v. 2, n. 14, 2022

Scientific Journal of Applied Social and Clinical Science

THE CHALLENGES OF THE SOFTWARE ARCHITECT IN AGILE METHODOLOGY PROJECTS

Ricardo do Carmo Martins
https://orcid.org/0000-0001-9206-1078

Abstract: Living with change is a reality in the business environment. Changes happen for a number of reasons: rules and laws change, people change their minds, and technology evolves. A software project must certainly have an initial goal to be achieved, but it must be flexible enough to accommodate changes when they come to light, so that it does not become irrelevant. In this context, the question arises around the challenges of a software architect in an agile methodology project, which, identified through systematic and critical bibliographic research, requires from this professional an accurate ability to adapt to changes, especially to manage them in people. that make up the project team, mediating conflicts and proposing solutions that guide the entire project development chain.
Keywords: Software architecture; Agile methodology; Adaptation; Challenges.

INTRODUCTION

Most agile methodology projects have flexible scope as their main characteristic, a fact that implies a series of changes in direction during the project.

In addition, the agile methodology opens the possibility for projects to be partially delivered, so it is not always possible to stick to the architecture proposed at the beginning of each project.

In this context, the question arises: What are the challenges of a software architect in agile methodology projects? Where the professional ceases to act only at the moment of designing the software, to exercise a fundamental responsibility at the time of its development, becoming responsible for making decisions in relation to the maintenance of the structure and organization of the software.

In traditional projects, where all requirements are known in advance, it is possible to create a complete architecture to meet all needs. In the agile context, for each new project need, it is necessary to review and improve the software architecture in order to adapt to the new assumptions and requirements that arise, causing a huge paradigm shift with regard to the original concept. of software architecture.

In the agile context, the software architect is challenged to create simpler, more adaptable and evolutionary architectures in a collaborative way with the project team, instead of a complete architecture that already addresses any and all needs.

This article aims to identify the behavioral and technical challenges faced by the software architect to work in agile methodologies projects.

METHODOLOGY

This work consists of a bibliographic research.

Research of this type is defined as a systematic and critical review of the most important publications on a specific subject, allowing the dissemination of current knowledge on the proposed topic.

Therefore, a review of recent literature will be carried out. The research will be carried out in texts, documents and books specific to the information technology sector, in addition to other materials related to the proposed theme.

The literature selection criteria will be full texts that provide data on the subject to meet the proposed objectives.

LITERATURE REVIEW

In this section, the main concepts are listed, with the objective of consolidating and familiarizing the reader with the understanding of the research carried out.

THE ROLE OF ARCHITECTURE IN A SOFTWARE DEVELOPMENT PROCESS

In the context of software development, factors such as cost and efficiency influence the choice of the best solution to be adopted. This is observed, above all, when analyzing the requirements for the construction of a software: there are several solutions that can be defined to meet these requirements, but a more in-depth analysis is necessary to define the development context of the software. application.

Software architecture emerges as one of the approaches that can be used in the representation of these solutions. Thus, in order to obtain the most adequate architecture to meet the software requirements, an evaluation of this structure must be carried out (SPÍNOLA; BARCELOS, 2008).

These requirements can be broadly classified as functional and non-functional requirements.

Functional requirements are responsible for describing the functionalities that the software must present (SPÍNOLA; BARCELOS, 2008). For Koelsch (2016), a functional requirement describes the functions the system (for example, hardware and software) must perform.

Functional requirements are categorized as business rules, administrative roles, authentication, authorization levels, auditing, tracking, compliance, legal or regulatory (KOELSCH, 2016), among others.

On the other hand, non-functional ones describe characteristics that the software must present, which can often be seen as restrictions or specialties of the final product (SPÍNOLA; BARCELOS, 2008). For Koelsh (2016), a non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors, they are contrasted with functional requirements that define specific behavior or functions.

Non-functional requirements are categorized as architectural, performance, security, quality, fault tolerance, efficiency, effectiveness, usability, scalability, recoverability, reliability, maintainability, interoperability, extensibility, availability (KOELSCH, 2016), among others.

Functional requirements are detailed during system design, while non-functional requirements are detailed in the system architecture.

Among the different types of requirements, both functional and non-functional, quality requirements are the ones that most influence the construction of architecture.

This is because, unlike functional requirements where in most cases a modification causes changes in a specific set of architectural elements, changes in a quality requirement can imply a complete restructuring of the architecture (BASS et al., 2003).

All these factors comprise the project at the architectural level and are directly related to the organization of the system and, therefore, affect the quality attributes (also called non-functional requirements) (FILHO, 2005).

If we make a comparison between software architecture (characterized, for example, by the layered style) and 'classical' architecture (relating to the construction of buildings), we can observe that the architectural design is decisive for the success of the system.

According to some authors, the software architecture still consists of a high-level model that allows an easier understanding and analysis of the software to be developed.

Shaw (1997) defines software architecture as a set of computational components and the relationships between these components.

Garlan (2000) states that it is a structure of components of a program/system, the

relationships between these components, the principles and guidelines that govern the projects and the evolution of software.

Clements et al. (2004) points to architecture as a complex entity that cannot be described in a one-dimensional way.

For Spínola and Barcelos (2008), an effective way to deal with this complexity is to describe it from different perspectives, also known as architectural views.

As shown by Spínola and Barcelos (2008), the use of architecture to represent software solutions was mainly encouraged by two trends (GARLAN AND PERRY, 1995; KAZMAN, 2001):

I. the recognition by designers that the use of abstractions facilitates the visualization and understanding of certain properties of the software; and

II. the increasing exploration of frameworks in order to reduce the effort of building products through the integration of previously developed parts.

Another particularity of the architecture is the possibility of using it as a tool to communicate the designed solution to the various stakeholders that participate in the software development process (GARLAN, 2000).

However, for this communication to be possible, the architecture must be represented through a document, known as an architectural document (SPÍNOLA; BARCELOS, 2008).

To obtain the architecture of a software, the requirements are the main information used. During the architectural specification process (demonstrated in Figure 1), sources of knowledge other than requirements can also be used to define architectural elements and how they must be organized.

The architect's experience, reasoning about requirements, in addition to architectural styles and tactics are sources that deserve to be highlighted (SPÍNOLA; BARCELOS, 2008).

There is, however, a lack of consensus in the community regarding both basic concepts and definitions and how to represent a software architecture (BUSCHMANN et al., 1996; CLEMENTS et al., 2004).

Some authors claim that software architecture represents the structure, or set of structures, which comprises the software elements, their externally visible properties and their relationships (BASS et al., 2003).

To create this structure, several authors agree that three types of basic elements can be used (DIAS; VIEIRA, 2000):

a) Software elements, which can also be called modules or components, are the abstractions responsible for representing the entities that implement specified functionalities;

b) Connectors, which can be called relationships or interfaces, are the abstractions responsible for representing the entities that facilitate communication between software elements;

c) Organization or configuration that consists of the way in which the software elements and connectors are organized.

For that, the structure and the entities that compose the architecture of a software must be represented in such a way that it is possible to use the designed architecture for its proper purposes. This representation is called an architectural document. Such a document is composed of a set of models and information that mainly describe the structure of the software specified to meet the requirements.

However, it is known that even though there are standards that indicate the type of information that must be described in an architectural document, it does not have an exact definition of the level of abstraction that must be used in the description of this information.

However, throughout the software development, the architecture undergoes

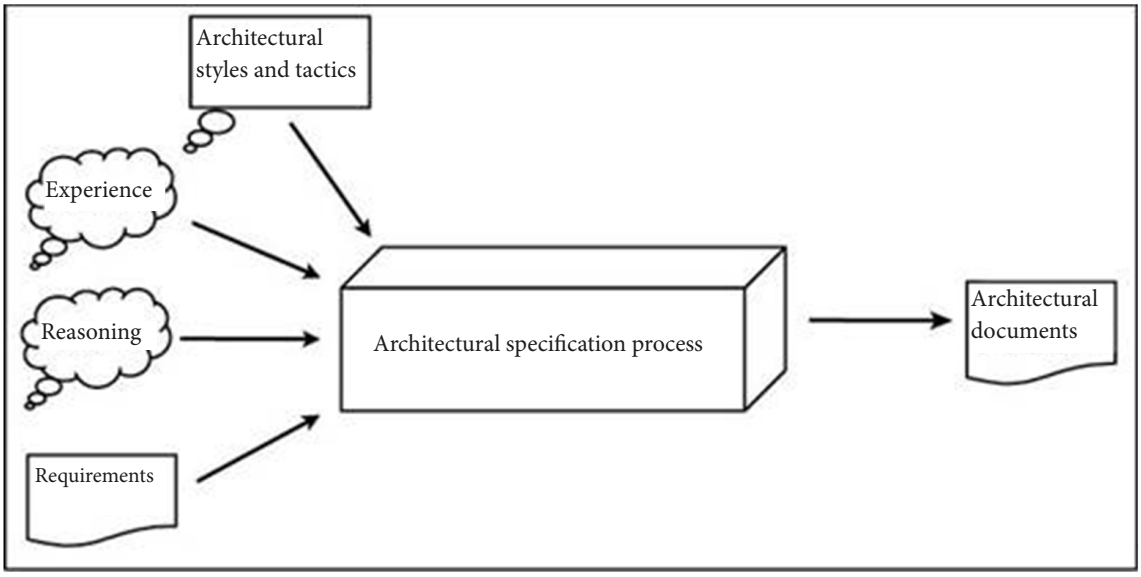


Figure 1 – Elements used in the construction of a software architecture (SPÍNOLA E BARCELOS, 2008).

	Module Views					C&C Views	Allocation Views				Other Documentation					
	Decomposition	Uses	Generalization	Layered	Data Model	Various	Deployment	Implementation	Install	Work Assignment	Interface Documentation	Context Diagrams	Mapping Between Views	Variability Guides	Analysis Results	Rationale and Constraints
Project managers	s	s		s			d			d		o				s
Members of development team	d	d	d	d	d	d	s	s	d		d	d	d	d		s
Testers and integrators	d	d	d	d	d	s	s	s	s		d	d	s	d		s
Designers of other systems					s						d	o				
Maintainers	d	d	d	d	d	d	s	s			d	d	d	d		d
Product-line application builders	d	d	s	o	s	s	s	s	s		s	d	s	d		s
Customers							o			o		o			s	
End users						s	s		o						s	
Analysts	d	d	s	d	d	s	d		s		d	d		s	d	s
Infrastructure support personnel	s	s			s	s	d	d	o					s		
New stakeholders	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Current and future architects	d	d	d	d	d	d	d	s	d	s	d	d	d	d	d	d
Key: d = detailed information, s = some details, o = overview information, x = anything																

Figure 2 – Example of a list of stakeholders with your interests (CUNHA, 2018).

refinements that decrease the level of abstraction and allow, for example, the representation of the relationships between the architectural elements and the source code files responsible for implementing them (CLEMENTS et al., 2004).

At this point, the architecture becomes part of the solution scope and also incorporates information related to design decisions, such as elements specific to the technology that will be used to implement the solution.

Some authors claim that the main motivation to evaluate the architecture of a software is related to its role within the development process (SPÍNOLA; BARCELOS, 2008).

According to these authors, the architecture of a software is based on different purposes for each stakeholder (shown in Figure 2):

The customer is the person or company that hires a development team to build a system that they need. In the initial phase of the project, this stakeholder needs an estimate of certain factors, usually economic, that can be obtained after defining the main structure of the software.

The customer, for example, is interested in software cost, accounting and maintainability estimates that can be obtained primarily through an architecture analysis.

Therefore, it is extremely important for the customer that the architecture meets the software requirements in order to represent their real expectations in relation to what was specified.

For managers, the architecture allows them to make certain design decisions by enabling the summarization of the various characteristics of the system. A manager can, for example, use the architecture as a basis to define the development teams according to the architectural elements that are identified in the architecture and that must be built.

The developer, on the other hand, looks for a specification from the software architecture that describes the solution in sufficient detail and that satisfies the customer's requirements, but that is not so restrictive as to limit the choice of approaches for its implementation. Developers use the architecture as a reference for composing and developing system elements, and for identifying and reusing architectural elements already built.

For software testers, the architecture provides, in a black box view, information related to the correct behavior of the architectural elements that integrate and compose the solution. A good architecture favors both the tests and the user of the system, but mainly the automated tests.

To the team of maintainers, the architectural description of the software provides a core structure of the application that ideally must not be violated. Any change must preserve it, seeking, if possible, a modification purely of the architectural elements and not the way they are organized.

Garlan (2000) states that the main role of software architecture is to serve as an instrument to communicate the proposed solution.

For Filho (2005), software architecture serves as a framework through which to understand the components of a system and their interrelationships.

The software architect has a very important role in the strategy adopted by the organization.

This professional needs to have in-depth knowledge of the domain, existing technologies and software development processes. A summary of a desired set of skills for a software architect and the tasks assigned to him are presented in Table 1 (FILHO, 2005).

However, the current scenario of the software development market has demanded from this professional a high degree of resilience, since there is a need for a continuous increase in competitiveness, following the

dynamism and speed with which information and knowledge circulate (RIBEIRO AND RIBEIRO, 2015).

AGILE METHODS

According to Gomes et al. (2014), for several years, Software Engineering was inspired by manufacturing processes to consolidate its working methods.

Born in the second half of the 20th century, the software development industry sought in emerging sectors of the industry at the time most of the theories and production methods. In particular, the automotive field, in a broad industrial rise, played an important role in the constitution of the new IT industry (GOMES et al., 2014).

Thanks to Henry Ford's serial production model, highly inspired by Frederick Taylor, all traditional thinking in the science of software development unfolded with an intense focus on the standardization of components and processes and the mechanization of movement (GOMES et al., 2014).

Over time, the complexity of software has increased more and more. Joining the problems inherent to software development and the current importance of computerized systems, some theorists began to disagree with the idea of treating software development as a serial production factory (MARQUES, 2012).

In the mid-1990s, alternative software development processes began to emerge, in response to the traditional ones, considered excessively regulated, slow, bureaucratic and inappropriate for the nature of the activity. These new processes were called "light", as opposed to the previous ones, "heavy" (GOMES et al., 2014).

In 2001, these processes became known as "agile", through the creation of the so-called Agile Manifesto, which established the principles of the methodology that was born there.

This manifesto was created by a group of 17 experts who met in Utah, in the United States, to discuss ways to develop software in a lighter way. They coined the terms "Agile Software Development" and "Agile Methods" and created the Agile Manifesto – widely spread as the canonical definition of agile development, composed of the values and principles that we will see next (GOMES et al., 2014).

It must be noted that most agile concepts and principles emerged with a focus on software development projects, but are currently used in various types of projects that have great uncertainties, such as advertising campaigns, new products, budget planning and many others. areas (RIBEIRO; RIBEIRO, 2015).

For Libardi and Barbosa (2010), a characteristic of agile methodologies is that they are adaptive rather than predictive. This way, they adapt and increment to new factors during the development of the project, instead of trying to analyze in advance everything that may or may not happen during the course of development. This pre- analysis is always difficult and expensive, in addition to becoming a problem when changes to the plans need to be made.

The Agile Manifesto makes an important message clear, that the process and tools are likely to be needed on the project; however, you must try to focus the team's attention on the individuals and interactions involved in the project.

Libardi and Barbosa (2010) emphasize that software is not built by a single person, they are built by a team, so they need to work together (including programmers, testers, designers and also the customer). Processes and tools are important, but not as important as working together.

Within this principle, one must focus primarily on the development of the individuals involved in the project, emphasizing

Desired Skills	Assigned tasks
Domain knowledge and relevanttechnologies	Modeling
Knowledge of technical issues for systemsdevelopment	Commitment and feasibility analysis
Knowledge of requirements gathering techniques, and systems modeling and development methods	Prototyping, simulation andexperimentation
Knowledge of the company's businessstrategies	Analysis of technological trends
Knowledge of processes, strategies andproducts of competing companies	'Evangelizer' of new architects

Table 1 - Skills and Tasks of a Software Architect (FILHO, 2005).

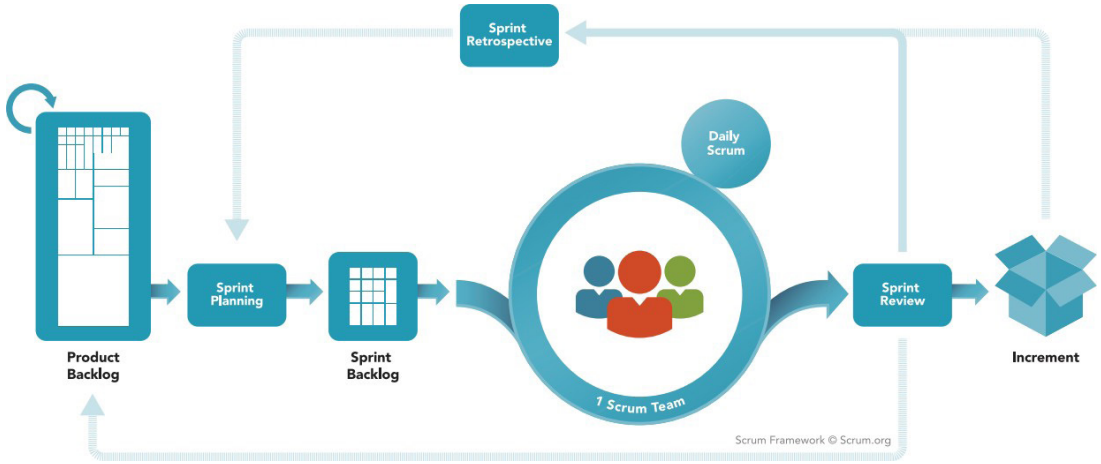


Figure 3 – SCRUM Framework (Scrum.org).

Traditional Methodology	Agile Methodology
Architect has knowledge of all functional requirements	Architect has little knowledge of functional requirements, they are identified as the software evolves
Architect thinks about the solution as a whole, as all features are known	Architect thinks about the solution only to meet the functionalities already identified
Complete and complex architecture that already meets any need	Creating simple architectures that are easily adaptable
Very well documented architecture at the beginning of the project	Focus on simple models, created from according to need
Architect produces several visions and abstractions as a means of communicating to others involved in the project	Architect produces simpler abstractions and views, sufficient for the team to understand
Isolated architects, far from the team	Architect is a member of the team, working collaboratively
The architectural definition is top-down, the development team implements	Architectural definitions emerge from the project team in a collaborative way
Architect produces predictive architectures	Architect produces evolutionary architecture
Architecture validated through reviews	Architecture validated with implementations concrete

Table 2 - Comparison of traditional methodology and agile methodologies

productive and effective interactions, with the objective of increasing the chances of project success (RIBEIRO; RIBEIRO, 2015).

For Gomes et al. (2014), we came to believe so blindly in processes and tools that we stopped communicating. We forget that people make software. Instead of conversations and discussions, developers were given written specifications. They are important, yes, but they don't communicate as well as a good face-to-face discussion, or sketches, doodles, and models. Obviously, tools are important. It's much harder to do things without them. Processes, too. Still, we mustn't stop valuing people and we mustn't stop communicating. This is part of teamwork. Therefore, if these issues start to fight for space, value the human side more and you will have a good chance of getting better results (GOMES et al., 2014).

Another positive point of agile methodologies is the constant delivery of operational parts of the software. This way, the customer does not have to wait long to see the software working and judge whether it faithfully meets their needs (LIBARDI; BARBOSA, 2010).

This, by the way, is another value contained in the manifesto, which highlights that software projects are normally initiated with the objective of creating value for the company through a high quality software product, often in deliveries in intermediate parts (increments of software).

Integration and continuous testing also make it possible to improve software quality. It is no longer necessary to have a module integration phase, as they are continuously integrated and any problems are constantly resolved.

Ribeiro and Ribeiro (2015) point out, however, that software without documentation is certainly problematic and makes support and maintenance difficult. But complete documentation without software adds absolutely nothing to any organization.

In Libardi and Barbosa's (2010) analysis of the Agile Manifesto, the authors state that documentation must exist to help people understand how the system was built, but it is much easier to understand how it works by seeing the system work than by seeing it work. diagrams that describe the operation or abstract the use.

The manifesto also reinforces the need to be flexible and efficient, rather than rigid and uncooperative. This applies to the numerous cases where the final product is delivered exactly as specified, but the customer signals the need for changes due to a change in idea, priority or market.

Only the customer can say what he expects from the software and, as these are usually people and organizations from different industries, it is normal for them to change their minds as they see the software working.

Having a contract is important to define responsibilities and rights, but it must never replace communication between the parties involved in the project. Successfully developed works have constant communication with the client to understand their needs and help them discover the best way to express them.

For Gomes et al. (2014), this is a weak point of the Manifesto and of the Agile Methods, constantly criticized due to its fragility and personality. For the authors, it's something that definitely needs to evolve. The parties need some security against acts of bad faith.

In order to minimize this risk, contracts usually have types of "control points", in which the relationship is reassessed to decide whether to continue or discontinue the contract without encumbrance. Naturally, if both parties are satisfied with the relationship, the commitment remains. Otherwise, the realignment of interests is sought and, if there is no agreement, the continuity of the project is suspended (GOMES et al., 2014).

The Manifesto admits that it is very difficult to address all the complex development issues in contracts. According to Gomes et al. (2014), trying to create protective walls will not solve anything if there is no collaboration between the team and the client. The solution would be, instead of trying to solve problems by including new clauses, writing overly complex contracts, working at another level with the client, creating a climate of trust and collaboration.

Libardi and Barbosa (2010) emphasize that the Agile Manifesto does not reject processes and tools, documentation, contract negotiation or planning, but simply shows that they are of secondary importance when compared to individuals and interactions, with the software working, with customer collaboration and rapid responses to changes and changes.

In projects with a large number of uncertainties, it is almost certain that the initial plans will change. Instead of investing efforts in trying to bring the project back to the original plans, effort and energy must be spent to respond to the inevitable changes in the project (RIBEIRO; RIBEIRO, 2015).

RESEARCH PRESENTATION

For the purpose of this research, a systematic and critical review of important and relevant publications on the subject was carried out, allowing the dissemination of current knowledge on the proposed topic.

The authors point out that, in a classic methodology, it can happen that a software is built entirely and then it is discovered that it no longer serves the purpose it was developed because the rules have changed and the adaptations become too complex to be worth the effort. worth developing them. Agile methodologies work with constant feedback, which allows you to quickly adapt to any changes in requirements.

These changes are often critical in traditional methodologies, which do not have the means to quickly adapt to changes (LIBARDI; BARBOSA, 2010).

In agile processes, however, delivery of working software is preferred over comprehensive, exaggerated and wasteful documentation. The expected result is the software working, with quality. Documentation and maintainability are part of this quality. However, there is a need to think more about "what" to document and "when" to document. One must reflect on what is really useful and what will quickly become outdated or not even be read someday. This generates waste and increases the cost of a project (GOMES et al., 2014).

The Agile Manifesto, in its second clause, proposes that working code is more important than extensive documentation. As already mentioned, documents and specifications are valid, but prioritizing them over well-made and functional software is a mistake. The 7th principle of the Manifesto ratifies this discourse, clarifying that the good progress of a software development project must be measured, primarily, through the amount of software delivered and working, which is what, in fact, matters to the end customer, and not by the volume of documents generated (GOMES et al., 2014).

The software architect, in this context, starts to live with the unlikely attribution of establishing communication between the work team and the contracting client, acting as a facilitator in this process.

The Agile Manifesto states that, among all types of information exchange between software development teams, the most effective is face-to-face communication. The less indirect communication, the lower the risks of misinterpretation. The more frequent the face-to-face conversations are, the less conflicts will arise, the less energy will be

spent on their reversal and the more effective and sustainable the work will be (GOMES et al., 2014).

The software architect is still faced with the difficult task of responding to changes rather than following a plan, as the learning process exists for both the development team and the customer. Changes, therefore, are natural and inevitable.

The changes must be seen as great opportunities for the developed system to be more responsive to the customer's needs, in addition to contributing greatly to the desired results. Therefore, the architect must do everything possible to receive them and welcome them with open arms, in addition to organizing the ideas that will be passed on to the development team.

One of the ways to avoid this is to adopt a constant process of collaboration between customers, the product owner and development teams; a relationship primarily driven by the software architect. It is his responsibility to promote a joint action of agile teams and direct representatives of the client, enabling a continuous flow of presentation, discussion and feedback, which is fundamental to guarantee the success of the project.

So, to really welcome the changes, we need to replan all the time. Agile planning processes usually include PDCA cycles at different levels (daily, weekly, monthly, quarterly, etc.), in which there is an opportunity for reflection and readjustment of the directions taken by the project (GOMES et al., 2014).

Realizing the inefficiency of the practices adopted against changes in the course of development, the agile philosophy chose to disagree with the secular premise that late changes are harmful and adopted a favorable stance towards their occurrence.

In this context, therefore, it is up to architects to naturally accept the fact that changes in the original scope of any project are expected and very welcome, even if this directly confronts everything that was preached in the past. As a result, changes of any nature are now seen as normal.

The great advantage is that agile methods bring techniques and tools to respond as quickly as possible to all kinds of changes, which is certainly a reflection of learning from some circumstance hitherto unnoticed by stakeholders.

The work rhythm is now dictated by pre-defined and predetermined time periods for software deliveries, making the work team aware of its speed, that is, it starts to better predict how much it is capable of produce in each cycle.

This evolution of software engineering has brought several processes, techniques and tools that, despite organizing and documenting the solution development life cycle, have become more important than the software to be delivered. On the other hand, it is necessary to deal with the ingenious brains of analysts and programmers, eager to apply the "state of the art" of the latest technologies, languages and tools, putting product quality at risk and leaving customer needs in the background. (GOMES et al., 2014).

This, therefore, is one of the duties of the software architect: to guarantee the delivery of the software working with quality, with fast and continuous iterations, always adding business value to the customer.

Agility is not about obeying pre-established production protocols, unlike in other development cultures, but about new patterns of behavior and attitude. Therefore, a team cannot call itself "agile" if it does not behave like that. Books and articles are great sources of knowledge, but no team becomes agile by simply reading them. After all, agility is not granted, but achieved with each small daily behavior transformation (GOMES et al., 2014).

DISCUSSION OF RESULTS

After conducting the research, it was possible to identify and list a comparison between the profile, and attributions, of the software architect in projects of the traditional methodology, and their correlation in agile methodologies, presented in Table 2.

Agile methodology projects go through several iterations of continuous improvement, enhancing positive aspects and acting on identified improvement points, guided and scored by the architect. The zipper metaphor (Figure 4) demonstrates the evolution and refinement of requirements, the extraction of relevant architectural requirements in each iteration, and the dependency between them.

It was found that in the agile context, it is up to the software architect to produce evolutionary architectures, with support for changes, extensible and flexible, such as architectures based on micro-services (Figure 5).

Use of evolutionary architectures, characterized by modularity and association with the business domain (Domain-Driven Design - DDD), aiming at the low coupling between the various components and interfaces, allows experimentation and minimizes the risks associated with the changes inherent to the construction and evolution of the application.

FINAL CONSIDERATIONS

It was identified that the software architect needs to recognize software development as an empirical process and subject to change throughout its life cycle, to effectively serve the end customer.

The architectural evolution, until then exclusive to the software architect, becomes shared and collaborative, enriched by the team, bringing more quality to the final product, as well as greater synergy, and understanding, before the entire project team.

Proposing and building simpler, flexible, modular and evolutionary architectures are the great challenges for the traditional software architect of software development, in projects of agile methodologies.

To continue this subject, it is suggested the evaluation of modeling techniques, and design, to form and enrich agile methodologies, considering not only one, but a complete set of activities related to software architecture, as well as techniques and approaches for building software. evolutionary architectures.

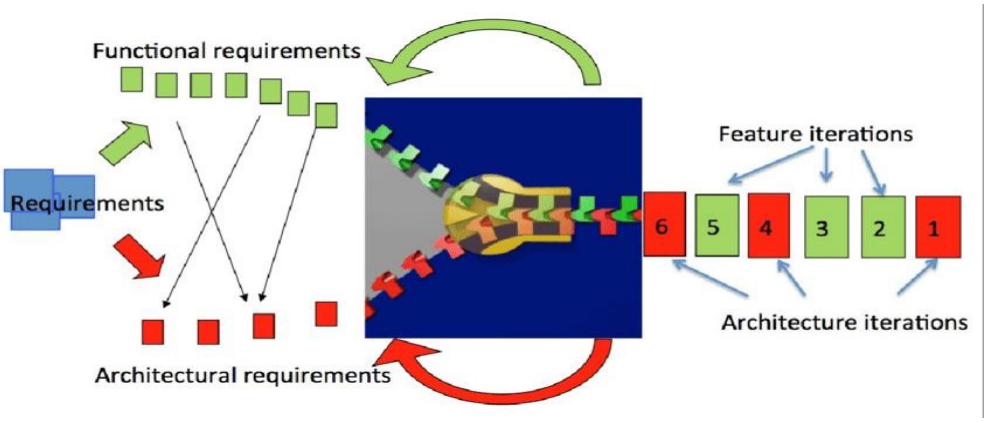
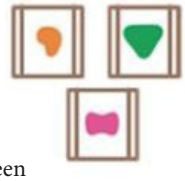


Figure 4 - Zipper metaphor for combining functional and architectural interactions. (NORD; OZKAYA; KRUCHTEN, 2014)

A monolithic application has all its functionality in a single process....

..and scale by replicating the monolith on multiple servers...

The microservices architecture puts each element of functionality into a separate service...



.... and scale distributing services between servers, replicating on demand...

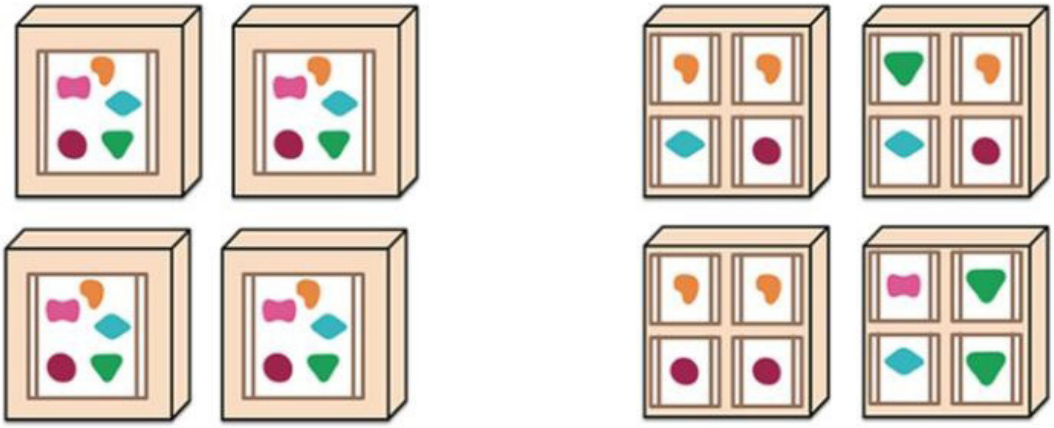


Figure 5 – Comparison between monolithic architecture and micro-services architecture (FOLWER; LEWIS, 2014)

REFERENCES

BACHMANN, F.; BASS, L.; CHASTEK, G.; DONOHOE, P.; PERUZZI, F. **The Architecture Based Design Method**, CMU/SEI, Relatório Técnico, CMU/SEI2000-TR- 001. 2000. Disponível em https://resources.sei.cmu.edu/asset_files/TechnicalReport/2000_005_001_13697.pdf. Acesso em 29 maio 2018.

BAHSOON, R.; EMMERICH, W. **Evaluating software architectures: development, stability, and evolution.** In: Book of Abstracts of the ACS/IEEE International Conference on Computer Systems and Applications, pp. 47, Tunis, Tunisia, July 2003. Disponível em https://www.researchgate.net/publication/4032742_Evaluating_software_architectures_ development_stability_and_evolution. Acesso em 27 junho 2018.

BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software Architecture in Practice**. Second Edition, Addison Wesley. 2003.

BECK, K. **Extreme Programming Explained:** Embracing Change. 1. ed. AdissonWesley, 1999.

BOSSAVIT, L. **The Unbearable Lightness of Programming**: a tale of two cultures. Cutter IT Journal, Massachusetts, v.15, n.9, 2002.

BUSCHMANN, F.; MEUNIER, R.; ROHNERT, H.; SOMMERLAD, P.; STAL, M. **Pattern- Oriented Software Architecture:** A System of Patterns, Jon Wiley and Sons. 1996.

CLEMENTS, P.; BACHMANN, F.; BASS, L.; GARLAN, D.; IVERS, J.; LITTLE, R.; NORD, R.; STAFFORD, J. **Documenting Software Architectures**. Addison-Wesley, 2004.

COPLIEN, J. O. **Lean Architecture:** for Agile Software Development. Wiley, 2010.

DIAS, M.S.; VIEIRA, M.E.R. **Software architecture analysis based on statechart semantics**. In: International Workshop on Software Specification and Design, pp. 133-137. Washington, 2000. Disponível em https://ieeexplore.ieee.org/document/891134/. Acesso em 9 junho 2018.

FAGUNDES, P. B. **Framework para Comparação e Análise de Métodos Ágeis.** Dissertação de Mestrado. Universidade Federal de Santa Catarina. Florianópolis, 2005. Disponível em https://projetos.inf.ufsc.br/arquivos_projetos/projeto_825/FR.COMP.ANAL.M.AGEIS.do c. Acesso em 22 maio 2018.

FILHO, A.M.S. **Arquitetura de Software:** Desenvolvimento orientado para arquitetura. Engenharia de Software Magazine, 1. ed. 2005. Disponível em http://www.garcia.pro.br/EngenhariadeSW/artigos%20engsw/art%201%20-%20Revista%20Engenharia%20de%20Software%20-%20edicao%201%20-%20Arquitetura%20de%20software.pdf. Acesso em 18 maio 2018.

FORD, N. **Arquitetura Evolucionária**: Considerações e técnicas para arquitetura ágil. DeveloperWorks, 2010. Disponível em https://www.ibm.com/developerworks/br/java/library/j-eaed10/index.html. Acesso em 11 abril 2018.

FRIED, J.; HANSSON, H.; LINDERMAN, M. **Getting real:** the smarter, faster, easier way to build a successful web application. 2006.

GARLAN, D. **Software architecture: a roadmap**. In: Proceedings of The Conference on The Future of Software Engineering, pp. 91-101, 2000. Disponível em https://dl.acm.org/citation.cfm?id=336537. Acesso em 26 maio 2018.

GOMES, A.; WILLI, R.; REHEM, S. **O Manifesto Ágil.** In: Prikladnicki, R.; Willi, R.; Milani, F. Métodos Ágeis para Desenvolvimento de Software. São Paulo: Bookman, 2014.

HIGHSMITH, J. Agile Software Development Ecosystems. Adisson-Wesley, 2002.

HUMMEL, A. D. **Como fica a arquitetura de software em um projeto** ágil? TI Especialistas, 2015. Disponível em http://www.tiespecialistas.com.br. Acesso em 11 maio 2018.

KAZMAN, R. **Handbook of Software Engineering and Knowledge Engineering**. In: CHANG, S.K. (eds), World Scientific Publishing, 2001.

KAZMAN, R.; BASS, L.; ABOWD, G.; WEBB, M. **SAAM: a method for analyzing the properties of software architectures**. In: Proceedings of the International conference on Software Engineering (ICSE), pp. 81-90, 1994. Disponível em https://ieeexplore.ieee.org/document/296768/. Acesso em 11 junho 2018.

KOCH, R. **O princípio 80/20:** o segredo de se realizar mais com menos. Rio de Janeiro: Rocco, 2001.

KOSCIANSKI, A.; SOARES, M.S. **Metodologias ágeis.** In: Qualidade de Software: Aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software. 1. ed. São Paulo: Novatec, 2006.

KRUCHTEN, P. **Architectural Blueprints - The "4+1" View Model of Software Architecture**. In: IEEE Software, v. 12, pp. 42-50, 1995.

LAITENBERGER, O.; ATKINSON, C. **Generalizing Perspective-based Inspection to handle Object-Oriented Development Artifacts**. In: Proceedings of the International conference on Software Engineering (ICSE), 1999. Disponível em: https://ieeexplore.ieee.org/document/841039/. Acesso em 17 junho 2018.

CUNHA, T. **Modelagem e Documentação Arquitetural**. Instituto de Gestão e Tecnologia da Informação, 2018.

NORD, R. L.; OZKAYA, I.; KRUCHTEN, P. **Agile in distress: Architecture to the rescue**. International Conference on Agile Software Development. Anais...2014

LEFFINGWELL, D.; MUIRHEAD, D. **Tactical Management of Agile Development:** Achieving Competitive Advantage. Colorado, 2004. Disponível em: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.495.4432&rep=rep1&type=pdf. Acesso em 22 junho 2018.

LIBARDI, P. L. O.; BARBOSA, V. **Métodos** Ágeis. Universidade Estadual de Campinas – UNICAMP, 2010. Disponível em: http://www.fatecsp.br/dti/tcc/tcc00064.pdf. Acesso em 27 junho 2018.

MARQUES, A. N. **Metodologias** ágeis **de desenvolvimento:** Processos e Comparações. São Paulo, 2012.

PALMER, S. R.; FELSING, J. M. **A Practical Guide to Feature-Driven Development.** New Jersey: Prentice Hall PTR, 2002.

FOWLER, M; LEWIS, J. **Microservices, a definition of this new architectural term**, 2014. Disponível em https://martinfowler.com/articles/microservices.html. Acesso em 06 outubro 2018.

POPPENDIECK, M.; POPPENDIECK, T. **Lean Software Development:** An Agile Toolkit. New Jersey: Addison Wesley, 2003.

PRESSMAN, R. **Desenvolvimento Ágil**. In: Engenharia de Software. 6. ed. São Paulo: McGraw Hill Interamericana, 2006.

RIBEIRO, R. D.; RIBEIRO, H. C. S. R. **Gerenciamento de projetos com métodos** ágeis. Rio de Janeiro, 2015.

SPÍNOLA, R. O.; BARCELOS, R. F. **Fundamentos de Arquitetura de Software**. Engenharia de Software Magazine. 6. ed., 2008. Disponível em: http://www.garcia.pro.br/EngenhariadeSW/artigos%20engsw/art%204%20-%20Revista%20Engenharia%20de%20Software%20-%20edicao%206%20-%20fundamentos%20de%20Arquitetura%20de%20Software.pdf. Acesso em 30 julho 2018.

XAVIER, J.R. **Criação e Instanciação de Arquiteturas de Software Específicas de Domínio no Contexto de uma Infra-estrutura de Reutilização**. Dissertação de Mestrado, Programa de Engenharia de Sistemas e Computação. COPPE/UFRJ, 2001. Disponível em http://reuse.cos.ufrj.br/prometeus/publicacoes/xavier-dissertacao- mestrado.pdf. Acesso em 27 julho 2018.

SCRUM.ORG, **SCRUM Framework**. Disponível em http://www.scrum.org. Acesso em 08 outubro 2018.

COSTA, M. **Características de Arquiteturas Evolutivas**. InfoQ, 2016. Disponível em: https://www.infoq.com/br/news/2016/04/evolutionary-architectures. Acesso em 01 outubro 2018.

SHAW, M.; GARLAN, D. **Characteristics of Higher-Level Languages for Software Architecture**. Carnegie Mellon University. 1994. Disponível em: http://www.dtic.mil/dtic/tr/fulltext/u2/a292215.pdf. Acesso em 14 julho 2018.

MENDONÇA, D; STAA, A. **Técnicas para Aplicação de Agilidade em Arquitetura de Software.** Monografia em Ciências da Computação. Pontifícia Universidade Católica do Rio de Janeiro, 2016. Disponível em ftp://ftp.inf.puc- rio.br/pub/docs/techreports/16_03_mendonca.pdf. Acesso em 05 outubro 2018.