some papers that deal with Agile and QA. [18] presents the scope of the ISO 9126 quality attributes (Correctness, Maintainability, variability, efficiency, availability, portability, testability, and reliability)through agile practices derived from XP.

In [19], the authors present Agile practices responsible for quality assurance: test-driven development, acceptance testing, code inspection, pair programming, refactoring, continuous integration, collaborative work, system metaphor, continuous feedback, and coding best practices.

[21] presents 3C approach, in which the authors achieved quality assurance from an improvement of the practice of continuous integration, continuous measurement, and continuous improvement.

[22] the author identifies some practices that promote quality assurance in the Scrum framework and continuous collaboration, integration testing, continuous feedback and development, knowledge sharing, retrospective, and daily meetings.

4 The Framework

The Background, Sect. 2, presents some paper for dealing with Agile, Service-Oriented Architecture (SOA), and Quality Assurance (QA). Nevertheless, none of the work presents a substantial solution for the entire Software Development Life-Cycle (SDLC) with these three concepts. Few of the papers deal directly with the principles of SOA and Agile. None of them present a complete process for QA. The NatVi Framework proposes to address these gaps not covered by the other solutions.

In this way, the NatVi Framework came to seek to improve Quality Assurance (QA) while dealing with Service-Oriented Architecture (SOA) and Agile during the Software Development Life-Cycle. As a framework, it is not exhaustive and presents a flexible set of phases. In this way, people responsible for a new project need to assess the phases in front of their reality and decide whether to use it. Furthermore, they can decide to customize some phase or create a new one and aggregate it to the Framework.

The NatVi Framework is composed of four layers with respective phases. The 'Stakeholder Support' is an extra layer that permeates all other layers and exists only to remember that the integration of each system stakeholders, whether

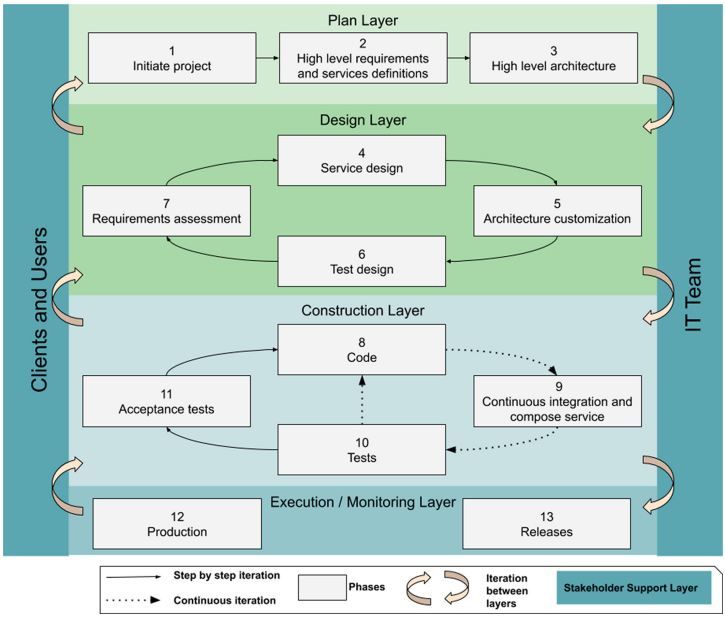

Fig. 1. NatVi Framework for Agile, SOA and QA - Figure created by the authors in the scope of this work.

clients, users and IT teams is critical to the success of the software project. The layers' position indicates they work sequentially, but the 'Iteration between layers' is indicated in the point of contact. In this way, if the development is facing problems in some layers, the work can jump virtually to any other layer to fix it (Fig. 1).

The Framework seeks to keep present the essential concepts of Agile, SOA, and QA throughout the process. From Agile, the values and principles (background Sect. 2.2). From SOA, the eight principles (background Sect. 2.1). From QA, the planning, and the process execution. Despite this, in some phases, some are highlighted, indicating that special attention is needed.

Phases of the Framework. As a framework, each iteration may need adjustments depending on the environment. Therefore, the terms used here are examples. In other environments, these terms may change and may extend. At each phase, a set of roles should be present, people who should be involved with the tasks of the phase. Each environment will be its own set of roles, defined by the environment according to their needs and organizational structure. Table 1 depicts examples of leading roles for each phase.

Table 1. Leading roles acting in each phase of the Framework.

Roles/phases	1	2	3	4	5	6	7	8	9	10	11	12
Sponsor	X										X	
Client	X	X		X		X	X				X	
End-user		X		X		X	X				X	
Business analyst	X	X	X	X		X	X					
Software designer		X	X	X	X							
Database administrator				X								
Developer						X	X	X	X	X		
Tester		X				X						
IT operator									X	X		X
QA manager			X	X	X	X	X	X	X	X	X	X

Table 2 depicts examples of a tool-set that may be present in each phase of the Framework. These tools can be electronic or not and include techniques that assist the execution, control, and documentation of tasks or the whole process.

The following subsections describe each phase of the Framework in terms of name, intention, inputs, tasks, and outputs. The items presented here are not exhaustive. For each environment, people responsible for roles may add new items and may exclude others, adapting these concepts to their reality. Inputs are all available material that can help people develop the tasks of the phase, e.g., the results of the previous phases, documents by the organization, and even outside documents. Tasks are specific actions that the roles need to do to achieve the phase objective. Outputs are the material result of the phase, e.g., report of elicited requirements, use cases, test cases, records in a control tool, developed code committed, ready system.

Table 2. Main tools used in each phase of the framework.

Tools/phases	1	2	3	4	5	6	7	8	9	10	11	12
Project management tools	X	X	X	X	X	X	X	X	X	X	X	X
Version control tool	X	X	X	X	X	X	X	X	X	X	X	X
Communication tool	X	X	X	X	X	X	X	X	X	X	X	X
Design tool		X	X	X	X	X	X					
Test tool		X				X	X	X	X	X	X	
SOA management tools			X		X		X					
SOA building tool			X		X		X					
Development tool								X		X		
Continuous integration tool									X	X		X
Documenting tool	X	X	X	X	X	X	X	X	X	X	X	X

Phase 1 - Initiate Project. *Intention:* This phase is composed of a set of tasks that allow for a general understanding of the project. At this stage, people document the origin of the demand and its main concerns. The resulting document does not need a more precise technical definition. Stakeholders and the technical team must carry out this phase in conjunction. The phase can follow a standardized protocol. Here, stakeholders can be free to expose all their thoughts and concerns without much interference from the technical team. The technical team will make technical adjustments to subsequent tasks.

In this phase, the Agile values 'Individuals and interactions' and 'Customer collaboration' should receive special attention to guarantee QA since the beginning of the project.

Inputs: Relation of business goals. The organization's strategic plan. Industry contracts and regulations. Deadlines and available budget. Relation of stakeholders who will participate in the project.

Tasks: Formalize demand in a control system. Create initial documentation. Register the project in a specific project control tool. Grant access to project information to all stakeholders.

Outputs: The formalized project. The documentation with the key characteristics recorded, e.g., people involved, schedules, budget, the main concern with business objectives, problems to be resolved, expectations about the possible new system. Possible, because the solution may be other than developing a new system. In this phase is the time to decide whether to make a whole new system, adapt the functionality of an existing system, or purchase a Commercial off-the-shelf (COTS). The formalities of this document depend on the level of formalization required by the organization. It can be a record in an information system, or it can be a record in the project management tool 'read-me' file.

Phase 2 - High-Level Requirements and Service Definition. *Intention:* In this phase, people who perform roles elicit requirements and break them down into concrete business needs, meaning that they identify the problems and concerns of the clients and end-users. The main concern of people is to identify the problem domain and not the solution to those problems. They need to identify and extract information from the business needs and interpret, analyze, validate, and gather together to compose the services. In this phase, people who perform roles identify all stakeholders and technical teams. They also initiate a more precise technical definition of the requirements. All stakeholders need to understand the goals and divide them as much as possible. Each of these parts defines a service. Thus, each service represents a specific functionality that explicitly maps to a step in a business process.

Inputs: The outputs generated in Phase 1 - Initiate project. Business process mapping, if any, or detailed description of the phases of how the work is done. Other documents that support business objectives. Some information from Phase 1 - Initiate project is vital for all next phases and should appear as input for all of them, e.g., schedule and budget.

Tasks: Elicit requirements for services. Design the initial tests. Link requirements, services, and tests.

Outputs: Relation of requirements. Record of backlogs, stories, and use cases. Draft of the architectural views of the services. Draft of the case tests.

Phase 3 - High-Level Architecture. *Intention:* In this phase, people who perform roles elaborate on high-level architecture as SOA as an architectural pattern does not have a fixed way to be implemented. In this phase, if SOA is new, the technical team should elaborate on the whole architecture. If SOA is a well-established environment, the work here will probably be simplified, and the technical team should do only a few setups.

Many of the SOA principles are planed in this phase, but two principles need more attention, 'Service Reusability' and 'Service Discoverability' (see background Sect. 2.1 about SOA). Because, if other services already achieve the goal of some new service, these other services must be discovered and reused, even with some level of adaptation.

Inputs: The outputs generated in previous phases. Information from Phase 2 - High-level requirements and Service definition, e.g., drafts of the architectural views and other infrastructure information, including information about external clouds, if they exist.

Tasks: Design the architecture for building the SOA. If SOA already exists, design the customization for the project. Decide between developing a new service or using or adapting an existing service.

Outputs: The built architecture. The documentation registered in the project management tool.

Phase 4 - Service Design. *Intention:* In this phase, people design the services (Consumer and Provider) and address the solutions to the SOA issues. This phase represents the beginning of the first loop, or iteration, in NatVi Framework. This iteration will repeat until the service design is considered good enough to go to the next iteration, the construction layer. Even if the process is already in the next layers, people can assess the requirements at any time, and the process can return to that layer.

All the eight SOA principles must be addressed in this phase (see background Sect. 2.1), but if someone needs to prioritize one, it will be 'Standardized Service Contract', because this principle can drive the final service to high-level quality, mainly in the security aspect.

Inputs: The outputs generated in previous phases, mainly the high-level requirements and the drafts of the architectural views from Phase 2 - High-level requirements and Service definition and the blueprint of the high-level architecture from Phase 3 - High-level Architecture.

Tasks: Refine elicited requirements. Identify the business objectives in their smallest parts. Design the services. Find alternative treatments if a principle cannot be achieved.

Outputs: All the necessary documentation to built the services, e.g., service design and refined requirements. Information about architecture customization needed. All documentation recorded in a specific tool.

Phase 5 - Architecture Customization. *Intention:* At this phase, people who perform roles evaluate adjustments in the architecture. Because SOA is a design pattern, not a whole architecture, people may need to do some customization in architecture, e.g., a new interface or a new term in the standardized service contract.

Inputs: The outputs generated in previous phases, mainly the documentation of service design from Phase 4 - Service design and the blueprint of the high-level architecture from Phase 3 - High-level Architecture.

Tasks: Refine the architecture. Update the architecture documentation. Implement the architectural changes.

Outputs: The built architecture and the documentation registered in the project management tool.

Phase 6 - Test Design (Test First). *Intention:* At this phase, before the Construction phase (code), people create the tests. The technical team uses information gathered in the previous phases to create these tests. Creating the test first can help developers make the code closer to service needs, that is, close the business goals.

Inputs: The outputs generated in previous phases, mainly the high-level requirements, use cases, and draft of the case tests from Phase 2 - High-level requirements and Service definition. The refined requirements and the documentation of the design services from Phase 4 - Service design.

Tasks: Refine the drafts of the case tests. Elaborate the unit tests, integration tests, system tests, and acceptance tests. Select a metric to guarantee the QA. Document and register the tests in a specific tool.

Outputs: Tests designed and registered in specific tools and tests documented in the project management tool.

Phase 7 - Requirements Assessment. *Intention:* This phase is the bottom of the first loop, the end of the first iteration and a control point. After the previous phases, the technical team has more understanding of the needs of the services. The clients, end-users, and the technical team evaluate the elicited requirements and all the done work until here. If they decide to make changes, they restart the loop. Otherwise, it is time to pass to the construction layer.

Inputs: All the inputs and outputs of the previous phases.

Tasks: Evaluate all the work done. Make decisions and document what will be done.

Outputs: A document with the assessment of the work. Record of the decision made.

Phase 8 - Code. *Intention:* This phase is the beginning of the construction layer of the framework. Although phases 8, 9, and 10 are in sequence, they are in a continuous iteration. A dotted line indicates the continuous iteration.

This phase is where the developers, technical team members, make the code itself, that is, build the service. All the artifacts generated in the previous iteration 'Analysis and design' are used in this phase as input to create the service. This 'Construction' layer phase is the first one of a new loop or iteration. This iteration will repeat until the service passes acceptance tests, made by clients and users.

Inputs: The outputs generated in previous phases, mainly the documentation from Phase 4 - Service design, e.g., service design, and refined requirements. The test design from Phase 6 - Test design (test first).

Tasks: Build the code of the services. Build the code of the unit tests. Evaluate unit tests. Daily deploy code, i.e., commit, pushing, build, and release in a development environment. Document and register the code in a specific tool.

Outputs: the source code and test code recorded in a version control tool, available to the pipeline of continuous integration tools.

Phase 9 - Continuous Integration and Compose Service. *Intention:* As already seen, the phases 8, 9, and 10 of the construction layer work almost at the same time. In this phase, people set tools that are responsible for continuous integration. These tools receive the new code, compile it, test it, e.g., unit test and integration test, build the software, and release in a development environment.

In this phase, the services are composed according to the designed architecture and the customization, if one. Composing services means putting the service in the right place into the SOA and accessible for the other services, even the external ones.

Inputs: The outputs generated in previous phases, mainly source code and the test code from Phase 8 - Code and test design from Phase 6 - Test design (test first).

Tasks: Set the continuous integration tools for build and test. Release the software in a testing/staging environment. Evaluate the tests. Give feedback to developers. Document the results.

Outputs: The services composed and released in a testing/staging environment. Testing report result.

Phase 10 - Test. *Intention:* This phase in the framework is more about giving special attention to the tests. As seen in the previous phases, the concern about tests is constant, before and after coding. So this test phase, in particular, has an academic meaning. In fact, in the real world, these three phases, Phase 8 - Code, Phase 9 - Continuous integration and compose service and Phase 10 - Test are implemented at the same time, in a constant iteration.

At these three phases, the testers make and test the services based on the tests designed in Phase 6 - Test design (test first). The developers make the unit tests, and they execute these tests in their machine to identify the errors of the first before committing to the code. The unit tests will be executed again, in Phase 9 - Continuous integration and compose service.

The testers, QA manager, and operator still make the integration and system tests in this phase. The operators' participation is fundamental because they can think in a set of services and consider the SOA. These tests will be executed only in Phase 9 - Continuous integration and compose service, where is expected a more significant number of errors than the unit tests.

Phase 11 - Acceptance Test. *Intention:* The end of the loop or iteration of the construction layer. The services are already composed and tested by clients and end-users, according to the definitions of Phase 6 - Test design (test first). These tests can be done manually or using specific tools for this.

Inputs: The software released in a testing/staging environment. The acceptance tests design in Phase 6 - Test design (test first).

Tasks: Evaluate the software against the acceptance tests. Give feedback to the technical team. Do formal approval or disapproval. Document the results and decisions.

Outputs: Testing report result. Feedback report. Formal approval or disapproval for the services to go into production.

Phase 12 - Production. *Intention:* In this phase, the services are finalized and approved. The built services are ready to release in a production environment, preferably in an automatic way.

Inputs: The source code from Phase 8 - Code. The documentation of the architecture, from Phase 3 - High-level Architecture and Phase 5 - Architecture Customization. The formal approval from Phase 11 - Acceptance test.

Tasks: Adjust the architecture. Set the production environment, Make automatic pipelines for production. Release services in the production environment. Document the activities.

Outputs: The adjusted or the new SOA ready. the services composed and ready for a production environment. The documentation up to date.

Phase 13 - Release. *Intention:* The service is released in the production environment to end-users. These two phases, 12 and 13, represented the framework's monitoring layer, where services are already in a production environment and need to be tracked to identify problems in their execution.

5 Conclusion

This work carried out research to seek a solution to better deal with developing software with quality and rapid deliveries in a software environment that tries to follow business objectives in constant change. The solution appears in the form of a Framework that deals with Agile Software Development, Service-Oriented Architecture (SOA), and Quality Assurance (QA).

The construction of the Framework made use of previous works found in background research and assessed the strengths, weaknesses, and gaps in the found solutions. The background research did not find studies that deal straight with the concepts of SOA, Agile, and QA together. However, it found studies that deal with these concepts in pairs: SOA and Agile, SOA and QA and Agile and QA. We made an effort to join those works made in pairs and fit our purpose, that is, a greater understanding of the three concepts working together.

Background research also identified trade-offs that generally arise when working with these concepts, but also in pairs, mainly the principles and values of SOA and QA versus the principles and values of Agile. These trade-offs were also a concern in the Framework construction that tried to address them.

The NatVi Framework presented in this paper is composed of four layers and 13 phases that combine in one life-cycle Agile, SOA, and QA. It concatenates each of its 13 phases through the outputs and inputs and defines specific roles and tasks for each phase. The fundamental of the Framework is continuously, in all phase, pay attention to values and principles of the three concepts.

In conclusion, the NatVi Framework is suitable for building a well-structured process to deal with the software development scenario with quality and fast deliveries in an ever-changing environment. The process created from the Framework can act on essential factors for fast delivery with quality of the final product: sufficient initial plan; adequate software architecture; business objectives divided into small pieces and directed to functional parts of the software; iterative development capable of absorbing changes instantly; code and integration tests to find errors during development; monitoring to detect problems while running the system.

However, there is a need to implement the Framework in real-world scenarios and gather empirical evidence to validate and improve on it. The authors are preparing a case study to expose the NatVi Framework to a real development environment, assessing its application, employing some measures before and after the use, and appraise the acceptance by stakeholders.

References

1. Abdelouhab, K.A., Idoughi, D., Kolski, C.: Agile & user centric SOA based service design framework applied in disaster management. In: 2014 1st International Conference on Information and Communication Technologies for Disaster Management (ICT-DM), pp. 1–8 (2014). https://doi.org/10.1109/ICT-DM.2014.6917792
2. Agile Alliance: Agile manifesto, **6**(1) (2001). http://www.agilemanifesto.org
3. de la Barra, C.L., Crawford, B., Soto, R., Misra, S., Monfroy, E.: Agile software development: it is about knowledge management and creativity. In: Murgante, B., et al. (eds.) ICCSA 2013. LNCS, vol. 7973, pp. 98–113. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39646-5_8
4. Bass, L.: Software Architecture in Practice. Pearson Education India, Delhi (2007)
5. Bianco, P., Kotermanski, R., Merson, P.F.: Evaluating a service-oriented architecture. Research Showcase @ CMU, 1 September 2007. http://repository.cmu.edu/sei
6. Bianco, P., Lewis, G.A., Merson, P., Simanta, S.: Architecting service-oriented systems. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst., August 2011
7. Carvalho, F., Azevedo, L.G.: Service agile development using XP. In: 2013 Proceedings of the IEEE 7th International Symposium on Service-Oriented System Engineering, SOSE 2013, pp. 254–259 (2013). https://doi.org/10.1109/SOSE.2013.25
8. Chaves, J.T.F., de Freitas, S.A.A.: A systematic literature review for service-oriented architecture and agile development. In: Misra, S., et al. (eds.) ICCSA 2019. LNCS, vol. 11623, pp. 120–135. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24308-1_11
9. Chehili, H., Seinturier, L., Boufaida, M.: FASOAD: a framework for agile service-oriented architectures development. In: Proceedings of the International Workshop on Database and Expert Systems Applications, DEXA, pp. 222–226 (2013). https://doi.org/10.1109/DEXA.2013.28
10. Chehili, H., Seinturier, L., Boufaida, M.: An evolutive component-based method for agile development of service oriented architectures. Int. J. Inf. Syst. Serv. Sect. (IJISSS) **9**(3), 37–57 (2017)
11. Demchak, B., Farcas, C., Farcas, E., Krüger, I.H.: The treasure map for Rich Services. In: 2007 IEEE International Conference on Information Reuse and Integration, IEEE IRI-2007, pp. 400–405 (2007). https://doi.org/10.1109/IRI.2007.4296653
12. Dutil, D., Rose, J., Suryn, W.: Software quality engineering in the new ISO standard: ISO/IEC 24748 - systems and software engineering—guide for life cycle management, pp. 89–96 (2010). https://doi.org/10.1145/1822327.1822339
13. Erl, T.: SOA: Principles of Service Design, vol. 1. Prentice Hall, Upper Saddle River (2008)
14. Farroha, D., Farroha, B.: Developing corporate services in an agile environment. In: Proceedings of the IEEE Military Communications Conference MILCOM, pp. 1535–1540 (2011). https://doi.org/10.1109/MILCOM.2011.6127525
15. Fortunato, C.A., Furtado, F., Selleri, F., de Farias Junior, I., Leitão Júnior, N.: Quality assurance in agile software development: a systematic review. In: Silva da Silva, T., Estácio, B., Kroll, J., Mantovani Fontana, R. (eds.) WBMA 2016. CCIS, vol. 680, pp. 142–148. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-55907-0_14

16. Galin, D.: Software Quality Assurance: From Theory to Implementation. Pearson Education India, Delhi (2004)
17. Hongying, G., Cheng, Y.: A customizable agile software quality assurance model. In: The 5th International Conference on New Trends in Information Science and Service Science, vol. 2, no. 60803110, pp. 382–387 (2011)
18. Hossain, A., Kashem, A., Sultana, S.: Enhancing software quality using agile techniques. IOSR J. Comput. Eng. **10**(2), 2278–661 (2013). https://doi.org/10.9790/0661-01028793. www.iosrjournals.org
19. Huo, M., Verner, J., Zhu, L., Babar, M.A.: Software quality and agile methods. In: 2004 Proceedings of the 28th Annual International Computer Software and Applications Conference, COMPSAC 2004, pp. 520–525 (2004). https://doi.org/10.1109/CMPSAC.2004.1342889
20. IEEE Computer Society: IEEE standard for software quality assurance processes. IEEE Std. 730-2014 (Revision of IEEE Std. 730–2002), pp. 1–138 (2014). https://doi.org/10.1109/IEEESTD.2014.6835311
21. Janus, A., Dumke, R., Schmietendorf, A., Jäger, J.: The 3C approach for agile quality assurance. In: Proceedings of the 2012 3rd International Workshop on Emerging Trends in Software Metrics, WETSoM 2012, pp. 9–13 (2012). https://doi.org/10.1109/WETSoM.2012.6226998
22. Khalane, T., Tanner, M.: Software quality assurance in Scrum: the need for concrete guidance on SQA strategies in meeting user expectations. In: IEEE International Conference on Adaptive Science and Technology, ICAST (2013). https://doi.org/10.1109/ICASTech.2013.6707499
23. Kitchenham, B., Pfleeger, S.L.: Software quality: the elusive target. IEEE Softw. **13**(1), 12–21 (1996). https://doi.org/10.1109/52.476281
24. Krogdahl, P., Luef, G., Steindl, C.: Service-oriented agility: an initial analysis for the use of agile methods for SOA development. In: 2005 IEEE International Conference on Services Computing (SCC 2005) Volume 1, vol. 2, pp. 93–100 (2005). https://doi.org/10.1109/SCC.2005.86
25. Kumar, G., Bhatia, P.K.: Comparative analysis of software engineering models from traditional to modern methodologies. In: 2014 Fourth International Conference on Advanced Computing Communication Technologies, pp. 189–196 (2014). https://doi.org/10.1109/ACCT.2014.73
26. Mnkandla, E., Dwolatzky, B.: Defining agile software quality assurance. In: 2006 International Conference on Software Engineering Advances (ICSEA 2006), p. 36 (2006). https://doi.org/10.1109/ICSEA.2006.261292
27. Posadas, J.V.: Application of mixed distributed software architectures for social-productive projects management in Peru. In: 2017 IEEE XXIV International Conference on Electronics, Electrical Engineering and Computing (INTERCON), pp. 1–4 (2017). https://doi.org/10.1109/INTERCON.2017.8079698
28. Rong, H., Zhou, N., Jin, M., Wu, J.: Research on service-oriented framework of interface prototype driven development. In: 2008 International Conference on Computer Science and Software Engineering, vol. 2, pp. 552–557 (2008). https://doi.org/10.1109/CSSE.2008.362
29. Shahrbanoo, M.: An approach for agile SOA development using agile principals. Int. J. Comput. Sci. Inf. Technol. **4**(1), 237–244 (2012). https://doi.org/10.5121/ijcsit.2012.4118

30. Sloane, E., Beck, R., Metzger, S.: AGSOA - agile governance for service oriented architecture (SOA) systems: a methodology to deliver 21st century military net-centric systems of systems. In: 2008 IEEE International Systems Conference Proceedings, SysCon 2008, pp. 106–109 (2008). https://doi.org/10.1109/SYSTEMS.2008.4518995
31. Sommerville, I., et al.: Software Engineering. Addison-Wesley, Boston (2007)
32. Sowunmi, O.Y., Misra, S., Fernandez-Sanz, L., Crawford, B., Soto, R.: An empirical evaluation of software quality assurance practices and challenges in a developing country: a comparison of Nigeria and Turkey. SpringerPlus **5**(1), 1–13 (2016). https://doi.org/10.1186/s40064-016-3575-5
33. Sullivan, K., Yang, J., Coppit, D., Khurshid, S., Jackson, D.: Software assurance by bounded exhaustive testing. ACM SIGSOFT Softw. Eng. Notes **29**, 133–142 (2004)
34. Timperi, O.: An overview of quality assurance practices in agile methodologies. Soberit. Hut.Fi 650 (2004)
35. Wang, B., Wen, C., Sheng, J.: A SOA based model driven rapid development architecture - SMRDA. In: ICETC 2010–2010 2nd International Conference on Education Technology and Computer, vol. 1, pp. 421–425 (2010). https://doi.org/10.1109/ICETC.2010.5529218
36. Zamudio, L., Aguilar, J.A., Tripp, C., Misra, S.: A requirements engineering techniques review in agile software development methods. In: Gervasi, O., et al. (eds.) ICCSA 2017. LNCS, vol. 10408, pp. 683–698. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62404-4_50