Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

# Service-Oriented Architecture (SOA), Agile Development Methods and Quality Assurance (QA): A case study

James Taylor Faria Chaves

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Computação Aplicada

Orientador
Prof. Dr. Sergio Antônio Andrade de Freitas

Brasília
2019

Ficha catalográfica elaborada automaticamente,
com os dados fornecidos pelo(a) autor(a)

FJ31s

Faria Chaves, James Taylor
Service-Oriented Architecture (SOA), Agile Development Methods and Quality Assurance (QA): A case study / James Taylor Faria Chaves; orientador Sergio Antônio Andrade de Freitas. -- Brasília, 2019.
81 p.

Dissertação (Mestrado - Mestrado Profissional em Computação Aplicada) -- Universidade de Brasília, 2019.

1. Service-Oriented Architecture (SOA). 2. Agile Development Methods. 3. Quality Assurance (QA). I. Andrade de Freitas, Sergio Antônio, orient. II. Título.

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

# Service-Oriented Architecture (SOA), Agile Development Methods and Quality Assurance (QA): A case study

James Taylor Faria Chaves

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Computação Aplicada

Prof. Dr. Sergio Antônio Andrade de Freitas (Orientador)
CIC/UnB

Prof.ª Dr.ª Monalessa Perini Barcellos
Departamento de Informática/UFES

Prof.ª Dr.ª Edna Dias Canedo
CIC/UnB

Prof.ª Dr.ª Aletéia Patrícia Favacho de Araújo
Coordenadora do Programa de Pós-graduação em Computação Aplicada

Brasília, 17 de julho de 2019

# Dedicatória

Dedico este trabalho ao meu pai Jesus e à minha Mãe Maria, que está no céu. À minha esposa Mara e aos meus filhos Pedro, Viviane e Natália.

# Agradecimentos

Agradeço primeiramente a Deus que me permitiu mais esta benção.

Agradeço meu pai Jesus e à minha mãe Maria, por sempre considerarem a educação uma das faces mais importantes da vida dos filhos e nunca medirem esforços para nos dar a melhor educação possível. À minha esposa Mara e aos meus filhos Pedro, Viviane e Natália, pelo apoio e paciência incondicionais.

Agradeço ao Professor Sergio, pelo orientação e principalmente pelo apoio. À Universidade de Brasília pela criação deste mestrado profissional, o que me deu a oportunidade de concretizar um antigo sonho.

Agradeço à Administração Pública Federal brasileira, instituição a qual tenho o prazer de fazer parte e que me concedeu o privilégio de mais um avanço nos meus estudos, entendendo o quão importante é para o país o investimento no conhecimento.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

# Resumo

Este trabalho propõe um framework batizado de NatVi e apresenta um estudo de caso que lidam com a interface entre Service-Oriented Architecture (SOA), Agile Development e Quality Assurance (QA). O framework NatVi busca apresentar uma solução para todo o ciclo de desenvolvimento de software, neste caso com foco em aplicações baseadas em serviços. NatVi foi resultado de uma revisão da literatura onde os 'trade off' conhecidos entre SOA e Métodos Ágeis foram identificados e as soluções possíveis avaliadas e incorporadas ao produto final. Também foram consideradas as melhores práticas baseadas tanto nos princípios de SOA quando nos princípios ágeis. Muito importante neste cenário foi não perder QA de vista, uma necessidade intrínseca aos projetos de software. Tudo isso para responder ao aumento no dinamismo dos ambientes de negócios que está aumentando a cada dia devido ao próprio dinamismo do avanço tecnológico. As organizações são chamadas a entregar valores com rapidez e confiança neste ambiente onde as possibilidades de soluções evoluem quase que diariamente. E os governos não são diferentes, obrigados a prestar mais e melhores serviços aos cidadãos e às empresas. O governo brasileiro não é uma exceção. As formas tradicionais de pensar o processo de engenharia de software vêm apresentando algumas dificuldades para lidar com este novo cenário, principalmente porque não são adequadas para lidar com constantes mudanças nos requisitos e entregas rápidas, conceitos que SOA e Métodos de Desenvolvimento Ágeis prometem ser capazes de responder. O estudo de caso foi realizado em uma pequena unidade do governo federal brasileiro. Um órgão responsável pela supervisão de um campo de interesse institucional para o país. O framework NatVi proposto foi aplicado em um ambiente onde SOA já estava em uso, apesar de insipiente. O estudo de caso avaliou a evolução da qualidade de software por meio do acompanhamento de métricas de erro no código fonte. Avaliou a evolução do entendimento sobre os métodos ágeis bem como o engajamento no processo por parte da equipe de desenvolvimento. Avaliou ainda a satisfação dos clientes com o novo processo de desenvolvimento. Durante o estudo de caso, aproveitou-se um treinamento em desenvolvimento ágil que foi ministrado pela instituição à equipe e TI e alguns clientes. Algumas limitações foram identificadas. Por exemplo, o tamanho da equipe de TI envolvida e a quantidade de clientes que participaram foi considerada pequena para

uma inferência estatística. Uma avaliação subjetiva teve que ser feita para melhorar o entendimento dos números. Desta forma, entrevistas semiestruturada foram feitas. Os resultados encontrados indicam que o caminho é promissor, mas indica também que muitos estudos ainda necessitam ser feitos, o que não é ruim, pois abre um campo vasto para pesquisas, ainda mais considerando outros ingredientes que foram identificados durante este trabalho, que podem muito bem fazer parte de estudos futuros, como containers e DevOps, por exemplo.

**Palavras-chave:** Arquitetura Orientada a Serviço (SOA), Metodologia Ágil, garantia de qualidade de software (QA)

# Abstract

This work proposes a framework named NatVi and presents a case study that deals with the interface between the Service-Oriented Architecture (SOA) with Agile Development Methods and Quality Assurance (QA). The NatVi framework seeks to present a solution for the entire software development cycle, in this case focusing on service-based applications. The framework was the result of a literature review where the known trade-offs between SOA and Agile Methods were identified and the possible solutions evaluated and incorporated into the final product. Best practices based on both the SOA principles and agile principles were also considered. It was very important in this scenario not to lose QA of view, an intrinsic need for software projects. All this to respond to the increase in the dynamism of business environments that is increasing every day due to the very dynamism of technological advancement. Organizations are called to deliver values quickly and confidently in this environment where solutions possibilities evolve almost daily. And governments are no different, obliged to provide more and better services to citizens and businesses. The Brazilian government is no exception. The traditional ways of thinking the software engineering process have presented some difficulties in dealing with this new scenario, mainly because they are not adequate to deal with constant changes in requirements and fast deliveries, concepts that SOA and Agile Development Methods promise to be able to respond. The case study was carried out in a small unit of the Brazilian federal government. A unit that is responsible for supervising a field of institutional interest to the country. The proposed NatVi framework was applied in an environment where SOA was already in use, though it was insipid. The case study evaluated the evolution of software quality through the monitoring of error metrics in the source code. It evaluated the evolution of the understanding of the agile methods as well as the engagement in the process by the development team. It also evaluated customer satisfaction with the new development process. During the case study, an agile development training was used that was given by the institution to the team and IT and some clients. Some limitations have been identified. For example, the size of the IT staff involved and the number of customers who participated was considered small for an inference statistics. A subjective assessment had to be made to improve the understanding of numbers. In this way, semi-

structured interviews were made. The results indicate that the way is promising, but it also indicates that many studies still need to be done, which is not bad, since it opens up a vast field for research, even more considering other concepts that were identified during this work, which can greatly well be part of future studies, such as containers and DevOps, for example.

**Keywords:** Software-Oriented Architecture (SOA), Agile Methods, Quality Assurance (QA)

# Contents

# List of Figures

# List of Tables

# Acronyms

**API** Application Programing Interface.

**ASD** Adaptive Software Development.

**AUP** Agile Unified process.

**BDD** Behaviour Driven Development.

**BPM** Business Process Modeling.

**BU** DoCS.

**CBD** Component-Based Development.

**CED** Code Error Density.

**COTS** Commercial off-the-shelf.

**DED** Development Error Density.

**DoD** U.S. Department of Defense.

**DSDM** Dynamic Systems Development Method.

**EA** Enterprise Architecture.

**EC** Error Code.

**ESB** Enterprise Service Bus.

**FDD** Feature-Driven Development.

**FR** NatVi.

**IMI** Intrinsic Motivation Inventory.

**ISD** Internet Speed Development.

**IT** Information Technology.

**KA** Knowledge in Agile.

**LSD** Lean Software Development.

**MDA** Model-driven Architecture.

**MTFS** Microsoft Team Foundation Server.

**OMG** Object Management Group.

**OOAD** Object-Oriented Analysis and Design.

**PP** Pragmatic Programming.

**QA** Quality Assurance.

**QoS** Quality of Service.

**REST** Representational State Transfer.

**RUP** Rational Unified Process.

**SBA** Service-Based Architecture.

**SLA** Service Level Agreement.

**SLR** systematic literature review.

**SOA** Service-Oriented Architecture.

**SOAD** Service-Oriented Analysis and Design.

**SOAP** Simple Object Access Protocol.

**SOC** Service-Oriented Computing.

**SOMA** Service-Oriented Modeling and Architecture.

**SS** Stakeholders Satisfaction.

**TDD** Test Driven Development.

**TM** Team Motivation.

**UAT** User Acceptance Testing.

**WCED** Weighted Code Error Density.

**WDED** Weighted Development Error Density.

**WSDL** Web Services Description Language.

**XP** Extreme Programming.

# Chapter 1

# Introduction

## 1.1 Context and justification

At the beginning of the 21st century, companies must respond flexibly, rapidly and dynamically to the demands of ever-changing customers, market opportunities, external threats, demands, etc. In terms of system solution to address this type of need, companies are facing new paradigms such as cloud computing and distributed software, and it is no different in the Brazilian government in this new era in which governments are called to provide more and better services online for citizens and businesses.

The technological dynamism reflects in the business world and consequently, the dynamism of the business world reflects in the world of software development. New ideas are born every day and ideas die every day. It is a lot of information to deal with. And all of this reflects in the software development process that has experienced a rapid change in business goals, customer interests, environments, and software requirements.

In this context, traditional forms of software development have some disadvantages, such as poor integration, delays in deliveries, and problems in sharing information among stakeholders. All these problems end up compromising the quality of the process as a whole.

## 1.2 The problem

Thus, in light of the above, there is a need for a software engineering solution to deal with the presented context, using modern concepts in terms of architectural design and development methods that are capable to handle all this dynamism, without losing quality and improving it, if possible.

So the main question addressed here is: **how to develop applications with quality and rapid deliveries in a distributed software environment that meet business objectives in constant change?**

To deal with the presented problem, two emerging concepts have been growing since the 1990s: Service-Oriented Architecture (SOA) and agile development methods. Both concepts promise to have the necessary resources that meet the required needs. Therefore, this work deals with a context of SOA, agile development methods, focusing on rapid deliveries, rapid changes in software requirements, and the maintenance or improvement of Quality Assurance (QA).

SOA is an architectural design pattern that aims to enhance the efficiency, agility, and productivity of an enterprise, prioritizing some values, such as business value over technical strategy, strategic goals over project-specific benefits and flexibility over optimization, among other. SOA uses standards-based infrastructure to forge large-scale systems and its components can act like service providers, service consumers or Information Technology (IT) elements that join providers and consumers. SOA has the following tenets: *Standardized service Contract*, *Loose Coupling*, *Abstraction*, *Reusability*, *Autonomy*, *Statelessness*, *Discoverability* and *Composability* [1].

Agile development methods are a lightweight method compared to traditional forms of software development. Although this concept was already known since the 1990s, its landmark is the publication of the Agile Manifesto, dated 2001. Seventeen developers and consultants representing light methods for software development gathered to discuss better ways to develop software and presented four values in the Agile Manifesto [2]: individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation and responding to changes over following a plan. From that point on, this type of development methods came to be known as Agile Methods. In summary, Agile Methods are dynamic, iterative, incremental, cooperative and adaptable to changes in environments and requirement and less documentation. There are several approaches to agile methods, each with some specificity, such as SCRUM, Extreme Programming (XP), Crystal Clear, Feature-Driven Development (FDD), Dynamic Systems Development Method (DSDM), and so on [3].

High-quality delivery is expected in any software engineering process, agile or otherwise, so Quality Assurance (QA) is itself a necessity. This is the third important concept for this work. In the literature, there are several descriptions of QA and some other concepts that share the same description, such as Software Assurance, Quality Attributes, and Software Quality. QA may refer to the ability to meet functional requirements, but it may also refer to non-functional requirements such as performance, reliability, availability, and maintainability. QA also refers to a process for ensuring quality in a software devel-

opment process and also the process for evaluating that development [4]. These and other concepts and descriptions are complementary and not excluding and can refer to different moments of a software project, development, deployment, maintenance, etc. This puts the scope of QA over a large range and it is needed to focus the approach, otherwise it will not be possible to complete this work in the time available. Thus, the concept that will be explored in this work is that, according to the IEEE definition, restricted to functional requirements and coding routines. Software quality assurance is: 1) A planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements; 2) A set of activities designed to evaluate the process by which the products are developed or manufactured [5].

Embracing change is an indispensable concept for SOA and Agile Development, so at first glance, they seem appropriate to one another. On the other hand, it can be said that SOA and Agile Development Methods do not fit together. As an architectural design pattern, SOA is complex in nature, which could be a problem for Agile Development that proposes a light process. The process of SOA can be a top-down approach, that is, the beginning is an overview of the elements that will compose the architecture, services and IT elements. The agile development method is known as a bottom-up approach, that is, to start the process, the first input is the specification, so that the process evolves. This type of trade-offs and others were evaluated in this work in order to evaluate if this set, Service-Oriented Architecture (SOA), agile methods and Quality Assurance (QA) are in fact a way of dealing with the problem of this work.

## 1.3 General and specific objectives

The main goal of this work is to present a solution able to deal with the presented problem by means of deploying Agile Development Methods in an Service-Oriented Architecture (SOA) scenario and maintain or improve Quality Assurance (QA). The specific goals are:

1. identifying the existing trade-offs between SOA and agile development methods and best practices to deal with them.
2. building a framework (principles, actions, and best practices) that enable agile development methods in a SOA scenario.
3. applying the framework in a case study, verify, evaluate and discuss the results, focusing mainly on QA.

## 1.4 Methodology

From the outline of the problem and the objectives of the work, a review of the literature was elaborated. During this phase, it was identified that Service-Oriented Architecture (SOA) and agile development methods seem to be able to handle the presented problem. But it was necessary to go deeper into the matters. Then, a literature review was done on Service-Oriented Architecture (SOA), agile methods and Quality Assurance (QA). A literature review was also done at the intersection of these three concepts, as well as on metrics that could help understand the evolution of the process. These metrics are related to software development errors, intrinsic motivation and understanding of agile development methods.

In the combined study between Service-Oriented Architecture (SOA) and agile methods, a systematic literature review (SLR) was developed following the guidelines and activities defined by Kitchenham [6]. This literature review generated a published article (Chaves and de Freitas [7]).

From the literature review, the framework proposed in this work was elaborated and it was named NatVi. In the NatVi framework were incorporated SOA principles as well as agile software development principles, with defined steps to guarantee Quality Assurance (QA) during the development process.

A case study was carried out in a small unit of the Brazilian federal government. This unit already used the concept of SOA, although in an incipient and initial way. In this environment, agile training was done for the IT team and for some clients, so that the main stakeholders were affected by the course. This course was already a prediction of the institution itself, and the present work only benefited from the process, without interfering in the application of the course. The development team itself chose to use SCRUM as the agile development method.

To evaluate the results and the possible evolution, measurements were made and, after or even during, the resulting data were tabulated, analyzed and the final considerations presented.

## 1.5 Expected results

Probably more studies need to be done, but it is expected that the results of this work will help organizations, especially in the Brazilian government, improve their services in software development.

## 1.6 Research limitations

Although this work has proposed the work with three great concepts, Service-Oriented Architecture (SOA), agile methods, and Quality Assurance (QA), it would be impossible to attempt to address all the possible problems that may arise from the study of these concepts together. Therefore, this work has some limitations.

Although the case study dealt with the three concepts when deploying the NatVi framework, it began by implementing the agile methods in an environment that already dealt with SOA, that is, the case study focused mainly on agile methods and QA.

This work did not present an initial hypothesis that the NatVi framework could improve Quality Assurance (QA). The idea was to evaluate the evolution of QA during the case study, using the Error Code (EC), Knowledge in Agile (KA), Team Motivation (TM) and Stakeholders Satisfaction (SS) indicators.

## 1.7 Structure

The structure of this work is: this introduction, chapter 1. Chapter 2 provides an overview of the state of the art in Service-Oriented Architecture (SOA), Agile Methods and Quality Assurance (QA), as well as global comparative studies among them. Chapter 3 highlights the main phases of the NatVi framework. In chapter 4 the case study is presented in some details. Chapter 5, some considerations, final conclusion, limitations and future work are taken.

# Chapter 2

# Background

This section presents some background on Service-Oriented Architecture (SOA) based applications, Agile Methods and Quality Assurance as well as comparative studies between them.

## 2.1 Service-Oriented Architecture (SOA)

Organizations have their own business goals and they need software systems to satisfy those goals, sometimes with crosscutting stakeholders concerns. One of the key challenges to building systems that meet the goals of large numbers of stakeholders is integrating and reusing existing systems, while adding new functionality to support the business processes and to respond to changes in the business. Software Architecture is a bridge between those goals and the final resulting system. Software Architecture comprehends the set of structures needed to reason about the system, which comprise of software elements, relations among them, and properties of both. In the ending of the 20th century, new approaches to system architecture have emerged which aim to structure a complex system around units of capability, called services [8–10].

This is the context in which Service-Oriented Architecture (SOA) is placed, that has emerged as a widely accepted solution to this challenge allowing a homogeneous enterprise-wide solution satisfying objectives that include an easy and flexible integration with legacy systems and simplifying business processes. Software architecture can be composed of several design patterns simultaneously to obtain its aim and SOA is a paradigm, a design pattern, not a complete architecture. This design pattern was already being used since the beginning of 1990s, mainly in Simple Object Access Protocol (SOAP) [1, 11] form. In the middle of the 2000s, the Representational State Transfer (REST) [11, 12] form began to spread and today it is hard to say which proportion news applications are made in one or other form, existing intermediary forms, between both [11]. SOA is a sort of

architecture that uses standards-based infrastructure to forge large-scale systems out of loosely coupled, inter-operable services, distributed, invocable, publishable and business oriented that can act like providers, consumers or Information Technology (IT) elements that join providers and consumers. SOA can create systems-of-systems by mapping existing systems into services, then orchestrating communication between the services. New functionality can be created by either adding new services or modifying communication among existing services with low costs, innovators services to clients, agile adaption, and reaction to opportunity and weakness competitiveness [8, 9, 13–18].

Many definitions have been given to the Service-Oriented Architecture (SOA). Some that are more connected to the technical aspects and some more connected to the business aspects. Let's continue with some definitions of bias in a business aspect, so the work aims to evolve into more technical aspects. One is a definition of SOA given by Erl [1] "SOA is an architectural model that aims to enhance the efficiency, agility, and productivity of an enterprise by positioning services as the primary means through which solution logic is represented in support of the realization of strategic goals associated with service-oriented computing". With this definition Erl [1] gives us a good view on SOA that goes beyond purely technical aspects and involves business logic. In fact, concepts such as efficiency, agility, and productivity are essential to a business process, and this definition connects business and technical processes. After all, a technical solution does not make sense if it does not solve real-world problems.

In the same way, other definitions were stated. Also from Erl [1], service-orientation is an implementation of the distributed logic solution paradigm, which aims to split a big problem into small parts, adhering to concern separation theory. According to Carvalho and Azevedo [19] "the greatest benefit of service-orientation is to produce small solutions that solve each small part of the original problem, while being agnostic to the bigger problem to be solved. This split of problems into smaller pieces fosters reuse". SOA as an architectural style for developing and integrating enterprise applications stresses that business must be able to respond to the market by building appropriate business services [20].

Towards more technical aspects, SOA can be understood like an architectural style and uses services as building blocks to embrace changes in the business environment by composing services and creating composite services already existing with applications in a technology heterogeneous environment [20, 21]. The business and technical processes are implemented as services and each service represents a particular functionality that maps explicitly to a step in a business process [22]. In this way, service can be understood as any task or function provided by an organization, aligned to business, well defined and isolated from other tasks (autonomy principle) [1] and Service orientation helps organizations con-

sistently deliver sustainable business value, with increased agility and cost effectiveness, in line with changing business needs [23]. In general, the basic Service Consumer software is simply a browser or browser-like "thin client", with little or no application software stored locally [21]. Figure 2.1 presents an overview of SOA[1].

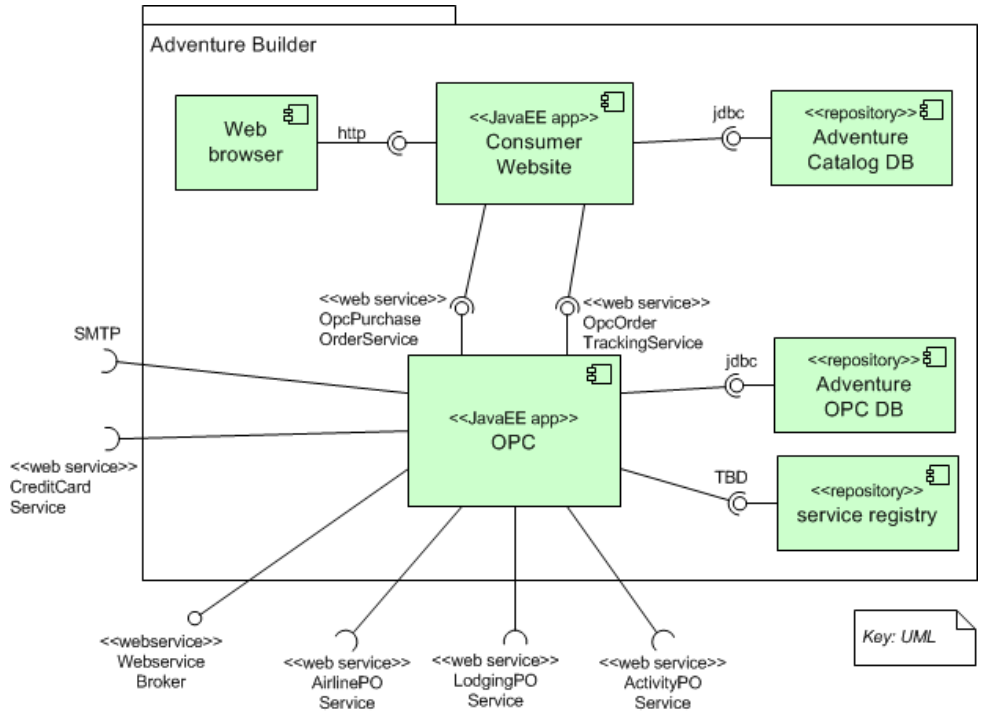


Figure 2.1: Top Level SOA View

This approach still enables interoperability because the services consumer and providers may be executed in different platforms, offering the necessary elements to interact with external elements, available on the internet. Service orientation helps organizations consistently deliver sustainable business value, with increased agility and cost effectiveness, in line with changing business needs. SOA-based systems are ideally suited for rapid reconfiguration because the Service Consumer - Service Broker - Service Provider architecture is inherently designed to facilitate on-the-fly changes [21, 23, 24].

According to Bianco and Lewis [25], "how the applications are distributed, exist the chance to rebuilt and solution that already exist in the organization, mainly in medium and big organizations. To prevent this kind of issues, all services have to be their own meta-data published in a central register service, enabling the discovery of services already built".

As a design pattern, SOA can work with other components, like Enterprise Service Bus (ESB) [25, 26] or a Directory Service [13]. The connectors can include synchrony

[1]https://wiki.sei.cmu.edu/confluence/display/SAD/Top+Level+SOA+View

and asynchrony calls, and can use diverse technology, like Simple Object Access Protocol (SOAP) or Representational State Transfer (REST), or a message infrastructure [27, 28].

In Erl [1] it is possible to see eight principles that a service-oriented solution should follow:

1. Standardized Service Contract: a service should express its purpose and capacities through a contract;
2. Service Loose Coupling: the dependencies among the service's contract, implementation and consumers should be minimal;
3. Service Abstraction: a service should hide its implementation details to keep its consumers loosely coupled;
4. Service Reusability: services should be corporate resources, agnostic to functional contexts;
5. Service Autonomy: a service should have control over its environment and resources, while remaining isolated from other services;
6. Service Statelessness: services should only hold state information when necessary in order to not compromise its availability or scalability;
7. Service Discoverability: services should be easily discovered and understood to foster reuse;
8. Service Composability: it should be possible to create services from the composition of other services in order to produce sophisticated solutions according to business needs requirements and fostering reuse of existing assets

According to Arsanjani et al. [23] Service-Oriented Architecture (SOA) is a type of architecture that results from applying service orientation and six values have to be prioritized:

- **Business** value over technical strategy
- **Strategic goals** over project-specific benefits
- **Intrinsic interoperability** over custom integration
- **Shared services** over specific-purpose implementations
- **Flexibility** over optimization
- **Evolutionary refinement** over pursuit of initial perfection

However, just because those good concepts showed until now, there are some challenges to cope relating to SOA. Good assets like Loose Coupling, Interoperability, Composability, Flexibility, and Abstraction can lead SOA to establish a complex interaction dependencies in favor of simplicity of structural decomposition over orchestration simplicity, hence the crosscutting orchestration concerns have to be reconciled [9]. The flexibility of SOA also represents a complex integration of a large number of disparate resources which can

exhibit a large number of critical interdependencies across many of the resources. All this complexity in SOA can also bring issues to its deploying, mainly in reliability [17, 21].

Stakeholder concerns also may be an issue to SOA. They can demand conflicting requirements which have to be handled properly. Aligning business requirements with IT solutions itself can raise some problems [19]. Performance is also a concern. If the processes present an undetermined number of request, SOA may generate bottlenecks in peaks of request [29].

Service providers may present various issues in a SOA environment. As a solution for a distributed system, problems among different providers that handle the solution can be an issue. Keeping only one provider could be a solution but it could sacrifice the reusability and interoperability [24, 25]. Providers may not reliably forecast service demand levels, which can make the investment and system scalability planning quite uncertain. Also, prior service provider versions may be discontinued, for various factors. Degradation of communication or other IT server systems offered by a provider may disrupt or stop most SOA systems [21].

Perhaps for all these issues and more, SOA is not a consensus if it is the emerging and recognized architectural style for architecting adaptive systems across the enterprise. Some doubts raised can be seen in Krogdahl et al. [10] and KrÃ¡l [30].

Besides the vision of SOA as an architectural design pattern, it can be seen as a system development process. In this way, SOA is a development approach decomposing all in services (software components). It addresses a specific need within reusability, simplicity and interoperability aims, and can hide the heterogeneity of the underlying information system [31]. Service-oriented applications are usually developed in an incremental fashion by building reusable services that may inter-operate with each other, but where a whole system definition including all the functionalities is missing [32]. Gu and Lago [33] state that service-oriented systems are developed differently from traditional ones. There are greater concerns with reuse-oriented software construction, stakeholder involvement, deep understanding of the business model, service distribution across enterprise boundaries, and business-IT alignment [19].

SOA development methodologies are not mature, although the goal is clear, the way to reach the goal is not yet clear since there is not an approach which is broadly accepted [10, 20]. But, in a systematic literature review (SLR) carried out by Lane and Richardson [34] some SOA models development are categorized according to meta-model development process that covers the following phases: Analysis and Design, Construction and Testing, Deployment and Provisioning and, Execution and Monitoring. According to Lane and Richardson [34] the most of the studies encountered lie down in phases Analysis and Design, Construction and End to End models, that is, cover all phases of meta-model

development process.

However, the widely adopted software design principle encountered lies down over Analyses an Design phase also known as Service-Oriented Analysis and Design (SOAD) that is an extension of the Object-Oriented Analysis and Design (OOAD) [35] and Component-Based Development (CBD) [36] paradigms [34, 37]. SOAD is a specially designed method of software modeling, analyzing and designing [38] and represents static and dynamic aspects of service-oriented software systems. These models that follow SOAD are partitioned according to the SOA Models. The SOAD models are generic in the sense of the generality supported by SOA [39] and it is defined in terms of the basic SOA elements (subscriber, publisher, and broker). SOAD largely focuses on defining processes to develop business processes and services without considering user-aware variability [40].

As a generic model, SOAD has different approaches that apply it to service design [34] but, according to Shahrbanoo [20] the most famous of these approaches is the approach proposed by Erl [41].

According to Shahrbanoo [20] Service-Oriented Modeling and Architecture (SOMA) [42] and Zimmerman's methodology [43] are instances of SOAD methodology.

Service-Oriented Modeling and Architecture (SOMA) [42] is a widely adopted SOAD methodology that is based on the identification, specification and realization of service components entirely based on the Rational Unified Process (RUP) framework [37]. It is a software development life-cycle method invented and initially developed by IBM for designing and building SOA-based solutions [42]. The SOMA methodology is essentially used for producing a service model artifact in the Elaboration phase through the identification, specification, realization and deployment of services. According to Arsanjani et al. [42] the SOMA method includes seven major phases but, no rigid sequencing is implied:

- Business modeling and transformation, where the business is modeled, simulated, and optimized;
- Identification phase, where is made the identification of the three fundamental constructs of SOA: services, components, and flows;
- Specification of services, components, flows and information;
- Implementation: build and assembly (Construction, generation, assembly) and, integration Testing (Unit, integration, User Acceptance Testing (UAT))
- Deployment, monitoring, and management: focus on packaging, provisioning, executing UAT, and deployment of services in the production environment.;
- Solution management: Hybrid solution type selection, method adoption and integration, project management, enterprise architecture
- Realization: Decisions, solutions templates and patterns, SOA reference architecture, technical feasibility prototyping

In Zimmermann et al. [43] the authors investigate suitable elements from Object-Oriented Analysis and Design (OOAD), Enterprise Architecture (EA), and Business Process Modeling (BPM) to SOA development. They conclude that this approach only cover part of the requirements needed to support the SOA paradigm. While the SOA approach reinforces well-established, general software architecture principles such as information hiding, modularization, and separation of concerns, it also adds additional themes such as service choreography, service repositories, and the service bus middle-ware pattern, which require explicit attention during modeling. The approach comprises elements from OOAD, BPM [44], and EA [45] in a best-of-breed fashion, and complements them with certain innovative elements. The new elements added in this approach are:

- Service categorization and aggregation: Services have different uses and purposes and atomic services can be orchestrated (composed) into higher level, full-fledged services;
- Policies and aspects: A service has syntax, semantics, and Quality of Service (QoS) characteristics that all have to be modeled; formal interface contracts have to cover more than the Web Services Description Language (WSDL) does.
- Meet-in-the-middle processes: As SOA leads with legacy applications, it always have to be taken into account. Therefore, a meet-in-the-middle approach is required, rather than pure, top-down or bottom-up process.
- Semantic brokering: The semantics issue (the meaning of parameters and so forth) has to be solved as well (domain modeling). SOA needs to take into account to be flexible and agile in response to the new business needs in a world of mergers and acquisitions, business transformation, globalization, and so forth.
- Service harvesting and knowledge brokering: Services should be identified and defined with reuse (and harvesting) as one of the main driving criteria of the SOA in mind. If a component (or service) has no potential for reuse, then it should probably not be deployed as a service. It can be connected to another service associated with the enterprise architecture, but will not be a service in its own right.

## 2.2 Agile Methods

A light method for software development can be understood as a method that the needs and goals of the stakeholders drive the development and enable the rapid and flexible development of software. The processes and tools itself are less important than deliverables, people, developers and stakeholders, communication, and the ability to adapt to

change. This type of method has to be iterative, incremental, cooperative and adaptable to changes in environments and requirements [16, 31, 46].

According to Carvalho and Azevedo [19] "during the 2000-year, many articles were published highlighting problems related to the documentation driven development and heavyweight processes. Authors advocate light methods for software development". But the search for light software development process began before, in the 1990s. As cited by Highsmith and Cockburn [47], in 1995 Goldman et al. [48] already offered a definition of agility which applies equally to today's software development: "Agility is dynamic, context-specific, aggressively change embracing, and growth-oriented. It is not about improving efficiency, cutting costs, or battening down the business hatches to ride out fearsome competitive 'storms'. It is about succeeding and about winning: about succeeding in emerging competitive arenas, and about winning profits, market share, and customers in the very center of the competitive storms many companies now fear".

The search for this type of software development method led to the "agile methods" that were stated in the Agile Manifesto, published in 2001 [2] where seventeen developers and consultants representatives of light methods for software development met to discuss better ways of developing software. The Agile manifesto consists of four values and twelve principles. The Agile manifesto four 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" [2].

The Agile manifesto twelve principles are as follows [2]:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity–the art of maximizing the amount of work not done–is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

In the reviewed literature we have found several approaches that claim to be agile development methods such as SCRUM, Rational Unified Process (RUP), Dynamic Systems Development Method (DSDM), Adaptive Software Development (ASD), Internet Speed Development (ISD), Lean, Crystal Clear, Extreme Programming (XP), Feature-Driven Development (FDD), Pragmatic Programming (PP), and so on [3, 19, 47, 49–51]. According to Malik et al. [52] the most commonly used are: FDD, XP, DSDM and SCRUM.

The following are some characteristics of each of the main methodologies mentioned, according to Timperi [51]:

- Feature-Driven Development (FDD) is based on several best practices and it emphasizes design and building activities. The requirements are captured first by constructing a domain object model and using it as a basis for requirements elicitation and inspections are the main quality assurance practice but testing is also mandatory.
- Extreme Programming (XP) is based on short iterations and incremental development with constant feedback from both the customer and other developers and most of the practices of XP are aimed at quality assurance and in particular getting timely feedback.
- Dynamic Systems Development Method (DSDM) relies heavily on prototyping in most development activities and proposes a pragmatic view to quality; the emphasis is on early validation while technical quality can be sacrificed.
- SCRUM is based on empirical management and it does not state engineering practices and consists of development in iterations called sprints. The requirements are captured in prioritized order in a product backlog and in a sprint backlog for the current sprint. Daily management is handled by scrum meetings in which participants answer three questions regarding what they have done since the last scrum meeting, what they will do between now and the next scrum meeting, and what problems they have.

## 2.3 Quality Assurance (QA)

Software quality is one critical criteria used to measure success of a software development project [53]. High-quality delivery is expected in any software engineering process, agile

or otherwise, so Quality Assurance (QA) is itself a necessity. In the literature, there are several descriptions of QA and some other concepts that share the same description, such as Software Assurance [54], Quality Attributes [55], and Software Quality [56].

Quality according to ISO 9000 is defined as "the totality of characteristics of an entity that bear on its ability to satisfy stated or implied needs."

Software quality according to IEEE is [4]:

1. The degree to which a system, component, or process meets specified requirements.
2. The degree to which a system, component, or process meets customer or user needs or expectations.

QA may refer to the ability to meet functional requirements, but it may also refer to non-functional requirements such as performance, reliability, availability, and maintainability. QA also refers to a process for ensuring quality in a software development process and also the process for evaluating that development. These and other concepts and descriptions are complementary and not excluding and can refer to different moments of a software project, development, deployment, maintenance, etc. This puts the scope of QA over a large range and it is needed to focus the approach otherwise, it will not be possible to complete this work in the time available.

Thus, the concept that will be explored in this work is that, according to the IEEE definition, restricted to functional requirements and coding routines. Software quality assurance is: 1) A planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements; 2) A set of activities designed to evaluate the process by which the products are developed or manufactured.

Quality Assurance (QA) is at the same time a planned and systematic process to provide adequate confidence that software conforms to requirements 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 to ensure the quality, like *demonstration of 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* [51, 57]. For each activity a set of features need to be defined according to each environment: guidelines, benefits, processes, best practices, templates and customization [57].

## 2.4 Related Works

Unlike the previous sections where the basics concepts of Service-Oriented Architecture (SOA), Agile Methods and Quality Assurance (QA) were presented, the intersections of these concepts will be presented in the subsections of this section. This work is more complex than just presented basics concepts, thus it was used three different approaches.

To compare SOA with Agile Methods, this researcher did a systematic literature review (SLR), called 'A Systematic Literature Review for Service-oriented Architecture and Agile Development' [7]. To compare Agile Methods with QA it was used an SLR, which had already been done by Fortunato et al. [58]. To compare SOA with QA it was made a traditional literature review in some related works.

### 2.4.1 Comparative study between Service-Oriented Architecture development and Agile Methods

Since embracing changes is the indispensable concept of SOA, it is seemed using agile methodology is natural fit to develop SOA-based systems. In this area, much debate has been done and despite consensus in the usefulness of an agile SOA methodology, there has not been considerable work in this area [20]. Although what was stated for Shahrbanoo [20], at the beginning of this 21st century, several authors reported studies involving the use of agile methods in Service-Oriented Architecture (SOA).

Inside the scope of this work, it was done a systematic literature review (SLR) entitled 'A Systematic Literature Review for Service-oriented Architecture and Agile Development' and aims to answer the main question 'What Agile solutions are proposed for developing Service-Oriented Architecture (SOA) based Applications?'. It was done a search string with the main synonyms of 'Agile Developmet' and SOA, table 2.1 and relevant electronic data sources were consulted, table 2.2.

| Agile Development | Service-oriented Architecture |
|---|---|
| Agile Method | SOA |
| Agile Programming | Service-Based Architecture (SBA) |
| Agile Process | Service-Oriented Computing (SOC) |
| Agile Practice | Webservice |
| Agile Requirement | Representational State Transfer (REST) |
| Agile Technique | Simple Object Access Protocol (SOAP) |
| | Service-based |

Table 2.1: Search synonyms

The data extraction occurred during August 2017 and it has found 18 studies that have proposed some type of solution to deal with Service-Oriented Architecture (SOA)

| Source | URL |
|---|---|
| IEEE | https://ieeexplore.ieee.org |
| ACM | https://dl.acm.org/ |
| Web of Science | https://www.isiknowledge.com |
| Springer | https://link.springer.com/ |
| Science Direct | https://www.sciencedirect.com |
| Scopus | https://www.scopus.com |

Table 2.2: Data sources

and Agile Development Methods. No filter per year was used and the selected studies are distributed between the years 2003 and 2017, and the peak is between the years 2010 and 2013, 9 studies. Between the years 2014 and 2016, only 3 studies and in the year 2017, 2 studies. This distribution of publication trend is following the Gartner Hype-Cycle [2]. It seems that today the subject is moving from the 'Slope of Enlightenment' to the 'Plateau of Productivity'. It is a sign of maturity of the subject. In this section, an overview of these studies is presented.

Table 2.3 shows the selected studies by type of source and the of solution they proposed. Some authors have explored SOA and Agile to address issues related to business objectives and rapid changes in requirements, in the environment, and even the customer, as an alternative to traditional monolithic systems [9, 16, 59–61]. But, other authors only addressed the need to deal with the SOA and Agile itself. They assumed that the problem itself would be the combination of these two concepts. They did not explain the need to use them. [10, 19, 20, 46, 62–65].

Although some studies have presented artifacts or experiments, none of them presented results or data capable of allowing any type of validation for the readers. The general conclusion of the authors is that the proposed solutions are able to solve the initial problems presented by them. In Rong et al. [61], for example, the authors presented a Service-Oriented Framework of Interface Prototype Driven Development. They did a comparison between two similar projects developed by the same team. Project 'A' used the proposed framework, meanwhile the project 'B' did not. They concluded the project that used the proposed framework supports an instant response to requirements changes and improves the quality and efficiency of analysis and design for data-centered application system, but, no data was presented that allow the readers to evaluate the results stated by the author.

Wang et al. [62] presented the 'SOA based Model-driven Rapid Development Architecture - SMRDA'. The authors presented a combination of SOA, Model-driven Architecture (MDA), proposed by Object Management Group (OMG) and agile methods. They ap-

[2]https://en.wikipedia.org/wiki/Hype_cycle

| Study | Source | Year | Solution |
|---|---|---|---|
| [31] | CP | 2014 | Framework/Model |
| [19] | SP | 2013 | Approach/Other |
| [59] | CP | 2012 | Approach/Framework |
| [16] | WP | 2013 | Framework |
| [66] | JP | 2017 | Model |
| [60] | JP | 2010 | Process |
| [9] | CP | 2007 | Model/Process |
| [46] | CP | 2011 | Framework |
| [65] | JP | 2003 | Approach |
| [63] | JP | 2008 | Approach |
| [10] | CP | 2005 | Other |
| [64] | CP | 2013 | Process |
| [61] | CP | 2008 | Framework |
| [67] | CP | 2010 | Model |
| [20] | JP | 2012 | approach |
| [68] | CP | 2017 | Approach |
| [62] | CP | 2010 | Model |
| [69] | SP | 2016 | Other |

CP - Conference Paper / JP - Journal Paper
SP - Symposium Paper / WP - Workshop Paper

Table 2.3: Categorical data extracted from selected studies

plied the SMRDA in a software education management platform. They concluded that the combination of SOA and MDA approaches is the main trend of modem software development in enterprise applications. The key of which, is modeling services correctly, and applying agile development technique.

Karam et al. [63] presented a tool called 'VisualWebC' (short for Visual Web Composition) to support the visual composition of Web services. The authors concluded that their approach presents several significant advantages, such as the creation of web applications with complex functionality with relatively little effort and time.

Chehili et al. [16] presented a platform called FraSCAti to be used with their proposed framework, the FASOAD, a Framework for Agile Service-Oriented Architectures Development. This FraSCAti platform was presented by the authors in an previous study. The authors also did a case study in a High Educational domain.

In Christou et al. [60] the authors presented an example using the Agile Unified process (AUP) in a process of a bank.

Abdelouhab et al. [31] concluded that a major benefit of the Agile-UCD-SOA (Agile & user centric SOA) framework is that it leads to highly flexible and agile software that should be able to meet rapidly changing business needs.

Chehili et al. [59] concluded that their approach, with successive division of the project,

the customer involvement, and the acceptance test parts of the architecture can meet the terms of the manifesto of agile methods.

Shahrbanoo [20] concluded that their approach is not only practical but also viable and valuable to develop agile architecture in an agile way since it is an easy approach to apply and also emphasizes on customer involvement. All the authors concluded by the success of their proposed solutions, but there is no way to evaluate the results.

Although Chehili et al. [66] did not present a way to validate their results, they presented a mature work, perhaps because it is a more recent work. ASOSDeM (Agile and Service Oriented Software Development Method) aims to overcome the complexity of web-based development projects by dividing it into sub-projects to allow the application of agile methods' practices. Defines how an SOA agile development project should be executed by a self-organizing team. Describes the concepts that may be used in an SOA project such as 'Artifacts', 'Tasks' and 'Roles'. Address development, analysis, architecture elaboration, granularity identification, components assembling, deployment and integration tests, and business processes assembling.

Many of the solutions are linked to the general concepts of agile methods. Some have borrowed specific methods used in methodologies that claim to be agile, such as Extreme Programming (XP), Lean Software Development (LSD), and Rational Unified Process (RUP).

Although some studies have adopted well-known development process tactics, few studies have addressed specific phases of agile development. Abdelouhab et al. [31] addressed design, implementation and deployment phases. Rao et al. [64] addressed exploration, planning, iterations to release, productionizing, maintenance, and death. Carvalho and Azevedo [19] construction phase. Christou et al. [60] inception, elaboration, construction, and transition phases. These results seems to corroborate the idea that defining the life-cycle in agile development methods is more difficult than in traditional development processes. Being agile is respecting other things besides processes and tools itself.

Like agile phases, few studies addressed specifics SOA principles. Zúñiga-Prieto et al. [69] addressed service loose coupling and service composability. Roy and Debnath [67] addressed standardized service contract, service loose coupling, and service abstraction. Demchak et al. [9] addressed service loose coupling and service composability . The only one study that addressed all 8 SOA principles was Carvalho and Azevedo [19], standardized service contract, service loose coupling, service abstraction, service reusability, service autonomy, service statelessnes, service discoverability, and service Composability .

Service-Oriented Architecture (SOA) presents a high level of complexity and what had been observed is that few authors have entered into SOA's core. Maybe they should have put more emphasis on the details, more attention to SOA principles.

### 2.4.2 Comparative study between Service-Oriented Architecture development and Quality Assurance

Quality Assurance (QA) in terms of Service-Oriented Architecture (SOA) is mainly treated in literature as concerns about quality attributes or non-functional requirements [70–74], such as performance, interoperability, usability, maintainability, reliability, portability, and security. According to Karthikeyan and Geetha [75] SOA-related quality attributes are divided into three categories: quality attributes of design, quality attributes of the main SOA characteristic, and quality attributes of business. Some quality attributes of design are reusability, modifiability, understandability, effectiveness. Some quality attributes of SOA characteristic are reusable, extensible, and interoperable [70].

SOA implies drawbacks in many cases because of its complexity and according to Voelz and Goeb [76], SOA compared with traditional software products does call for a differentiated picture and it has a general impact on quality modeling rather than affecting a single quality attribute or quality assurance technique. The main approaches to assure quality are testing [76–80], monitoring [71, 81] and static analysis [82–84].

In the literature reviewed, most authors treat systems based on SOA primarily for the perspective of services that only specifications are available, but no design or code is available to the application builder and the intermediary [78, 85]. Although many authors have been cited in cases where the code is available, they have not done a thorough evaluation, relegating this subject to a testing of traditional systems. In some cases where only the specifications are available, testing SOA is equivalent to the black box test [86], but while testing remains the predominant quality assurance technique [76], traditional software testing practice is often not sufficiently addressing and previous paradigms must be modified to cope with the new SOA characteristics, with numerous runtime behaviors and multiple standard protocols [79, 85, 87]. Therefore, the test needs to be adaptable to changes in service-oriented applications at runtime and must also be achieved dynamically and automatically, rather than in the traditional human-intensive approach [80].

According Tsai et al. [84] SOA testing techniques focus on the composition and the interoperability among services, including functionality testing, interface testing, message testing, stress testing, and performance testing and they aim to guarantee desired properties, such as deadlock avoidance, bounds on resource usage, and response times. Still according to Tsai et al. [84], some tests in service-oriented may be: in collaboration among WS providers, clients, and independent service brokers; distributed, remote, multi-agent and multi-phase; on-line just-in-time in an application environment; on-line regression using data dynamically collected; dynamic configuration and systems are linked at runtime and verified at runtime;

According to Golshan and Barforoush [88], "as SOA environments become more dynamic and unpredictable, the importance of monitoring and tracing quality becomes more obvious". But, monitoring SOA can become a fairly complex activity, it could in itself become a resource intensive activity [71] and can be expensive, as a number of services from different providers need to be tested [89]. Many authors treat the monitoring as a problem relating an Service Level Agreement (SLA), negotiated between service providers and consumers, where service provider have to guarantees quality of the services and They argue that the SLA plays a key role in service architectures [88, 90]. Besides monitoring, SLA deals with negotiation, provisioning, scheduling and strategies [71].

Unlike the test approach, static analysis is used to extract the static structure from the source code, components, interfaces, and connectors [91]. Static code analysis or white box [86] testing allows the code to be analyzed without executing it [92]. According to Masood and Java [92], some vulnerability that static code analysis help to prevent in SOA are: SQL Injection, Recursive and Oversize Payload Attacks; JSON (Java Script Objection Notation) Hijacking; Action Tampering / Spoofin; WS-Security Attack Obfuscation and External Entity At- tacks (XXE); Coercive Parsing and Oversized Cryptography; BPEL State Deviation and Flooding Attacks; and Middleware Hijacking and Session Management Middleware.

### 2.4.3 Comparative study between Agile Methods and Quality Assurance

Fortunato et al. [93] made a systematic literature review (SLR) of Agile Development and Quality Assurance (QA) and they discovered the key practices that Agile Development enables QA: Refactoring, Driven Development, Pair Programming, Inspection, System Metaphor, Joint Planning Meeting. The best practices encountered are knowledge sharing meetings, simple design, continuous integration, retrospective meetings, stand-up meetings, Sprint Review and Planning.

Refactoring is a programming technique by changing the code in smaller chunks in a disciplined manner to improve its design, ensuring that code is always kept simple, making it easier to work in order to make code maintainable and database design more flexible, improving the quality of the code without affecting the external functionality [94–96].

According to Timperi [51] "Test-driven development is a practice in which unit tests are written before the source code and ran directly after the implementation is complete. Test-driven development forces the source code to be testable and guarantees that unit tests are written"

Pair programming means two programmers continuously working on the same code, sitting at a single computer. It can also improve design quality, reduce defects, share each others knowledge and identify their mistakes. The two programmers can act as a driver or a partner. The responsibility of the driver is to write the code and to focus on the current matters at hand. The responsibility of the partner is to check that the code written by the driver is correct and to think ahead [51, 96, 97]. According to Upadhyay et al. [98] "pair programming in agile environment results show that short term productivity might decrease but due to the high quality produced code the resulting long term productivity and quality goes high"

According to Timperi [51] "inspections are the main quality assurance practice". An inspection is a formal evaluation technique used for achieving both internal quality of the source code as well as promoting knowledge transfer and learning. The main purpose is to verify that the system meets the agreed requirements. Software requirements, design, or code are examined in detail by a person or group other than the author to detect faults, violations of development standards, and other problems.

System metaphor presents a simple shared story of how the system works instead of using a formal architecture, and provides the basic understanding of system architecture. It reduces the possibility of system failure if development work carried out according to the architecture. System metaphor contributes to the team's development of a software architecture [96, 97, 99].

According to Timperi [51], "joint planning meeting is a requirements gathering practice used in Crystal Clear, XP, and Scrum. In a joint planning meeting, customers and developers come together to discuss requirements, ask questions, and confirm that people understand the requirements in a similar way".

## 2.5 Metrics

In this paper, four types of metrics were used. In the 4.4 section, the metrics used are presented in more detail. But in summary, these four metrics are related to three concepts, quality metrics in software development, intrinsic motivation and understanding of agile development methods. Each of these three concepts is presented below.

### Quality metrics

According to Galin [4] the IEEE definition of quality metrics in the software industry refers to a process and an outcome for that process:

1. The outcome: A quantitative metric of the degree to which an item possesses a given quality attribute.

2. The process: A function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which the software possesses a given quality attribute.

It is commonly believed that quality metrics should be included in the software industry in three basic areas, control of software development projects (process metrics) and software maintenance (product metrics), support of decision-making, and initiation of corrective actions, although it has not been applied at an adequate level [4]. Still according to Galin [4] the main objectives of software metrics are: "To facilitate management control as well as planning and execution of the appropriate managerial interventions" and "To identify situations that require or enable development or maintenance process improvement in the form of preventive or corrective actions introduced throughout the organization".

Software quality metrics can still be classified according to the measurement subject [4], where each classification will assign a category of the process metrics:

- Quality
  - Error density metrics (table 2.4);
  - Error severity metrics (table 2.5);
- Timetable
- Effectiveness
- Productivity

| Code | Name | Calculation formula |
|---|---|---|
| CED | Code Error Density | $CED = \frac{NCE}{KLOC}$ |
| DED | Development Error Density | $DED = \frac{NDE}{KLOC}$ |
| WCED | Weighted Code Error Density | $WCED = \frac{WCE}{KLOC}$ |
| WDED | Weighted Development Error Density | $WDED = \frac{WDE}{KLOC}$ |
| WCEF | Weighted Code Errors per Function point | $WCEF = \frac{WCE}{NFP}$ |
| WDEF | Weighted Development Errors per Function point | $WDEF = \frac{WDE}{NFP}$ |

Table 2.4: Error density metrics

Key:

- NCE = number of code errors detected in the software code by code inspections and testing. Data for this metric are culled from code inspection and testing reports.

| Code | Name | Calculation formula |
|---|---|---|
| ASCE | Average Severity of Code Errors | $ASCE = \frac{WCE}{NCE}$ |
| ASDE | Average Severity of Development Errors | $ASDE = \frac{WDE}{NDE}$ |

Table 2.5: Error severity metrics

- KLOC = thousands of lines of code.
- NDE = total number of development (design and code) errors detected in the software development process. Data for this metric are found in the various design and code reviews and testing reports conducted.
- WCE= weighted code errors detected. The sources of data for this metric are the same as those for NCE.
- WDE = total weighted development (design and code) errors detected in development of the software. The sources of data for this metric are the same as those for NDE.
- NFP = number of function points required for the development of the software. Sources for the number of function points are professional surveys of the relevant software.

### Intrinsic Motivation

According to Csikszentmihalyi and Rathunde [100], cited by Ryan and Deci [101], "The construct of intrinsic motivation describes this natural inclination toward assimilation, mastery, spontaneous interest, and exploration that is so essential to cognitive and social development and that represents a principal source of enjoyment and vitality throughout life".

In this work Intrinsic Motivation Inventory (IMI) was used to evaluate participants' subjective experience related to the target activity in a controlled experiment.

The IMI instrument allow the assessment of participants interest/enjoyment, perceived competence, effort/importance, pressure and tension, perceived choice and value/usefulness while performing a given activity, thus yielding a sub-scale score. Here, this sub-scale consisted of seven statements that had scored on a 7-point Likert scale [102] where 1 represents 'Absolutely disagree', 4 represents 'Indifferent' and 7 represents 'Completely agree'.

In some parts of this work, the concepts of the agile development motivator found in systematic literature review (SLR), 'Developers motivation in agile teams', by De O. Melo et al. [103], were used. In that work, in addition to showing the motivators and demotivators identified in the SLR, the authors also carried out three case studies in which they

identified more motivators. According to the authors, some factors could be motivating and demotivating at the same time. Table 2.6 shows this motivators and demotivators found by the authors in literature. Following, the motivators found by the authors in the case studies:

1. Technically challenging work;
2. Team working;
3. Identify with the task (clear goals, personal interest, know purpose of task, how it fits in with whole, job satisfaction; producing identifiable piece of quality work);
4. Employee participation/involvement/working with other;
5. Working in a successful product;
6. Development needs addressed;
7. Autonomy;
8. Equity;
9. Good Work/life balance (flexibility in work times, caring manager/employer, work location);
10. Development practices (object oriented, XP and prototyping practices);
11. Problem solving (the process of understanding and solving a problem in programming terms).

Although motivation may be intrinsic (derived from the pleasure of doing the work itself) or extrinsic (related to factors external to the job, such as work conditions), according to Sharp et al. [104] "the motivators inherent in software engineering are all intrinsic factors as they relate solely to software engineering", but in the case of software development, motivators can also be related to the work environment and individual's personality. Figure 2.2 shows the model used by the authors to guideline their work.

Another important concept related to this type of research is the reliability of the answers collected. One way to assess this reliability is through Cronbach's Alpha ($\alpha$), where, according to Cronbach [105] *'$\alpha$ is therefore an estimate of the correlation between two random samples of items from a universe of items like those in the test'*.

According to Maroco and Garcia-Marques [106], it is considered that a Cronbach's alpha between 0.7 and 0.9 would be acceptable and the responses reliable. According to this same study, a Cronbach's Alpha above 0.9 shows high reliability.

### Understanding in Agile Development Methods

In this work, to evaluate the understanding of agile development methods, an option was made to evaluate the evolution of knowledge about some critical success factors in agile software projects. In this way, the concepts of critical success factors in agile software

| *General motivators* | *Motivators in agile development* | *Demotivators in agile development.* |
|---|---|---|
| Autonomy | X | |
| Changing | | X |
| Development Needs Addressed (e.g. training opportunities to widen skills; opportunity to specialize) | X | |
| Feedback | X | |
| Good Management (senior management support, team-building, good communication) | X | |
| Work/life balance (flexibility in work times, caring manager/employer, work location) | | X |
| Identify with Task (clear goals, personal interest, know purpose of task, how it fits in with whole, job satisfaction; producing identifiable piece of quality work) | X | X |
| Software process/lifecycle (software development, project initiation and feasibility studies, and maintenance) | X | X |
| Problem Solving | X | |
| Recognition of work done (for a high quality, good job done based on objective criteria) | X | X |
| Rewards and incentives (e.g. scope for increased pay and benefits linked to performance) | | X |
| Sense of belonging/supportive relationships | | X |
| Teamwork | X | X |
| Variety of Work | X | |

Table 2.6: Motivators and demotivators found in the literature

projects, extracted from the work of Chow and Cao [107], were used. In that paper, the authors selected in the literature and in some case studies, some factors that could be considered success and failure factors in agile development projects.

Chow and Cao [107], at first identified, 12 main success factor for agile software development in literature, distributed in five factor dimensions: technical, people, process, organizational, and project. With those 12 factors the authors have elaborated 48 hypotheses and they conducted a survey to test those hypotheses that consisted in evaluating each one of the 12 success factors in four success dimensions in regard to the agile software development process: quality, scope, time and cost. After the analysis of the results of the survey, the authors concluded that only 3 factors of the 12 initial factors are actually critical for the agile development: a correct delivery strategy, a proper practice of Agile software engineering techniques, and a high-caliber team. The authors add three more factors that could be critical to certain success dimensions: a good Agile project management process, an Agile-friendly team environment, and a strong customer involvement.

The table 2.7 presents the 12 factors used in this work. Those that contain a value in the 'Rank' column are those 6 factors actually considered as success factors in the agile development process for Chow and Cao [107]. The table also shows the dimensions in which the 6 final factors were considered as success factors.

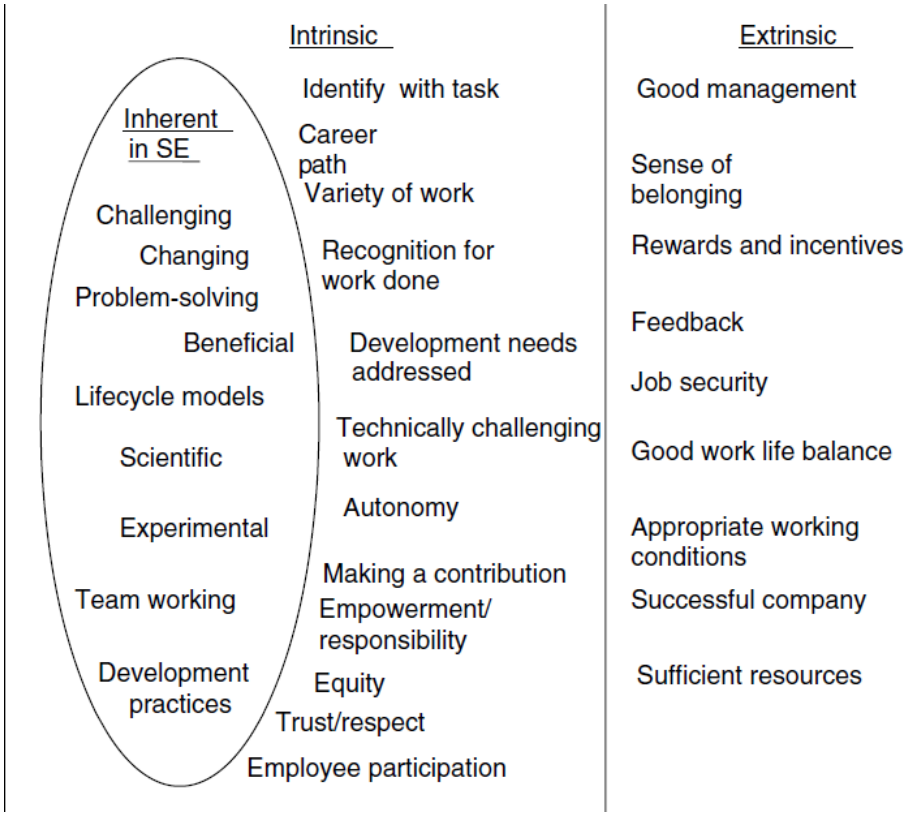


Figure 2.2: Motivators, Outcomes, Characteristics and Context)

| Number | Rank | Dimensions | Factor |
|---|---|---|---|
| 1 | | | The existence of a strong management commitment. |
| 2 | | | The presence of agile-friendly organizational environment. |
| 3 | 5 | Q | The existence of agile-friendly project team environment. |
| 4 | 3 | T/C | Having a team of high caliber. |
| 5 | 6 | S | Having a strong customer involvement. |
| 6 | 4 | Q | The practice of agile project management process. |
| 7 | | | The practice of a methodical project definition process. |
| 8 | 2 | Q/S | The practice of agile software engineering techniques. |
| 9 | 1 | S/T/C | The execution of a correct delivery strategy. |
| 10 | | | Limiting only to non-life-critical projects. |
| 11 | | | Limiting only to projects of variable scope with emergent requirements. |
| 12 | | | Limiting only to projects with dynamic, accelerated schedule. |

Dimensions: **Q**uality / **S**cope / **T**ime / **C**ost according to Chow and Cao [107] survey

Table 2.7: Factors that can be success factors in agile development

# Chapter 3

# The Proposed Framework

In chapter 2 several solutions for dealing with Service-Oriented Architecture (SOA) and Agile Development were discussed. But none of the work reviewed presents a substantial solution for the entire software development life-cycle. Few of the solutions presented deal directly with the principles of SOA and Agile. This chapter proposes a framework that addresses these gaps and also Quality Assurance (QA). To facilitate communication, the framework was named NatVi.

Distributed software is already a reality and has allowed a great dynamism in the organizations' businesses, represented by innovations, rapid product to market and constant changes in their goals. Traditional ways of thinking about software architecture and software development may not be able to handle these new business needs. But since the 1990's, two concepts have been gaining ground in the software engineering world, Service-Oriented Architecture (SOA) (background 2.1), and Agile development methods (background 2.2), and they seem not only capable of handling such issues but also seem to be able to maintain or improve Quality Assurance (QA) (background 2.3).

The proposal of this work is to create a framework that deals with SOA, Agile development and QA. In terms of architecture, some other design patterns could be able to deal with the business needs presented such as client-server architecture, distributed Internet architecture or even a hybrid service architecture. SOA was selected because its main features seem to represent an advantage over the other architecture options: standardization, interoperability, loose coupling, reusability, composability, and discoverability.

The Agile Development Methodology was selected because it proposes a short and iterative process, which can improve the integration of team members, leading to more team members' efforts, better sharing of knowledge, software coding, and maintenance of the software itself. The Agile methodology is recognized as flexible and suited to deal with constant changes in requirements and fast deliveries, which seems adequate to deal with the needs of the business.

Using these two technologies, SOA and Agile development methodology can also be a factor in improving QA, which is a necessity in itself. Indeed, the delivery of high quality is an obligation for the successful software engineering. That is why QA was selected to be part of the proposal of this work.

With the background (chapter 2) studies on Service-Oriented Architecture (SOA), Agile Development and QA and the cross-referencing of these three concepts, it was possible to answer the research questions that were elaborated to guide this work and to give a solution to develop applications with quality and fast deliveries in a distributed software environment that meets changing business goals. The answer came as a framework, depicted in figure 3.1.

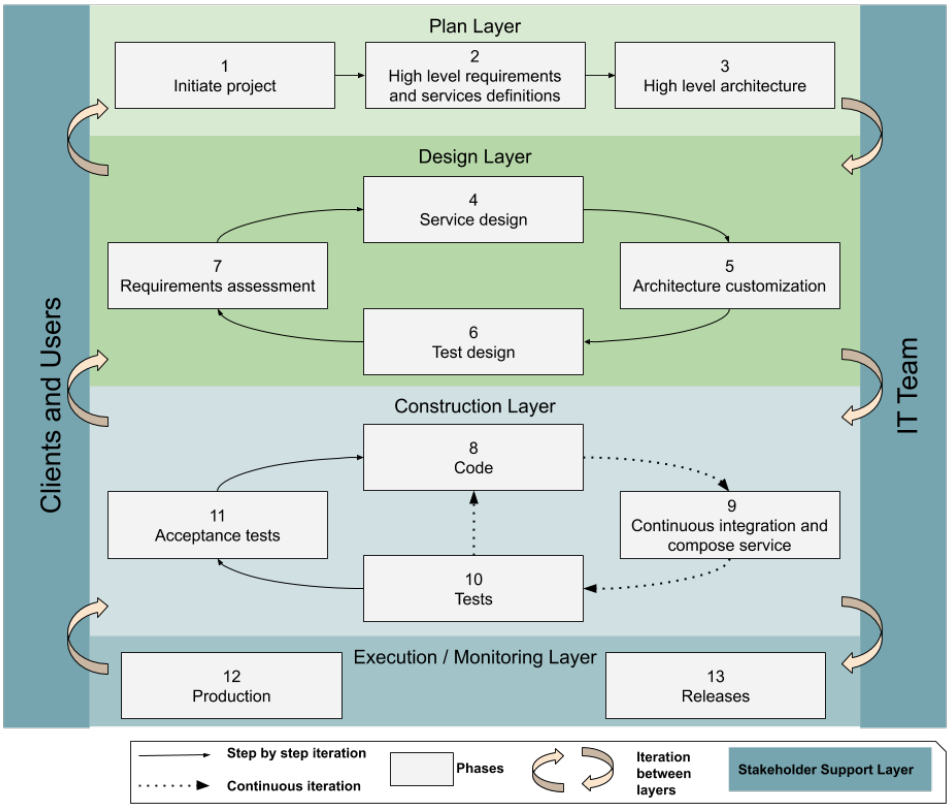


Figure 3.1: NatVi Framework for Agile, SOA and QA

The NatVi framework does not join together both life-cycles, agile development and SOA, on the contrary, it is an attempt to depict in only one life-cycle the big phases necessary to implement services solution, using SOA in an agile way. As a framework, it is not exhaustive and it is presented as a flexible set of phases that needs to be assessed for each project and reality. Not all phases need to be implemented and new phases with better detailing can be aggregated, if necessary.

The NatVi framework is composed of five layers with respective phases. The 'Stakeholder Support' layer permeates all other layers, which means that the integration of clients, users and IT team is critical to the success of the software project. The other layers are positioned in a way that indicates they work sequentially, and that is right, but the 'Iteration between layers' arrows indicate that if the development is facing problems in some layer, the work can return to the previous layer. It could even return to the 'Plan' layer if new business goals arise during the other layers.

## 3.1 Phases of the framework

As a framework, each iteration needs to be adjusted to the environment. Therefore, the terms used here, as well as the examples, are borrowed from the case study environment described in chapter 4. In other environments, these terms may change and may be extended.

In the following subsections, each phase of the structure is described in terms of:

- main roles: people who should be involved with the tasks of the phase. In this explanation the following roles are used:
  - Sponsor: The main responsible for the system. It is usually who pays for the system. This role can be accumulated with the roles of Clients and End User;
  - Client: who is responsible for the system operationally. This role can be accumulated with the role End User;
  - End User: who will interact with the system;
  - Technical team: roles in this category represent skills that some team members need rather than a formal role within the team, meaning that within the team there is no need for a member responsible for specific roles, but somehow these skills must be distributed among team members:
    * Business Analyst: the ability to connect business people with the development team. People with these skills need to understand the business environment and domain language to some degree and must be able to translate business concerns into a technical language;
    * Software Designer (or software architect): the ability to make views of software architecture and compose the services of SOA;
    * Database Administrator: the ability to understand the data involved throughout the process and create a solution for data entry, maintenance and retrieval, or make it easy to create the solution.

    - Tester: Ability to design, execute and evaluate software tests. This test can be unit tests, integration tests, acceptance tests and etc;
    - Developer: the ability to build system source code. This role also includes the ability to create unit tests and deal with some QA concerns during the development phase;
    - QA Manager: the ability to monitor and evaluate Quality Assurance (QA) throughout a software development life-cycle;
    - Operator: the ability to maintain constant integration and constant delivery on an automated basis.
- input: all available material that can help people to develop the tasks of the phase. These can be documents created by the project itself or outside the project and the results of previous phases;
- tasks: the work that needs to be done in each phase to achieve the phase objective;
- tools: any tools, electronic or otherwise, and even techniques that assist in performing tasks and in controlling and documenting the process;
- outputs: the material result of each phase. It can be a report of elicited requirements, use cases, recorded in a control tool, or even the finished system and in production;
- examples: examples of each phase borrowed from the case study done during this research.

### 3.1.1 Phase 1 - Initiate project

This phase is composed of a set of tasks that allow for a general understanding of the project. At this stage, some bureaucracy must be made to document the origin of the demand and its main concerns, although without a more precise technical definition. This task can follow a standardized protocol and must be done in the union of the stakeholders and the technical team, but at this stage, stakeholders can be free to expose all their thoughts and concerns without much interference from the technical team. Technical adjustments will be made in subsequent tasks.

**Main roles:** Sponsor, Client, Technical member - Business Analyst

Inputs: Business Objectives. The strategic plan of the organization. Deadline for the solution to be ready. Budget available. List of people from the business areas who will participate in the project.

**Tasks:**

- meetings;
- Formalization of demand;
- Creation of initial documentation;

- Project Registration in Control Tools;
- Give project information access to all stakeholders.

**Tools:**

- project management: a tool or toolkit that allows project control, tasks, schedules, budget, people involved, documentation repository, etc;
- version control: a tool that allows versioning of generated artifacts, documentation, source code, etc;
- Communication: definition of tools and communication process among all the people involved.

**Outputs:** the formalized project. Document with key characteristics recorded: people involved, schedules, budget, main concern with business objectives, problems that need to be resolved, expectation about the possible new system. Possible, because the solution may be other than developing a new system. This 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 readme file.

**Example:** Customers have formalized their need in a document in the organization's information system, linking their needs with the organization's strategic plan that takes the time and budget available. For the initial meeting, they raised concerns about the need to automate the monitoring of compliance by some companies being monitored.

The project was called 'Compliance' and was registered in Microsoft Team Foundation Server (MTFS) (now known as Azure DevOps Server). The 'readme' file was created and the initial information from the entries and other information collected at the initial meeting was recorded.

### 3.1.2 Phase 2 - High level requirements and Service definition

This phase is where the requirements are elicited, meaning that problems and concerns of the clients and end users are identified and broken down into concrete business needs. The main concern is to identify the problem domain and not the solution to those problems. Information is identified and extracted from the business needs and interpreted, analyzed, validated and gathered together to compose the services. In this phase, the clients, end users, and the technical team are identified and a more precise technical definition is initiated. All stakeholders need to understand the goals and break them to be transformed into services.

Ideally, goals need to be divided as much as possible. Each of these parts defines a service. Thus, each service represents specific functionality that explicitly maps to a step in a business process.

**Main roles:** Client, end Users, Technical members - Business Analyst, Software Designer and Tester;

**Inputs:** the documentation generated in Phase 1 - Initiate project (3.1.1). Business process mapping, if any, or detailed description of the phases of how the work is done. Other documents that support business objectives;

**Tasks:**

- meetings;
- requirements elicitation;
- initial tests design;
- link among requirements, services and tests;

**Tools:**

- project management;
- version control;
- communication;
- design: tools that allow the software designer to create blueprints and schematics for structures, systems, machines, and equipment and work collaboratively with the team members. There are many specialized tools available in the market with a lot of features. But there are more simple suits that could be used, such as Microsoft Office Visio[1], Bizagi[2] and even the Google Drawings[3];
- test: some techniques use specific tools to support building tests an execution, such as Selenium[4] or Cucumber[5].

**Outputs:** requirements, backlogs, stories, use cases, draft of the architectural views of the services and draft of the case tests.

**Example:** The clients brought the list of items to monitor and a document with the description of how the work is done manually. In the meeting it was decided that each item for monitoring is a specific goal and each one will be a service. The clients prioritized the items for monitor and the technical members registered backlogs and built use cases for the first five. The idea is to get the expertise with the first five services and then build the next. The technical members initiated the built of the architectural views.

[1]https://products.office.com/pt-br/visio/
[2]https://www.bizagi.com
[3]https://docs.google.com/drawings
[4]https://www.seleniumhq.org/
[5]https://cucumber.io/

### 3.1.3 Phase 3 - High level Architecture

This is the phase where a high level architecture is elaborated. Service-Oriented Architecture (SOA) as an architectural pattern does not have a fixed way to be implemented. In this phase, if SOA is new in the organization, the technical team, preferably the software designer, should elaborate on the whole architecture. If SOA is a well-established environment, the work here will probably be simplified, and only a few set-ups will be made.

**Main roles:** Technical members - Business Analyst, Software Designer and QA Manager;

**Inputs:** the outputs generated in previous phases, mainly the drafts of the architectural views from Phase 2 - High level requirements and Service definition (3.1.2). Some information from Phase 1 - Initiate project (3.1.1) are also important such as schedule and budget. The infrastructure architecture, including information about external clouds, if exists.

**Tasks:**

- based on the budget, time and available tools, designing the architecture;
- building the SOA, or if it already exists, making the initial customization, if necessary.

**Tools:**

- project management;
- version control;
- communication;
- design;
- specific tools to SOA management, such as Composite Application Manager for SOA[6] and AmberPoint SOA Management System[7]
- specific tools to build SOA solution, such as Oracle Enterprise Service Bus (ESB)[7] or Red Hat Fuse[8]

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

**Example:** SOA is in course in the example environment. The environment uses the Red Hat Fuse as an Enterprise Service Bus (ESB) and the services and data are routing across various applications. In this phase the technical team took the opportunity to update the existing architectural views of SOA. If some architecture view was deprecated, it was updated or marked as deprecated. The ideal is to maintain all views updated, but

[6]https://www.ibm.com/support/knowledgecenter
[7]https://www.oracle.com
[8]https://developers.redhat.com/products/fuse

if it is impossible to update a view because of time problems, it is better to exclude that view rather than maintaining it out of date.

### 3.1.4 Phase 4 - Service design

This phase is where the services (Consumer and Provider) are designed and the SOA issues are handled. This phase represents the beginning of the first loop, or iteration, in NatVi framework. This iteration will be done 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 construction or execution / monitoring layers, the requirements can be reevaluated and the process can return to this layer.

The eight principles of the service-oriented solution must be analyzed in this phase, Standardized Service Contract, Service Loose Coupling, Service Abstraction, Service Reusability, Service Autonomy, Service Statelessness, Service Discoverability, and Service Composability.

**Main roles:** Client, end user, Technical members - Business Analyst, Software Designer, Database Administrator and QA Manager;

**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 (3.1.2) and the blueprint of the high level architecture from Phase 3 - High level Architecture (3.1.3).

**Tasks:**

- refinement of requirements;
- identification of the various business goals in their smallest parts;
- deciding between developing a new service or using or adapting an existing service. In this task, there is special attention to the Discoverability principle.
- design of the services themselves. In this task all of the eight principles of SOA must be analysed.
- deciding what to do if any principle cannot be reached.

**Tools:**

- project management;
- version control;
- communication;
- design;

**Outputs:** all the views necessary to built the service, including information about architecture customization. These views must be recorded in the project management tool.

**Example:** The requirements were broken down into its smallest parts and were decided that each article of the norms will be a service. As the project was in the beginning, some control services were needed. All identified services will be built from the beginning, meaning new services. This example and the next ones will be treating only the services for control and one business service, regarding the delivery of documents on time by the monitored companies, called Submission of Trial Balance (STB) Service. See chapter 4 to understand better the context of these examples.

The eight principles of the service-oriented solution were analyzed. This explanation took only the STB Service:

- Standardized Service Contract: the request to the service will be authenticated. Service will receive an 'xml' file with the following information: code of the company and month and year of the Trial Balance; The response to the service will be an 'xml' file with the information if the Trial Balance of the company at the period was sent. The response 'xml' file still counts with other sections where more detail could be described, but, to this service, only the main information is necessary, that is, if the Trial Balance was sent by the company or not. All service of this type, that is, analysing if some document were sent in time, have this same contract.
- Service Loose Coupling: this service was built as independent, that is, it does not depend on that other application. But, the decoupling in fact is not total. This service accesses directly the institutional database, so there is some level of coupling;
- Service Abstraction: all implementation of the service are hidden, that is, looking at the contract it is not possible to identify other entry point other than that described in the contract. This also facilitates the loose coupling;
- service Reusability: the service contract was designed in a way that if other application needs the same information the service will be available;
- Service Autonomy: as explained in the loose coupling, the service was built as independent. The environment is based in containers, which facilitates this principle;
- Service Statelessness: the service does not hold any information. The service receives the request, processes and sends the response and that is it.
- Service Discoverability: all information about the existence and the service contract are recorded in a tool to facilitate posterior manual discoverability.
- Service Composability: this principle has not been achieved. But it was not considered a problem.

### 3.1.5 Phase 5 - Architecture Customization

At this phase, it is evaluated whether some adjustments are required in the architecture. Because SOA is a design pattern, not a whole architecture, each service may require some customization such as a new interface or new term in the standardized service contract.

**Main roles:** Technical members - Software Designer and QA Manager;

**Inputs:** the outputs generated in previous phases, mainly the views of the services from Phase 4 - Service design (3.1.4) and the blueprint of the high level architecture from Phase 3 - High level Architecture (3.1.3).

**Tasks:**

- refinement of architecture;
- update the documentation of the architecture;
- implement the modifications on the architecture.

**Tools:**

- project management;
- version control;
- communication;
- design;
- specific tools to SOA management;
- specific tools to build SOA solution.

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

**Example:** no customization was done in the original architecture;

### 3.1.6 Phase 6 - Test design (test first)

At this phase, before the Construction phase (code), the tests are created. 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, business goals.

**Main roles:** Client, end user, Technical members - Tester, Business Analyst, Developer and QA Manager;

**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 (3.1.2), the refined requirements and the views of the services from Phase 4 - Service design (3.1.4).

**Tasks:**

- refinement of the drafts of the case tests;
- elaborate the unit tests, integration tests, system tests and acceptance tests;

- select metric to guarantee the QA;
- documenting

**Tools:**

- project management;
- version control;
- communication;
- design;
- specialized tools for test, such as Selenium, jUnit[9] and Sonarqube[10]

**Outputs:** tests designed and documented in the project management tool.

**Example:** as the service will be automated and the end users have their own way to do the monitoring manually, they agree to do the acceptance tests comparing the result of some iteration of the new system with their manual results. The integration tests, system tests were designed in term of tasks in the pipe line of the operational tasks. The unit tests were designed to be implemented during the coding.

### 3.1.7 Phase 7 - Requirements assessment

This phase is the bottom of the first loop, the end of the first iteration. After the previous phases, the technical team already has more understanding of the needs of the services. In a new interaction among clients, end users and the technical team, the initial elicited requirements are evaluated and some customization can be done, if necessary, in which case the loop will be restarted. This phase is like a control point, where all the work done until here is evaluated and it is decided if it is time to pass to the construction layer.

**Main roles:** Client, end user, Technical members - Business Analyst, Developer and QA Manager;

**Inputs:** all the outputs generated in previous phases.

**Tasks:**

- evaluating all the work done until this phase;
- documenting

**Tools:**

- project management;
- version control;
- communication;
- design;
- other specialized tools used until here.

[9]https://junit.org
[10]https://www.sonarqube.org

**Outputs:** a document with the evaluation of the work done until here recorded in the project management tool.

**Example:** it was necessary some iterations until the work could pass to the next layer. With each iteration, all stakeholders are made more aware of the goals and functionality of the new system.

### 3.1.8 Phase 8 - Code

This phase is the beginning of the construction layer. Although the phases are ordered, it could be said that there is a short circuit among the phases 8, 9 and 10, which are done almost at the same time, in a 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 phase is the first one of a new loop, or iteration, the 'Construction' layer. This iteration will be done until the service passes acceptance tests, made by clients and users.

**Main roles:** Technical members - Developer and QA Manager;

**Inputs:** the outputs generated in previous phases, mainly the views of the services from Phase 4 - Service design (3.1.4), the refined requirements and the views of the services from Phase 4 - Service design (3.1.4) and tests design from Phase 6 - Test design (test first) (3.1.6).

**Tasks:**

- coding the services;
- coding the unit tests;
- evaluate unit tests;
- daily commit and pushing;
- deploying system in a development environment
- documenting;

**Tools:**

- project management;
- version control;
- communication;
- specialized tools for test;
- specialized tools used to development, such as Eclipse IDE[11] and Microsoft Visual Studio[12];

[11] https://www.eclipse.org
[12] https://code.visualstudio.com

- specialized tools for continuous integration, such as Jenkins[13] and Microsoft Azure Devops[14];
- documenting.

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

**Example:** the source code and unit tests were done in java and the daily commit and pushing guarantees that the code was always available to the MTFS (Azure Devops) and to the Sonarqube.

### 3.1.9 Phase 9 - Continuous integration and compose service

As the concept of continuous integration is used, in this phase the tools are configured to do a set of actions. Tests, unit and integration, compiled and deployment and etc. At the build of the pipeline of the integration, the services are also composed.

The service is composed according to the designed architecture and the eventual customization for the specific service. Composing services means putting the service in the right place into the SOA and accessible for the other services by the same SOA. In this phase, the pipeline does the deployment for a test environment.

**Main roles:** Technical members - Operator, developer and QA Manager;

**Inputs:** the outputs generated in previous phases, mainly source code and the test code from Phase 8 - Code (3.1.8) and test design from Phase 6 - Test design (test first) (3.1.6).

**Tasks:**

- setting the integration tests and system tests;
- make the pipelines to tests;
- deploying system in a testing / staging environment;
- evaluating unit tests, integration tests and system tests;
- feedback to developers;
- documenting;

**Tools:**

- project management;
- version control;
- communication;
- specialized tools for test;
- specialized tools for continuous integration;
- documenting.

[13]https://jenkins.io
[14]https://azure.microsoft.com

**Outputs:** the system deployed in testing / staging environment and in acceptance test environment. Testing report result;

**Example:** operator and QA manager setting the MTFS building the pipelines to deploying the system to the test environment and to pass the Sonarqube. At each iteration, the MTFS sends an e-mail to the technical members with the tests results and deploying results.

### 3.1.10 Phase 10 - Test

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 separate has an academic meaning. In fact, in the real world, these last 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 this phase the tests designed in the 'Test design' phase are made and the services are tested. The unit tests are made by the developers, and they execute these tests in their own machine to identify the firsts errors before committing to the code. The unit tests will be executed again, in Phase 9 - Continuous integration and compose service, but it is expected that a few errors arise in that phase originated by the unit tests, because the proper developer should have fixed them.

The integration and system tests are already made in this phase, but, preferably, they need to be made by the QA manager and operator, with the capability of thinking in a set of services and considering the SOA. These tests will be executed only in Phase 9 - Continuous integration and compose service, where it is expected a greater number of errors than the unit tests.

### 3.1.11 Phase 11 - Acceptance test

The end of the loop or iteration of the construction layer. The service, already composed according to the proposed architecture, is tested by clients and end users, according to the definitions of the Phase 6 - Test design (test first) (3.1.6). These tests can be done manually or using specific tools for this.

**Main roles:** Sponsor, Clients, end users, Technical members - QA Manager;

**Inputs:** the system deployed in the acceptance test environment. The acceptance tests design in Phase 6 - Test design (test first) (3.1.6).

**Tasks:**

- evaluating system by the acceptance tests;
- feedback to technical members;

- formal approval or not;
- documenting;

**Tools:**

- project management;
- version control;
- communication;
- specialized tools for test;
- documenting.

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

**Example:** the clients and end users compare the result of the new system with their own results made manually.

### 3.1.12 Phase 12 - Production

In this phase, after the services are considered finalized and approved by the sponsor, clients and end users, all the work to deploy the services in the production environment is done. Preferably in an automatic manner, where the integration tools allow for the construction of pipelines that obtain the source code, run all the tests again and put the system into production.

**Main roles:** Technical members - operator, QA Manager;

**Inputs:** the source code from Phase 8 - Code (3.1.8). The documentation of the architecture, from Phase 3 - High level Architecture (3.1.3) and Phase 5 - Architecture Customization (3.1.5).

**Tasks:**

- adjust the architecture;
- setting the production environment;
- make the pipelines to production;
- deploying system in a production environment;
- documenting;

**Tools:**

- project management;
- version control;
- communication;
- specialized tools for continuous integration;
- documenting.

**Outputs:** the system deployed in the production environment. The documentation up to date;

### 3.1.13 Phase 13 - Release

The service is released 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.

# Chapter 4

# Case Study

To evaluate the NatVi framework, a case study was carried out. The case study consisted of introducing agile development into an environment that already uses Service-Oriented Architecture (SOA). It was carried out in a Small department of the Brazilian Federal Government. It is an executive branch department of the federal government charged with coordinating and supervising a field of public interest. On the rest of this work, this Department will be identified by the acronyms DoCS. In DoCS there are four sub-departments and the unit responsible by the Information Technology (IT) is under the Administration Sub-department. The Information Technology (IT) unit is subdivided into Infrastructure Coordination and Software Development Coordination, responsible for system development and maintenance.

The IT unit has eighteen people, among its own servers and outsourced personnel. The software development coordination is made up of eight people, including a coordinator, a division head and three trainees. The development team is composed of four people, including the division head. Although trainees are from the development area, they do not count as members of the development team because they can not develop without a responsible developer.

At the beginning of the case study, the software development methodology used in DoCS was classified as traditional, that is, not agile methodology. The process consisted of a big design up front and a confusing process of documentation. In the urgent need of development, sometimes much documentation was recorded and sometimes no documentation was recorded. It was a difficult job for a small team. In addition, each project was carried out almost by only one member. Due to this situation, some issues arise. Some of them are well known and agile methods seem to be adequate to address them, such as requirements changes, short time delivery needs, team integration, and product maintenance. Service-Oriented Architecture (SOA) was not new to the development team because other systems already use this type of architecture in DoCS, although in an in-

cipient and initial way. The solution found by DoCS was to implement SCRUM, and the first step was to train the team in this methodology. It seemed to be a good environment to do a case study.

The project, object of the case study, was a new project, named 'Compliance'. DoCS is in charge of monitoring a set of private companies that perform a particular type of public interest service. The objective of the project was to automate the verification of compliance of these companies according to the respective laws that regulate the environment in which they operate.

There are many and complex items that DoCS oversees, and some of them are under confidentiality but, some of them are not. The DoCS receives some periodic electronic information from the companies. Creating an automated routine to track the delivery of this information within the legal deadline is the phenomena that this case study investigated.

The system was built from scratch, as the only thing that was already done was the SOA, used in other systems. In order to organize the implementation of the agile development methods and the collection of metrics needed for the case study, conceptually the process was divided into five main phases, besides the writing of the final report (this work). Actually, some of these phases were executed at the same time:

1. Agile training.
2. Adaptation to the agile development.
3. Development, applying the NatVi framework.
4. Measurement.
5. Analysis.

The project 'Compliance' itself is larger than this case study. The entire project schedule is more than a year and this case study followed only a few weeks from the start of the project. In figure 4.1 an overview of the schedule of the case study is shown. The SCRUM method uses the concept of sprints and the sprint initially would take a week, but in the end, after the adaptation phase, the sprints passed to two weeks each.

The case study followed the project for 18 weeks. During these weeks, the agile training, the adaptation of the agile methodology, the development itself (five sprints), the collection of data following the defined indicators, the application of the Intrinsic Motivation Inventory (IMI) forms and the semi-structured interviews and the analyses of collected data were carried out.

## 4.1 Agile training

The case study began in an environment that SOA was in operation and the development methodology was not agile. The first step was to implement agile development in

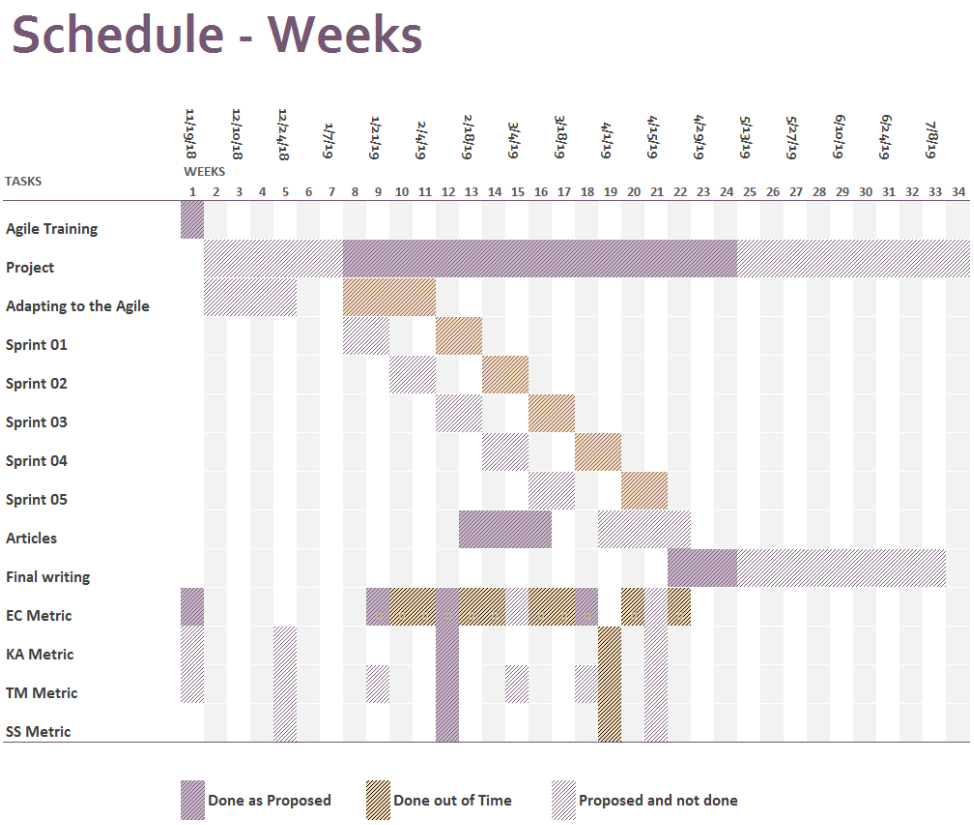


Figure 4.1: Case study schedule

the environment. To achieve this, a specific methodology needed to be selected, such as SCRUM, Extreme Programming (XP), Crystal Clear, Feature-Driven Development (FDD), etc. The team itself selected SCRUM as their preferred methodology and a week of training in the SCRUM methodology was given to the development team and to some clients and users.

There was a delay of 4 months at the beginning of the agile training in relation to the expected one. The training was scheduled to take place in July and it took place in November. This delay has caused some practical problems, as we will see in the development phase below.

## 4.2 Adaptation to the agile development

The adaptation to the agile development was necessary for the paradigm shift, from a traditional way of development to an agile way. As the team itself chose SCRUM as the agile method, some work needed to be done, since SCRUM follows a well-defined protocol with some well-defined personas as well. This phase of adaptation lasted four weeks. Note that there was a gap of 6 weeks between the agile training and the beginning of

the development phase. This gap occurred because during the period between the end of one year and the beginning of the next one there are many routines to be closed in the Information Technology (IT) unit and many people are on vacation. As agile training was postponed, as seen in the agile training phase above, the start of the project began in this atypical period.

The adaptation phase was carried out at the same time as the initial tasks of the development phase in the planning layer of the framework.

Some adjustments in the software development tool-kit were made. These adjustments were a necessity of adaptation to the agile development itself and were not specifically made for the case study. But the case study took advantage of some of these changes to make it easier to monitor the phenomena.

The adjustment in the tool-kit was more in the way of using it than in the introduction of new tools, as these tools were already used in traditional forms of development. The figure 4.2 shows an overview of these tools that allows continuous integration and unit testing. The test repository and selenium are the only two new tools in this tool-kit:

- Test repository: in this case study it was used the SonarQube tool[1].
- Microsoft Team Foundation Server (MTFS) as continuous integration ( source code management; reporting, requirements management, project management, automated builds, testing and release management capabilities.
- Maven as manager of a project's build;
- JUnit for unit tests in Java programming language.
- Selenium for web browser and web services tests

Besides this tool-kit, presented in figure 4.2, other tools were used in the development, such as Eclipse IDE for Java and PHP development, Microsoft Visual Studio for C#, .Net and SSIS development, Git for version control for Java and PHP.

## 4.3 Development

This phase addressed the implementation of the NatVi framework. After the agile training and the adaptation to agile development, the development was initiated. The solution was built following the NatVi framework.

The planning layer, as prescribed on the framework, was carried out at the same time as the adaptation phase of this case study. In this first layer, the general documentation and the formalization of the project were carried out. High-level business goals were identified and the necessary services were thought out. The high-level requirements began to be elicited. The high level of SOA has also been defined.

[1]https://docs.sonarqube.org

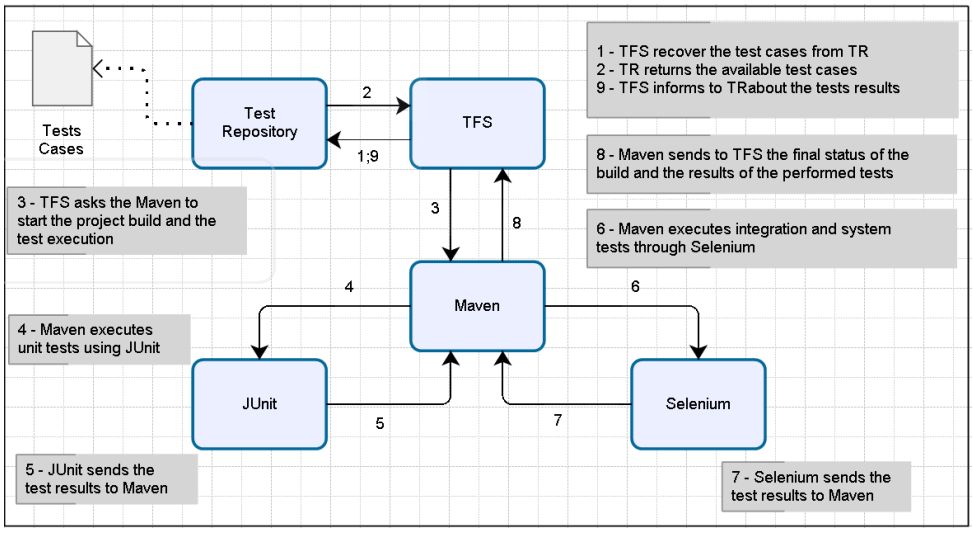


Figure 4.2: Tools composition

After the planning layer, the sprints began, taking two weeks each. The first and second sprints were used to cover the tasks in the design layer:

- Architecture customization: in this task, an Enterprise Service Bus (ESB) was used, with the setting of the Red Hat fuse tool. In this task the database was also designed and constructed.
- Service design: to deliver the product of this case study, four services needed to be designed:
  - Thin client web application: a service consumer. Interface web to manipulate the data in the database, consuming the CRUD service;
  - CRUD ( Create, Read, Update and Delete): a provider service to manipulate the data in the database;
  - Executor: a consumer service that calls ESB to start the job according to the default schedule. When executed, ESB in turn consumes the CRUD service.
  - Monitoring A: a service provider that receives a call from the ESB, process the service to monitor the delivery of document A and returns data to the ESB. Document A is a nickname for the real document that is being monitored. This service is considered the core of the work. The other services are part of the platform, they will serve to other monitoring services.
- Test design: based on the requirements elicited, on the services designed, the tests were also designed.
- Requirements assessment: At this point, the knowledge of problems and solutions is already more advanced, so the requirements can be evaluated and, when necessary, restart the iteration.

The construction layer used the other three sprints to complete the development of the first module of the 'Compliance' project, the monitoring of the delivery of document A. The following phases of the NatVi framework were carried out in an iterative way in these three sprints:

- Code: the code of the designed services was done in java language. The commit was, at least, daily. In this phase, the setting of the ESB, the Red Hat fuse tool, was done;
- Tests: the unit tests were built, according to the designed tests;
- Continuous integration and compose service: for each commit, MTFS performed the integration tests and the composition of the services through the configuration of the respective pipelines.
- Acceptance tests: at the end of each sprint, as recommended for the SCRUM methods, a meeting took place with all stakeholders to evaluate the work done and if the work would be approved or not. In case it is not approved, the service must return to the backlog to be done again with the requirements adjusted or simply better detailed.

The 'Execution and Monitoring' layer of the NatVi framework, composed of the 'Production' and 'Release' phases, was not a part of this case study.

## 4.4 Measurement

As seen in the 2.3 section, Quality Assurance (QA) is both a planned and systematic process to provide adequate confidence that the software is compliant with the requirements and a set of activities designed to evaluate the process by which the software is developed.

In the NatVi framework, QA is present as the concept of a planned and systematic process, where many tasks are directed towards this end, especially in planning and testing, as it can be seen in the figure 3.1 in chapter 3.

In this case study, QA was evaluated by a set of activities. The main idea was to evaluate not only the development itself but also the engagement and perception of motivation of both the IT team and other stakeholders represented by the clients who requested the software, which can also influence the quality of the software. For that, four simultaneous indicators were used, one objective, through analyzes of the developed code, and three subjective ones, through the evaluation of the team knowledge about the agile methodology and motivation of IT team and stakeholders:

1. Error Code (EC): This indicator aimed to evaluate the reliability of the developed code in relation to the requirements elicited. The test cases and the code metrics were defined by the team and researchers according to the development

environment. Data collection was automated using tools during the development and testing phases. Because agile is intrinsically iterative, although data collection was done for each commit, only those collected at the end of each sprint were used. See section 2.5, quality metrics, for more about the concepts of the metrics of this indicator.

2. Knowledge in Agile (KA): With this indicator, agile knowledge of IT team has been assessed. Factors that can be considered successful in an agile development project were used to evaluate this knowledge, through a questionnaire and semi-structured interview. See section 2.5, Understanding in Agile Development Methods, for more about the concepts of the metrics of this indicator.
3. Team Motivation (TM): This indicator aimed to evaluate the motivation of the IT team towards the activities of the framework. The metrics were collected through the application of a specific Intrinsic Motivation Inventory (IMI) questionnaire and a semi-structured interview, based on the IMI questionnaire questions. See section 2.5, intrinsic motivation, for more about the concepts of the metrics of this indicator.
4. Stakeholders Satisfaction (SS): In this indicator, the stakeholders' satisfaction was evaluated. The metrics were also collected through the application of a specific IMI questionnaire and a semi-structured interview, based on the IMI questionnaire questions. See section 2.5, intrinsic motivation, for more about the concepts of the metrics of this indicator.

All four indicators presented needed a baseline, that is, an initial measurement to be the basis for comparison. After collecting the initial values, each metric needed to be collected as needed. These questions and more details of the four indicators are addressed in the following subsections.

### 4.4.1 Automated metrics

Error Code (EC) was evaluated by code and development error density metrics that were collected automatically. The error metrics used were collected during the development process. These metrics were shown in the 2.5 section. In short they are Code Error Density (CED), Weighted Code Error Density (WCED), Development Error Density (DED) and Weighted Development Error Density (WDED).

As seen in the section 4.2 above, a tool-kit was deployed and configured to the new development environment, allowing automated deployment and continuous integration, including automated tests.

As a test repository it was used the SonarQube tool and it was used its default configuration. As it is possible to see in the SonarQube website, this tool works with the idea of issues [2]:

*While running an analysis, SonarQube raises an issue every time a piece of code breaks a coding rule. The set of coding rules is defined through the associated quality profile for each language in the project.*

Each issue is classified according to its severity. SonarQube works with five types of severity:

- Blocker: Bug with a high probability to impact the behavior of the application in production: memory leak, unclosed JDBC connection, etc. The code MUST be immediately fixed.
- Critical: Either a bug with a low probability to impact the behavior of the application in production or an issue which represents a security flaw: empty catch block, SQL injection, etc. The code MUST be immediately reviewed.
- Major: Quality flaw which can highly impact the developer productivity: uncovered piece of code, duplicated blocks, unused parameters, etc.
- Minor: Quality flaw which can slightly impact the developer productivity: lines should not be too long, "switch" statements should have at least 3 cases, etc.
- Info: Neither a bug nor a quality flaw, just a finding.

In this case study, an indicator similar to that proposed by U.S. Department of Defense (DoD) [4] was used to weigh the issues, where each gravity receives a specific weight. INFO severity was not used in the case study. The weights used were Minor (1), Major (3), Critical (6) and Blocker (9). These issues raised by SonarQube were used in the calculation of the Code Error Density (CED), Weighted Code Error Density (WCED) metrics.

For the calculation of the Development Error Density (DED) and Weighted Development Error Density (WDED) metrics, were used other data extracted from SonarQube such as line code duplication, Cyclomatic Complexity, and lines to cover. The respective weights, 1, 2 and 3 were chosen according to to the historic probability of their occurrence in the project

As a continuous integration tool, the Microsoft Team Foundation Server (MTFS) (now called Azure DevOps Server[3]) was used. In its pipeline, SonarQube has been configured to do the automated standard tests. In this way, SonarQube retained the results of these tests. SonarQube publishes some Application Programing Interface (API) that allowed

[2]https://docs.sonarqube.org/latest/user-guide/issues/

[3]https://docs.microsoft.com/en-us/azure/devops/?view=azure-devops

the extraction of the test result in an automated way. For this case study the R language[4] was used, a powerful language to deal with data in a statistical way.

As a baseline for these automated metrics, SonaQube issues from another project, similar to the project object of this case study, which uses SOA but not using agile development methods, were measured at the beginning of the 'Compliance' project. This measurement was used as a basis for evaluating the evolution of the metrics of the 'Compliance' project.

In short, the R code accesses the web services available in API and extracts the data for each commit that was made. To extract only the data of interest, some manual interference was made in the SonarQube, where a special tag was included in the measurement at the end of each SCRUM sprint, allowing the R language to filter only this measurement.

### 4.4.2 Questionnaires and semi structured interview

In order to evaluate the Knowledge in Agile (KA), Team Motivation (TM) and Stakeholders Satisfaction (SS), questionnaires and semi-structured interviews were carried out. At the beginning of the case study, the idea was to use only the questionnaires, but since DoCS is a small unit of the Brazilian government, the population to be consulted by these questionnaires was very small. In this way, a statistical result could be compromised.

It was decided that an alternative way was needed to understand the evolution of motivators and also in relation to agile knowledge. As seen in chapter 4, a case study has these characteristics, that is, to be dynamic, and then the semi-structured interview seemed to be a good option. Then, with the questions of the three initial questionnaires, three semi-structured interviews were elaborated.

For evaluate the Team Motivation (TM) and Stakeholders Satisfaction (SS) two versions of the Intrinsic Motivation Inventory (IMI) were carried out. See section 2.5 for more about the concepts of this metric.

#### Knowledge in Agile (KA)

Both the questionnaire and the semi-structured interview to evaluate the Knowledge in Agile (KA) were elaborated using the concepts of critical success factors in agile software projects, drawn from the work of Chow and Cao [107]. See section 2.5 for more about the concepts of this metric.

The objective here was to verify the degree of Knowledge in Agile by the IT team, identifying the perception of the development team members of each success factor for

[4]https://www.r-project.org/

agile development in four different dimensions: quality, scope, time and cost, and compare with the results found by Chow and Cao [107].

As this work is a case study into a small agile team, the statistical analysis could be impaired. So, in order to assess how much the team members were rationalizing about the agile development, it was presented the initial 48 hypothesis to the team members, over the time, and evaluate how the responses match to the work of Chow and Cao [107]. It was used a sub-scale scores that consists of seven statements that have to be scored on a 7-point Likert scale [102] where 1 represents 'Absolutely disagree', 4 represents 'Indifferent' and 7 represents 'Completely agree'. The team members had to evaluate the 4 success dimensions for each of 12 success factors.

### Team Motivation (TM)

Applying a new solution in a software development environment can change Quality Assurance (QA). There are several possible reasons for that. This NatVi framework may present some technical advantage over the old way of developing software. But if the people involved are not motivated to do the new activities, this may have the power to do QA. More motivation by itself does not mean that QA is better or worse, but it means that a good environment has been developed for software development and thus better quality can be achieved.

To evaluate the motivation of the team in participating in the implementation of the NatVi framework, object of this work, the concepts of motivators were used. See the section 2.5 for more details about this concept. In this work, only the motivators factors were analyzed, and only those found in at least two studies of the systematic literature review (SLR) carried out by De O. Melo et al. [103]:

1. Software process/life-cycle (software development, project initiation and feasibility studies, and maintenance): appeared in 7 (seven) studies;
2. Teamwork: appeared in 4 (four) studies;
3. Good Management (senior management support, team-building, good communication): appeared in 3 (three) studies;
4. Development Needs Addressed (e.g. training opportunities to widen skills; opportunity to specialize): appeared in 2 (two) studies;
5. Identify with Task (clear goals, personal interest, know the purpose of the task, how it fits in with whole, job satisfaction; producing an identifiable piece of quality work): appeared in 2 (two) studies;

In these five selected motivators, four are considered intrinsic. Two are inherent in software engineering (items 1 and 2) and two are not (items 4 and 5). Item 3 is considered

| | Motivator | IMI Sub-scale | IMI Question | |
|---|---|---|---|---|
| 1 | Software process/life-cycle | Interest/Enjoyment | I enjoyed doing this activity very much | |
| 2 | | | This activity did not hold my attention at all | R |
| 3 | | | I would describe this activity as very interesting | |
| 4 | | Effort/Importance | It was important to me to do well at this task. | |
| 5 | | | I didn't put much energy into this | R |
| 6 | Teamwork | Relatedness | I'd like a chance to interact with the team more often | |
| 7 | | | I felt really distant to the team | R |
| 8 | | | I fell close to the team | |
| 9 | Good management | Pressure/Tension | I felt very tense while doing this activity | |
| 10 | | | I was very relaxed in doing the necessary activities | R |
| 11 | | Perceived Choice | I believe the activities could be of some value to me | |
| 12 | | | I think this is important to do because it can give me the opportunity to specialize in software development | |
| 13 | Identify with Task | Value/Usefulness | I think doing this activity could help me to understand the purpose of my job to the organization as a whole | |
| 14 | | | I think this is an important activity | |
| 15 | | Perceived Competence | I am satisfied with my performance at this task | |
| 16 | | | This was an activity that I couldn't do very well | R |

Table 4.1: IMI by agile development motivators

an extrinsic motivator. Likewise, all five were used in the evaluation, because it was chosen to consider all the motivators that appear in at least two studies. It was an objective criterion and in the end, it was possible to identify them all with one or more items of the Intrinsic Motivation Inventory (IMI), in this way, it is believed that there was an improvement in this work, since it has worked with a greater number of motivators.

In this way, it was assessed the team motivation evaluating the agile development motivators [103] at the light of Intrinsic Motivation Inventory (IMI), a multidimensional measurement device intended to assess participants subjective experience related to a target activity in laboratory experiments [108]. In this way, the IMI intended to assess participants subjective experience related to a target activity in controlled experiments. The instrument assesses participants interest/enjoyment (IE), perceived competence (PC), effort/importance (EI), pressure and tension (PT), perceived choice (PC), value/usefulness (VU) and Relatedness (RL) while performing a given activity. Since the original IMI is long and repetitive, it is necessary to adjust the instrument according to specific tasks and fields [101] and a short version will be used attempting fit to the selected motivators [103]. This distribution allowed us to evaluate both the general intrinsic motivation and the level of motivation in each of the motivators selected in [103]. It was used a sub-scale scores that consists of seven statements that have to be scored on a 7-point Likert scale [102] where 1 represents 'Absolutely disagree', 4 represents 'Indifferent' and 7 represents 'Completely agree'.

The scale includes five control questions that are scored in reverse of the participants response on that item, indicated in table 4.1 following by '(R)'. The actual questionnaire will not exhibit the indication of revers scored (R) and the questions were randomly distributed. The scale was validated for use in software development through Cronbach's Alpha scores. The entire survey was designed to be completed in less than five minutes.

Scoring information Begin by reverse scoring items 2, 5, 7, 10 and 16 by subtracting the item response from 8 and using the result as the item score for that item. Then calculate sub-scale scores by averaging the items scores for the items on each sub-scale.

### Stakeholders Satisfaction (SS)

The way IMI was used to evaluate Stakeholders Satisfaction (SS) can be considered a direct way to use the questionnaire. Only the number of questions in the original form has been reduced, which is a practice used by researchers [101]. It was necessary to adjust the instrument according to specific tasks and fields and even to different populations. Thus, since the original IMI is long and repetitive, a short version was used across the following five subcategories: interest/enjoyment (I), effort/importance (E), perceived competence (P), pressure/tension (T), and value/Usefulness (V). The sub-scales of perceived choice (C) and relatedness (R) were eliminated because all team members are employees of the organizations and they will be doing their jobs during the research. The scale was validated for use in software development through Cronbach's Alpha scores. The entire survey was designed to be completed in less than five minutes.

Table 4.2 shows all IMI questionnaire. The scale includes 8 control questions that are scored in reverse, indicated in the table in column 'Tp' with the 'R'. The subcategories are indicated in the table in the column 'Sc'. The actual questionnaire has not exhibited the indication of reverse scored 'R' neither the initials of the subcategories.

As the case study environment occurred in a Brazilian organization, the questionnaire is in Portuguese. Table 4.2 presents the questions in English and in Portuguese. When the team members answered the questionnaire, the question was as followed: "Please indicate how much of the following statements are true for you by using the following scale of integers: from 1 = 'strongly disagree' to 7 = 'fully agree' (*"Por favor, indique o quanto as afirmações a seguir são uma verdade para você, usando a seguinte escala de números inteiros: de 1 = 'Discordo totalmente' a 7 = 'Concordo totalmente'*").

Figure 4.1 shows the moment when the metrics were collected. The baseline to the Error Code (EC) metrics, the Knowledge in Agile (KA) metrics, and the Team Motivation (TM) metrics occurred at the beginning of the process, in the first week. The baseline for the Stakeholders Satisfaction (SS) metric occurred at the beginning of the project.

The next collections of metrics were done as follows. The metric for EC was collected at the end of each sprint, generated automatically by the set of tools used in the development environment (figure 4.2).

The metrics of KA were collected after the agile training and after the fourth sprint, with the application of the IMI questionnaire. Also after the fourth sprint the semi-structured interview was applied.

| Sc | Tp | English/Portuguese |
|---|---|---|
| I | | I believe that agile software development activity could be of some value to me.<br>*Acredito que a atividade de desenvolvimento ágil de software poderia ter algum valor para mim.* |
| I | | I believe that I had some choice in participating in agile software development.<br>*Acredito que tive alguma possibilidade de escolha no tocante a participar do desenvolvimento ágil de software.* |
| I | R | While participating in the development activity, I realized how much she liked me.<br>*Enquanto participava da atividade de desenvolvimento, eu me dei conta do quanto ela me agradou.* |
| I | R | I think performing agile software development activity is useful for improving my finalist activities.<br>*Acho que realizar a atividade de desenvolvimento ágil de software é útil para melhorar as minhas atividades finalísticas.* |
| I | | It was fun to participate in the development activity.<br>*Foi divertido participar da atividade de desenvolvimento.* |
| I | | I think participating in agile software development is important to my professional growth.<br>*Acho que participar do desenvolvimento ágil de software é importante para meu crescimento profissional.* |
| I | | I enjoyed doing this activity very much<br>*Tive muito prazer em realizar esta atividade.* |
| P | | I really had no choice about performing agile software development activity.<br>*Realmente não tive escolha sobre realizar a atividade de desenvolvimento ágil de software.* |
| P | | I did this development activity because I wanted to.<br>*Realizei esta atividade de desenvolvimento porque queria.* |
| P | | I think agile software development is important.<br>*Acho que o desenvolvimento ágil de software é importante.* |
| P | | While I was doing this activity, I was thinking about how much I enjoyed it.<br>*Eu senti como se estivesse gostando da atividade enquanto eu a estava fazendo.* |
| P | | I found the agile software development activity monotonous.<br>*Achei a atividade de desenvolvimento ágil de software monótona.* |
| P | R | I believe that participating in agile development could help me to improve my work.<br>*Creio que participar do desenvolvimento ágil poderia me ajudar a melhorar meu trabalho.* |
| E | | I felt like it was not my own choice to do this task. (R)<br>*Senti que não tinha escolha na realização desta atividade.* |
| E | R | I would describe agile development activity as very interesting.<br>*Eu descreveria a atividade de desenvolvimento ágil como muito interessante.* |
| E | | I would be willing to do this again because it has some value to me.<br>*Eu estaria disposto(a) a realizar esta atividade de novo, porque ela tem algum valor para mim.* |
| E | | I found the activity of agile software development very pleasant.<br>*Achei a atividade de desenvolvimento ágil de software bastante prazerosa.* |
| E | R | I felt that I should perform this agile development activity.<br>*Senti que deveria realizar esta atividade de desenvolvimento ágil.* |
| T | R | I believe that participating in agile software development could help me.<br>*Acredito que participar do desenvolvimento ágil de software poderia me ajudar.* |
| T | | I did this activity because I had to do it.<br>*Realizei esta atividade porque tinha que realizar.* |
| T | R | I believe that carrying out the development activity could help me do my job better.<br>*Acredito que realizar a atividade de desenvolvimento poderia me ajudar a fazer melhor o meu trabalho.* |
| T | | I believe I had some choice about doing this activity.<br>*Ao fazer esta atividade, senti que tinha uma escolha.* |
| T | | I would describe agile software development activity as very fun.<br>*Eu descreveria a atividade de desenvolvimento ágil de software como muito divertida.* |
| T | | I felt like it was not my own choice to do this task. (R)<br>*Eu senti que não era minha escolha fazer esta atividade.* |
| T | R | I am willing to participate in agile development again because I think it is useful.<br>*Estou disposto a participar do desenvolvimento ágil novamente porque acho que é útil.* |

Table 4.2: IMI - Stakeholders Satisfaction (SS)

The TM metrics were collected at the end of the first and fourth sprint. They were collected using the IMI questionnaire. Also after the fourth sprint the semi-structured interview was applied.

The metrics of SS were collected at the end of the second and fourth sprint. They were collected using the respective IMI questionnaire. Also after the fourth sprint was applied the semi-structured interview.

## 4.5 Analysis

In fact, the analysis phase was performed along with the measurement phase, but for a better explanation, it was treated here as a separate phase.

After the measurements, the raw data was available to the analyses. To Error Code (EC), the four metrics that were calculated were Code Error Density (CED), Weighted Code Error Density (WCED), Development Error Density (DED), and Weighted Development Error Density (WDED). These metrics were presented in the 2.5 section. For the metrics Knowledge in Agile (KA), Team Motivation (TM) and Stakeholders Satisfaction (SS), the result of the application of the questionnaires and the semi-structured interview had to be tabulated. More details of the analyses are addressed in the following subsections.

### 4.5.1 Error analysis

As seen in the 2.3 section the Quality Assurance (QA) can present a plethora of different concepts, but, in this work, it was used the concept related to the process of software development and the evaluation of the product in terms of its errors. In the next subsections the metrics selected to this end are analyzed, CED, WCED, DED and WDED showed in the 2.5 section.

#### Code error analysis

To calculate CED and WCED the data concerns the issues. CED used the issues' data in a natural way and the WCED with the following weights according to their severity: Minor (1), Major (3), Critical (6), and Blocker (9).

The graphs in figure 4.3 show the measurement over time of the case study, where it is possible to see the evolution of the calculated metrics. The red marker, at the beginning of both graphs, represents the baseline, that is, the measurement of the same metrics for another project, which works with SOA but did not use agile methods for development.

As it is possible to see in the CED graph, the project starts with 44.1 CED, 15 points above the CED baseline of 29.1 CED. This initial CED held for some period and then wobbled and ended the case study at zero.

In the WCED graph, figure 4.3, the project starts with 88.1 WCED, 75.2 points under the WCED baseline of 163.3 WCED. This initial WCED held for some period and then wobbled and ended the case study at zero too, like CED, as the raw data were the same.

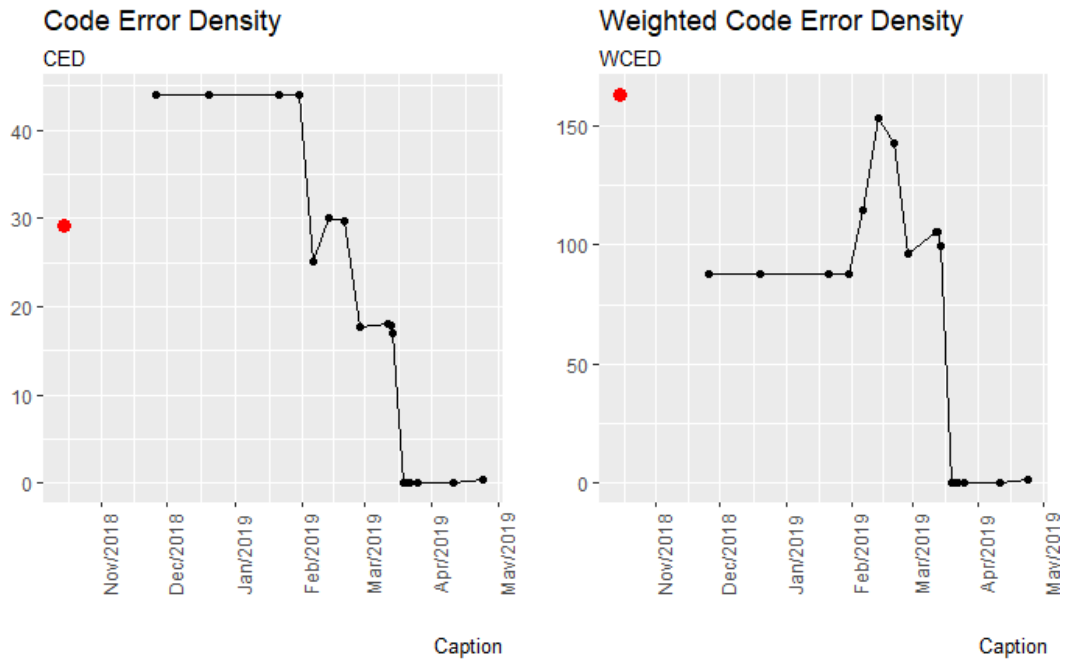


Figure 4.3: Code error graphs

While both graphs show similar lines, the WCED graph allowed for a better analysis, considering the severity of each error. At a glance on the CED graph, one might think that the code starts with many problems compared to the baseline. Well, it is true, but with the help of the WCED graph, it is possible to assess that the problems, though larger, were of lesser severity.

Although the case study followed only a few weeks of the entire project, these two analyzes demonstrated that during the period, Quality Assurance (QA), measured here by Code Error Density (CED) and Weighted Code Error Density (WCED) was not compromised by the application of the NatVi framework, on the contrary, after the application of the framework, after a period of adaptation, the metrics improved.

### Development error analysis

To calculate Development Error Density (DED) and Weighted Development Error Density (WDED) it was used the data for code items that the developer needed to worry about.

These data were related to duplication of code's lines, Cyclomatic Complexity, and lines not covered by unit tests. DED used these data in a natural way and WDED with the following weights according to to the probability of their occurrence in the project: line code duplication (1), Cyclomatic Complexity (2) and lines to cover (3). This probability was calculated based on historical data of other projects of DoCS. This idea was drawn from the work of Porter and Selby [109], although in simplified form, where the lowest probability item received weight 1, the next item in terms of probability received the next weight 2 and so on.

As in the subsection above, the graphs in figure 4.4 show the measurement during the time of the case study and the red marker represents the baseline for the comparison.

As it is possible to see in the DED graph, the project starts with 96.9 DED, 558.1 points under the DED baseline of 655 DED. this initial DED held for some period and then wobbled and ended the case study with 168.9 DED.

In the WDED graph, figure 4.4, the project starts with 255.5 WDED, 1297.5 points under the WDED baseline of 1553 WDED. This initial WCED held for some period and then wobbled and ended the case study with 369.8 WDED.

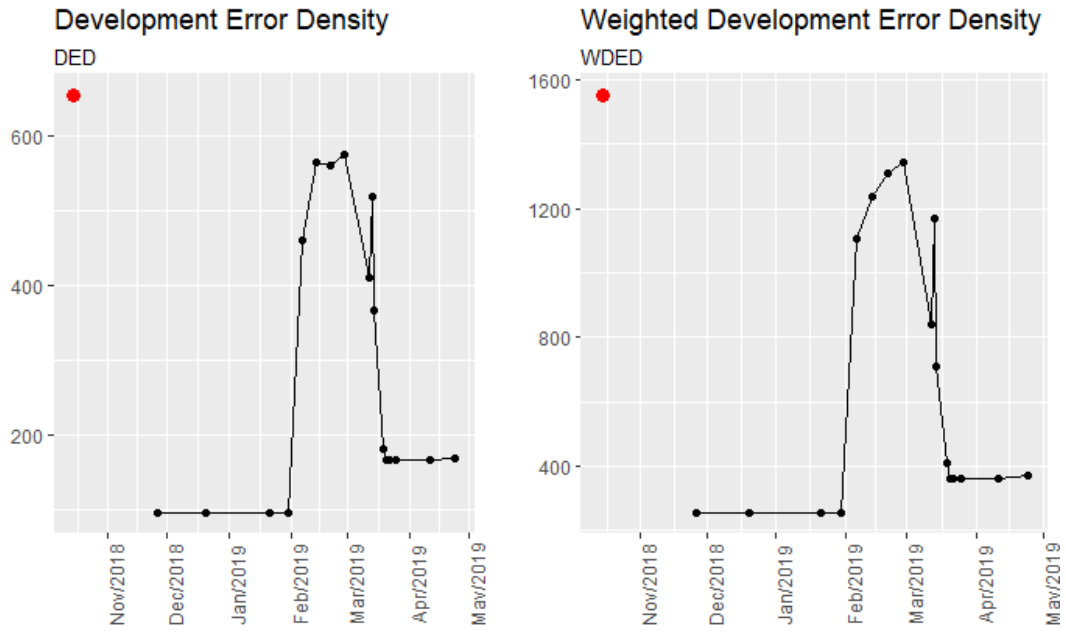


Figure 4.4: Development error graphs

As in the analysis in subsection above, both graphs show a similar line and, likewise, the weighted graph helps to enlighten the situation. Here again, the analyses demonstrated that during the period, Quality Assurance (QA) was not compromised by the application of the NatVi framework and, by the result of the metrics, QA even improved.

### 4.5.2 Knowledge in agile (KA) analysis

As seen in the subsection 4.4.2 above, after the training in agile, during the course of the case study, the agile development knowledge of the IT team was assessed. To achieve this, the Knowledge in Agile (KA) indicator was used through the application of one questionnaire and a semi-structured interview.

Both the questionnaire and the semi-structured interview was done based on the paper by Chow and Cao [107] where the authors raised the possible success factors in the implementation of agile development in four dimensions, quality, time, scope and cost. In the result of that work, the authors concluded that only 6 factors of the 12 initial factors are actually success factors in an agile development project. Table 2.7 shows a summary of these 12 factors.

The questionnaire was applied twice. Table 4.3 shows the average result of the 12-factor questionnaire for the two periods, called 'Initial' and 'Final' periods. In the table, the factors number in gray-box are those 6 factors considered actually success factors in software development projects in the result of the study of Chow and Cao [107]. The rank they appear in Chow and Cao [107] is in the row 'Rank'. The averages in bold font are that 6 larger for each period. The averages in the gray-box are the two largest for each period.

As is possible to see in table 4.3 factors 5 and 9 were first. They presented average for the initial period equal to 6.35. In the final period, factor 5 remained in the first place with a average of 6.5 and in the second place factor number 3 appeared with average equal 6.45.

| Period | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Initial | 5.75 | 5.40 | **5.80** | **6.20** | **6.35** | **6.05** | 5.10 | **6.10** | **6.35** | 4.00 | 4.00 | 4.50 |
| Final | **6.25** | 5.65 | **6.45** | **6.20** | **6.50** | 6.05 | **6.20** | 5.90 | 6.00 | 3.10 | 4.75 | 3.75 |
| Rank | | | 5 | 3 | 6 | 4 | | 2 | 1 | | | |

Period in gray-box: the 6 final factors considered actually by success in [107]
Average in **bold font**: the 6 larger average in each period
Average in light gray-box: the two larger average in each period
Rank: the final rank of the 6 factors considered actually by success in [107]

Table 4.3: Factors by period - general mean

Comparing the 2.7 table and the 4.3 table it is possible to see that in this case study the same 12 factors that Chow and Cao [107] have been used in their work were exposed to the IT team. The IT team average in this paper bears similarities to the work of Chow and Cao [107]. In the initial period, the similarities were better where all 6 best results were combined with the final success factor 6.

The graph 4.5 shows the general average of the 12 factors, that is, considering both periods analyzed. With this graph it is possible to compare the average of the IT team into the dimensions of the work by Chow and Cao [107] for the final 6 success factors. Based on this graph,the table 4.4 was elaborated. In this table, the average of the Knowledge in Agile (KA) metrics in this work was classified and compared with the 6 final factors and their dimensions. The gray-box is just to enlighten the classification in comparison with the affected dimensions.

According to the table 4.4, none of the 6 analyzed factors corresponds completely. Factor number 9, the first in the [107] rank, has a great similarity. In [107] this factor is affected by scope, time and cost dimensions. In this work, time dimension appears in first place and scope and cost tied in third place. The quality dimension, which took second place, is not considered a success dimension in  cite chow2008survey.

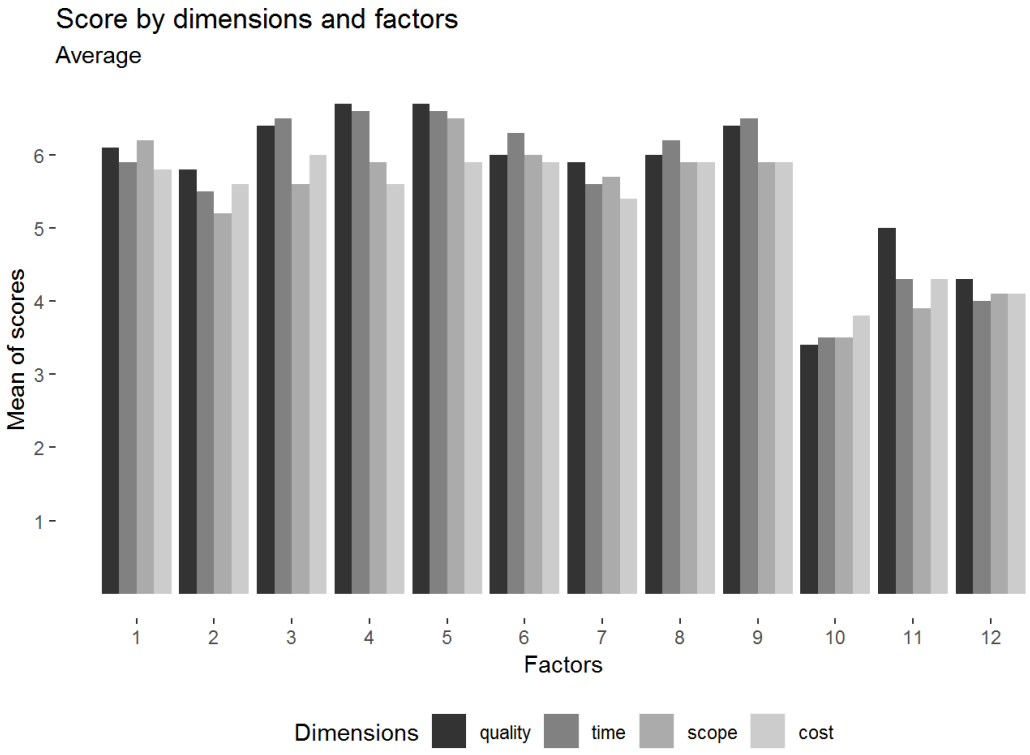


Figure 4.5: Score by factors and dimensions - general average

Factors 3 and 6, that are affected only by the quality dimension in Chow and Cao [107], had this dimension in second place in this work. In this work, the time dimension took first place. Factor 8, affected only by the quality and scope dimension, had these two dimensions in second and third place, respectively in this work. Time dimension took the first place. Factors 4 and 5 are those that fit less. Although the classification does not completely match, it is possible to see similarities.

| Factor | Rank | Dimension | Mensurement | | | |
|---|---|---|---|---|---|---|
| | | | Q | S | T | C |
| 3 | 5 | Q | 2 | 4 | 1 | 3 |
| 4 | 3 | T/C | 1 | 3 | 2 | 4 |
| 5 | 6 | S | 1 | 3 | 2 | 4 |
| 6 | 4 | Q | 2 | 2 | 1 | 4 |
| 8 | 2 | Q/S | 2 | 3 | 1 | 3 |
| 9 | 1 | S/T/C | 2 | 3 | 1 | 3 |

Measurement: classification of the average of dimensions in this work

Dimensions (Chow and Cao [107]):
**Q**uality / **S**cope
**T**ime / **C**ost

Table 4.4: Average dimension classification

This similarities in both analyses, general average by factors and general average by dimensions and factors, demonstrated that the answers of the IT team, on average, had good adequacy to the Chow and Cao [107] research. This may mean that the IT team had a good understanding of agile development methods. Since this is the first case study, the IT team is small and the time spent was little, expanding this study might be a good future study.

### Knowledge in agile (KA) - semi-structured interview analysis

Analyzing the semi-structured interview, the results seem similar to the results of the questionnaire, although some members understand that everything is still very initial.

The factors of numbers 3, 4 and 5 were cited as the most important for the 'Quality' and 'Time' dimensions by several of the interviewees. Compared to the baseline study [107], some discrepancies are perceived. In that study, the factor 3 is concerned with only the 'Quality' dimension, the factor 4 with 'Time' and 'Cost' dimensions and the factor 5 only with 'Scope' dimensions. Despite this, some quotes were interesting to emphasize the thoughts of the interviewees. For example, related to 'Quality', to factor 3 (The existence of agile-friendly project team environment), it was said that it would be very difficult to achieve quality if there is a member who is not willing or interested in the implementation of the methodology. Related to 'Time', to factor 3, in the face of problems, if the team is not aligned with the methodology, it can take time to get everyone to speak the same language, which will directly affect the project time. To factor 5 (Having a strong customer involvement), in 'Quality', it was said that this is one of the main ideas of Scrum, the agile method applied. Therefore, for product quality, certainly customer involvement is very important.

For the 'Scope' dimension, highlight the factor number 5 (Having a strong customer involvement) that is strongly related to the result of the study [107] used as the basis of this work. In that work, this factor number 5 is among the final success factors and precisely in the 'Scope' dimension. Ultimately the project is to serve the clients, so the scope has to be born of them.

To the 'Cost' dimension, highlight the factor number 3 (The existence of agile-friendly project team environment), which also presents a discrepancy in relation to the base study [107], because in that study, only the quality dimension affects this factor of number 3. It seems that respondents had a greater difficulty analyzing the factors in the 'Cost' dimension. Apparently because they do not know how to estimate the cost of the project. It may be a feature of the context of the case study, that is, a unit of government.

In general, it can be seen that the interviewees showed that they are interested in understanding the new methodology. At some points they eventually agreed to the results of the Chow and Cao [107] study, although in several cases they did not agree. But that does not mean they are wrong. The divergences may be due to the characteristics of the context.

### 4.5.3 Team motivation analysis

In 4.4.2 section was presented the Intrinsic Motivation Inventory (IMI) questionnaire for obtaining answers to assess the Team Motivation (TM). Besides the presented questionnaire a semi-structured interview was also done based in the questions of the questionnaire. In this subsection was analyzed what happened with the motivation of the IT team during the course of the case study.

Before the application, the TM IMI questionnaire was tested and presents a Cronbach's coefficient alpha of 0.80. In table 4.5 is possible to see the statistics values of this test in row 'General - Test'.

The data actually collected presents good reliability with a raw Cronbach's Alpha at 0.9085 as it is possible to see in table 4.5 in row 'General'. An overview of the correlation can be seen in the 'Correlogram' (a diagram that shows the correlation among items) depicted in figure 4.6. Some pairs of questions presented negative correlation, but, in general, the correlation was acceptable for this work.

In table 4.5 the general mean is 5.49 with a standard deviation of 0.78. This indicates that the motivation of the IT team with the news activities involving Service-Oriented Architecture (SOA) and agile development is good. But, in graph in figure 4.7 some questions related mainly with subcategories pressure/tension and relatedness got low mean. In fact, during the semi-structured interview, the main concern of the IT team was about pressure to deliver software in time.

| Motivator | raw_alpha | std.alpha | average_r | mean | sd | median_r |
|---|---|---|---|---|---|---|
| General - Test | 0.8043 | 0.7859 | 0.1775 | 4.6176 | 0.5029 | 0.2425 |
| General | 0.9085 | 0.9137 | 0.3838 | 5.4941 | 0.7833 | 0.4564 |
| Process/Life cycle | 0.8744 | 0.9026 | 0.6494 | 5.0800 | 0.8786 | 0.7335 |
| Teamwork | 0.3571 | 0.3931 | 0.1776 | 4.4667 | 0.8367 | 0.2440 |
| Good management | 0.6154 | 0.7773 | 0.4659 | 4.7500 | 0.9014 | 0.6179 |
| Identify with task | 0.3871 | 0.3057 | 0.1280 | 4.4000 | 0.8300 | 0.1111 |

**raw_alpha** - alpha based upon the covariances;
**std.alpha** - The standardized alpha based upon the correlations
**average_r** - The average inter item correlation
**mean** - The general mean; **sd** - The standard deviation of each item
**median_r** - The median inter item correlation

Table 4.5: Team Motivation (TM) Statistics

Continuing to look at the table 4.5, the Cronbach's Alpha by motivators shows some fluctuation. Cronbach's Alpha was only acceptable to the 'Process / Life Cycle' (0.8744) and 'Good management' (0.6154) groups and the mean of this two groups are 5.08 and 4.75 respectively, which indicates that the motivation for these groups of motivators is also good. Although the mean of the other groups also indicates good motivation, since Cronbach's Alpha is less than 0.6, it is difficult to make any inference.

### Team motivation (TM) - semi-structured interview analysis

In this section the results of the semi-structured interview are analyzed. The questions were elaborated based in the TM IMI questionnaire questions. In general, the results of this semi-structured interview seem similar to the results of the questionnaire. This interview was carried out with four developers and one trainee.

In the 'Interest/Enjoyment' sub-scale, only one of the respondents did not show much interest in the new process. Some interviewees considered the new process a challenge, in a good way.

Related to 'Perceived Competence' sub-scale, the feeling of satisfaction with the execution of the new tasks was almost unanimous, although some have mentioned that the process is still at the beginning and much needs to be explored until a more mature evaluation.

In the 'Effort/Importance' sub-scale, despite a general feeling of recognition of the importance, many have mentioned that it is too early to really assess the real importance of the process.

Related to 'Pressure/Tension' sub-scale, only two respondents said they felt a little more pressure on the old mode of development. They mentioned that there is a learning curve that needs to be addressed to improve knowledge and performance on tasks.

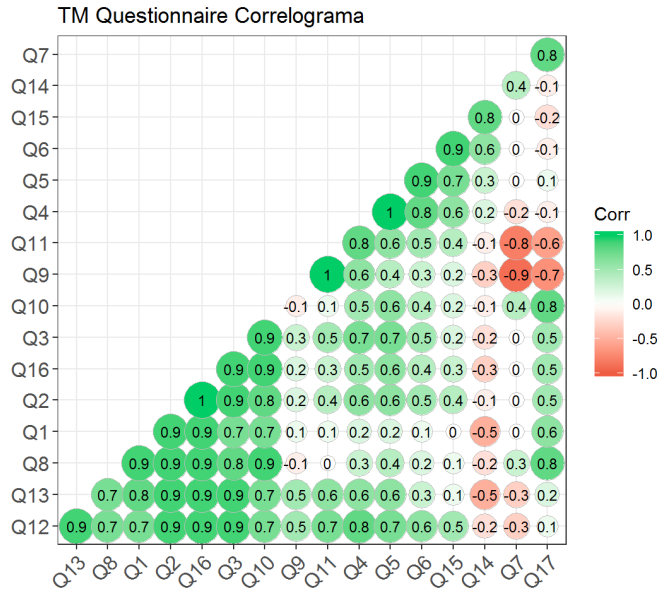


Figure 4.6: Team Motivation (TM) Questionnaire Correlogram

In the 'Value/Usefulness' sub-scale, all respondents agreed to consider the usefulness of the new process.

Related to 'Relatedness' sub-scale, the answers of the respondents follow three different ways. Two interviewees did consider themselves very close and within the group, two other did express feelings in the opposite direction and one interviewee did not know how to express on this point.

Despite being a semi-structured interview, following the same concepts of the questionnaire, some interesting subjects emerged beyond the questionnaire. As already mentioned, some interviewees considered the new process a challenge, in a good way. The new methodology has brought more convenience, more ease of definition with the team, to define what each one will do and the deadlines for delivery. In case of difficulties someone has, it is easier to find someone with experience to help solve the problem. Things get clearer, more organized in the context of the team. Being able to meet daily and be able to tell what was done and what is having difficulty making and receiving help. Periodically, finish a well-defined step and start a new step. Better synergy with people in the business areas by better understanding the meaning of what is being done.

### 4.5.4 Stakeholders satisfaction analysis

In 4.4.2 section was presented the Intrinsic Motivation Inventory (IMI) questionnaire for getting answers to assess the Stakeholders Satisfaction (SS). Besides the presented questionnaire a semi-structured interview was also done based in the questions of the question-

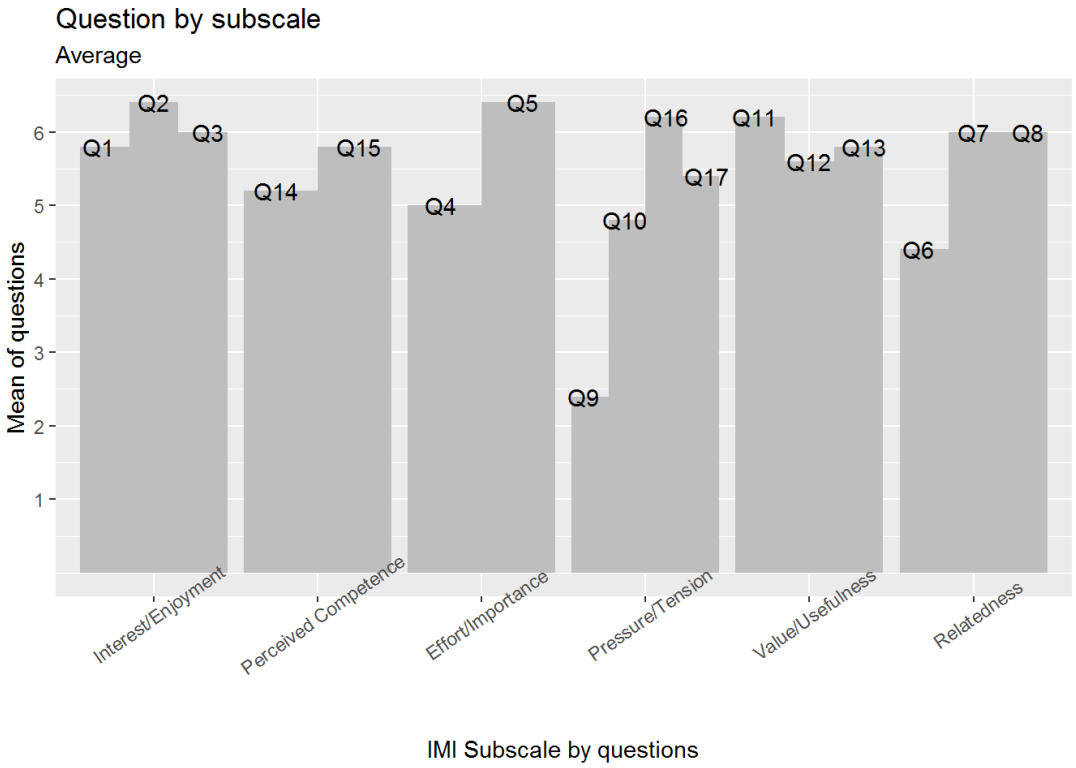


Figure 4.7: Team Motivation (TM) Question by sub-scale

naire. In this section, stakeholder satisfaction was analyzed by assessing its motivation during the course of the case study.

The IMI questionnaire was tested and presents a Cronbach's coefficient alpha of 0.85. In table 4.6 is possible to see the statistics values of this test in row 'Test'.

The data actually collected presents good reliability with a raw Cronbach's Alpha at 0.9223 as it is possible to see in table 4.6 in row 'General'. An overview of the correlation can be seen in the 'Correlogram' depicted in figure 4.8. Some pairs of questions presented negative correlation, but, in general, the correlation was acceptable for this work. The question 'Q24' (I felt like it was not my own choice to do this task. (R)) had no variance and needed to be excluded from the analysis.

In table 4.6 the general mean is 5.50 with a standard deviation of 0.69. This indicates that the motivation of the stakeholders with the news activities involving Service-Oriented Architecture (SOA) and agile development is good. But, in graph in figure 4.9 some questions related mainly with subcategory pressure/tension got low mean.

### Stakeholders Satisfaction (SS) - semi-structured interview analysis

In this section the results of the semi-structured interview are analyzed. The questions were elaborated based in the IMI questionnaire questions. In general, the results seem

| Motivator | raw_alpha | std.alpha | average_r | mean | sd | median_r |
|---|---|---|---|---|---|---|
| Test | 0.8518 | 0.8483 | 0.1889 | 5.5139 | 0.447 | 0.2176 |
| General | 0.9223 | 0.9232 | 0.3337 | 5.5052 | 0.6906 | 0.3936 |

**raw_alpha** - alpha based upon the covariances;
**std.alpha** - The standardized alpha based upon the correlations
**average_r** - The average inter item correlation
**mean** - The general mean; **sd** - The standard deviation of each item
**median_r** - The median inter item correlation

Table 4.6: Stakeholders Satisfaction (SS) Statistics

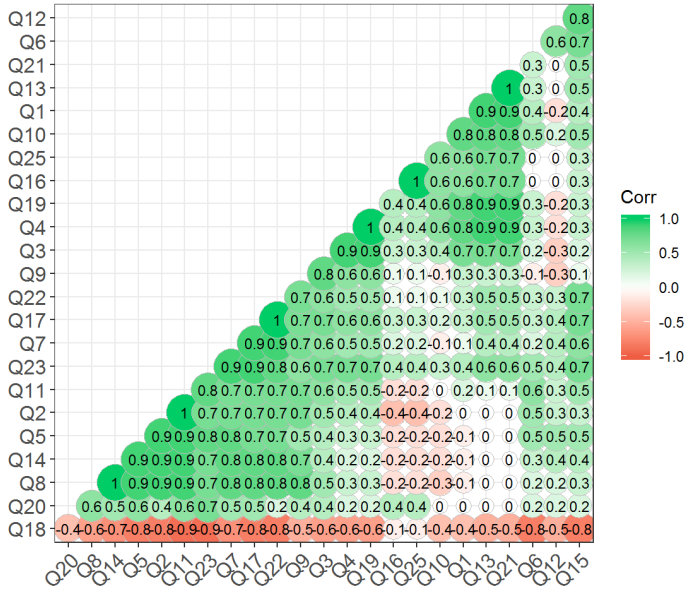


Figure 4.8: Stakeholder Satisfaction (SS) Questionnaire Correlogram

similar to the results of the questionnaire. This interview was conducted with two clients, who were the ones who participated most in the project.

Related to the 'Interest/Enjoyment' sub-scale, respondents expressed that the new tasks aroused great interest and are very hopeful with the project results. For example, participation in the activity was considered important because the agile methods advance in relation to the way it was done before, allowing the results of the work to be seen in a faster and clearer way.

In the 'Pressure/Tension' sub-scale, respondents demonstrated that the new tasks did not bring more pressure than they were accustomed to. It was said that although the new activities had incited them to leave the comfort zone, they did not feel the pressure. They adapted well to the process by realizing the benefits when they saw the results start to come out faster than they used to be.

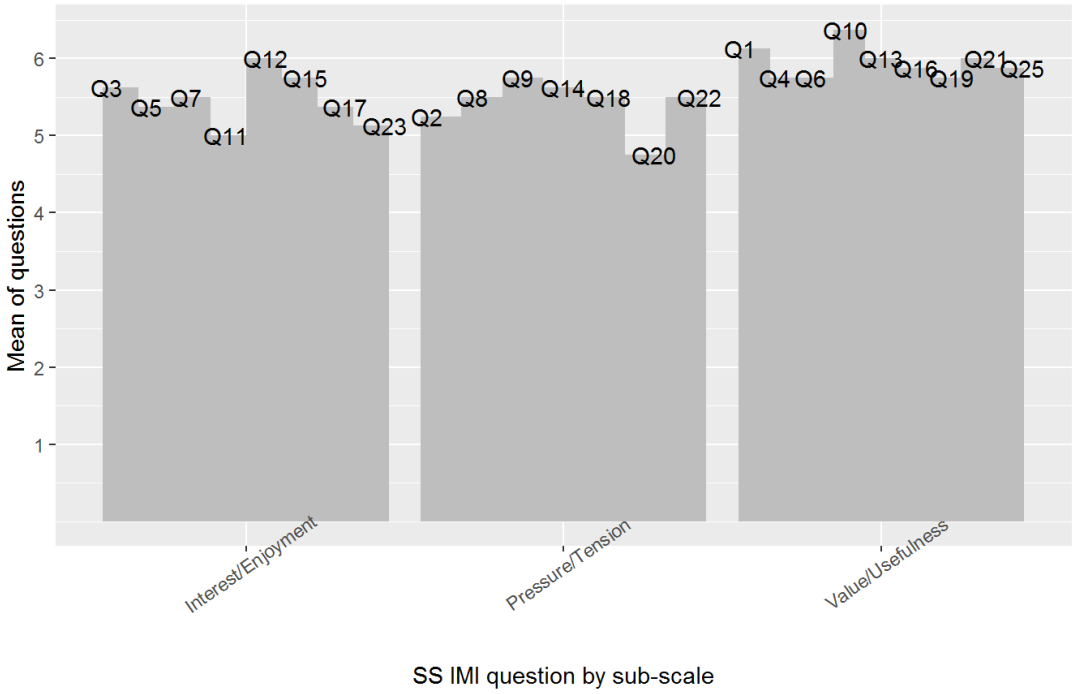


Figure 4.9: Stakeholders Satisfaction (SS) questions mean by sub-scale

Related to the 'Value/Usefulness' sub-scale, the respondents expressed that they consider the new tasks very useful and showed some confidence that the results will have a positive impact on their final activities. For example, it was said that if with the agile methodology the results really are faster, the work of the final areas also develops more. So it is totally useful because it will improve the effectiveness of the final activities.

Although it was a semi-structured interview, following the same concepts of the questionnaire, some interesting subjects emerged beyond the questionnaire. It was mentioned that the new process is innovative, where there is no hierarchy among the areas involved but rather a group that works together. This improves knowledge and professional life, making them deal with situations that go beyond normal work situations. A two-way path where clients can influence the pace and priorities of software development.

# Chapter 5

# Conclusions and future work

The initial intention of this research was to identify a solution to better deal with developing applications with quality and rapid deliveries in a distributed software environment that meet business objectives in constant change.

In this way three objectives were presented that this work sought to meet. The specific objective 1 (*identifying the existing trade-offs between SOA and agile development methods and best practices to deal with them*) was addressed in the 2.4.1 section where a systematic literature review (SLR) was made to identify what has already been researched on these two concepts together, SOA and Agile Methods of development. It was seen in that section that trade-offs exist, primarily with respect to the complexity of SOA and the lack of well-defined processes in Agile Development Methods. But it was identified in the literature, papers that addressed the trade-offs. This work sought to include the solutions presented in those paper in the NatVi framework.

The specific objective 2 (*building a framework (principles, actions, and best practices) that enable agile development methods in a SOA scenario*) was discussed in chapter 3 where the NatVi framework elaborated based on the literature review was presented.

The specific objective 3 (*applying the framework in a case study, verify, evaluate and discuss the results, focusing mainly on QA*) was presented in chapter 4 where the case study was performed. In that chapter, there is the presentation of the context in which the case study was carried out and the presentation of the main parameters used, as well as the analyzes made.

The case study presented some problems, mainly related to the context in which it was carried out. There were some aspects of the environment that led to problems, such as delays in some steps that ultimately compromised the outcome of the case study to some degree. Another aspect is related to the size of the team involved. Few stakeholder members got involved in the case study and the small size of the IT team brought problems in terms of statistical inferences.

When working in this kind of context, a better plan needs to be done, considering the possible bureaucracy that can be faced. With regard to the number of personnel involved, it is necessary to do a better job of engagement. All of this can be seen as lessons learned. But despite these problems, the case study came to an end, with some adaptations, such as the introduction of a semi-structured interview not foreseen at the start of the study.

As a contribution, this paper presents a framework that can work in an environment with Service-Oriented Architecture (SOA) and Agile Development Methods together, while at the same time worrying about Quality Assurance (QA). The study indicates that framework phases that addresses the principles of SOA and agile development, as well as in phases that emphasize the tests, can lead to a better QA. In addition to those phases of the framework that straight address QA, the study further indicates that the motivation of the personnel involved and the constant increase in knowledge of SOA and especially of Agile Methods can also lead to a better QA.

## 5.1 Future works

In this work the framework was evaluated by the results of its application, but the use of the framework itself was not evaluated. This is future work that can be done in another case study applying the framework.

The study of the literature indicates that the selected subject is a concern in several environments and that there is a lack in this area of knowledge. Other studies need to be done. This same study, for example, can be done in a context with more people involved.

Besides the concepts explored in this work, other concepts were identified during the literature review that are probably part of the context presented, such as DevOps, Test Driven Development (TDD), Behaviour Driven Development (BDD), and Microservice architecture design pattern. These concepts could not be addressed in this paper because they would make the work very large and it would be impractical to conclude. But future work on these concepts seems to be a good way forward in the studies presented here.

# References


[1] Thomas Erl. *Soa: principles of service design*, volume 1. Prentice Hall Upper Saddle River, 2008. 2, 6, 7, 9

[2] Agile Alliance. Agile manifesto. *Online at http://www. agilemanifesto. org*, 6(1), 2001. 2, 13

[3] Raoul Vallon, Bernardo José da Silva Estácio, Rafael Prikladnicki, and Thomas Grechenig. Systematic literature review on agile practices in global software development. *Information and Software Technology*, 96:161–180, 2018. ISSN 0950-5849. doi: https://doi.org/10.1016/j.infsof.2017.12.004. URL `http://www.sciencedirect.com/science/article/pii/S0950584917302975`. 2, 14

[4] Daniel Galin. *Software quality assurance: from theory to implementation*. Pearson Education India, 2004. ISBN 0201709457. URL `http://books.google.com/books?hl=en&lr=&id=p5jDETUc2K8C&oi=fnd&pg=PR17&dq=Software+Quality+Assurance:+From+Theory+to+Implementation&ots=OJJtBcHL2B&sig=fnD_Z9APND4DEQs3Uak19mKk7R0`. 3, 15, 22, 23, 51

[5] P. Runeson and P. Isacsson. Software quality assurance-concepts and misconceptions. *Proceedings. 24th EUROMICRO Conference (Cat. No.98EX204)*, 2:853–859, 1998. ISSN 1089-6503. doi: 10.1109/EURMIC.1998.708112. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=708112`. 3

[6] Barbara Kitchenham. Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33(TR/SE-0401):28, 2004. doi: 10.1.1.122.3308. 4

[7] James Taylor Faria Chaves and Sergio Antônio Andrade de Freitas. A Systematic Literature Review for Service-Oriented Architecture and Agile Development. In *International Conference on Computational Science and Its Applications*, pages 120–135, Saint Petersburg, Russia, 2019. doi: https://doi.org/10.1007/978-3-030-24308-1{_}11. 4, 16

[8] Len Bass. *Software architecture in practice*. Pearson Education India, 2007. 6, 7

[9] Barry Demchak, Claudiu Farcas, Emilia Farcas, and Ingolf H. Krüger. The treasure map for Rich Services. *2007 IEEE International Conference on Information Reuse and Integration, IEEE IRI-2007*, pages 400–405, 2007. doi: 10.1109/IRI.2007.4296653. 6, 7, 9, 17, 18, 19

[10] P Krogdahl, G Luef, and C Steindl. Service-oriented agility: an initial analysis for the use of agile methods for SOA development. In *2005 IEEE International Conference on Services Computing (SCC'05) Vol-1*, volume 2, pages 93–100, 2005. doi: 10.1109/SCC.2005.86. 6, 10, 17, 18

[11] Michael Zur Muehlen, Jeffrey V. Nickerson, and Keith D. Swenson. Developing web services choreography standards - The case of REST vs. SOAP. *Decision Support Systems*, 40(1 SPEC. ISS.):9–29, 2005. ISSN 01679236. doi: 10.1016/j.dss.2004.04.008. 6

[12] Charles Severance. Roy T. Fielding: Understanding the REST Style. *Computer*, 48 (6):7–9, 2015. ISSN 00189162. 6

[13] S. S. B. Shi, E. Stokes, D. Byrne, C. F. Corn, D. Bachmann, and T. Jones. An enterprise directory solution with DB2. *IBM Systems Journal*, 39(2):360–383, 2000. ISSN 0018-8670. doi: 10.1147/sj.392.0360. URL `http://ieeexplore.ieee.org/ielx5/5288519/5386991/05387000.pdf?tp=&arnumber=5387000&isnumber=5386991`. 7, 8

[14] Djilali Idoughi, Moussa Kerkar, and Christophe Kolski. Towards new web services based supervisory systems in complex industrial organizations: Basic principles and case study. *Computers in Industry*, 61(3):235–249, 2010. 7

[15] Erik Meijer and Gavin Bierman. A co-relational model of data for large shared data banks. *Communications of the ACM*, 54(4):49, 2011. ISSN 00010782. doi: 10.1145/1924421.1924436. URL `http://portal.acm.org/citation.cfm?doid=1924421.1924436`. 7

[16] Hamza Chehili, Lionel Seinturier, and Mahmoud Boufaida. FASOAD: A framework for agile service-oriented architectures development. *Proceedings - International Workshop on Database and Expert Systems Applications, DEXA*, pages 222–226, 2013. ISSN 15294188. doi: 10.1109/DEXA.2013.28. 7, 13, 17, 18

[17] Philip Bianco, Rick Kotermanski, and Paulo F Merson. Evaluating a Service-Oriented Architecture. *Research Showcase @ CMU*, 1(September), 2007. ISSN 03636143. URL `http://repository.cmu.edu/sei`. 7, 10

[18] Barry Demchak, Vina Ermagan, Emilia Farcas, To-ju Huang, Ingolf H Krüger, and Massimiliano Menarini. A Rich Services Approach to CoCoME. In *The Common Component Modeling Example*, number January, pages 85–115. 2015. ISBN 7612228890. doi: 10.1055/b-0035-108273. URL `http://link.springer.com/chapter/10.1007/978-3-540-85289-6_5`. 7

[19] Felipe Carvalho and Leonardo Guerreiro Azevedo. Service agile development using XP. *Proceedings - 2013 IEEE 7th International Symposium on Service-Oriented System Engineering, SOSE 2013*, pages 254–259, 2013. doi: 10.1109/SOSE.2013.25. 7, 10, 13, 14, 17, 18, 19

[20] Majlesi Shahrbanoo. An Approach for Agile SOA Development using Agile Principals. *International Journal of Computer Science and Information Technology*, 4(1): 237–244, 2012. ISSN 09754660. doi: 10.5121/ijcsit.2012.4118. 7, 10, 11, 16, 17, 18, 19

[21] Elliot Sloane, Robert Beck, and Sue Metzger. AGSOA - Agile Governance for Service Oriented Architecture (SOA) systems: A methodology to deliver 21st Century military net-centric systems of systems. *2008 IEEE International Systems Conference Proceedings, SysCon 2008*, pages 106–109, 2008. doi: 10.1109/SYSTEMS.2008.4518995. 7, 8, 10

[22] A Arsanjani and K Holley. The Service Integration Maturity Model: Achieving Flexibility in the Transformation to SOA. In *2006 IEEE International Conference on Services Computing (SCC'06)*, page 515, 2006. doi: 10.1109/SCC.2006.104. URL `https://ieeexplore-ieee-org.ez54.periodicos.capes.gov.br/document/4026979/`. 7

[23] Ali Arsanjani, G Booch, T Boubez, P Brown, D Chappell, J DeVadoss, T Erl, N Josuttis, D Krafzig, M Little, and others. The soa manifesto. *SOA Manifesto, October*, page 35, 2009. URL `http://serviceorientation.com/soamanifesto/original`. 8, 9

[24] Paul Clements, David Garlan, Reed Little, Robert Nord, and Judith Stafford. Documenting software architectures: views and beyond. In *Proceedings of the 25th International Conference on Software Engineering*, pages 740–741. IEEE Computer Society, 2003. 8, 10

[25] Philip Bianco and Grace A Lewis. Architecting Service-Oriented Systems. (August), 2011. 8, 10

[26] Sixto Ortiz. Getting on board the enterprise service bus. *Computer*, 40(4):15–17, 2007. ISSN 00189162. doi: 10.1109/MC.2007.127. 8

[27] James Casey, Lionel Cons, Wojciech Lapka, Massimo Paladin, and Konstantin Skaburskas. A messaging infrastructure for WLCG. *Journal of Physics: Conference Series*, 331(PART 6), 2011. ISSN 17426588. doi: 10.1088/1742-6596/331/6/062015. 9

[28] Michael Bennett and Peter Weill. Exploring the use of electronic messaging infrastructure: The case of a telecommunications firm. *Journal of Strategic Information Systems*, 6(1):7–34, 1997. ISSN 09638687. doi: 10.1016/S0963-8687(97)00002-4. 9

[29] Jorge Valenzuela Posadas. 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)*, pages 1–4, 2017. doi: 10.1109/INTERCON.2017.8079698. 10

[30] JarJaroslav KrálMichal Žemlička KrÃ¡l. Support of Service Systems by Advanced SOA. Number 1993, pages 2049–2058, 2013. doi: 10.1007/978-3-642-35879-1{_}9. URL `http://link.springer.com/chapter/10.1007/978-3-642-35879-1_9`. 10

[31] K A Abdelouhab, D Idoughi, and C Kolski. 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)*, pages 1–8, 2014. doi: 10.1109/ICT-DM.2014.6917792. 10, 13, 18, 19

[32] Patricia Lago and Maryam Razavian. A pragmatic approach for analysis and design of service inventories. In *International Conference on Service-Oriented Computing*, pages 44–53. Springer, 2011. URL `http://dare.ubvu.vu.nl/bitstream/handle/1871/33178/A?sequence=2`. 10

[33] Qing Gu and Patricia Lago. A stakeholder-driven service life cycle model for SOA. In *2nd international workshop on Service oriented software engineering: in conjunction with the 6th ESEC/FSE joint meeting*, pages 1–7. ACM, 2007. 10

[34] Stephen Lane and Ita Richardson. Process models for service-based applications: A systematic literature review. *Information and Software Technology*, 53(5):424–439, 2011. ISSN 09505849. doi: 10.1016/j.infsof.2010.12.005. URL `http://dx.doi.org/10.1016/j.infsof.2010.12.005`. 10, 11

[35] David E. Monarchi and Gretchen I. Puhr. A Research Typology for Object-Oreinted Analysis and Design. *Communications of the ACM*, 35(9):35–47, 1992. ISSN 00010782. doi: 10.1145/130994.130995. 11

[36] Alexander Pokahr and Lars Braubach. The active components approach for distributed systems development. *International Journal of Parallel, Emergent and Distributed Systems*, 28(4):321–369, 2013. ISSN 17445760. doi: 10.1080/17445760.2013.785546. 11

[37] Juan Pablo Napoli and Kalinka Kaloyanova. An integrated approach for RUP, EA, SOA and BPM implementation. In *Proceedings of the 12th International Conference on Computer Systems and Technologies*, pages 63–68. ACM, 2011. doi: 10.1145/2023607.2023620. URL `https://dl-acm-org.ez54.periodicos.capes.gov.br/citation.cfm?doid=2023607.2023620`. 11

[38] Yonghong Tian, Yila Su, and Xufei Zhuang. Research on service identification methods based on SOA. In *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*, volume 6, pages 6–27, 2010. doi: 10.1109/ICACTE.2010.5579359. 11

[39] H Sarjoughian, S Kim, M Ramaswamy, and S Yau. A simulation framework for service-oriented computing systems. In *2008 Winter Simulation Conference*, pages 845–853, 2008. doi: 10.1109/WSC.2008.4736148. 11

[40] A kenzi, B El Asri, M Nassar, and A Kriouile. A model driven framework for multiview service oriented system development. In *2009 IEEE/ACS International Conference on Computer Systems and Applications*, pages 404–411, 2009. doi: 10.1109/AICCSA.2009.5069357. 11

[41] Thomas Erl. *Service-oriented architecture: concepts, technology, and design*. Pearson Education India, 2005. 11

[42] A Arsanjani, S Ghosh, A Allam, T Abdollah, S Ganapathy, and K Holley. SOMA: A method for developing service-oriented solutions. *IBM Systems Journal*, 47(3): 377–396, 2008. ISSN 0018-8670. doi: 10.1147/sj.473.0377. URL `http://www.cs.jyu.fi/el/tjtse54_09/Artikkelit/ArsanjaniEtAlIBMSsJ.pdf`. 11

[43] Olaf Zimmermann, Pal Krogdahl, and Clive Gee. Elements of service-oriented analysis and design, 2004. URL `https://www.ibm.com/developerworks/library/ws-soad1/index.html`. 11, 12

[44] Ruth Sara Aguilar-Savén. Business process modelling: Review and framework. *International Journal of Production Economics*, 90(2):129–149, 2004. ISSN 09255273. doi: 10.1016/S0925-5273(03)00102-6. 12

[45] HENK JONKERS, MARC LANKHORST, RENÉ VAN BUUREN, STIJN HOPPENBROUWERS, MARCELLO BONSANGUE, and LEENDERT VAN DER TORRE. Concepts for Modeling Enterprise Architectures. *International Journal of Cooperative Information Systems*, 13(03):257–287, 2004. ISSN 0218-8430. doi: 10.1142/S0218843004000985. URL `http://www.worldscientific.com/doi/abs/10.1142/S0218843004000985`. 12

[46] Deborah Farroha and Bassam Farroha. Developing corporate services in an agile environment. *Proceedings - IEEE Military Communications Conference MILCOM*, pages 1535–1540, 2011. ISSN 1996756X. doi: 10.1109/MILCOM.2011.6127525. 13, 17, 18

[47] J Highsmith and A Cockburn. Agile software development: the business of innovation. *Computer*, 34(9):120–127, 2001. ISSN 0018-9162. doi: 10.1109/2.947100. 13, 14

[48] Steven L Goldman, Roger N Nagel, and Kenneth Preiss. *Agile competitors and virtual organizations: strategies for enriching the customer*, volume 8. Van Nostrand Reinhold New York, 1995. 13

[49] Irum Inayat, Siti Salwah Salim, Sabrina Marczak, Maya Daneva, and Shahaboddin Shamshirband. A systematic literature review on agile requirements engineering practices and challenges. *Computers in Human Behavior*, 51:915–929, 2015. ISSN 0747-5632. doi: https://doi.org/10.1016/j.chb.2014.10.046. URL `http://www.sciencedirect.com/science/article/pii/S074756321400569X`. 14

[50] Tore Dybå and Torgeir Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9):833–859, 2008. ISSN 0950-5849. doi: https://doi.org/10.1016/j.infsof.2008.01.006. URL `http://www.sciencedirect.com/science/article/pii/S0950584908000256`. 14

[51] O. Timperi. An Overview of Quality Assurance Practices in Agile Methodologies. *Soberit.Hut.Fi*, 650, 2004. 14, 15, 21, 22

[52] Usman M Malik, Haseeb M Nasir, and Ali Javed. An Efficient Objective Quality Model for Agile Application Development. *International Journal of Computer Applications*, 85(8):975–8887, 2014. ISSN 09758887. doi: http://dx.doi.org/10.5120/14861-3234. 14

[53] Tiisetso Khalane and Maureen Tanner. Software quality assurance in Scrum: The need for concrete guidance on SQA strategies in meeting user expectations. *IEEE International Conference on Adaptive Science and Technology, ICAST*, 2013. ISSN 23269448. doi: 10.1109/ICASTech.2013.6707499. 14

[54] Kevin Sullivan, Jinlin Yang, David Coppit, Sarfraz Khurshid, and Daniel Jackson. Software Assurance by Bounded Exhaustive Testing. *ACM SIGSOFT Software Engineering Notes*, 29:133–142, 2004. 15

[55] Daniel Dutil, José Rose, and Witold Suryn. Software Quality Engineering in the new ISO standard : ISO / IEC 24748 - Systems and software engineering — Guide for life cycle management. *Management*, pages 89–96, 2010. doi: 10.1145/1822327.1822339. 15

[56] Barbara Kitchenham and Shari Lawrence Pfleeger. Software quality: the elusive target. *IEEE Software*, 13(1):12–21, 1996. ISSN 07407459. doi: 10.1109/52.476281. 15

[57] Gu Hongying and Yang Cheng. A Customizable Agile Software Quality Assurance model. *The 5th International Conference on New Trends in Information Science and Service Science*, 2(60803110):382–387, 2011. 15

[58] Carlos Alberto Fortunato, Felipe Furtado, Fernando Selleri, Ivaldir de Farias Junior, and Nelson Leitao Junior. Quality Assurance in Agile Software Development: A Systematic Review. In TS DaSilva, B Estacio, J Kroll, and RM Fontana, editors, *AGILE METHODS, WBMA 2016*, volume 680 of *Communications in Computer and Information Science*, pages 142–148, GEWERBESTRASSE 11, CHAM, CH-6330, SWITZERLAND, 2017. Araucaria Fdn; Natl Council Sci & Technol Dev; Fed Univ Parana; Fed Univ of Sao Paulo; Pontifical Univ Catholic Rio Grande Sul, SPRINGER INTERNATIONAL PUBLISHING AG. ISBN 978-3-319-55907-0; 978-3-319-55906-3. doi: 10.1007/978-3-319-55907-0{_}14. 16

[59] Hamza Chehili, Mahmoud Boufaida, and Lionel Seinturier. An Agile Approach for Service-Oriented Architectures. In *ICSOFT*, pages 468–471, 2012. 17, 18

[60] I Christou, S Ponis, and E Palaiologou. Using the Agile Unified Process in Banking. *IEEE Software*, 27(3):72–79, 2010. ISSN 0740-7459. doi: 10.1109/MS.2009.156. 17, 18, 19

[61] H Rong, N Zhou, M Jin, and J Wu. Research on Service-Oriented Framework of Interface Prototype Driven Development. In *2008 International Conference on Computer Science and Software Engineering*, volume 2, pages 552–557, 2008. doi: 10.1109/CSSE.2008.362. 17, 18

[62] Bin Wang, Chunyan Wen, and Jinfang Sheng. A SOA based Model driven Rapid Development Architecture - SMRDA. *ICETC 2010 - 2010 2nd International Conference on Education Technology and Computer*, 1:421–425, 2010. ISSN 2155-1812. doi: 10.1109/ICETC.2010.5529218. 17, 18

[63] Marcel Karam, Sergiu Dascalu, Haidar Safa, Rami Santina, and Zeina Koteich. A product-line architecture for web service-based visual composition of web applications. *Journal of Systems and Software*, 81(6):855–867, 2008. ISSN 01641212. doi: 10.1016/j.jss.2007.10.031. 17, 18

[64] G. Sivanageswara Rao, C. V. Phani Krishna, and K. Rajasekhar Rao. Rational unified process for service oriented application in extreme programming. *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*, pages 1–6, 2013. doi: 10.1109/ICCCNT.2013.6726586. URL `http://ieeexplore.ieee.org/document/6726586/`. 17, 18, 19

[65] Alexander Ivanyukovich, G R Gangadharan, Vincenzo D'Andrea, and Maurizio Marchese. TOWARDS A SERVICE-ORIENTED DEVELOPMENT METHODOLOGY. *To appear in Transactions of the SDPS pp*, 1:10, 2003. 17, 18

[66] Hamza Chehili, Lionel Seinturier, and Mahmoud Boufaida. An Evolutive Component-Based Method for Agile Development of Service Oriented Architectures. *International Journal of Information Systems in the Service Sector (IJISSS)*, 9(3):37–57, 2017. 18, 19

[67] S Roy and M Kumar Debnath. Designing SOA based e-governance system using eXtreme Programming methodology for developing countries. In *2010 2nd International Conference on Software Technology and Engineering*, volume 2, pages 2–277, 2010. doi: 10.1109/ICSTE.2010.5608805. URL `https://ieeexplore.ieee.org/abstract/document/5608805/`. 18, 19

[68] B Wang, D Rosenberg, and B W Boehm. Rapid realization of executable domain models via automatic code generation. In *2017 IEEE 28th Annual Software Technology Conference (STC)*, pages 1–6, 2017. doi: 10.1109/STC.2017.8234464. 18

[69] Miguel Zúñiga-Prieto, Emilio Insfran, and Silvia Abrahão. Architecture Description Language for Incremental Integration of Cloud Services Architectures. *Proceedings - 2016 IEEE 10th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Environments, MESOCA 2016*, pages 16–23, 2016. doi: 10.1109/MESOCA.2016.10. 18, 19

[70] A Nuraini and Y Widyani. Software with service oriented architecture quality assessment. *Proceedings of 2014 International Conference on Data and Software Engineering, ICODSE 2014*, 2014. doi: 10.1109/ICODSE.2014.7062707. URL `http://www.scopus.com/inward/record.url?eid=2-s2.0-84927585861&partnerID=40&md5=575a8d0b9bc415cc17455690fda28149`. 20

[71] Shelly Saunders, Margaret Ross, Geoff Staples, and Sean Wellington. The software quality challenges of service oriented architectures in e-commerce. *Software Quality Journal*, 14(1):65–75, 2006. ISSN 15731367. doi: 10.1007/s11219-006-6002-2. 20, 21

[72] Hamid Mcheick and Yan Qi. Quality attributes and design decisions in Service-Oriented Computing. *2012 International Conference on Innovations in Information Technology (IIT)*, pages 283–287, 2012. doi: 10.1109/INNOVATIONS.

2012.6207749. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6207749. 20

[73] Matthias Galster and Paris Avgeriou. Qualitative analysis of the impact of SOA patterns on quality attributes. *Proceedings - International Conference on Quality Software*, pages 167–170, 2012. ISSN 15506002. doi: 10.1109/QSIC.2012.35. 20

[74] Guillermo Rodríguez, J Andrés Díaz-Pace, and Álvaro Soria. A Case-based Reasoning Approach to Reuse Quality-driven Designs in Service-Oriented Architectures . *Information Systems*, 77:–, 2018. ISSN 0306-4379. doi: https://doi.org/10.1016/j.is.2018.06.003. URL https://www.sciencedirect.com/science/article/pii/S0306437915300181. 20

[75] T. Karthikeyan and J. Geetha. A Quantitative Measurement and Validation of Granularity in Service Oriented Architecture. *International Journal of Computer Science Issues*, 9(2):377–382, 2012. 20

[76] Dirk Voelz and Andreas Goeb. What is Different in Quality Management for SOA? *2010 14th IEEE International Enterprise Distributed Object Computing Conference*, pages 47–56, 2010. ISSN 15417719. doi: 10.1109/EDOC.2010.27. URL http://ieeexplore.ieee.org/document/5630238/. 20

[77] Heiko Koziolek, Bastian Schlich, Carlos Bilich, Roland Weiss, Steffen Becker, Klaus Krogmann, Mircea Trifu, Raffaela Mirandola, and Anne Koziolek. An industrial case study on quality impact prediction for evolving service-oriented software. *Proceeding of the 33rd international conference on Software engineering - ICSE '11*, page 776, 2011. ISSN 02705257. doi: 10.1145/1985793.1985902. URL http://portal.acm.org/citation.cfm?doid=1985793.1985902. 20

[78] W. T. Tsai, R. Paul, Yamin Wang, Chun Fan, and Dong Wang. Extending WSDL to facilitate Web services testing. *Proceedings of IEEE International Symposium on High Assurance Systems Engineering*, 2002-Janua:171–172, 2002. ISSN 15302059. doi: 10.1109/HASE.2002.1173119. 20

[79] W. T. Tsai, R. Paul, Weiwei Song, and Zhibin Cao. An XML-based framework for Web services testing. *Proceedings of IEEE International Symposium on High Assurance Systems Engineering*, 2002-Janua:173–174, 2002. ISSN 15302059. doi: 10.1109/HASE.2002.1173120. 20

[80] Xiaoying Bai, Dezheng Xu, Guilan Dai, Wei Tek Tsai, and Yinong Chen. Dynamic reconfigurable testing of service-oriented architecture. *Proceedings - International Computer Software and Applications Conference*, 1(Compsac):368–375, 2007. ISSN 07303157. doi: 10.1109/COMPSAC.2007.106. 20

[81] Luciano Baresi, Carlo Ghezzi, and Sam Guinea. Smart Monitors for Composed Services. *International Conference On Service Oriented Computing*, page 193, 2004. doi: 10.1145/1035167.1035195. URL http://portal.acm.org/citation.cfm?id=1035195. 20

[82] Danilo Ardagna and Barbara Pernici. Adaptive service composition in flexible processes. *IEEE Transactions on Software Engineering*, 33(6):369–384, 2007. ISSN 00985589. doi: 10.1109/TSE.2007.1011. 20

[83] Holger Schlingloff, Axel Martens, and Karsten Schmidt. Modeling and model checking web services. *Electronic Notes in Theoretical Computer Science*, 126:3–26, 2005. ISSN 15710661. doi: 10.1016/j.entcs.2004.11.011. URL `http://dx.doi.org/10.1016/j.entcs.2004.11.011`. 20

[84] Wei Tek Tsai, Yinong Chen, and Ray Paul. Specification-based verification and validation of web services and service-oriented operating systems. *Proceedings - International Workshop on Object-Oriented Real-Time Dependable Systems, WORDS*, pages 139–147, 2005. ISSN 15301443. doi: 10.1109/WORDS.2005.51. 20

[85] Samer Hanna and Malcolm Munro. An Approach for Specification-based Test Case Generation for Web Services. *2007 IEEE/ACS International Conference on Computer Systems and Applications*, pages 16–23, 2007. doi: 10.1109/AICCSA.2007.370859. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4230934`. 20

[86] Christopher Henard, Mike Papadakis, Mark Harman, and Yves Le Traon. Comparing White-box and Black-box Test Prioritization. pages 523–534, 2016. 20, 21

[87] Wei Tek Tsai, Xinyu Zhou, Yinong Chen, and Xiaoying Bai. On testing and evaluating service-oriented software. *Computer*, 41(8):40–46, 2008. ISSN 00189162. doi: 10.1109/MC.2008.304. 20

[88] Farnoush Golshan and Ahmad Abdollahzade Barforoush. A new approach for tracing quality attributes in service oriented architecture using graph transformation systems. *2009 14th International CSI Computer Conference*, pages 10–16, 2009. doi: 10.1109/CSICC.2009.5349605. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5349605`. 21

[89] Marcelo Teixeira, Richardson Ribeiro, Cesar Oliveira, and Ricardo Massa. A quality-driven approach for resources planning in Service-Oriented Architectures. *Expert Systems with Applications*, 42(12):5366–5379, 2015. ISSN 09574174. doi: 10.1016/j.eswa.2015.02.004. URL `http://dx.doi.org/10.1016/j.eswa.2015.02.004`. 21

[90] Krasimir Baylov and Aleksandar Dimov. Quality Characteristics for Service Oriented Architectures. *Proceedings of the 2015 European Conference on Software Architecture Workshops - ECSAW '15*, pages 1–5, 2015. doi: 10.1145/2797433.2797488. URL `http://dl.acm.org/citation.cfm?doid=2797433.2797488`. 21

[91] Steffen Becker, Mircea Trifu, and Ralf Reussner. Towards supporting evolution of service-oriented architectures through quality idlpact prediction. *Aramis 2008 - 1st International Workshop on Automated engineeRing of Autonomous and runtiMe evolvIng Systems, and ASE2008 the 23rd IEEE/ACM Int. Conf. Automated Software Engineering*, pages 77–81, 2008. ISSN 2151-0830. doi: 10.1109/ASEW.2008.4686297. 21

[92] Adnan Masood and Jim Java. Static analysis for web service security - Tools & techniques for a secure development life cycle. *2015 IEEE International Symposium on Technologies for Homeland Security (HST)*, pages 1–6, 2015. doi: 10.1109/THS.2015.7225337. URL `http://ieeexplore.ieee.org/document/7225337/`. 21

[93] Carlos Alberto Fortunato, Felipe Furtado, and Fernando Selleri. Agile Methods. 680:142–148, 2017. doi: 10.1007/978-3-319-55907-0. URL `http://link.springer.com/10.1007/978-3-319-55907-0`. 21

[94] A. Ahmed, S. Ahmad, N. Ehsan, E. Mirza, and S. Z. Sarwar. Agile software development: Impact on productivity and quality. *2010 IEEE International Conference on Management of Innovation & Technology*, pages 287–291, 2010. doi: 10.1109/ICMIT.2010.5492703. URL `http://ieeexplore.ieee.org/document/5492703/`. 21

[95] Scott Ambler. Quality in an Agile World. *Sqp*, 7(4):34–40, 2005. ISSN 0260-2938. doi: 10.1080/02602930120082032. URL `http://www.ambysoft.com/downloads/agileQuality.pdf`. 21

[96] S I Hashmi and J Baik. Software Quality Assurance in XP and Spiral-A Comparative Study. *International Conference on Computational Science and its Applications, 2007. ICCSA 2007.*, pages 367–374, 2007. doi: 10.1109/ICCSA.2007.65. 21, 22

[97] Ming Huo, J Verner, Liming Zhu, and M a Babar. Software quality and agile methods. *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*, pages 520–525, 2004. ISSN 0730-3157. doi: 10.1109/CMPSAC.2004.1342889. 22

[98] Priyanka Upadhyay, Abhishek Singh, Naveen Garg, and Uttar Pradesh. Modeling Software Maintainability and Quality Assurance in the Agile Environment. 7(3): 83–90, 2014. 22

[99] A. Jenila. Agile Software Development Methodologies and Practices. *International Journal of Innovative Science*, 80(3):2010–2011, 2010. 22

[100] Mihaly Csikszentmihalyi and Kevin Rathunde. The measurement of flow in everyday life: toward a theory of emergent motivation. 1993. 24

[101] Richard M Ryan and Edward L Deci. Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *American psychologist*, 55(1):68, 2000. 24, 54, 55

[102] Jamal Munshi. A method for constructing Likert scales. 2014. 24, 53, 54

[103] Claudia De O. Melo, Célio Santana, and Fabio Kon. Developers motivation in agile teams. *Proceedings - 38th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2012*, 1(September):376–383, 2012. doi: 10.1109/SEAA.2012.45. 24, 53, 54

[104] H Sharp, N Baddoo, S Beecham, T Hall, and H Robinson. Models of motivation in software engineering. *Information and Software Technology*, 51(1):219–233, 2009. doi: 10.1016/j.infsof.2008.05.009. 25

[105] Lee J Cronbach. Coefficient alpha and the internal structure of tests. *Psychometrika*, 16(3):297–334, 1951. ISSN 1860-0980. doi: 10.1007/BF02310555. URL `https://doi.org/10.1007/BF02310555`. 25

[106] João Maroco and Teresa Garcia-Marques. Qual a fiabilidade do alfa de Cronbach? Questões antigas e soluções modernas? *Laboratório de psicologia*, pages 65–90, 2006. 25

[107] Tsun Chow and Dac-Buu Cao. A survey study of critical success factors in agile software projects. *Journal of systems and software*, 81(6):961–971, 2008. 26, 27, 52, 53, 60, 61, 62, 63

[108] E L Deci and R M Ryan. Intrinsic Motivation Inventory. Self-Determination Theory [On-line], 2003. URL `http://selfdeterminationtheory.org/intrinsic-motivation-inventory/`. 54

[109] A A Porter and R W Selby. Empirically guided software development using metric-based classification trees. *IEEE Software*, 7(2):46–54, 1990. ISSN 0740-7459. doi: 10.1109/52.50773. 59