Master Thesis
Software Engineering
Thesis no: MSE-2013-139
September 2013

Overcoming the Limitations of Agile Software Development and Software Architecture

Carlos García Álvarez

School of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona
Sweden

This thesis is submitted to the School of Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:
Author(s):
Carlos García Álvarez
E-mail: carlos.garciarez@gmail.com

University advisor(s):
Dr. Darja Šmite
School of Computing

School of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona
Sweden

Internet : www.bth.se/com
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

ABSTRACT

Context. Agile Software Development has provided a new concept of Software Development based in adaptation to changes, quick decisions, low high-level design and frequent deliveries. However, this approach ignores the value that Software Architecture provides in the long term for increasing the speed in delivery working software, which may have catastrophic consequences in the long term.

Objectives. In this study, the combination of these two philosophies of Software Development is investigated. Firstly, the concept of Software Architecture in Agile Projects; then, the major challenges faced concerning Software Architecture in Agile Projects, the practices and approaches that can be used to overcome these challenges and the effects that these practices may cause on the project.

Methods. The research methodologies used in this study are Systematic Literature Review for gathering the highest amount possible of contributions available in the Literature at this respect, and also the conduction of Semi-Structured Interviews with Agile Practitioners, in order to obtain empirical knowledge on the problem and support or deny the SLR findings.

Results. The results of the Thesis are a unified description of the concept of Software Architecture in Agile Projects, and a collection of challenges found in agile projects, practices that overcome them and a relation of effects observed. Considering the most frequent practices/approaches followed and the empirical support, it is enabled a discussion on how to combine Software Architecture and Agile Projects.

Conclusions. The main conclusion is that there is not a definite solution to this question; this is due to the relevance of the context (team, project, customer, etc.) that recommends the evaluation of each situation before deciding the best way to proceed. However, there are common trends on the best-recommended practices to integrate these two concepts. Finally, it is required more empirical work on the issue, the conduction of controlled experiments that allows to quantify the success or failure of the practices implemented would be most helpful in order to create a body of knowledge that enables the application of certain practices under certain conditions.

Keywords: Software Architecture, Agile Software Development, Systematic Literature Review, Semi-Structured Interviews.

ACKNOWLEDGEMENTS

First of all, I want to thank my supervisor at Blekinge Tekniska Högskola, Dr. Darja Šmite, for all her support, time, quick feedback and constructive reviews of my work. Without her advice and contributions this Thesis would have not been possible.

I would like to thank my co-supervisor at Universidad Politécnica de Madrid, Dr. Ana María Moreno. Her suggestions, attention, and effort in critical moments of the Thesis have been essential to reach a successful end. I would also like to specially thank Dr. Óscar Dieste for his endless collaboration and advices in bureaucratic issues, which have given me the tranquility to know that every problem has a solution.

I would like to express my special thanks to the participants in the interviews for sharing their time and knowledge with nothing in reward.

I would like to mention to all my classmates of the European Master in Software Engineering, both at UPM and BTH, and very especially to my 'brother', Ali Demirsoy, we had a great time in Sweden and I am sure we will meet again. Also, I am very thankful to all my friends at Ponferrada, who have stood me during these difficult months, trying to 'keep me sane'.

I am also deeply grateful to my family. To my Mother, who has been extremely tolerant and supporting day by day even though she was not in her better moment; and to my Sister, I will never be able to pay the effort and time you devoted to review this Thesis and your continuous encouragement to go on.

Finally, and in the most special way possible, I want to thank to my Father for all his support during all his life. You gave everything everyday for me, and although you are not able to be here, I will try to make you feel proud of me, wherever you are.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

The world changes every second; these days changes happen very rapidly and continuously; the world is smaller, news travel instantly and communications provoke that something that occurs in one corner of the world immediately affects to the opposite part of the globe. Companies need to respond quickly to changes, and Software Development Companies are no exception at all.

The needs in Software Development projects change extremely fast; factors like technological evolution, market demands, rivals' products, etc. may cause that a project that is barely finished is already outdated; we are in a context where the requirements of a customer may evolve within a project's life span, and this is an urgent need that should be met in order to achieve success.

Traditional Software Development methodologies (like Waterfall [2]) mainly propose a static approach. Their vision of the project is tight and most of them are mainly sequentially structured; although approaches like V-Model [3], Spiral model [4] or Rational Unified Process (RUP) [5] tried to solve Waterfall problems and present some variations in the traditional phases [1], they are still heavyweight. These traditional approaches perform a set of what we could call "pre-construction" tasks where the analysis of the problem itself is performed, in this stage the aim is to achieve a full understanding and an unambiguous definition of the problem; this way a solution can be designed, implemented and deployed attached to the static definition of the problem. This approach is usually called Big Design Up Front (BDUF) [6]

Despite this way of thinking is probably the most structured one, what happens if the needs that constitute the problem that have been analyzed at the beginning of the project, change during the life of the very project? With this question in mind, other Software Development philosophies appeared little by little.

Although it is possible to track the origins of the agile methodologies back to 1957, as Larman and Basili mentioned in their paper "Iterative and Incremental Development: A Brief History" [7]; and in 1985 Tom Gilb introduced the EVO method [8] as an alternative to Waterfall, it is in mid-1990s when some of the most popular agile methodologies are presented. Ken Schwaber presented the first version of Scrum in 1995 [9], Alistair Cockburn introduced Crystal in 1997 [10], the Extreme Programming project started in 1996, although the first book is the Extreme Programming: Explained, by Kent Beck, which was published in 1999 [11]; during these years other methodologies like Adaptive Software Development [12], Feature Driven Development [13] and Dynamic Systems Development Method [14] were also introduced.

On February 13th 2001; representatives from these approaches met to talk and relax and find a common ground, and what emerged was the "Manifesto for Agile Software Development" [15]. This manifesto states a new philosophy for Software Development based in communication, collaboration within the team and with the customer, flexibility and adaptability to changes and continuous delivery of working software.

With the birth of the Agile Manifesto, all the methodologies previously mentioned (Scrum, XP, DSDM, Crystal Methodologies, ASD, FDD, etc.) were encompassed and referred from then on as agile methodologies. These agile methodologies were refined progressively and become

more and more popular, this popularity can be justified due to its potential benefits [16]; in fact, statistics show [17] very interesting data related to the strength of agile methodologies:

- 49% of the businesses say most of their companies use agile development.
- 52 % of customers are happy or very happy with agile projects.
- There was 15% increase in the number of respondents who work where there are at least 5 Agile teams from 2011 to 2012.
- The number of those who plan to implement agile development in future projects has increased from 59% in 2011 to 83% in 2012.

This statistics shows not only the 'healthy state' of the Agile Software Development methodologies, but also the great perspectives of growth that the usage of these methodologies has.

But not everything is an advantage with Agile Software Development; there are also limitations. In their systematic review about Agile Software Development, Dybå and Dingsøyr [16] point out a limitation that repeatedly appears in literature; this is the lack of attention that agile methodologies seems to pay to architectural design issues [5] [18] [19].

Concerning architectural design, agile radical proponents consider architectural design and documentation as extra work that does not add value to the product that is being constructed, basically what they think is that You Ain't Going to Need It (YAGNI), this school of thought is openly contrary to the Big Design Up Front (BDUF) that has been previously mentioned [5].

Many authors indicate that, without Software Architecture, the project is most likely to fail [54] and also, long-term problems related with maintainability, scalability and quality of the software developed appear [5].

So the question seems to be clear, is it possible to be agile without giving up on architecture? Finding the solution to this question is the main driver of this Thesis Work.

1.1 Aims & Objectives

The main aim of this Thesis Work can be stated as:

"To study and characterize how agile methodologies and architectural design are combined in industrial projects."

This goal will be achieved through the following objectives:

1. **Objective 1:** Collect data from agile practitioners regarding architectural challenges in agile projects. This objective can be split into two different sub-objectives:

    a. **Sub-objective 1.1:** Obtain information from practitioners about the problem and its context, this is:
        i. Define what agile practitioners understand with architecture in their projects, this is, which artifacts they create, how much effort they devote to it, relevance of this kind of work, etc.

        ii. Find out the main challenges that agile practitioners face regarding architecture in agile projects.

    b. **Sub-objective 1.2:** Gather a collection of approaches (practices, methods, techniques) followed by agile practitioners to overcome challenges concerning

architecture in agile projects and its effects on the project where they are applied.

2. **Objective 2:** Compare contributions extracted from electronic databases with empirical data obtained from industry. This comparison will be established considering the challenges found, the approaches proposed, and the effects observed; The aim here is to find out how the empirical evidence supports or denies the approaches found in literature (with special interest to those ones based solely on expert opinion).

1.2 Research Questions

The general research question that drives this Thesis Work is:

How can Software Architecture be designed in agile projects in industry?

This research question can be split into the following ones:

1. **RQ 1:** *What do agile practitioners understand by architecture in the context of an agile project?*
2. **RQ 2:** *Which challenges regarding Software Architecture are found in agile projects?*
3. **RQ 3:** *Which practices/approaches are followed to overcome these challenges?*
4. **RQ 4:** *Which effects these practices/approaches provoke on the project?*

It is possible to check a map between Objectives and Research Questions in the next table (Table 1. Objectives/Research Questions mapping):

Objective		Research Question
Objective 1	**Sub-objective 1.1**	RQ 1, RQ 2
	Sub-objective 1.2	RQ 3, RQ 4

Table 1. Objectives/Research Questions mapping

The Objective 2 is addressed through the analysis of the information obtained by answering to the previous four research questions and the discussion enabled on the most important contributions and how the empirical findings support or deny SLR findings (see Results Discussion).

1.3 Expected Outcomes

The expected outcomes of this Thesis Work are:

1. **Outcome 1:**
   A list of challenges/problems faced in agile projects regarding Software Architecture.
2. **Outcome 2:**
   A collection of practices, approaches and recommendations about how to combine Software Architecture and Agile Methodologies. This collection will be created considering the effects that the practices cause in the project.
3. **Outcome 3:**
   A discussion containing a comparison between the information extracted from electronic databases and the information obtained from the interviews. The key point of this comparison is to establish a relation between claims found in literature and the empirical evidences found with agile practitioners.

1.4 Structure of the Thesis

This Thesis Work is structured as follows: In section 2 (Background & Related Work) the background on the main related topics is described. This section provides knowledge on Agile Methodologies, Software Architecture and Plan-driven development before considering the main question of the study. This section also contains a summary of the related work regarding Agile and Architecture combination and current gap that will be filled with the result of this research.

The research design is presented in Section 3 (Research Methodology). This section exposes the methods employed for conducting this research along with its motivation.

Section 4 (Systematic Literature Review: Results & Analysis) presents the results and analysis of the findings obtained from the Systematic Literature Review. Section 5 (Semi-Structured Interviews: Results & Analysis), in a similar way, presents and analyzes the findings from the Semi-Structured interviews conducted with agile practitioners.

Section 7 presents a discussion concerning the results obtained from both sources, Systematic Literature Review and Semi-Structured Interviews. This section is focused in describing the most relevant findings of this Thesis Work, reviewing which of the information obtained from Literature finds support in the empirical information got through the Interviews.

Section 8 describes the validity threats of this research, ensuring the trustworthiness of this study.

Finally, section 9 presents the conclusions of the Thesis Work, providing a final comparison between the two types of information found, and revisiting each research question and its corresponding answer. Also, the future lines of research will be contained in this section.

2 BACKGROUND & RELATED WORK

2.1 Plan-Driven Software Development

Plan-Driven approaches are based on a structured conception of Software Development; as their very name indicates, they defend planning everything from the inception to the closure of a project.

This kind of Software Development Methodologies are most recommended for well-known stable domains, they enable a whole vision of the problem to be solved, allowing the consideration of alternative solutions and making possible the creation of accurate estimations in terms of costs, effort and time.

The most popular Plan-Driven Software Development Methodologies is the Waterfall Model.

2.1.1 Waterfall Model

The Waterfall model [2] is a Plan-Driven sequential process for Software Development; it is divided in different stages that structure the process of software creation. It is interesting to point out that the first time this model was proposed by Royce, it was not referred as "Waterfall". The basis of the model is that every stage begins when its predecessor is finished, and it takes as input their predecessor output. These phases are established as Figure 1. Waterfall methodology [2] shows:

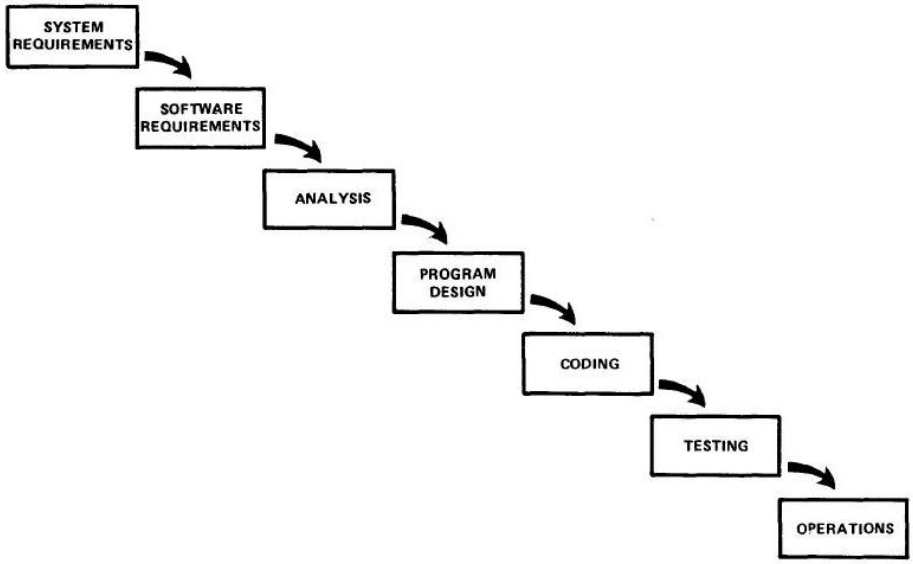


Figure 1. Waterfall methodology [2]

2.2 Agile Software Development

Contrary to the Plan-Driven Software Development methodologies, which are mainly focused on planning everything in advance, and then follow the plan; Agile Software Development defends iterative and incremental development in order to enable the change of the requirements defined at the beginning of the project.

Rapid respond to change, collaboration with the customer, short deliveries of working software, are some of the values defended by agile proponents. This Software Development philosophy is shown in the "Manifesto for Agile Software Development", published in February 2001.

The publication of the Manifesto is the official appearance of Agile Software Development, although its origins are long before 2001. According to Larman and Basili [7], iterative and incremental development was in use as early as 1957.

There are lots of agile variations, different methodologies with the same underlying principles; some of them were published before the Manifesto and have been continuously reviewed, and some others appeared after the Agile Manifesto. In this Thesis Work, the major methodologies that will be presented are: Extreme Programming (XP), Scrum, Crystal Methodologies, Feature Driven Development (FDD), Adaptive Software Development (ASD), Test Driven Development (TDD), and Agile Unified Process (AUP)

2.2.1 Agile Manifesto

The Agile Manifesto was the result of a meeting of the main representatives of different Software Development Methodologies openly different from the traditional Waterfall Model and its derivatives.

On February 11-13, 2001, the "Manifesto for Agile Software Development" was created; in this manifesto, these representatives state their intentions and expose a new philosophy for Software Development.

This "Manifesto for Agile Software Development" [15] is expressed as follows:

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

***Individuals and interactions** over processes and tools*

***Working software** over comprehensive documentation*

***Customer collaboration** over contract negotiation*

***Responding to change** over following a plan*

That is, while there is value in the items on the right, we value the items on the left more."

From this manifesto it is possible to abstract twelve guiding principles, which are:

1. Early and continuous delivery of Software for satisfying the customer.
2. Welcome changing requirements.
3. Deliver working software frequently.
4. Business and technical people working together.
5. Motivation as a driver of the project.
6. Face-to-face conversation is the most effective method of sharing information.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity 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.

As it has been stated, the agile manifesto collects a way of thinking, not propose any specific method for Software Development, it is more about the philosophy beneath the methods. Concerning Agile Methodologies, there are dozens of different ones, the most relevant in this Thesis Work are Extreme Programming (XP) and Scrum.

2.2.2 Extreme Programming (XP)

Kent Beck, Ward Cunningham and Ron Jeffries introduced Extreme Programming (XP) in late 1990s [10]. XP is one of the oldest agile methodologies nowadays. This methodology is based on frequent releases in short cycles, in order to improve quality and be able to adopt new requirements.

2.2.2.1 Values

In its first version [11], Extreme Programming found its basis in four main values, which are:

1. ***Communication:*** Good communication is considered as essential for building software, most of the problems in software projects are originated in the lack of communication among the team. XP aims to enhance communication by the usage of techniques that are not possible to conduct without communication.
2. ***Simplicity:*** This value proposes that it is better to do the simplest thing today, and pay a little more tomorrow if it is needed, than make a bigger effort today and never need that tomorrow. In other words, work just what is required when it is required.
3. ***Feedback:*** Feedback value would encourage learning from the very system that the team is building. It is possible to obtain feedback in the short term (minutes, seconds) by the creation of tests, and also in a longer term (days, weeks, months) by presenting the system to the functional tests of customers.
4. ***Courage:*** This value enables the accomplishing of the previous three ones, it is not contained in practices per se, but it is a required value for being able of performing the actions recommended by communication, simplicity and feedback values.

In a later review of the XP method [22] a new value has been added:

5. ***Respect:*** This value proposes to respect your own work and everybody else's work; if this is true, then loyalty among team members appears and all this, respect and loyalty, improves motivation in the team.

2.2.2.2 Practices

Extreme programming materializes these five values, through 12 recommended practices [11], which are:

1. ***Pair Programming:*** All the code is produced by couples of programmers in a single workstation. One of them writes code and the other one reviews it at that very moment. Pairs should change frequently, so everybody is aware of what everybody is doing.
2. ***On-Site Customer (Whole Team):*** XP proposes to include the customer as part of the project. This way, the customer is available for all the possible questions that might arise during project.

3. ***Planning Game:*** It determines the scope of the next release, considering business priorities and technical estimations.
4. ***Testing:*** Programmers write unit tests continuously, these tests must be passed before continuing coding. Customers write tests in order to check that features are finished.
5. ***Continuous Integration:*** This practice indicates that the team must continuously integrate and build the system; this is required in order to avoid delays and integration problems caused by differences in the versions that the different programming pairs work with.
6. ***Refactoring:*** This practice consist in restructure the code of the system, by removing duplicated sections of code, simplifying the code and making it more generic.
7. ***Small Releases:*** This encourages the team to perform short cycles and to deliver working software in short spans of time.
8. ***Simple Design:*** Developers should apply the famous KISS principle (Keep It Simple, Stupid) to their system design. The system should be design as simply as any given moment allows. Every time a piece of code is written, the team must ask: is there a simpler way to do this? If so, the simplest approach must be taken.
9. ***System Metaphor:*** The System Metaphor is a story that explains how the system should work to any stakeholder of the project. These stories should guide the development.
10. ***Collective Code Ownership:*** This practice can be summarized in: anyone can change anything in the code at anytime.
11. ***Coding Standard:*** In order to enable the communication among the developers through the code, this code must be written according to certain standards, so everybody can understand it.
12. ***Sustainable Pace (40-hours week):*** Workers should not work more than 40 hours in a week. If there is overtime one week, it must not be overtime the next one. The aim of this practice is to keep the team rested for being more productive and creative.

2.2.2.3 XP Project Lifecycle

An ideal XP project lifecycle contains the following six phases [11]:

1. **Exploration Phase:**

   In this pre-production phase, customers write down the stories that the first release of the project should implement. Programmers explore different technologies and experiment with them in order to find the most suitable one for the development of the project. Also, they explore possibilities for system architecture.

   This phase finishes when the team is confident to go into production phases, when they believe they are able to get the program finished. This phase can last from few weeks, if the team knows the domain and the technologies, to few months if the technologies and the domain are new.

2. **Planning Phase:**

   In this phase, the planning game is played. Stories are prioritized; developers and customers agree which of the stories are going to be implemented for the first small release. If the team has prepared properly during exploration phase, planning phase may last just a couple of days.

3. **Iterations to Release Phase:**

   The release is divided into different iterations, typically from one to four iterations. In this phase the customer is responsible to pick the stories that will enable the creation of the "whole architecture" of the system, even in a skeletal form. The customer is also responsible to select each iteration stories and to complete functional tests that should run at the end of each one. Once the last iteration is finished, the system is ready to go into production.

4. **Productionizing Phase:**

   This is the end game of the release; this phase can be described as XP's deployment phase. The aim here is to certify that the system is ready to go into production, so testing plays a key role in Productionizing Phase. In this phase the evolution of the system slows down, and the team is more concerned about addressing risks at the time of deciding whether a change should be included in the current release. When the testing is finished, the system is ready to go into production.

5. **Maintenance Phase:**

   This phase is the normal state of an XP project; this is because during maintenance the system keeps evolving over time. Maintenance phase encompasses all the previous ones, as far as new releases are launched. It should be noted that, when a system is in production, every change must be more carefully and conservatively evaluated, so change management is very helpful in this phase.

6. **Death Phase:**

   This phase appears when the customer has no more stories to be implemented. This moment is the right one to create some documentation of the system if it is needed.

Figure 2. XP Lifecycle [23] contains a graphical representation of the ideal Extreme Programming project lifecycle:

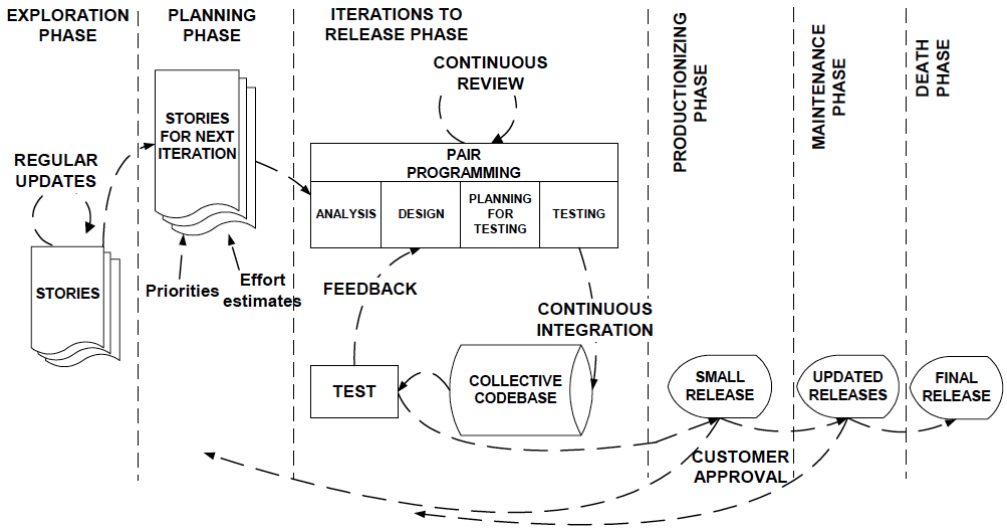


Figure 2. XP Lifecycle [23]

2.2.3 Scrum

Scrum is an Agile Software Development framework that encourages iterative and incremental work. Although the first references to Scrum are found in the article of Takeuchi and Nonaka in 1986 [24], the first appearance of the term Scrum was introduced by Ken Schwaber and Jeff Sutherland in 1995 [9] at the OOPSLA research conference. The term 'Scrum' comes from the rugby formation, rugby is used as an example for this methodology, because in both (rugby and Scrum methodology) "the team tries to go the distance as a unit, passing the ball back and forth" [24].

Scrum is defined at The Scrum Guide [25] as a framework for developing and sustaining complex products. Scrum is a framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value.

Scrum is based in the empirical process control theory, this is, knowledge comes from experiences, and decisions are made depending on what is known. Three pillars support this principle [25]:

- **Transparency:** The process must be visible to those responsible of the outcome. Standard definitions for the different aspects of the process are required for sharing a common understanding.
- **Inspection:** Scrum users must frequently inspect Scrum Artifacts for checking that there are no unexpected variations. These inspections must not interrupt the way of working.
- **Adaptation:** If a deviation of what was expected is detected in one of the inspections, then an adjustment is required for avoiding further deviations.

For describing Scrum, it is needed to depict three main aspects of this methodology: Team, Events and Artifacts.

2.2.3.1 The Scrum Team

The Scrum Teams are self-organized and cross-functional. These teams consist of a Product Owner, the Development Team and the Scrum Master.

- **Product Owner:**

  The Product Owner is the responsible of maximizing the value of the final product and the work of the Development Team [25]; he is also responsible of managing the product backlog, which is the list of requirements of the system and what states what goes next during the development process.

  The entire organization must respect his decisions for succeeding; all this decisions must be reflected in the product backlog; and as far as the Development Team is not allowed to work on anything different from the Product Backlog, he commands the Development Team on what to do.

- **Development Team:**

  The Development Team is self-organized; this means that they organize their own work, the product backlog indicates what they must do, but the way the work is organized inside the team is completely up to the very team.

Also, the Development Team is cross-functional, what means that there is no distinction between the workers, all of them share the same title: "Developers"; although as individuals they may be more focused in different aspects of the work (coding, testing, etc.).

- **Scrum Master:**

  Scrum Master is the facilitator of the Scrum Project. His mission in the project is to ensure that the team is functional, and that everything keeps working. The Scrum Master keeps the team attached to the Scrum theory, practices, and rules. Also, he is the responsible of solving the different difficulties that arise during projects, e.g.: helping the product owner with the Product Backlog, coaching the Development Team or working with other Scrum Masters for improving Scrum productivity in the Project.

2.2.3.2 Scrum Artifacts

Scrum provides a set of artifacts that represents value and transparency in the project and that enables inspection and adaptation [25]. These artifacts are:

- **Product Backlog:**

  This is a list of system requirements and other features needed in the project. This list is dynamic; is continuously evolving as new product needs are incorporated. The Product Owner maintains the Product Backlog.

  The Product Backlog lists all features, functions, requirements, enhancements, and fixes, what constitute the changes to be made to the product in future releases. Product Backlog items have the attributes of a description, order, and estimate.

- **Sprint Backlog**

  The Sprint Backlog is a set of the Product Backlog that must be implemented in a specific sprint. The Development Team is the responsible of selecting this set of work, and also of including a plan for delivering the product increment.

  For the Development team, the Sprint Backlog works as a plan that makes visible the work needed to reach the Sprint Goal, so all the team share the same vision.

  The Sprint Backlog is a dynamic element, it emerges during the Sprint; the Development Team continuously modifies it, depending on the work that is needed to reach the Sprint Goal. The Sprint Backlog is as a real-time picture of what is being done during the Sprint.

- **Increment**

  The Increment is the sum of all the Product Backlog items completed during a Sprint and all previous Sprints.

2.2.3.3 Scrum Events

- **The Sprint**

  In the Scrum Guide [25], this is defined as the "heart" of Scrum. It is a time-box of one month or less where a releasable product increment is produced. When a Sprint finishes, a new Sprint begins, and this way the project advances. Each sprint consists in four

different phases, which are: Sprint Planning Meeting, Daily Scrum, Sprint Review and Sprint Retrospective.

- *Sprint Planning Meeting:*

  The work to be done in the sprint is planned in the "Sprint Planning Meeting"; all the entire Scrum Team collaborates in this meeting. Typically, for a 30-day Sprint, this meeting lasts for eight hours; although shorter Sprints may require shorter Sprint Planning Meetings.

  In this meeting there are two issues that are clarified: "What will be done in this sprint?" And "how will the chosen work get done?"

- *Daily Scrum:*

  The Daily Scrum is a short meeting (15 minutes approximately) that takes place at the beginning of each day. The aim of the Daily Scrum is to synchronize activities and create a plan for the next 24 hours.

  In this meeting, each member of the Scrum Team explains three questions:

  - What has been accomplished since the last meeting?
  - What will be done before the next meeting?
  - What obstacles are in the way?

- *Sprint Review:*

  The Sprint Review takes place at the end of each Sprint; the aim of this is to inspect the product increment and to adapt the Product Backlog if it is needed. Typically, it lasts for four hours for a one-month sprint, proportionally less for a shorter sprint.

  The result of the Sprint Review is a revised Product Backlog that defines the probable Product Backlog for the next Sprint.

- *Sprint Retrospective:*

  The Sprint Retrospective is a meeting that takes place after the Sprint Review, and just before the Sprint Planning Meeting of the next Sprint. Normally it takes three hours for a one month Sprint.

  The aim of the Sprint Retrospective is to:

  - Inspect how the last Sprint went.
  - Identify and order the major things that went well and potential improvements.
  - Create a plan for implementing improvements to the Scrum Team.

A graphical representation of the Scrum Process can be seen in the Figure 3. Scrum Process

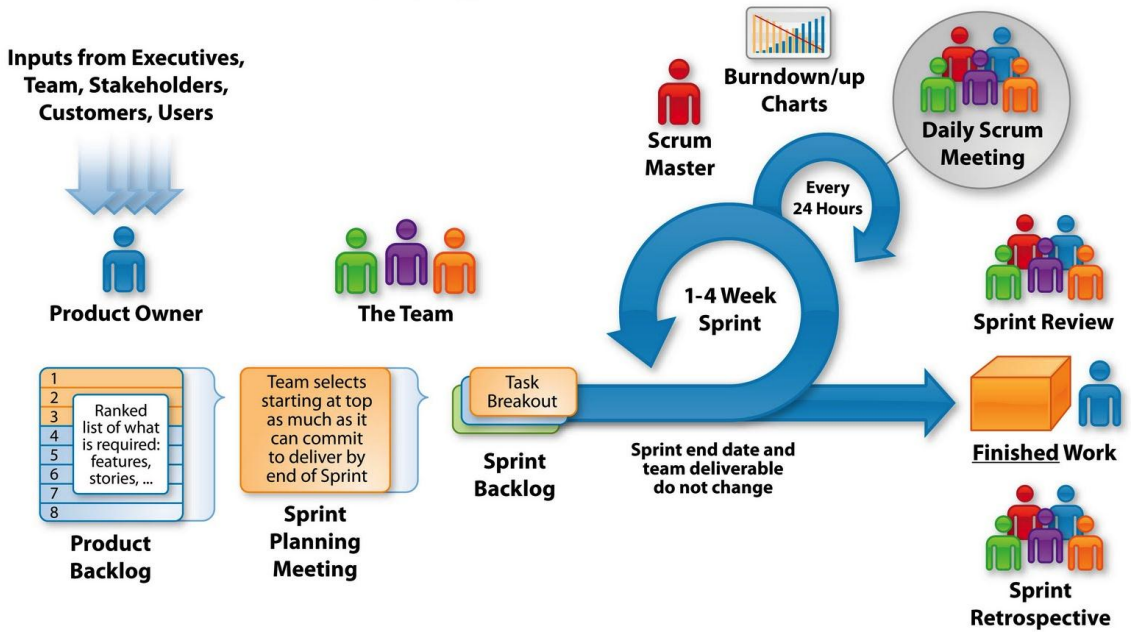


Figure 3. Scrum Process

2.3 Software Architecture

In the beginning, programming was considered as an art. Each developer understood what the system must do, and tried to formalize that into a program without any guidance but his own intuition.

This might work for small programs, but when systems are bigger and more complex, this 'artistic' philosophy will lead to complete chaos. As Garlan and Shaw state in their *"Introduction to Software Architecture"* [31]: "As the size and complexity of software system increases, the design problem goes beyond the algorithms and data structures of the computation: designing and specifying the overall system structure emerges as a new kind of problem". This new problem is addressed with the Software Architecture level of design.

2.3.1 Concept of Software Architecture

As it has been mentioned before, the Software Architecture level of design is above algorithms and data structures, the overall system structure must be specified at this level. Different authors refer to a broad range of elements of the system that should be included in Software Architecture: structure, behavior, processes, guidelines for future changes, perspectives, etc.

Some interesting definitions of architecture are:

- *"Software Architecture = {Elements, Form, Rationale} That is, a Software Architecture is a set of architectural elements that have a particular form"* (Perry and Wolf, 1992) [32].
- *"The structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time"* (SEI, 1994) [33].
- *"Software Architecture deals with the design and implementation of the high-level structure of the software. It is the result of assembling a certain number of architectural*

elements in some well-chosen forms to satisfy the major functionality and performance requirements of the system, as well as some other, non-functional requirements such as reliability, scalability, portability, and availability" (Kruchten, 1995) [34].

Together, these three definitions cover the main three aspects of a System, the same ones that Software Architecture must reflect. These main aspects are: the structure of the system, its behavior and its "future perspectives".

2.3.2 Architectural Activities

As happens with the concept, there are published many different processes for designing Software Architecture; one interesting general model, because it reflects the abstract common activities that any architectural creation process should contain, is the one proposed by Hofmeister et al. [35] which proposes the following activities:

- Architectural Analysis: this serves to define the problems that are going to be solved.
- Architectural Synthesis: this is the core of the architectural design; here the potential solutions are proposed.
- Architectural Evaluation: this ensures that the solutions proposed are the right ones.

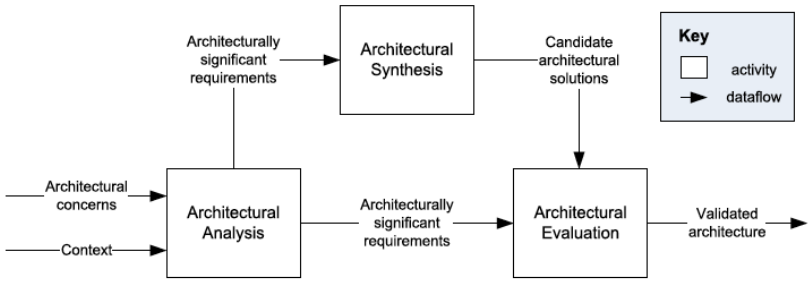


Figure 4. Architectural Activities [36]

These three activities are aimed to the architecture design of new systems, not to the evolution of existing systems. For this, a fourth activity is required:

- Architectural Evolution: this is concerned with the maintenance and adaptation of an existing architecture to environmental changes.

2.3.3 Importance of Software Architecture

Software Architecture is determinant for the success of a project, because failing at architectural level can be catastrophic. Also, it has a beneficial effect on five aspects of the project [33]:

1. **Understanding:** Software Architecture simplifies the comprehension of the system by providing a higher-level of abstraction that is more easily understood.
2. **Reuse:** Architectural descriptions supports reuse of large components and frameworks in which components can be integrated. Along these lines, it is possible to find architectures, frameworks, architectural patterns that are applicable to different projects.
3. **Evolution:** Software Architecture makes easier the maintenance of the system, by simplifying the understanding of the implications that a change has.
4. **Analysis:** Software Architecture enables new mechanisms of analysis, especially to check conformance against architecture (consistency, style, quality attributes, etc.). Benefits in terms of analysis are also enhanced by Perry and Wolf in [32]

5. **Management:** making Software Architecture a key milestone in the Software Development process involves specifying the initial requirements, expectations of growth, the Software Architecture and a rationale that demonstrates that architecture will satisfy both the initial requirements and the expectations of growth.

2.4 Related Work

As it has been mentioned before there are opposite positions towards the need of Software Architecture; on the one hand, proponents of the Big Design Up Front (BDUF) and, on the other hand, proponents of You Ain't Going to Need It (YAGNI) principle [36], where it is possible to place Agile Proponents.

This "fight" repeatedly appears mentioned in Literature, authors like Philippe Kruchten [58] or M. Ali Babar [59] indicate that, very often, Agile Proponents identify Software Architecture with massive documentation implementing unnecessary features, this is, Big Design Up Front, which is considered as a bad thing. On the other hand, those practitioners of the architectural work consider that agile approaches are [58] "amateurish, unproven, and limited to small web-based socio-technical systems".

According to Kruchten [58], the origins of the problems lie in the "axis adaptation versus anticipation"; Agile Approaches are based in the *last moment responsible* and perceive that Architectural Approaches plan too much in advance.

Despite these contrary points of view, most of the software engineering community is aware of the need of combining these two concepts: agile methodologies and architectural work; as it is reflected by Abrahamsson et al. in [6] who consider that "a healthy focus on architecture isn't antithetic to any agile process". These authors consider that it is needed to precisely define a series of concepts in order to reach a combination of these two philosophies. These concepts are the concept of Software Architecture, the moment in the lifecycle to "do architecture", the role of the architect, the amount of documentation needed, and the value of the architectural work, among others.

Along these lines, J. Madison [55] defends that "Agility and architecture are not odds", he declares that Agile offers the architect the chance of "working closely to the business and technical teams", this involves challenges, but adaptation is possible.

Mancl et al. [57] defend the need of Architecture in Agile from the architectural importance point of view. They state that architecture is essential for enabling "the economics of scale and scope". Architecture should be the driver of the system development in order to ensure that the project will finish where it is intended. Also according to them, Software Architecture reduces development effort and cost, improves reusability, enables a more complete testing and allows new members of the team to know the System. The adaptation is the key, for them [57], Architecture can be progressively created in Agile Projects

Despite all this reflections, there is a lack of specific research work that addresses the problem down to the specific practices that solve the *Architectural issue in Agile*. Agile Software Development and Software Architecture are such vast fields of study that there are contributions that tangentially deal with the topic; most of the contributions found are referred to specific contexts (methodologies, projects, teams, etc.) and they are not applicable to other ones. Almost none of them try to take a general view of the problem and gather a collection of practices to overcome the problems of integrating Agile and Software Architecture.

One of the most important studies that tries to gather the different contributions on the matter and present them synthetized is an existing Systematic Review conducted by Breivold et al.[19]. In this SLR, they state that there is an important need of merging these fields (agile and architecture) and that the combination of agile and architecture "is insufficiently supported by empirical work"; and concludes that more studies in industry should be conducted in order to better understand the interrelations of agile and architecture.

Along these lines, Babar [56] claims that the key step for creating a strategy that integrates architectural and agile approaches is "a good understanding of the current industry practices and challenges related to Software Architecture"; this idea is also encouraged by Abrahamsson et. al [6]

Despite there are similar studies related in the area, the conduction of further studies seems advisable. It is noticeable that the scope of Brievold's et al. study [19] is quite narrow, this study does not contain specific practices that can be implemented to combine the two philosophies, or the effects that those practices may cause. It does not provide any special reference to specific methodologies apart form the mapping of the different contributions included in their Systematic Literature Review. Also the concept of Software Architecture is not discussed, an issue that has been considered as highly relevant in [6]. These facts make advisable the conduction of further studies with a broader scope in order to reach a deeper understanding of the problem (a more complete view of the "big picture" of the issue concerning Agility and Architecture).

Besides the practices to solve the issue, the concept of Software Architecture requires a special interest, as it is the basis of the problem. As it has been shown in Software Architecture section, the concept of Software Architecture has been, and it still is, opened for discussion. It is impossible to determine what Software Architecture is for absolutely all the possible projects, as far as different projects have different needs. Thus, Software Architecture can be considered as an abstract and relative concept. One of the objectives of the Thesis Work will be to determine what is meant with Software Architecture in Agile Projects or, at least, if there is any common trend on what Agile Practitioners mean when they talk about Software Architecture (if they do so).

Also, it is advisable to complement this theoretical study with empirical work that support the evidences found. That way, it will be possible to clarify the possible shortages found in Literature and elicit which of the findings are empirically supported

Finally, this Thesis Work pursues to fill this gap by conducting a broader study that depicts the current state-of-the-art of the relations between Agile Software Development and Software Architecture; this is, the concept of Software Architecture in Agile Projects, the challenges confronted, how they overcome them, and also, the possible effects that might be caused in the project. Everything from a dual point of view, the contributions that can be found at scientific databases (containing both empirical and expert opinion contributions) and empirical knowledge collected from agile practitioners, that supports or denies the findings obtained from Literature.

3 RESEARCH METHODOLOGY

This section introduces the Research Methodologies that have been used in order to reach the Aims and Objectives stated. This chapter also contains discussions regarding the motivation behind each choice, alternative research methods considered, descriptions of the execution of the different research methods chosen and finally, it will establish a correlation between objectives, research questions and methods selected for making clearer the big picture of the steps taken in this Thesis Work (See Appendix C – Objectives – Research Questions – Research Methodologies Mapping).

The general process that has been done during the execution of this Thesis Work consists of different stages structured as is shown in Figure 5. Thesis Research Methodology:

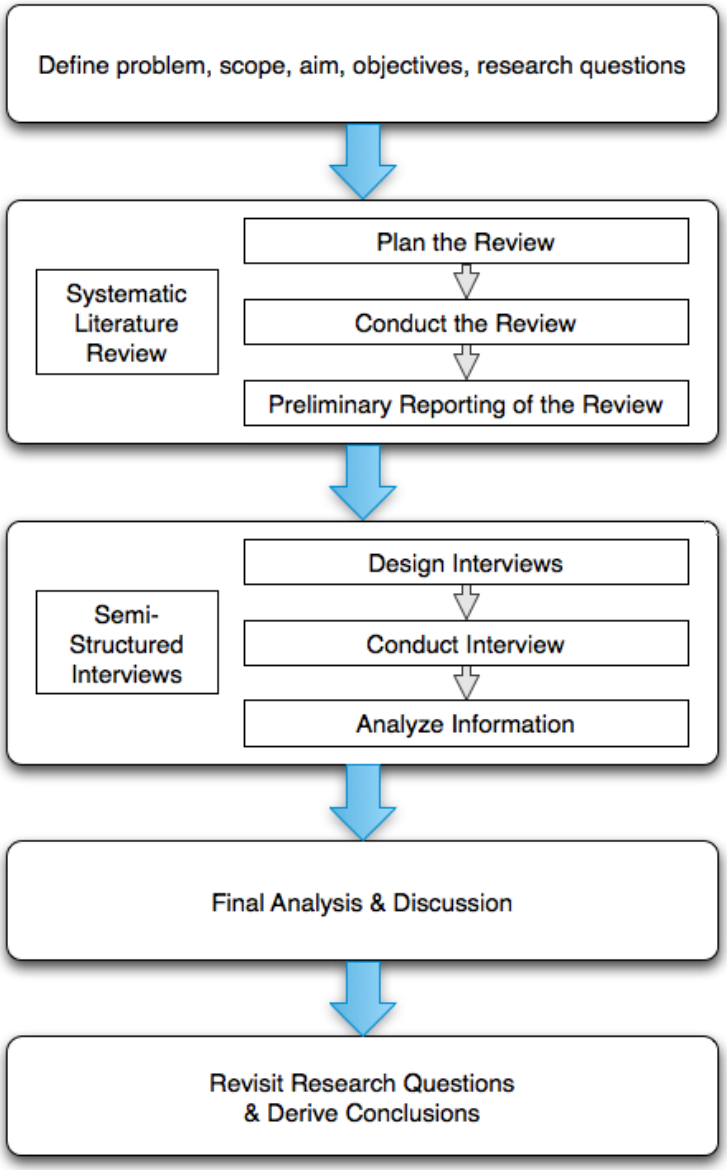


Figure 5. Thesis Research Methodology

The first step taken in the execution of this Thesis Work was the definition of the problem to be solved; this enabled setting the general aim of the thesis and its subsequent objectives. All this initial work was oriented to the formulation and refinement of the research questions, which actually drive the Thesis Work.

In order to obtain a deep understanding of the problem itself and the state-of-the-art of the potential solutions that can be found in literature, a Systematic Literature Review was conducted, by following Kitchenham's *Guidelines for performing Systematic Literature Reviews in Software Engineering* [37]. These guidelines establish three main stages for conducting a SLR, which are: Planning the review, Conducting the review and Reporting the Review (the specifics of how was carried out each of this stages will be discussed in Part I: Systematic Literature Review).

As it was pointed out in the Related Work section, it is needed to understand the perceptions of the practitioners, along with the practices conducted in industry to reach a full understanding of the problem and find out potential solutions. This idea, along with the demand in software engineering community of more empirical work in the field [4] makes necessary the conduction of interviews with agile practitioners for providing a more accurate answer to the research questions. The motivation of the choice of Semi-Structured Interviews [38] as research methodology for this stage of the Thesis Work will be provided in its correspondent section.

These two sources of information make possible to analyze the convergences and divergences between them; this way deeper discussion on the topic can be hold and a more complete answer to the aim of the thesis can be provided.

Finally, the Research Questions were revisited to provide a compact answer to each one of them and final conclusions on the issue were derived, including further lines of research that might be interesting to explore in near future.

3.1 Part I: Systematic Literature Review

The main reference used for conducting the Systematic Literature Review is *Guidelines for performing Systematic Literature Reviews in Software Engineering* [37] by Barbara Kitchenham. The reasons for using these guidelines are that those guidelines are supported with hundreds of citations, they are widely used, and also they were the specific recommendation of the supervisor for conducting the review. These guidelines establish to conduct the SLR in three main stages, each one of them composed by several activities, these are:

1. Planning the Review
    a. Identification of the need for a review
    b. Development of a review protocol
2. Conducting the Review
    a. Identification of research
    b. Selection of primary studies
    c. Study quality assessment
    d. Data extraction and monitoring
    e. Data synthesis
3. Reporting the Review

The following subsections contain the specifics about the adaptation of these guidelines for conducting Systematic Literature Reviews.

3.1.1 Motivation – Identification of the need for a review.

According to Kitchenham [39], a "Systematic Literature Review is a means of identifying, evaluating and interpreting all available research relevant to a particular research question, or topic area, or phenomenon of interest". This Thesis Work requires knowledge of the state-of-the-art about the problems concerning Software Architecture design in Agile Projects and the proposed solutions in each particular context, so a Systematic Literature Review has been seen as an appropriate research methodology in order to gather all the relevant material about that particular.

It should be noted that this is not the first SLR on the topic; there is an existing one that is fully focused on the issue about agile and architecture. This is Breivold et al. [19] Systematic Literature Review. This SLR intends to find out the perceptions in the research literature towards the relationship between agile and architecture. Although it is related and it is a very useful source of references for this Systematic Literature Review, it is noticeable that the scope of Breivold's SLR [19] is quite general for the purpose of this Thesis Work and there is a need of conducting a new SLR that is more focused on finding out the architectural problems associated to specific agile methodologies and practices to overcome them.

Nonetheless, Breivold et al. Systematic Literature Review [19] was used as a very useful starting point in order to conduct a new review of the existing literature until 2010, which is the year of its publication, but focusing more specifically on the scope mentioned in the previous paragraph.

Another point considered was the suitability of Systematic Literature Review as the appropriate research method for the purpose of the thesis; as Kitchenham remembers, there are other types of review like Systematic Mapping Studies or Tertiary Reviews.

Tertiary Reviews was directly not considered as an option because one of the keys of this Thesis Work is to obtain information directly from the source (practitioners, experts), so that discards automatically the option of conducting a tertiary review. Moreover, there is just one systematic review found on the topic, so it is impossible to conduct such a study.

More troublesome was to decide whether to conduct a Systematic Mapping Study or a Systematic Literature Review; but considering the needs of the thesis, it was decided that a deep understanding on the issue was required, the kind of understanding that a synthesis of the information found provide. Systematic Mapping Studies do not require making a synthesis of the data found, and that was the key aspect considered for choosing Systematic Literature Review as research methodology.

3.1.2 Related Research Questions

There is one research question that drives the Systematic Literature Review, this RQ is:

How can Software Architecture be designed in agile projects in industry?

It is impossible to cover this general research question stated for the Thesis Work as a whole, due to the wideness of its related fields of study. As far as it is not intended to narrow down the scope of the thesis (for example, restricting the study to specific methodologies); it is very helpful, even required, to decompose this general aim into smaller objectives that enable the achievement of the general answer. The research question is split into the next four research

questions, which will actually drive the Systematic Literature Review; these research questions are:

1. **RQ 1:** *What do agile practitioners understand by architecture in the context of an agile project?*
2. **RQ 2:** *Which challenges regarding Software Architecture are found in agile projects?*
3. **RQ 3:** *Which practices/approaches are followed to overcome these challenges?*
4. **RQ 4:** *Which effects did these practices/approaches provoke on the project?*

Descriptions of the research questions that drive the Systematic Literature Review are provided here in order to make more understandable and clear their purpose:

1. **RQ 1:** As related work points out, it is needed to understand agile practitioners' perceptions of architecture. This research question is aimed to find an answer to this issue; understanding what agile practitioners think about architecture, is the first step to discover what they expect when they are told to "do architectural work".

2. **RQ 2:** This research question goes further into the agile practitioners' perceptions; it is aimed to find out which problems exist in projects that are originated in the agile-architecture conflict. It is needed to specify that these problems might be related either with the execution of architectural work in agile, or with the lack of architectural work in agile.

3. **RQ 3:** This research question is the part of the main core of the Thesis Work; it is aimed to find out the practices or approaches that are used to merge agile and architecture, the answer to this question will be a collection of methods that serve that purpose.

4. **RQ 4:** This research question is pointed to perform an evaluation of the practices proposed, which benefits and drawbacks they cause to the project.

Answering this four research questions will enable the answer to the big research question pointed out at the beginning of this section; an answer that is articulated through the comprehension of the concept of Software Architecture for agile practitioners, the problems originated by the presence or absence of architecture in agile projects, the measures, methods or practices that are taken for merging the two concepts, and the effects that these have in the project.

3.1.3 Search Strategy

3.1.3.1 Snowballing Approach

Originally, it was intended to follow a "conventional" approach as a Search Strategy, this would consist in translating the driving Research Questions to Search Strings and executing those strings in the principal scientific databases. But as it has been mentioned in Section 1, there is an existing Systematic Literature Review aimed to depict the state-of-the-art research concerning the correlations between agile and architecture [19], this SLR was used as starting point to identify, locate and review the most relevant material by following the "Snowball" [40] technique.

This searching strategy consisted of several stages across different levels of references and citations:

1. *1st Stage: Breivold et al. study:* This stage consisted in reviewing the references included in Breivold et al. SLR [19], the selected papers from this stage will be identified as *1st level papers*.

2. *2nd Stage:* Two different parts composed this stage. The selected material from this stage is identified as *2nd level papers.*
    a. *References of the 1st level papers:* This consisted in reviewing the references that had been used by the papers in the previous stage.
    b. *Citations of the 1st level papers:* This consisted in reviewing the citations of the papers in the previous stage.

3. *3rd Stage:* In this stage, the approach was exactly the same than in the previous one:
    a. *References of the 2nd level papers:* This consisted in reviewing the references that had been used by the papers in the previous stage.
    b. *Citations of the 2nd level papers:* This consisted in reviewing the citations of the papers in the previous stage.

This search strategy enabled the location of the highest amount possible of relevant material downwards and upwards in time. The aim of this strategy was to reach a point where the references and the citations found were mostly repeated, that point indicated that the body of relevant material available on the topic had been located and reached.

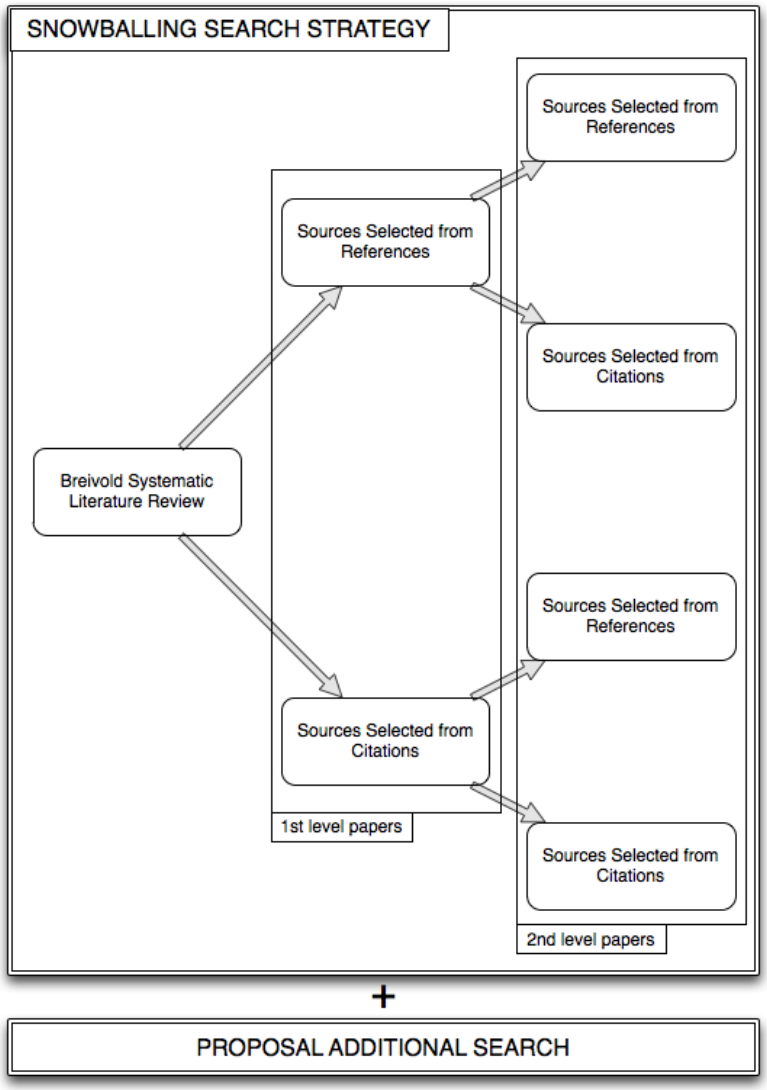


Figure 6. Search Strategy

It should be noted here that for the elaboration of the Thesis Proposal, a preliminary search in scientific databases had been performed; the results of this search were also reanalysed to find any possible contribution that had not been reached with the snowball approach explained above; in Figure 6 is shown as *Additional Search*.

3.1.3.2 Keywords

The proposed approach for searching relevant material caused that there are no search strings to be used in scientific databases; instead more effort in filtering found material was required. This fact made useful to define a set of Keywords that help in order to distinguish relevant from irrelevant material.

The first step for defining such a list was to identify the key concepts in the research questions that drive the Systematic Literature Review, those RQ were:

- *RQ 1:* What do **AGILE** practitioners understand by **ARCHITECTURE** in the context of an **AGILE PROJECT**?

- *RQ 2:* Which **CHALLENGES** regarding **SOFTWARE ARCHITECTURE** are found in **AGILE PROJECTS**?

- *RQ 3:* Which **PRACTICES** are followed to overcome the **CHALLENGES** regarding **SOFTWARE ARCHITECTURE** in **AGILE PROJECTS**?

- *RQ 4:* Which **EFFECTS** did the **PRACTICES** that overcome **CHALLENGES** regarding **SOFTWARE ARCHITECTURE** in **AGILE PROJECTS** provoke on the **PROJECT**?

The key concepts of the research questions are the words in bold, it is true that two of them are more relevant per se (agile, Software Architecture) and the others are key concepts that should be put in the appropriate context (project, challenge, practice, effect). But through these concepts it was possible to summarize what the Systematic Literature Review was aimed to find.

Once the key concepts were clear, a set of related Keywords for helping in the review process, which would identify relevant material, was created:

Key Concept	Keywords
Agile	Agile, Scrum, Extreme Programming, XP, Dynamic Software Development Method, DSDM, Crystal, Feature-Driven Development, FDD, Test-Driven Development, TDD, Pair Programming, Metaphor, Lean Development, Adaptive Software Development, ASD, You Ain't Going to Need It, YAGNI
Software Architecture	Software Architecture, Architecture, Reference Architecture, Software Design, Design, High-Level Design, Detailed Design, Pattern, Artefact, View, Model, Lifecycle, Big Design Up Front, BDUF, Methodology, Discipline, Plan-Driven Development, Waterfall
Projects	Industrial case, empirical study, case study, experiment, experience
Challenges	Problem, issue, interaction
Practices	Recommendation, solution, integration, hybrid, process
Effects	Benefits, drawbacks

Table 2. Keywords

This list of keywords helped in recognizing which papers could be relevant at a first sight, again, it is important to remark that the mandatorily one of the keywords related with agile or Software Architecture should be present, the other keywords (project, challenges, practices, effects) are just helpful to identify material that could answer our research questions.

3.1.3.3 Additional Search

In Snowballing Approach section it was mentioned that, in order to make the search more complete, the result of the preliminary search that was conducted for the thesis proposal would be reviewed.

That search was conducted in the following five databases:

- ACM Digital Library
- IEEE Xplore
- EngineeringVillage2
- SciVerse Scopus
- Google Scholar

The results obtained were not systematically registered and reviewed, but it is sure that they correspond to the search string ("agile" AND "architecture"); in total, there are 11 papers found that could be potential contributions, related with the topic of the thesis. This set of 11 papers was reviewed after the systematic process depicted in Snowballing Approach section, for making the review more complete.

3.1.4 Study Selection

3.1.4.1 Study Selection Criteria

The inclusion or exclusion of a paper was decided taking as basis the following criteria:

3.1.4.1.1 Inclusion Criteria

1. The papers included should be peer-reviewed papers.
2. The papers included should be written in English.
3. The papers should be available in full text.
4. The papers should be related to software engineering, Software Architecture or Agile Software Development.
5. The papers should contain information that addresses any of the research questions stated.
6. The papers could expose industrial cases or expert opinions.
7. In case a paper has been published twice or more, the most recent version of the paper will be included in the study.

3.1.4.1.2 Exclusion Criteria

1. Papers written in other languages than English.
2. Papers not available in full text.
3. Papers that do not relate to Software Development.
4. Papers not relevant for the research questions.
5. Papers that despite being related to agile, do not mention architectural issues at all.
6. Papers that are not primary studies.
7. Duplicated papers will be excluded (Just one copy of each study will be included).

3.1.4.2 Study Selection Procedure

As far as the Systematic Literature Review should be repeatable, in this section it is depicted how the study selection process was performed. The selection of papers was performed using an adapted Tollgate approach [41] that divides the process in different stages:

1. *Stage 1 - Title and keywords inclusion.* In this stage, those studies that are inside of the scope according to their title and keywords were selected. Here the duplicated papers were detected and eliminated.
2. *Stage 2 – Abstract inclusion.* This stage selected those papers that were relevant of the driving research questions.
3. *Stage 3 – Light Read*: A light read of the full papers trying to identify key words and concepts that are relevant for the purposes of the Systematic Literature Review (see Table 2. Keywords). This Light read also helped in acquiring the required immersion on the topic itself for being familiar with the depth and breadth of the evidence [42].
4. *Stage 4 – Deep read and conclusions.* It consisted in reading the full paper for detecting internal duplicities (like different papers that refer to the same study), irrelevant articles and just selects those ones with relevant content for the purposes of the study. This deep read is also thought to be useful for completely understand the concepts that each paper try to expose, and identify relevant sections of the text, that helped in subsequent phases of the SLR, specifically Data Extraction phase.

It should be noted that the inclusion and exclusion criteria (see section 4.) were simultaneously applied along with the selection process depicted above. The result of this process was a list of pre-selected papers ready for the quality assessment stage.

3.1.5 Study Quality Assessment

Once that a set of pre-selected papers that fulfils the inclusion/exclusion criteria previously stated (see Study Selection Criteria section) was obtained, the quality of the studies was assessed using the following quality questionnaire:

No.	Criteria	Assessment (*)
1	Are the aims of the study clearly described?	Yes/No/Partially
2	Are the findings clearly described?	Yes/No/Partially
3	Does the study refer to a specific agile methodology?	Yes/No/Partially
4	Does the study refer to specific problems regarding architecture?	Yes/No/Partially
5	Does the study depict the measures taken for designing architecture in the agile project?	Yes/No/Partially
6	Are the effects on the overall project explained?	Yes/No/Partially
7	Is the data collection method adequately described?	Yes/No/Partially
8	Are the validity threats of the study properly considered?	Yes/No/Partially
9	Is the article cited?	Yes/No/Partially

Table 3. Quality Assessment Checklist

(*) The scale for the assessment is:

- Yes = 2
- Partially = 1
- No = 0

Initially, the papers included in the SLR should get a minimum quality mark of 6 points according to the previous scale. This criterion is based on the belief that every included study should fulfil at least partially with 2/3 of the quality criteria established for the evaluation of the selected studies.

3.1.6 Data Extraction and Synthesis

3.1.6.1 Data Extraction

A Systematic Literature Review requires conducting a formal Data Extraction process that explains how is the information extracted from the selected sources. In this study, it has been used a specific method for extracting data that consists in an adaptation of the structured reading technique exposed by Cruzes et al. [43]. The procedure has consisted in conducting four main steps:

1. Read the paper to get an overview of it.
2. Read the paper looking for the following information:

a. Tested results
b. Context description

3. Highlight and number the relevant information found.
4. Transfer key details to code data forms.

The main variation from the generic method proposed by Cruzes et al. has consisted in simplifying the fourth step; the relevant information found has been transferred directly to the coding data forms (first step of the Data Synthesis process) instead of devoting time in the creation and formalization of individual data forms for each selected source. This measure was taken in order to reduce time by avoiding the generation of redundant documentation.

3.1.6.2 Data Synthesis

As its last step, a Systematic Literature Review requires carrying out a synthesis of the data collected [37]. At this point it is essential to consider the nature of data obtained, in this case this is qualitative data, so qualitative synthesis was performed.

There are many different methods for synthesizing qualitative data, in [44] Cruzes and Dybå present a summary of different methods available; among them, *"Thematic Synthesis"* [42] has been considered to be the most suitable one for the purposes of this study.

Thomas and Harden originally presented the research approach called "Thematic Synthesis" [45] in the context of Medical Research. This approach consists in reviewing the pieces of relevant information found in the selected sources line-by-line, group them, and perform successive abstractions for generating *themes* and *high-order themes*. This way, it is possible to model the general idea that synthesizes all the selected contributions and maintain a link between conclusions and the very text of the studies included.

These characteristics have been essential in the choice of Thematic Synthesis for the Data Synthesis stage of the Systematic Literature Review. In the next section, Systematic Literature Review: Results & Analysis, it will be shown that most of the contributions expose specific approaches that suits each specific context, it was required to abstract the underlying approach from each specific contribution for grouping them and generate families of strategies. It was also interesting to keep track of the origins of each strategy to enable the "going downwards" to each contribution to check different implementations of the same approach.

There is a specific adaptation for conducting Thematic Synthesis in Software Engineering [42] by Cruzes and Dybå. They extend the original method for SE and establish five steps for conducting it, these steps are:

1. **Extract data:** see Data Extraction section
2. **Code data:** Coding data consists in identify interesting segments of data by labelling. The labels should be repeatedly used for enabling the creation of categories across the entire data set. Harden & Thomas [45] recommend performing a line-by-line codification of the data, nevertheless this has been considered as not feasible due to the amount of relevant data found, instead of line-by-line, it has been considered more appropriate to work with chunks of data, as Cruzes and Dybå suggest in [42].
3. **Translate code into themes:** According to Cruzes and Dybå [42], this process consists in joining the units of information from the previous step (codes), for creating a smaller number of units that organize the information attending its nature.
4. **Create a model or high-order themes:** This step is aimed to return to the original research questions. It consists in connecting the different themes creating a general

model that addresses the research questions, which drive the study, with arguments sustained in the themes and codes and segments of text under it.

5. **Assess the trustworthiness of the synthesis:** As the name of the stage indicates, this is aimed to ensure that findings are as trustworthy as possible. The trustworthiness of the study depends on the quality of the data included and the methods used. The measures to ensure the quality of the contributions are depicted in Study Quality Assessment section. Along these lines the trustworthiness of the synthesis will be also object of discussion in Validity Threats section.

3.2 Part II: Semi-Structured Interview

It is required to clarify that this section was conducted once the Systematic Literature Review was finished; so much of the reasoning shown in the following subsections is based on the experience achieved in the SLR stage.

3.2.1 Motivation

The Systematic Literature Review depicted in the previous section provides information about the concept of Software Architecture in agile projects, the challenges that are faced concerning Software Architecture in Agile Projects, the practices to overcome them and the effects provoked on the project; but most of the relevant sources included are based in pure expert opinion or in theories yet to be proved, as Abrahamsson et. al [6] recommended, "more empirical work is required".

So, without this empirical work, the results of the Systematic Literature Review are somehow invalid; in other words, how much of the information found is actually useful in a real industrial environment. Having this idea in mind, it has been considered as a very interesting complement to conduct a set of interviews that allows the contact with the real situation of the topic.

Two main options have been identified concerning the best way to carry out this task; the first of them is to take the interviews as a validation phase for the information obtained in the Systematic Literature Review, what would involve asking agile practitioners' opinion on the results of the SLR. Although this is interesting, it has been considered as unfeasible due to the amount of practices and challenges obtained.

On the other hand, a different approach for understanding the interviews is to take them as a study per se, not dependent on the SLR, and later establish the relation with the results of the SLR through by establishing a comparison where possible. This has been considered as a much more feasible approach, and more suitable, because it would eliminate the possible predisposition that might be created in the interviewees when they saw the results of the SLR. Instead the interviewees are completely free to talk about their direct experience, and not their mind on others' experience or opinion.

The purpose of the interviews is to obtain information about experiences, thoughts, feelings, and decisions of agile practitioners in industrial projects regarding Software Architecture. As Hove and Anda [38] state, "Interviewing people provides insight into their world".

For the purpose of this work, it seems to be more appropriate to perform Individual Interviews [46] instead of group interviews; this is because the aim of the thesis it is not to create arguments in order to arise new issues regarding the topic, instead, it is trying to learn the approaches followed by the industry workers in order to solve the architectural issue in agile projects. Furthermore, an individual interview is more controllable than an open group

discussion among people who work together, because that way it is more probable that the conversation derives towards related topics of their daily work that are useless for the purposes of the Thesis Work. Finally, the feasibility of getting contacts is also a factor to keep in mind; it is more probable to obtain individual volunteers for participating in an interview than a whole team or a part of it.

The type of interview conducted is a Semi-Structured Interview [47]; these interviews combine prepared questions (to come up the topic) and open questions (that are useful to react to the information received 'on the fly'). This suits the nature of the study because, although with the help of the information obtained in the Systematic Literature Review, we can foresee the course of the interview; we need flexibility to go deep into specifics that may be unique depending on the context [51].

It has also been considered the option of performing online Surveys [48] instead of individual interviews. An online survey can be seen as a feasible way to augment the amount of data that will be collected, a feasible option is the creation of an on-line survey because, as Cook et al. state, through an electronic survey "very large or representative numbers of a population can be reached" [49]. However, including the execution of an online survey in the context of this Thesis Work will provide several problems that advise against its use, for example:

- ***Profiling:*** this Thesis Work is aimed to obtain information from agile practitioners, ensuring the identity of the responders could be difficult and would take an extra effort, so it is reasonable to think that the quality of the data obtained will not suite the purpose of this thesis. On the other hand, with personal interviews it is possible to ensure the source of the information obtained.
- ***Nature of the related topics:*** Agile and Architecture are "abstract concepts", which methodology? Which challenges? What is understood for architecture? It is quite difficult to obtain useful data with a fixed questionnaire on the Internet, because of the ambiguity of the topics. A flexible conversation is needed in order to reach the information pursued in this thesis, i.e. some questions will be based on previous answers (this motivates also the choice of Semi-Structured Interviews).
- ***Nature of the data collected:*** Interviews and Surveys provide different types of data, mixing those two different sorts of data will require an extra effort that would mean a risk in terms of time to the overall Thesis Work. So, in order to delimit a more feasible scope for the thesis, it is advisable to carry out either Interviews or Surveys but not them both.

So it has been considered that Semi-Structured Individual Interviews is the most suitable research methodology to obtain the information from agile practitioners for this Thesis Work.

3.2.2 Participants Selection

3.2.2.1 Interviewee Ideal Profile

A key issue in the conduction of these interviews is to know who is questioned. This aspect has been one of the decisive reasons for choosing Semi-Structured Interviews as research methodology to obtain information from industrial environments.

Ideally the profile of the interviewee should be a person with real experience in the usage of agile methodologies in industry, also should be interesting if that person has experience in traditional methodologies too, especially in Software Architecture usage. A professional background like this would be very interesting for the purposes of this Thesis Work, because the interviewee would talk with real knowledge of the difficulties and implications of not doing any

architectural work in agile, how can be that architectural work combined with the agile approaches, and the implications that such combination may have on the project.

Finally, a last interesting characteristic of the interviewee would be that this person has high-level knowledge on how is his team organized, even the whole project; this would be extremely interesting to find out how the agile methodology is actually implemented in his company in terms of people, tools, processes, etc.

3.2.2.2 Interviewees Search

The measures taken for finding candidates to be interviewed have been mainly three:

- **University provided contacts:** the co-supervisor of the thesis at Technical University of Madrid (UPM) has provided these contacts.
- **LinkedIn / Agile Spain groups:** other interviewees have been found by posting debates in these forums asking for volunteers.
- **Personal contacts:** personal contacts provided by the author that are currently working in agile projects in industry.

At this point, it is needed to remark a major circumstance that has made difficult to find interviewees: the interviews stage has been conducted in July 2013, when most of the workers are on holiday.

3.2.3 Interviews Protocol

3.2.3.1 Interview Strategy

As it has been said, the interviews seek to provide a deeper understanding on the perceptions of the agile practitioners [56], finding what actually happens in industry, complementing the results of the Systematic Literature Review. From this point of view, it is clear that the Research Questions that drives the Semi-Structured interviews are the same than the ones that drives the Systematic Literature Review, which are:

1. **RQ 1:** What do agile practitioners understand by architecture in the context of an agile project?
2. **RQ 2:** Which challenges regarding Software Architecture are found in agile projects?
3. **RQ 3:** Which practices/approaches are followed to overcome these challenges?
4. **RQ 4:** Which effects did these practices/approaches provoke on the project?

The guidelines for creating the procedure for conducting the Semi-Structured Interviews have been obtained from C. B. Seaman [51]. These guidelines recommend the creation of an Interview Guide; this consists in a list of open-ended questions that help in organizing the meeting. The general script for the interview can be structured in three different phases:

1. Characterization of the interviewee.
2. Characterization of the project.
3. Research Questions

3.2.3.1.1 Characterization of the interviewee

In this section, the aim is to create a profile of the interviewee in order to be able to classify the information found, and also to make easier the conduction of the interview. Information like total experience in software engineering projects, experience in agile, methodologies used in the past, role in the team, etc. might be interesting to be considered.

3.2.3.1.2 Characterization of the project

This section pursues to define the context in which the agile methodology is being applied. Information relevant here would be:

- Complexity of the project: size, knowledge of the domain, etc.
- Agile methodology used: culture, adoption, etc.

- Team: size, experience, roles, etc.
- Customer: involvement in the team, demands, etc.

3.2.3.1.3 Research Questions

This phase, the most open and broad of the three phases, will make questions aimed to solve the research questions that drove the Systematic Literature Review. This way, it is easier to establish a comparison between the information obtained from the SLR and the interviewee's testimonies.

3.2.3.2 Interview Guide

This is the structured side of the interviews, created to maintain coherence and order in the conversation, although it will be allowed to ask questions outside this script (see Appendix B – Interview Guide),

3.2.3.3 Interview Conduction

3.2.3.3.1 Anonymous Participants

In order to make the interviewee feel more comfortable sharing details about their work in the companies, it will be ensured anonymity of the answers provided.

3.2.3.3.2 Interview Duration

The intention is that the approximated duration of the interviews will be 30 minutes each one. This is because it is more likely that a tired interviewee provide less relevant information. The interview guide will also help in maintain the focus on the relevant topics and, that way, to avoid extending unnecessarily the talk.

3.2.3.3.3 Information collection

The collection of the information of the interviews will be enabled through three main methods

- *Recording Interviews:* For making easier the process of collection of data, the interviews will be recorded by using a mobile phone or capturing the screen and voice, in case of videoconferences.
- *Interviews Transcription:* The interviews recordings will not be completely transcribed to written words; instead, short notes on the content of the interviews will be taken. Interviewees' quotes that are especially meaningful will be the only content that will be literally transcribed.
  Information Translation: The interviews will be conducted in English and in Spanish, for those ones conducted in Spanish the information will be translated to English for its extraction, especially interviewees' literal quotes.

4 SYSTEMATIC LITERATURE REVIEW: RESULTS & ANALYSIS

4.1 SLR Results

As it has been mentioned in the previous chapter, the strategy for locating relevant sources has consisted in reviewing all the references and citations of Breivold et. al. Systematic Literature Review, which has been taken as a starting point, and successively repeating the process to find the whole body of relevant material available.

This strategy has provided 3086 potential sources to be analyzed, which are a considerable amount of contributions related with the topic, although it should be noted that some of these initial potential sources are repeated.

The first stage of the selection procedure consisted of reviewing the *Title* and the *Keywords* of each contribution trying to find relevant material, also in this step duplicated sources have been removed; in total 2491 references have been removed in this step.

After reviewing titles and keywords, the aim of the review has been the *Abstract* of the potential sources. The level of relevance required in this step has been that the abstract is related not just to the area or field of study object of this Thesis Work, but also to any of the research questions that drive the Systematic Literature Review. In this step, 374 sources have been eliminated.

At that point, it was possible to ensure that the "surviving" sources are in some way related to the aim of the thesis. Following the recommendations of Cruzes and Dybå [42], a light read has been conducted for getting immersed in the topic itself and identifying key aspects or ideas that are relevant for answering the research questions. The *Light Read* step eliminated 104 sources more.

Almost at the end of the process, the *Deep Read* of the remainder sources took place. This step is aimed to detect pieces of information that can be related to specific research questions, it is about locating sources that provide relevant contributions that help in answering the research questions. At the same time, the inclusion and exclusion criteria defined in the Systematic Literature Review protocol (see Part I: Systematic Literature Review section) are strictly applied. In this step, 73 sources were removed.

A detail decomposition of the potential sources found and the successive filters applied to remove irrelevant sources can be found at Table 4:

		Citations	References	Additional	Results
Potential sources		1128	1947	11	3086
Filters	Title	958	1533	0	2491
	Abstract	73	296	5	374
	Light Read	38	66	0	104
	Deep Read	41	26	6	73
Pre-Selected		18	26	0	44

Table 4. Study Selection Summary

After all this process, the pre-selected sources considered as relevant for the purposes of the Systematic Literature Review were 44 contributions. But still one last step was required to ensure the validity of the study, *Quality Assessment*. Using the questionnaire shown in Study Quality Assessment section, the quality of the pre-selected sources was assessed. The quality criteria for including or not a source was to obtain at least a mark of 6 in the questionnaire. The distribution of the quality assessment of the pre-selected sources is shown in Figure 7.

Figure 7. Quality Assessment Results

As it is noticeable in the chart, 2 sources got 5 in the questionnaire; consequently they were eliminated from the final selection, which was composed by 42 contributions. The whole selection procedure is graphically summarized in Figure 8:

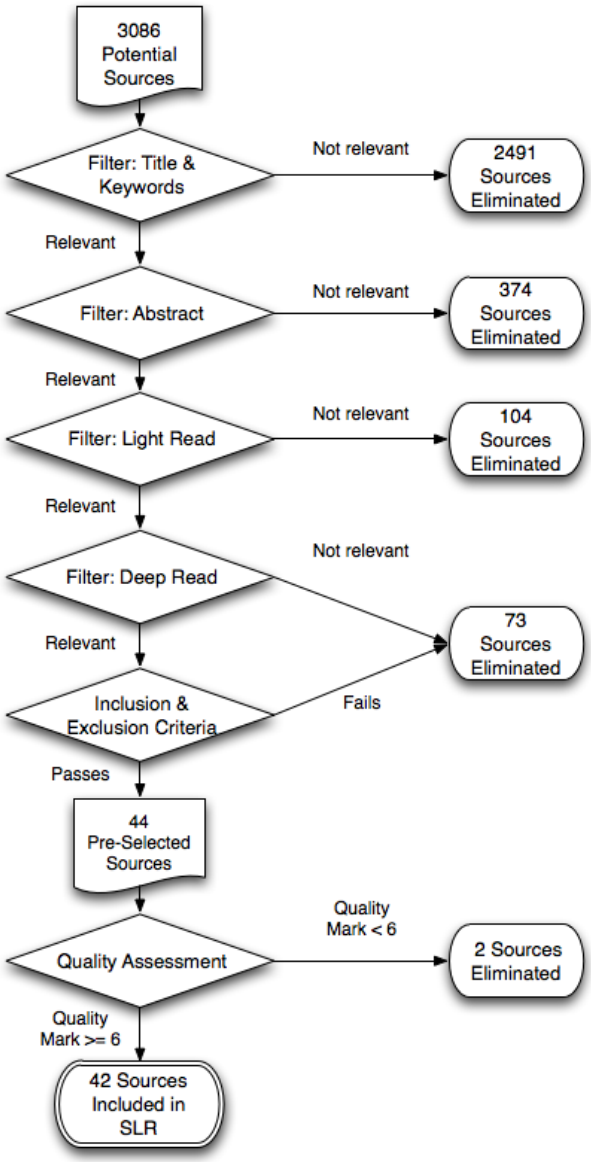


Figure 8. Study Selection Process & Results

4.1.1 Particularities of the Study

At this point it is required to make some clarifications on the process to derive the information that this section contains. The nature of the data obtained in this Systematic Literature Review required performing successive abstractions to generalize common approaches and techniques from individual adaptations that suited each particular context.

This process, as it has been mentioned in Data Extraction and Synthesis section, has been done using *Thematic Synthesis* technique [42] with some variations, to know:

- Due to the diversity of the information found, no Extraction Forms have been created; instead every relevant piece of information has been highlighted in its containing source.
- These pieces of relevant information are located in the *Coding Tables (*See Appendix D – Data Coding Tables), which presents the code assigned to each segment of relevant

data and its thematic translation; working as data extraction forms for the purposes of the SLR and, at the same time, being the first abstraction process conducted.

- The themes generated in the process depicted in the previous point, and the high-order themes that interrelate them are exposed in this section, but also, they can be checked in Appendix E – List of Themes & High-Order Themes in a schematic Version.

4.1.2 Characterization of the Included Sources

It is possible to establish different classifications for characterizing the whole body set of contributions included in the Systematic Literature Review; this is due to the diversity and the broadness of the fields of study of this Thesis Work.

- **Type of data:**

  Although the primary intention was only to include industrial studies in the sources selected for this Systematic Literature Review, the shortage of that type of papers has caused that also expert opinion contributions and mixed contributions have been included in the Study.

  - *Empirical Studies:* 6 sources – 14% of the total.
  - *Expert Opinion:* 22 sources – 53% of the total.
  - *Mixed Contributions:* 14 sources – 33% of the total.

- **Methodology related:**

  It is needed to specify that the same contribution might be related to more than one methodology.

  - *No Specific Method:* 14 sources – 33 % of the total.
  - *Extreme Programming (XP):* 16 sources – 38 % of the total.
  - *Scrum:* 9 sources – 21 % of the total.
  - *Adaptive Software Development (ASD):* 2 sources – 5% of the total.
  - *Crystal:* 2 sources – 5% of the total
  - *Feature Driven Development (FDD):* 1 source – 2% of the total.
  - *Test Driven Development (TDD):* 1 source – 2% of the total.
  - *Agile Unified Process (AUP):* 1 source – 2% of the total.

Table 5 shows a mapping between the type of study and the methodology employed for each contribution:

	Empirical Study	Expert Opinion	Mixed Contribution
No Method	[S9], [S11], [S16]	[S1], [S2], [S4], [S5], [S7], [S10], [S17], [S38]	[S3], [S6], [S15]
XP	[S23], [S28]	[S18], [S19], [S20], [S21], [S22], [S24], [S27], [S33], [S36]	[S13], [S25], [S26], [S34], [S35]
Scrum		[S32], [S36]	[S12], [S29], [S30], [S31], [S34], [S35], [S39]
ASD		[S8], [S37]	
Crystal		[S33], [S40]	
FDD		[S41]	
TDD	[S42]		
AUP			[S14]

Table 5. Type of Study / Methodology classification Mapping

4.2 SLR Findings

It is noticeable that different Agile Methodologies will present different challenges concerning Software Architecture; the approaches to solve those challenges and problems will be different too, and so they will be the consequences of implementing those measures.

The general schema of the SLR findings can be graphically seen in Appendix G – SLR Thematic Map, this graphic can provide with a mental schema of the contributions found in this study. The description of each one of the elements shown in the mentioned graphic will be explained in this section

In order to make the results found more understandable, readable and applicable to other contexts, the synthesis of the results has been done trying to abstract the spirit beneath each piece of information. A more specific relation of information attached to each specific context can be obtained in Methodological Analysis of Results.

Revisiting the driving research questions of this Systematic Literature Review, it has been possible to find the following results (see the following sub-sections).

4.2.1 Research Question 1 – What do agile practitioners understand by architecture in the context of an agile project?

The findings of the Systematic Literature Review show that there is not any special concept of Software Architecture that can be extracted from the included sources. This is probably the biggest shortage in the findings of this Systematic Literature Review; it has not been possible to find much information regarding the concept of Software Architecture in Agile Projects, or the perceptions of the Software Architecture in these contexts.

4.2.1.1 Structural concept of Software Architecture

Most of the Authors [S02] [S03] [S18] [S27] [S35] share a structural vision of Software Architecture, they are much more concerned about specifying the structure of the software than about specifying its behavior. This structural vision mainly involves specifying the components of the system and its relations [S03] [S27].

4.2.1.2 Agile Architectural Artifacts

On the other hand, there are some contributions or approaches that the *agile world* proposes towards Software Architecture [S09] [S24]. One of the agile methodologies that propose a specific artifact to conduct architectural tasks is Extreme Programming, which proposes System Metaphors [S23] as a mean to depict the general architecture of the system that is being built and also to enable communication and share a common vision among the team. But most of these artifacts, especially System Metaphors, have proved themselves to be useless (this issue is more deeply addressed in the Research Question 2 – Which challenges regarding Software Architecture are found in agile projects? section).

4.2.2 Research Question 2 – Which challenges regarding Software Architecture are found in agile projects?

In order to reach a global vision of the challenges that arise concerning Software Architecture, it is needed to distinguish between the types of challenges that may appear. Some of the problems can be prior to the project; other ones can be generated in the team or with the customer, etc.

This understanding is the first key step in order to achieve success, as Sun-Tzu says, “If you know your enemy and know yourself, you need not fear the result of a hundred battles.” [50].

4.2.2.1 Agile Suitability Constraints

Agile does not work for everything; this is a shared opinion among the majority of the authors of the sources included in the Systematic Literature Review [S4] [S9] [S10] [S13] [S16] [S25] [S28] [S32]. There are certain conditions that restrict a priori the applicability of the Agile Principles; these can be categorized as follows:

4.2.2.1.1 Agile is not suitable for large projects

Agile is not recommended for large projects, in these cases, a lack of architectural design may cause problems in terms of coupling, integrity, and scalability. Also in this type of projects, very often the customers required a complete documentation of the system due to the big dimensions of the project, so it seems clear that agile are not suitable for this purpose [S13] [S16] [S25] [S28] [S32].

4.2.2.1.2 Domain knowledge

Although not every source makes explicit mention to this challenge, it can be deduced the importance of experience in the field in the agile context. When a team is experienced in a specific area, their mastery enables the possibility of creating more evolvable solutions [S9] as far as the team possesses a deeper understanding of the problem.

4.2.2.1.3 Project complexity

Some authors point out the problems that the usage of agile in complex and/or critical projects may have [S4]. This happens because in agile environments most of the decisions are done very fast and learning on the fly, without considering all the possible situations that a system may confront.

4.2.2.1.4 Team size

This is extremely important; agile requires a great coordination and communication among the team. Working with very large teams, even distributed teams may mean an unavoidable obstacle for the correct application of the agile principles. [S10]

4.2.2.1.5 Environment Characteristics

It is very important that the general characteristics of the project meet the enhanced values of agile [S32]. For example, if the requirements of the project are stable and the changes are minimal, most probably traditional approaches will offer higher effectiveness. In such cases, agile is not recommended.

4.2.2.2 Project Challenges

This category of challenges is composed of problems that appear during the execution of the project, not prior to it like in the previous section. These difficulties are obvious either to the team or the customer and may mean very costly efforts to solve them on the fly [S3] [S7] [S9] [S10] [S11] [S13] [S16] [S19] [S20] [S21] [S23] [S24] [S26] [S27] [S28] [S29] [S32] [S40].

4.2.2.2.1 Lack of Quality

This refers to the fact that very often in agile projects, quality is not considered as relevant; time, budget and functionality get all the attention of the team [S9] and the resulting software does not present the required quality [S16] [S19] [S29].

4.2.2.2.2 No alternative design solutions considered

The effectiveness of the design in traditional methodologies results from the consideration of different possible solutions; in agile methods there is not such a discussion [S9].

4.2.2.2.3 Lack of architectural evaluation

Agile methodologies do not provide any mean of evaluating architecture. Thus, there is no way to guarantee that the system meets quality goals [S24] [S40].

4.2.2.2.4 Lack of professional management

Software architects usually take a mentoring role; they attend doubts of the developers, guide the development of the system and organize the next steps to be taken. Without this guidance, all the development becomes more confusing [S16].

4.2.2.2.5 Lack of conceptual integrity

Conceptual integrity refers to how well software conforms to a single, simple set of design ideas. Without Software Architecture, it is most probable to lose it [S3] [S27].

4.2.2.2.6 Non-reliable software is produced

This refers to the bigger possibilities of failure of a system that is not supported by Architectural work behind [S10] [S11] [S13] [S32].

4.2.2.2.7 XP proposals for architecture do not address the problem

Although these findings are presented without focus on any particular agile methodology, due to the repeated appearance of the issue, this theme has been included in the synthesis. Extreme Programming proposes System Metaphor as a mean of creating some kind of architecture, but very often in the sources included in the SLR [S7] [S19] [S20] [S21] [S23] [S24] [S26] [S27] [S28] it is assured that System Metaphors are useless.

4.2.2.3 Team Challenges

This family contains challenges that directly affect the team, either in their perception of the tasks they are carrying out, their communication (misunderstandings, discussions, difficulties in establishing communication) or in the dependency or their experience for performing an effective work [S9] [S16] [S18] [S28] [S29] [S32].

4.2.2.3.1 Reliance on Experience

This reflects that agile methodologies rely too much on the experience of the people who composes the team, not in the method itself. This can lead to a

dependency of the individuals that might mean huge problems in the case those individuals are not in the team [S9] [S16] [S18] [S28] [S32].

4.2.2.3.2 Perceived irrelevance of the architectural work

This challenge is related with the perceptions of the agile practitioners; it has to do with the low relevance that the architectural work has for the team, which often sees it as extra work that does not produce anything [S16].

4.2.2.3.3 Team Communication Problems

Software Architecture is a very useful vehicle to enable communication among the team; all the team members can share a common vision of the software by consulting the architecture of the system. Removing this vehicle may lead to lose the common vision of the system [S9]. Also this challenge makes reference to the difficulties in coordinating teams that are not co-located, in those cases communication turns difficult, and the lack of architecture makes it even harder [S29]

4.2.2.4 Customer Challenges

These challenges share the common characteristic that they all directly involve the customer; either during the project [S01], or when the software is delivered [S16] [S26].

4.2.2.4.1 Customer Communication Problems

Besides providing a complete description of the system, Software Architecture is basic for enabling a fluent communication with the customer in order to ensure the validity of the Software that is being developed. The lack of Software Architecture makes this communication much more complicated and misunderstandings between team and customer appear [S01].

4.2.2.4.2 Usability Problems

If there is not any Architectural reference, it is most probable that important problems concerning the usability appear. If the Software delivered is not usable, it will provide no value to the customer, regardless how internally right it is or how quickly it is delivered [S26].

4.2.2.4.3 Lack of documentation

Without Software Architecture, the customer does not have any reference document that allows him to fully understand his system; instead, there is a total dependency on the people who built the system for getting to know it [S16]. If this happens, the customer could find himself in a situation that he does not know his own system and has no tool (except from the Source Code) for getting this knowledge.

4.2.2.5 Long-Term Challenges

These challenges are not obvious at the end of the project; they are problems that affect later in the life of the software. Without architectural work, what enables the capability of foresee this kind of issues, they are extremely difficult to detect until it is too late (e.g.

add new functionality to the original system) [S01] [S04] [S06] [S13] [S16] [S17] [S19] [S26] [S27] [S28] [S29] [S31].

4.2.2.5.1 Lack of reusability

Software produced with agile methodologies lack the capability of being reused; this is caused mainly due to the absence of a good design that helps in the creation of well-defined portions of software with well-defined functionalities, which could be used in future systems [S06] [S13] [S16] [S29] [S31].

4.2.2.5.2 Maintenance problems

Agile software often presents long-term problems. Difficulties in the maintenance appear because there is not an architecture document where the information of the system is kept, what makes much harder to carry out tasks like: detect and fix incidences, add functionalities, optimize current functionalities, etc. [S01] [S16] [S29].

4.2.2.5.3 Scalability problems

Without architectural work behind, it is quite impossible to think of a system in the long term. Without Software Architecture, the only choice for understand how a system is built, is reading source code what may become a nightmare and makes very difficult to scale a project [S04] [S17] [S27] [S28] [S29].

4.2.2.5.4 Structural problems

The software produced without considering Software Architecture design does not present a good structure, leading to problems in terms of reusability, reliability and scalability [S13] [S19] [S26].

4.2.2.5.5 Lack of flexibility

The same way, it becomes very difficult to scale a system without Software Architecture, it also makes extremely difficult to adapt the system for different purposes in the future [S26].

4.2.3 Research Question 3 – Which practices are followed to overcome these challenges?

In order to show the findings in the clearest way possible, this section is structured as follows:

First, the general approaches followed to merge Agile Software Development and Software Architecture are presented. Each approach contains a list of practices that can be assigned to it, but this does not mean that a specific practice cannot occasionally appear in contributions that follow other approaches; it only means that the “spirit” of that technique corresponds to the approach where it is located.

After that, a second categorization of the practices is provided; this time the categorization has been done attending to the nature of the practice and the area where the practice is focused.

Finally, the practices are briefly described, including the sources where the specific application of each one of them can be found. It should be remarked that for finding the specific application

of one of the practices gathered here, it is needed to consult (ref to annex coding tables), in order to locate the specific segment of text where the application is exposed.

4.2.3.1 Proposed Approaches

Contrarily to the initial intuition that the best way to combine Software Architecture and Agile Software Development would be to modify Agile processes for supporting Architectural Design tasks, the findings of the Systematic Literature Review has shown that adding architecture to agility is one possibility, but another possibility is to make architectural work more agile, and also there is another option that consists in forgetting everything and developing a completely new methodology that combines the two worlds: the dynamism of agile with the discipline of Software Architecture and traditional approaches.

4.2.3.1.1 Agile with Architecture Approach

This approach encompasses practices that are added to an agile project in order to enable architectural design. It consists in varying the activities that are typically carried out in the agile project with architectural practices that can be inserted and used trying to affect the less possible to the agility of the overall process. The sources included in the Systematic Literature review that follow Agile with Architecture approach are: [S7] [S9] [S11] [S16] [S18] [S19] [S21] [S22] [S23] [S24] [S26] [S27] [S28] [S29] [S30] [S39] [S40] [S41]. The practices attached to this approach are:

- Embed architectural practices into agile.
- Architectural evaluation.
- Preliminary high-level design.
- Design upfront.
- Design patterns usage.
- Perform impact analysis in every iteration.
- Iteration Zero.
- Refactoring techniques.
- Simple Design (KISS principle).
- Architectural Scrum sprints.
- Design meetings.
- Customer involved in architectural tasks.
- Devoted roles.
- Usage of knowledge.
- Experience-based architecture.
- Enhance quality.
- Quality attributes enable architecture.

4.2.3.1.2 Agile Architecting Approach

This approach proposes to make architectural modeling more agile. It consists in applying techniques that simplify the architectural process; this way, it is possible to obtain the benefits of the agile methodologies in a traditional project. The sources included in the Systematic Literature review that follow Agile Architecting approach are: [S2] [S3] [S5] [S6] [S14] [S31] [S34] [S37]. The practices attached to this approach are:

- Prioritization of input.
- Iterative & Incremental process.
- Embed XP practices into RUP iterations.
- Communication tools usage.
- Architect as communication enabler.
- Divide teams.
- Integration effort.
- Two-cycle methods.

4.2.3.1.3 Mixed Methodology Approach

These approaches are a mixture of agile and traditional development that is balanced according to a certain criteria. The methodologies that arise from this approach should vary in time; they need to be readjusted as projects and team evolve. The sources included in the Systematic Literature review that follow Mixed Methodology approach are: [S1] [S4] [S8] [S10] [S12] [S13] [S15] [S17] [S20] [S25] [S32] [S33] [S35] [S36] [S38] [S42]. The practices attached to this approach are:

- Hybrid approach.
- Risk driven methodology.
- Project characterization.
- Monitor and readjust methodology.

4.2.3.2 Practices Categorization

Previously, the practices have been presented grouping them attending to the general approach where they belong. But a different categorization may arise following the spirit of the classification of the challenges; this is attending to the nature of the practices proposed. As far as a description of each practice has been provided in the previous subsection, it will be not repeated again here.

4.2.3.2.1 Architecture-inspired practices

These practices have in common that they are conceived of architectural practices. The idea here is taking architectural activities that are performed in traditional projects and incorporating them into the agile methodology. These practices are:

- Embed architectural practices into agile [S9] [S29] [S32] [S41].
- Architectural evaluation [S9] [S12] [S19] [S29] [S36] [S37] [S38].
- Preliminary high-level design [S7] [S8] [S9] [S12] [S13] [S16] [S27] [S36] [S37] [S39] [S41].
- Design upfront [S28] [S42].
- Design patterns usage [S18] [S31].
- Perform impact analysis every iteration [S13] [S18].
- Risk driven methodology [S1] [S5] [S15] [S17].
- Monitor and Readjust methodology [S4] [S15].

4.2.3.2.2 Agile-inspired practices

In this case, these practices are conceived of processes or principles from agile approaches. This consists in transforming agile procedures to make them compatible with architectural work (e.g. take a Scrum Sprint and make it an

Architectural Scrum Sprint by focusing it in architectural work). These practices are:

- Iteration Zero [S9] [S10] [S12] [S13] [S16] [S18] [S26] [S27] [S37].
- Refactoring techniques
  - Refactoring [S5] [S9] [S19] [S36] [S40].
  - Reviewing [S16] [S26].
  - Refinement [S17] [S34].
  - (Re)-Design [S21].
- Simple design (KISS principle) [S2] [S5] [S26] [S35].
- Architectural Scrum sprints [S31] [S35].
- Prioritization of Input [S2] [S8] [S13] [S26] [S30] [S31] [S38] [S39].
- Iterative & Incremental process [S5] [S6] [S7] [S8] [S16] [S17] [S19] [S21] [S25] [S26] [S27] [S30] [S31] [S33] [S34] [S37] [S38] [S39] [S40].
- Embed XP practices into RUP iterations [S20].
- Project characterization [S4].
- Monitor and Readjust methodology [S4] [S15].

4.2.3.2.3 Communication practices

These practices are focused in enhancing communication; they pursue to enable architectural work through the improvement of the communication among the team and with the customer. These practices are:

- Design meetings [S12] [S26] [S32] [S40].
- Customer involved in architectural tasks [S2] [S5] [S9] [S18] [S25] [S39] [S40].
- Communication tools usage [S9] [S26].
- Architect as communication enabler [S7].

4.2.3.2.4 Quality practices

In a similar approach than *Communication practices*, this type of practices is oriented to improve the quality of the software created, and in that context architectural design is needed. These practices are:

- Enhance quality [S9] [S16] [S18] [S29].
- Quality attributes enable architecture [S18] [S29].

4.2.3.2.5 Costly practices

This kind of practices shares the characteristic that they may result expensive for the project. This cost can be expressed in terms of money (hiring extra people, buying external solutions, etc.), people (depending on individuals), time (stop the flow of the activities to perform extra practices or creating new *ad hoc* methodologies), etc. These practices are:

- Devoted roles [S7] [S9] [S10] [S16] [S19] [S26] [S36] [S39].
- Usage of knowledge [S33].
- Experience-based architecture [S5] [S11] [S16].
- Divide teams [S13].
- Integration effort [S13] [S21] [S25].
- Two-cycle methods [S3] [S6] [S14] [S30].

- Hybrid approach [S9] [S42].

4.2.3.3 Practices Definition

4.2.3.3.1 Embed architectural practices into agile

This technique consists in embedding defined processes for architectural activities into an agile method; this way, it is possible to address architectural work in those contexts [S9] [S29] [S32] [S41].

4.2.3.3.2 Architectural evaluation

4.2.3.3.2.1 Performed at high-level:

This consists in evaluating the architecture at the beginning of the project, before any implementation [S9].

4.2.3.3.2.2 Performed at the end of each iteration/sprint:

This consists in evaluating the architecture at the end of each sprint or iteration, so the evaluation of the architecture is done iteratively [S12] [S19] [S29] [S36] [S37] [S38].

4.2.3.3.3 Preliminary high-level design

This makes reference to the creation of a high-level design generated at the early stages of the project; this early design guides the actions during the rest of the life of the project [S7] [S8] [S9] [S12] [S13] [S16] [S27] [S36] [S37] [S39] [S41].

4.2.3.3.4 Design upfront

It consists in including a "traditional" design phase in an agile project; it addresses the problem of architecture, but provokes several drawbacks like losing agility, so it is recommended in combination with other approaches or in hybrid approaches [S28] [S42].

4.2.3.3.5 Design patterns usage

It consists in using architecture patterns in agile context, either to add more architectural work to agile tasks or as a way of making architectural work more agile [S18] [S31].

4.2.3.3.6 Perform impact analysis every iteration

This practice analyzes the consequences of the changes introduced each iteration in order to identify risks and address them [S13] [S18].

4.2.3.3.7 Iteration Zero

This practice consists in including a "Zero" iteration (previous to the common ones) for carrying out general analysis of the projects, create preliminary designs and setting up the general context of the project [S9] [S10] [S12] [S13] [S16] [S18] [S26] [S27] [S37].

4.2.3.3.8 Refactoring techniques

These practices share a defining characteristic; they all progressively modify the design. These changes might depend on:

- Code implementation (Refactoring) [S5] [S9] [S19] [S36] [S40].
- Revision of a supervisor (Reviewing) [S16] [S26].
- Specification of the lower levels of the design (Refinement) [S17] [S34].
- New functionalities that arise during the project ((Re)-Design) [S21].

4.2.3.3.9 Simple Design (KISS principle)

It consists in creating a design as simple as possible, it makes reference to the agile principle that says: "Simplicity is essential". It could be implemented by defining an adjusted set of artifacts that must be created to satisfy the architectural needs of the project and no more [S2] [S5] [S26] [S35].

4.2.3.3.10 Architectural Scrum sprints

This is specially indicated for Scrum, it consists in introducing sprints where all the work is related to Software Architecture. Also, it can be introduced sprints that are mixed architecture-implementation. This way, architectural work is carried out in the same work routine than implementation [S31] [S35].

4.2.3.3.11 Design meetings

This practice consists in including design in daily meetings; this way, design becomes a joint practice of all the team. This approach combines the design work with the agile principle of "working together" [S12] [S26] [S32] [S40].

4.2.3.3.12 Customer involved in architectural tasks

This practice consists in including the customer in meetings with the team to contribute in architectural tasks. Thus, the product is more adjust to the customer's demands [S2] [S5] [S9] [S18] [S25] [S39] [S40].

4.2.3.3.13 Devoted roles

This indicates that extra people fully devoted to architectural tasks is required. This means an extra cost for being able to address Software Architecture design [S7] [S9] [S10] [S16] [S19] [S26] [S36] [S39].

4.2.3.3.14 Usage of knowledge

This refers to the reliance on the experience of the team members and also on capturing architectural knowledge for later reuse or buying predefined solutions that meets the needs of the project (e.g. COTS) [S33].

4.2.3.3.15 Experience-based architecture

This "practice" consists in obtaining architecture by relying on the experience of the team members. This practice may lead to a dependency of those very individuals who built the system [S5] [S11] [S16].

4.2.3.3.16 Enhance quality

This practice consists in remarking the importance of quality at the time of measuring the success of a project; it also refers to specific measures taken for caring about quality aspects of the software [S9] [S16] [S18] [S29].

4.2.3.3.17 Quality attributes enable architecture

This practice is based in using quality attributes as a driver to obtain architectural design. The identification of the quality attributes takes place in meetings (workshops) where the customer may be present [S18] [S29].

4.2.3.3.18 Prioritization of inputs

This practice consists in the classification of the input of the project (whatever it is: User Stories, Sections or Requirements) according to its priority in order to be addressed first [S2] [S8] [S13] [S26] [S30] [S31] [S38] [S39].

4.2.3.3.19 Iterative & Incremental process

This indicates that the design tasks are addressed in several times and, sometimes, incrementally; this is, focusing progressively in bigger domains. It should be remarked that this two characteristics may appear together or separated, this is, iteratively, incrementally or both at the same time [S5] [S6] [S7] [S8] [S16] [S17] [S19] [S21] [S25] [S26] [S27] [S30] [S31] [S33] [S34] [S37] [S38] [S39] [S40].

4.2.3.3.20 Embed XP practices into RUP iterations

This practice consists in inserting agile practices from XP method into Rational Unified Process iterations (e.g. coding by using pair programming) [S20].

4.2.3.3.21 Communication tools usage

It makes reference to the usage of tools as a mean for improving team communication (e.g. using a wiki). This practice may mean extra costs for acquiring and deploying these tools [S9] [S26].

4.2.3.3.22 Architect as communication enabler

This practice consists in using the software architect role as a communication vehicle for the team and with customers. Software Architect is a figure that has a general vision of the system that is being built; thus, this figure can enable that the team share a common vision of the system; and also, it can be the best way to show this vision to the customer and check if everything is accepted [S7].

4.2.3.3.23 Divide teams

This practice is indicated to overcome limitations concerning large teams and the communication problems that this entails. Usually, team division goes together with problem decomposition or with concurrent work (both of these practices require extra costs in terms of integration or tools usage) [S13].

4.2.3.3.24 Integration effort

This indicates that extra effort must be devoted to the integration, this happens when there is decomposition of the general problems [S13] [S21] [S25].

4.2.3.3.25 Two-cycle methods

This refers mainly to application of agile in the context of Product Line Engineering, and reflects the usage of agile methods in the Product Development cycles. This consists in performing architectural tasks in the context of the Domain Engineering phase, and just performing small adaptations in the Product Engineering phase, where agile methodologies can be applied [S3] [S6] [S14] [S30].

4.2.3.3.26 Hybrid approach

This practice tries to overcome the difficulties that agile methodologies find in some contexts (e.g. in unknown domains). The approach consists in following a hybrid traditional-agile methodology in order to meet the domain and at the same time apply agile principles where possible [S9] [S42].

4.2.3.3.27 Risk driven methodology

This practice consists in creating a methodology thought to be agile but addressing the risks that the application of agile might have on the project [S1] [S5] [S15] [S17].

4.2.3.3.27.1 Risk assessment:

This consists in the examination of the project identifying potential risks.

4.2.3.3.27.2 Develop a hybrid methodology addressing risks:

To create a methodology that balances traditional approach and agile approach attending to the risk assessment done.

4.2.3.3.28 Project characterization

This practice consists in assessing the project attending to five factors (personnel, dynamism, culture, size, and criticality) [S4].

4.2.3.3.28.1 Project assessment:

Analyze project and evaluate the 5-axes of the method (personnel, dynamism, culture, size, and criticality).

4.2.3.3.28.2 Develop a hybrid methodology based on 5-axes assessment:

It consists in creating a hybrid traditional-agile methodology that addresses the previous characterization.

4.2.3.3.29 Monitor and readjust methodology

It consists in reviewing the methods that are being applied and reevaluate them in order to maintain the right balance between traditional and agile approaches in hybrid methodologies [S4] [S15].

A graphical mapping of the practices according to the approaches and the nature classification can be seen in Appendix F – SLR Practices (RQ 3) Dual Classification.

4.2.4 Research Question 4 – Which effects did the practices that overcome these challenges provoke on the project?

Following the logical flow of reasoning, after knowing the main challenges concerning Software Architecture in Agile Software Development projects and the practices used to overcome them, it is time now to check which effects these practices may have on the project.

4.2.4.1 Enhanced agile principles

This family of effects shares the feature that they all are contained in the agile principles behind the agile manifesto [15]: principles like "more communication", "quick deliveries", "customer involvement", etc. are present here [S3] [S6] [S9] [S22] [S25] [S26] [S28] [S29] [S39].

4.2.4.1.1 Less documentation

Fewer documents are produced [S9].

4.2.4.1.2 Enhanced communication

Communication is improved both among the team and with the customers [S9] [S26].

4.2.4.1.3 Simpler design

The design created is less complex, more understandable and easier to handle [S3] [S9] [S22] [S26].

4.2.4.1.4 Agility maintained

Agility of the project is not affected by the practices introduced to obtain architecture [S28] [S29] [S39].

4.2.4.1.5 Enhanced customer involvement

The practices make that the customer is more involved in the tasks of the project, this way, the customer is more aware of what is been doing in the project [S25] [S26].

4.2.4.1.6 Reduced time to market

The adopted practices reduce the general duration of the project (until it is on the market) [S6] [S25]

4.2.4.1.7 Reduced development time

The practices introduced reduced the time required for the development (understood as coding) [S28] [S29]

4.2.4.2 Architectural benefits

This category makes reference to the capabilities that are enabled through architectural design and that are absent in the usual agile approaches [S6] [S22] [S25] [S28] [S29] [S38] [S42].

4.2.4.2.1 Flexibility enabled

The resulting software is more versatile in order to be modified for addressing different purposes [S6].

4.2.4.2.2 Scalability enabled

The adopted practices make possible to scale the project [S22] [S25].

4.2.4.2.3 Improved quality

The adopted practices enhance the quality of the software produced [S22] [S29] [S38]

4.2.4.2.4 Fewer defects

The adopted practices introduce fewer defects in the system or help in the early detection of them [S42].

4.2.4.2.5 Enhanced reusability

The software produced is more reusable, so it is possible to save time and cost in future projects [S6] [S28] [S38].

4.2.4.3 Negative Effects

This *family* makes reference to some drawbacks that the usage of certain practices to overcome the "architectural issue in agile" may cause on the project [S3] [S26] [S42].

4.2.4.3.1 Architectural work reduces agility

The adopted practices augment the duration of the project, however this investment in architectural work may produce a reduction of the time required for performing *"long-term tasks"* like scale the project in the future. From this point of view this agility reduction would be concerning the immediate process, which takes more time as far as there are more tasks to conduct [S26].

4.2.4.3.2 Increased development time

The required time for the development is higher because of the adopted practices [S42].

4.2.4.3.3 Team experience dependency

The practices introduced are based on the experience of the team. This raises a problem; if there is no reference documentation of the system, every time it is desired to change something, the team that built the system is needed, because they are the only ones that possess the required knowledge about how is the system. This dependency may be a very important problem for any organization [S3].

4.2.5 Methodological Analysis of Results

For practical purposes, it has been considered as an interesting contribution to include a mapping of the results found, attending to the Agile Methodology.

It has been mentioned that there are some sources that do not make any reference to a specific method, and others that are focused in one or several specific agile methodologies (See Table 5). In this section, it is presented a classification of the challenges, practices and effects caused grouped by agile methodology (for the description of each one of them, consult previous sections).

4.2.5.1 Extreme Programming (XP)

4.2.5.1.1 Challenges detected

- XP proposals for architecture do not address the problem [S19] [S20] [S21] [S23] [S24] [S26] [S27] [S28].
- Agile is not suitable for large projects [S13] [S25] [S28].
- Structural problems [S13] [S19] [S26].
- Reliance on Experience [S18] [S28].
- Scalability problems [S27] [S28].
- Lack of reusability [S13].
- Non-reliable software is produced [S13].
- Lack of quality [S19].
- Lack of architectural evaluation [S24].
- Usability Problems [S26].
- Lack of flexibility [S26].
- Lack of conceptual integrity [S27].

4.2.5.1.2 Practices implemented

- Iterative and incremental process [S19] [S21] [S25] [S26] [S27] [S33] [S34] [S35].
- Refactoring techniques [S19] [S21] [S26] [S34] [S36].
- Iteration Zero [S13] [S18] [S26] [S27].
- Preliminary high-level design [S13] [S21] [S27] [S36].
- Integration effort [S13] [S21] [S25].
- Devoted roles [S19] [S26] [S36].
- Prioritization of input [S13] [S26].
- Perform impact analysis in every iteration [S13] [S18].
- Customer involved in architectural tasks [S18] [S25].
- Simple design (KISS principle) [S26] [S35].
- Architectural evaluation at the end of each iteration [S19] [S36].
- Embed XP practices into RUP iterations [S20].
- Design meetings [S26].
- Divide teams [S13].
- Design upfront [S28].
- Communication tools usage [S26].

- Enhance quality [S18].
- Quality attributes enable architecture [S18].
- Design patterns usage [S18].
- Risk assessment [S18].
- Usage of knowledge [S33].

4.2.5.1.3 Effects observed

Positive:

- Agility maintained [S28].
- Reduced development time [S28].
- Enhanced reusability [S28].
- Scalability enabled [S22] [S25].
- Enhanced customer involvement [S25] [S26].
- Reduced time to market [S25].
- Enhanced communication [S26].
- Simpler design [S22] [S26]
- Improved quality [S22]

Negative:

- Architectural work reduces agility [S26]

4.2.5.2 Scrum

4.2.5.2.1 Challenges detected

- Lack of reusability [S30] [S31].
- Lack of quality [S29].
- Maintenance problems [S29].
- Team communication problems [S29].
- Scalability problems [S30].
- Reliance on Experience [S32].
- Non-reliable software is produced [S32].
- Agile is not suitable for large projects [S32].
- Agile is not suitable for large projects [S32].

4.2.5.2.2 Practices implemented

- Iterative and incremental process [S30] [S31] [S34] [S35] [S39].
- Architectural evaluation at the end of each iteration [S12] [S29] [S36].
- Preliminary high-level design [S12] [S36] [S39].
- Prioritization of input [S30] [S31] [S39].
- Refactoring techniques [S34] [S36].
- Architectural Scrum Sprints [S31] [S35].
- Devoted roles [S36] [S39].
- Simple design (KISS principle) [S34].

- Design meetings [S12] [S32].
- Embed architectural practices into agile [S29] [S32].
- Enhance quality [S29].
- Iteration Zero [S12].
- Quality attributes enable architecture [S29].
- Two-cycle method [S30].
- Design patterns usage [S31].
- Customer involved in architectural tasks [S39].

4.2.5.2.3 Effects observed

Positive:

- Agility maintained [S29] [S39].
- Reduced development time [S29].
- Improved quality [S29].

4.2.5.3 Adaptive Software Development (ASD)

4.2.5.3.1 Challenges detected

- No specific findings

4.2.5.3.2 Practices implemented

- Iterative and incremental process [S8] [S37].
- Preliminary high-level design [S8] [S37].
- Architectural evaluation at the end of each iteration [S37].
- Iteration Zero [S37].
- Prioritization of requirements [S8].

4.2.5.3.3 Effects observed

- No specific findings

4.2.5.4 Crystal Clear

4.2.5.4.1 Challenges detected

- Lack of architectural evaluation [S40].

4.2.5.4.2 Practices implemented

- Iterative & Incremental process [S33] [S40].
- Usage of knowledge [S33].
- Design meetings [S40].
- Customer involved in architectural tasks [S40].
- Refactoring techniques [S40].
- Embed architectural practices into agile [S40].

4.2.5.4.3 Effects observed

- No specific findings

4.2.5.5 Feature-Driven Development (FDD)

4.2.5.5.1 Challenges detected

- No specific findings

4.2.5.5.2 Practices implemented

- Embed architectural practices into agile [S41].
- Preliminary high-level design [S41].

4.2.5.5.3 Effects observed

- No specific findings

4.2.5.6 Test-Driven Development (TDD)

4.2.5.6.1 Challenges detected

- No specific findings

4.2.5.6.2 Practices implemented

- Design upfront [S42].
- Hybrid approach [S42].

4.2.5.6.3 Effects observed

Positive:

- Fewer defects [S42].

Negative

- Increased development time [S42]

4.2.5.7 Agile Unified Process (AUP)

4.2.5.7.1 Challenges detected

- No specific findings

4.2.5.7.2 Practices implemented

- Two-cycle method [S14]

4.2.5.7.3 Effects observed

- No specific findings

5 SEMI-STRUCTURED INTERVIEWS: RESULTS & ANALYSIS

5.1 Interviewees Profiles

This section shows the results obtained from the Semi-Structured Interviews conducted with agile practitioners. For the purposes of this Thesis Work, quality has been more important than quantity, and the interviewees have been five professionals with a broad experience in Agile Software Development and Software Engineering in general.

The anonymity of the interviewees was ensured at the moment of conducting the interviews, so each interviewee will be referred with a code: I1, I2, I3, I4, and I5.

As it has been mentioned in the Characterization of the interviewee section, the profile of the different participants is interesting in order to fit the information provided in the appropriate context, and also was very helpful during the interview in order to formulate additional questions that are familiar to them. The profile of the interviewees can be seen in the following table:

	I1	I2	I3	I4	I5
Experience Software Engineering	25 years	7 years	7 years	19 years	10 years
Experience Agile	13 years	3 years	5 years	8 years	5 years
Agile Meth. Used	Scrum, XP	Scrum (current)	Scrum (current), Kanban	Scrum (current), Kanban, BDD, TDD	Scrum, Kanban, TDD, BDD, XP (Pair programming), Lean
Current Position	Developer, communication with the customer.	Mixed role: tester, coder, Scrum master assistant	Scrum Master	Scrum Master	Agile coach & consultant

Table 6. Interviewee's profiles

As it is noticeable from the profile shown in the previous table, most of the interviewees have a high experience working in Software Engineering, and also, a considerable experience in working with Agile Methodologies (the less-experienced one has been working for 3 years); considering this, their experiences can be considered as most valuable for the purposes of this Thesis Work.

Additionally, their background is also interesting. They all have different backgrounds with similarities, i.e. they all have worked with Scrum; Extreme Programming is not as used as it would be expectable, other Agile Methodologies and Practices like TDD, BDD or Kanban, or Lean appear in the interviewee's background. This diversity of techniques used by the interviewees gives credibility to their testimonies as far as they are originated in different experiences and contexts. This diversity of origins also may positively speak about the generalizability of the information obtained.

5.2 Characterization of the Project

As it has been said in the previous section, the experiences of the interviewees are originated in different contexts. Despite this, it has been considered as useful to characterize the current project where the interviewees are working, since the results of the Systematic Literature Review shown a concern towards the conditions of applicability of Agile Software Development philosophy. This section shows the characteristics of the contexts where Agile is used.

Generally, the participants were working in medium-size projects; this point was quite difficult to clarify, as far as the references given by the participants were quite sparse and vague. None of them expressed clearly that they were working on very huge projects; however, most of the interviewees were prone to transmit the idea of being working in medium-size projects.

The same happened regarding the complexity of the system, it was quite difficult for them to assess the degree of complexity of their systems. Most of the systems built were management information systems; consequently, most of them expressed that they considered their projects as medium-complex projects.

Concerning the criticality of the system, every participant indicated that his project was not critical. The common expression used could be stated, "the project is not critical in the sense that there are no human lives depending on the system, obviously it is not life-critical; but it could be considered somehow critical in the terms of the business of the company".

In this study it was also interesting to find out the experience that the team had in the domain, so the interviewees were asked about previous experiences in the same type of systems. With one exception, all of them worked in projects known by the team, even with legacy systems they had to update from scratch. These participants worked in teams whose customer was their own company. The exception previously mentioned was the only one of the interviewees whose team worked for external customers; so he stated that it is possible that his team had not previous experience in similar projects, as he said: "you never know what is coming up next".

5.3 Characterization of the Team

Knowing each interviewee's background and the general characteristics of each project is very helpful, but the picture of the context where Agile Software Development is applied would not be complete without getting information about the characteristics of the team.

The most typical situation depicted by the participants was that a co-located small team carries out the project; despite this, some of the participants told that their teams collaborate with other teams that were in other cities, even in different countries. Concerning the size of the teams, all of them worked in teams that did not exceed 10 people, so they worked in small size units.

For establishing the collaboration with the distributed teams, they use communication tools: phones, videoconferences, online repositories, shared blackboards, etc. These tools are not needed for working in the co-located team, where face-to-face conversation is the prominent mean of communication; along with whiteboards for visually organize the work.

Most of these teams were born as agile and, in the opinion of the participants, this is much better, because moving from a traditional way of working towards an agile approach is very complicated. One of the interviewees, whose team made this transition, pointed out that "it was a real nightmare", because people didn't adapt well to the change of routines, and doubts arose continuously. In his opinion, it is much better to constitute an agile team from scratch, with predefined working protocols, instead of changing them little by little.

As it was pointed in the previous section, many of the participants work in teams whose customer is the very company; the role of the customer for them is quite interactive, as far as they are in the same place, and quite comprehensive, in the sense that the company do not demand a heavy set of documentation.

In the case of the participant that worked for external customers, he expressed that his company had strict rules for the customers, some of these rules are: no paper documentation at all (no negotiation on this matter), the customer collaborates at the end of each iteration providing feedback to the team, and customers' influence must be limited. This limitation of influence refers to situations where the customer intends to internally manage the work of the team (micromanagement), which in the experience of this participant is very negative because it affects the normal functioning of the team.

5.4 Contributions

All the previous information was interesting in order to set up the context where Agile Software Development is applied, now the pieces of information that are directly related to any of the driving research questions (see Interview Strategy section) are exposed (see the following sub-sections).

5.4.1 Research Question 1 – What do agile practitioners understand by architecture in the context of an agile project?

Regarding the concept of Software Architecture, the answers provided are very varied. Each participant understands Architecture in the most convenient way for his interests.

I3 explained that in his company the concept of architecture was "The general structure of the whole system", and also that the importance of this structure is extremely high for his team.

Agreeing with I3 on the matter that Software Architecture is extremely important, the participant I1, self-defined as an extreme radical proponent of agile software methodologies, showed himself "offended" for the mere suggestion that agile does not propose architectural design, he defined it as "a misconception, agile DOES propose architecture, but not in a traditional way". His concept of Software Architecture was that "There should be no written documentation at all, the architecture of the system is depicted on the sets of executable tests created for the system. So we might be talking about an *executable Architecture*", he also stated that their company refuses any request from the customer oriented to obtain a written description of the system.

This opinion coincides with the one that I2 stated, "We never thought about architecture as a whole big thing, we don't talk about architecture; for us, architecture is a evolutionary thing. It is not documented anywhere, no UML diagrams or anything like that". Agreeing with I1, I2 said that the only architectural description they do is described in the classes of the tests created (as they use TDD) and its hierarchy.

Along these lines, I4 also stated that considering that agile does not have architecture is an important prejudice, this participant justifies the commonality of this prejudice because of the "trauma caused by the transition from one paradigm that separates *thinking* from *acting*, towards a different paradigm that unifies these two concepts".

5.4.2 Research Question 2 – Which challenges regarding Software Architecture are found in agile projects?

Concerning the applicability of Agile Software Development most of the participants agreed that there are important limitations that make that agile cannot be applied to every domain, but some of them specified that this is applicable to any Software Development paradigm. Literally, I1 stated, "Show me any Software Development Methodology that perfectly works for any project, and I will pay 1.000.000 € for it".

I2 also pointed out some of the limitations of their way of working, in this case the limitations were concerning to the flexibility of the system at the time of modifying some element of the system, because in his team "We don't have a flexible structure that enables the scalability of the system. We live thinking in the immediate change, we don't plan any change that we don't know if it is going to happen". This interviewee also admitted, "This way of working relies completely on the experience of the team who built the system".

I4 just stated, "An agile project is like any other project, if you don't have any architectural support, you will have all the problems that any other type of project would have".

I3 pointed, "It is very serious not performing any architectural work in the agile project". He said that problems related to architecture do not appear immediately, in the first sprints; problems appear when requirements change, even when new features are discovered, that the customer even had not thought of them. Generally the main problems are "related to scalability issues".

As an interesting reflection, I5 stated that the key issue with adapting architectural work to agile methodologies is "how can you combine a rigid structure as architecture with changing requirements over time? That's the crux of all the questions".

5.4.3 Research Question 3 – Which practices are followed to overcome these challenges?

I1 strongly emphasized that the key practice to develop architecture was to develop it Iteratively and incrementally, he depicted a process where the important thing was to reach as soon as possible a first release to present to the customer, and with the feedback keep improving the system as they learn about it. An Iteration Zero is adopted for enabling the development of the first prototype; in this iteration a very small model of the system is created, understanding model not as static documentation, but something it is possible to show to the customer (e.g. a mock-up or working skeleton). I1 also stated that, "later in the development, all the team participates in the architectural decision taking, so it becomes a joint practice; but it should be

noted that if someone has a strong experience of the type of system is being built, that person's opinion rules."

I2 explained that in his Scrum team, they have introduced a pre-plan phase that is previous to the joint meeting of all the team; in this pre-plan some responsible team members discuss the best way to implement a User Story, also they plan how the all-team meeting will be conducted, and in that meeting the solution is brought up for discussion. So this *Pre-Plan* stage works as a preliminary design phase for the user story that is going to be implemented. Once the User Story is approved, it is developed.

I3 explained that in their company they had adopted some variations over the conventional Scrum Lifecycle; the main one was the inclusion of a Sprint Zero. This additional sprint involves "creating a quick model of the system to be developed (this process is assisted by experts on architecture). In general, architecture DDD [52] is used, so before start developing, we model our interpretation of the project". For this, they use predefined design patterns that are adapted to the specific needs of each project.

I4 expressed that "the agile view suggest doing things on demand: do architecture when architecture is needed, do documentation when documentation is needed, enhance quality when the quality of the system decreases, etc." what constitutes a flexible reacting approach, or in other words: *architecture "on demand"*. Some of the techniques used for building architecture within the agile project were: Light documentation, face-to-face discussion and joint meetings.

On the other hand, interviewee I5 proposed a middle point between the Big Design Up Front and the You Ain't Going to Need It, this consists in "allowing that the architecture emerges as the product evolves, that's the true agile approach". This participant also explained, "obviously, it is required to refactor time and time again in order to build and adapt the architecture and the software itself, but that is feasible and even recommendable".

5.4.4 Research Question 4 – Which effects did the practices that overcome these challenges provoke on the project?

Referring to the Pre-Plan stage adopted by his team, I2 pointed that "having this slight design phase improves the team meeting in the sense that people are a little influenced on the potential solution, they don't start from scratch so the pressure is less and people is more relaxed and more productive".

I3 pointed that introducing Sprint Zero and modeling at the beginning of the project take away some agility, but it is worth it during the rest of the project, because "we obtain many more benefits".

6 RESULTS DISCUSSION

At the light of the results obtained from both sources of information, Systematic Literature Review and Semi-Structured Interviews, it is possible to obtain interesting findings concerning the issues that compose the aim of this Thesis Work. This section introduces the most important, relevant and empirically supported findings of this Thesis Work: the general concept of Software Architecture in Agile Project that is proposed, the most common challenges concerning Architectural Design and Agile Software Development, the best recommended practices to overcome them, along with the possible effects caused in the project by implementing those practices.

As it has been mentioned in the Related Work section, it was most needed for the purposes of this Thesis (find out how can Software Architecture and Agility be combined) to define ***The Concept of Software Architecture in Agile Projects***. The results obtained from the Systematic Literature Review in that sense show that, when most of the Agile Practitioners talk about Software Architecture, they refer to the ***Structure of the System***. Although it is not explicitly defined like that, it is easily abstracted from the descriptions of Software Architecture they make (components and relations most of the times). They don't make any reference to the need of defining the behavior of the system under certain circumstances, or the perspectives of deployment or expansion of the system; nothing that has to do with foreseeing possible changes.

This concept has been corroborated by the testimonies of the Agile Practitioners interviewed, some of them in a very fancy way. Several of them specified that in their projects they do not create any written documentation or diagrams of the System Architecture; instead, they talked about a new concept of Software Architecture for Agile Projects called ***Executable Architecture***. This kind of architecture consists of the set of tests created for the system. According to the interviewees, the source code of the test contains all the information needed to provide a deep knowledge of the system. Indirectly, the interviewees are confirming the Structural Concept with this proposal of executable architecture, as far as the classes and methods contained in the source code of the test, show nothing but the very structure of the System at its lowest level possible.

This kind of documentation seems not feasible even for medium-size systems in the long time, depending on the source code for reaching an understanding of the application involves two options: a complete dependency on the same people who built the system, or forcing the new team to conduct a hard, slow and discouraging task for understanding how the system is built by reading code.

Concerning the Structural Concept, it could be enough with describing the structure of the system; if the components are well specified, the behavior of the system may be inferable in case of need. And, as far as Agile Methodologies provide the mechanisms to adapt to changing requirements, the architecture may vary the same way, without needing the anticipation of possible changes.

Another important finding of the Systematic Literature Review concerning the Concept of Software Architecture is the existence of some ***Agile Architectural Artifacts***, which are proposals that some Agile Methodologies suggest for addressing the Architectural level of design. There is nothing remarkable concerning these proposals, as far as their appearance in

literature is sparse, however System Metaphors deserve a special attention. They appear more frequently than any other Agile Architectural Artifact; these system metaphors are proposed in the context of Extreme Programming (XP), but their usage is limited, and most of the times not successful. At this respect, the Agile Practitioners interviewed showed no interest in such things, as far as they did not even mentioned them.

On a different matter, the Systematic Literature Review has produced different types of problems that require special measures to be addressed, there are some of them that appear more frequently, and consequently, it is logical to think they have major impact on Agile Projects.

Firstly, there are several situations that have been considered as Agile Challenges related with Architecture in the context of this Thesis Work, but that does not require the implementation of any special measure of practice to be solved. This type of challenge is concerned about the ***Agile Suitability Constraints***. It has to do with the characteristics that a certain project must accomplish for recommending the usage of Agile Methodologies. There are several conditions (see 4.2.2.1 Section) that identify an "optimal candidate", some of them have to do with the team, some others with the customer, and the most important ones (the unavoidable ones) have to do with the project itself. Agile is not recommended for complex or critical projects, and mainly it is not recommended for big projects.

The findings from the Semi-Structured Interviews conducted support the existence of these limitations in the applicability of the Agile Software Development philosophy, so the first recommended practice to conduct is to ***evaluate the characteristics of the project and its context*** for assessing if agile is suitable or not. This issue does not mean that Agile is good or bad; the existent approaches or methodologies for Software Development cannot be judged in absolute terms, instead it is needed to evaluate the specific circumstances in order to choose the most optimal solution possible. Panaceas do not exist in Software Engineering.

There are many other problems in Agile Projects that are related to Software Architecture, some of them are solved with the very inclusion of Software Architecture in the project, and some others require specific measures to be solved. Although many other challenges have been found (they can be consulted in Research Question 2 – Which challenges regarding Software Architecture are found in agile projects? Section), there are several of them that appear more frequently in the Literature and that are also, empirically supported.

One of these is the ***Uselessness of the Agile Proposals for Architecture***, especially in the case of Extreme Programming proposal, System Metaphors. Very often in literature, it is expressed the inefficacy of those artifacts for addressing the Architectural level of design in Agile Projects. At this respect, agile practitioners interviewed did not mention anything concerning System Metaphors or any other Agile Architectural Artifact. In this context, their silence can be interpreted as a confirmation that these proposals are not even considered.

The main challenges concerning the absence of Software Architecture in Agile are related with ***Long-Term problems***. These challenges can be specified as: ***Structural Problems, Scalability Problems, Maintenance Problems,*** and ***Lack of Reusability***. This kind of challenges has been pointed as the most important ones in the Semi-Structured Interviews conducted with Agile Practitioners.

There are no specific measures that individually address each one of these challenges; they all are issues that shall be enhanced through the Architectural Design. Thus, the relevant question is: how can be Architectural Tasks conducted in an Agile Project?

For answering this question, the Systematic Literature Review has provided a wide set of practices (the whole contributions can be consulted in Practices Definition Section). Among these practices there are some of them that are most recommended, and that found empirical support, these are:

- ***Iterative & Incremental Process***
- ***Iteration Zero***
- ***Preliminary High-Level Design***
- ***Prioritization of Inputs***
- ***Devoted roles***

Most of the sources consulted in this Thesis Work, both literature and interviews, agree in indicating that the most appropriate way to perform Architectural Design in Agile is by doing it ***Iteratively and Incrementally***. Agile practitioners consulted indicated that architecture emerges progressively, as the team learns how the System should be. Conducting this task require to divide the available input in several iterations (or sprint) and ***Prioritizing the inputs (whatever they are User Stories, Requirements or Sections)*** seems advisable, this way the most urgent or relevant functionalities demanded by the customer are more time implemented and thus, they can be more tested.

Another recommended practice, easily implementable in an Agile context, is the introduction of an ***Iteration Zero***, this should take place before start iterating for developing the system. Iteration Zero is a very interesting technique as far as it allows to "take some time" for analyzing the inputs provided by the customer and implementing the previous technique (Prioritization of inputs).

Furthermore, Iteration Zero enables the creation of a ***Preliminary high-level design or model***. This high-level design is supposed to work as guidance for the development of the System. Also, as some of the empirical sources obtained indicate, this general model can be useful for the quick ***Development of a Working Skeleton of the application*** for obtaining ***Customer's Feedback*** that provides knowledge about the system and enables the advance in the project.

An interesting example of an adaptation of these techniques was provided by one of the interviewees who pointed an interesting technique that consists in embedding in each iteration a ***Pre-plan Stage***. This stage consists of taking two members of the team, allow them to analyze a subset of the existent input (this practice was explained in the context of a Scrum Project, so the input would be User Stories) and these two members would select an appropriate pack of the pending User Stories to be developed next, and also, with a general model that explains how the solution could be built. Then, the selected User Stories and the High-Level Model created are presented to the team in a ***Design Meeting*** (this practice has been repeatedly found in Literature) where they are discussed and accepted or rejected. This way, Architectural Design becomes a joint practice of all the team members.

Sometimes, it is not enough with the participation of all the team, or maybe the team is not experience enough for creating an appropriate Architecture; in these cases, another broadly used technique is the inclusion of ***people Fully Devoted to architecture***. This practice can be implemented as Extra Workers to conduct Architectural Tasks without delaying the project or Experts in Software Architecture that work as Consultants.

These practices are adaptable to most of the Agile Projects; as it has been pointed at the beginning of the section, considering the particularities of each project and its context (team, experience, customer…) it is essential for successfully implementing them. These techniques will provide none other than Software Architecture with all its consequent benefits: ***Scalability enabled, Flexibility enabled, Enhanced reusability***. On a different matter, as it was pointed out by some of the interviewees, having a general model of the system or a preliminary preparation of the next steps to be conducted and how (pre-plan stage), makes ***the Team feel relaxed, more confident and it is more productive.***

It is true, and it should be remarked, that the conduction of the Architectural Practices for Agile here presented takes its time; therefore ***some of the Agility is taken away from the project***. Nevertheless, some of the Agile Practitioners interviewed agree that ***it is worth it***, this is because the benefits of having Architectural Design in the later phases of a project makes that some tasks become easier, and even ***quicker***. From this point of view, it is possible to state that loosing some time in Architectural Work can ***reduce the development time***, or at least, ***the Agility of the Project is maintained***.

As it has been mentioned before, there are some challenges that are inherent to the very absence of Software Architecture, and thus, enabling the Architectural Design solves them. However, there are others that require special measures to be addressed. One of this, although it is not supported by the interviewees' testimonies, appears several times in the sources included in the Systematic Literature Review, this is the ***Lack of Quality*** of the Software Produced.

According to the sources included in the Systematic Review, this happens because the attention of the team is more focused on delivering functional software on time and within the fixed cost, than in enhancing quality attributes that completely satisfy the customer. A recommended practice to overcome this challenge is to ***enhance Quality as a Measure of Success***; so not just delivering the software on time or within the budget is what determines the success of a project, also, quality attributes as reliability, efficiency, security, etc. will be required.

However, quality is an abstract concept that may not mean the same in every project. The best way to ensure that a project fulfills the quality expectations of the customer is to ***involve the Customer in Architectural Tasks in the project***. This practice has to do with the obtaining of Customers' Feedback pointed previously. Consequently, it is most probable that the system behaves as the customer expects, if it is successively in contact with him. Another practice that may help in this task is to ***evaluate the architecture at the end of every iteration***, so it is more certain that the system will address its expected behavior.

Collaterally, with these practices, another challenge that appears in the Literature is solved. This is the ***lack of Reliability of the Software produced***, which makes reference to the possibilities of failure of the project. Involving the customer and evaluating iteratively the architecture of the system reduce these possibilities.

Besides this, the participation of the customer and the evaluation of the architecture involve other benefits for the project, like ***the communication is enhanced, the customer is more involved in the process and the general quality of the system improves***. However, there is an important risk, pointed out by one of the Agile Practitioners Interviewed, which appears when the Customer is too involved with the team. This possible negative effect is that, generally, ***customers tend to perform micromanagement to the development team***. For avoiding this, it is

needed to strictly define which are the tasks that the customer must perform in his collaboration with the team, and keep vigilant to avoid that the customer goes too far in his demands.

Finally, the last of the big challenges found in Literature (also supported by some of the interviewees) is the ***Reliance on Experience of the Team***. This challenge has to do with the dependency that exists of the team who built the system for reaching a deep understanding of how the system is built. Applying the recommended practices depicted previously for conducting Architectural Design should address the problem. However, there is a rejection of creating documentation of the System. If the team applies architectural techniques for building a "better" system, but does not create the means for transmitting the knowledge on how is the system, the ***Dependency of the Team Experience*** will persist.

At this respect one of the Agile Practitioners interviewed stated a possible solution to overcome this situation, this is to create ***written documentation on demand***. This would consist in defining what information is enough for describing the system and create documentation just for that, avoiding unnecessary work. The immediate effect of this approach is that ***the amount of documentation created decreases***, and so it does the dependency on the team experience.

In this section, a set of most recommended practices has been presented, along with the challenges they overcome and their possible effects. However, it is needed to remark that there is not a unique approach to combine Software Architecture and Agile Software Development. Findings from the Systematic Literature Review show that there are three main approaches for merging these philosophies (see Proposed Approaches Section).

- ***Agile with Architecture Approach***
- ***Agile Architecture Approach***
- ***Mixed Methodology Approach***

The most obvious one is to apply those techniques in the context of an Agile Project for enabling Architectural Design, but also it is possible to apply them in a "Traditional" Project in order to make them more Agile.

As final consideration of the findings of this Thesis Work, it is needed to remark that the key point of conflict of these two philosophies of Software Development is originated in their nature. Combining Software Architecture, which is based in a rigid conception of the System, with Agile Software Development, which is based in a dynamic conception of the system, might seem impossible; but if the positions approach enough it is possible to obtain advantages from both of them.

In this Thesis Work, a set of practices to reach such combination has been depicted and analyzed, but what is most important is the adaptation, the analysis of the context of the project, the characteristics of the team and the customer expectations, and apply what suits these issues better.

Generally, it is possible to state that the most important findings of the SLR have been endorsed by the testimonies gathered in the Semi-Structured Interviews, although not all of them. Furthermore, a schematic summary on which themes among the findings obtained in the Systematic Literature Review can be seen at Appendix H – Comparative: Findings SLR – Interviews.

7 VALIDITY THREATS

In this section, the main validity threats concerning the Systematic Literature Review and the Semi-Structured Interviews are introduced. These are:

7.1 Systematic Literature Review Threats

7.1.1 Study Search & Selection

7.1.1.1 Identification of relevant primary studies

The first threat that was considered in the Systematic Literature Review was to decide the best way to ensure the inclusion of all the relevant material (or at least the highest amount of contributions possible).

Keeping in mind the broadness of the fields of study of this thesis (Agile Software Development and Software Architecture), and the fact that there is an existing literature review [19] with a similar focus; the method chosen for conducting the review was "Snowballing technique"[40].

The process that this approach proposes was conduced time and time again, until only repeated sources were found, so this was considered as the proof that the main body of knowledge available on the topic was covered.

Despite this, the results obtained during the elaboration of the thesis proposal were also reviewed. This additional "manual" search was conducted on the most relevant scientific databases, and helps in ensuring that some relevant source that might be outside of the "Snowball" was also covered.

7.1.1.2 Publication Bias

As Kitchenham points out in [37], "publication bias refers to the problem that positive results are more likely to be published than negative results". This thread is not that important for this Systematic Literature Review. This study does not formulate questions in terms of good or bad, it just intends to find out how to integrate architecture and agility, as it will be shown in Conclusions section, there are very different ways to confront the situation, and approaches that work in some context, might not work in a different one.

Despite this, a detailed protocol was created, providing inclusion/exclusion and quality criterion agreed with the main advisor at BTH and the co-advisor at UPM (as far as there is just one reviewer). A detailed protocol, with a strict selection procedure, helped in overcoming this thread.

7.1.2 Data Extraction & Synthesis

7.1.2.1 Confirmability

In their *Recommended steps for Thematic Synthesis in Software Engineering* [42], Cruzes and Dybå state that Confirmability *"is concerned with how the extracted data*

are coded and sorted and whether or not various researchers with the way those data were coded and sorted".

One reviewer has conducted the whole process of the Systematic Literature Review, so a second reviewer did not directly assess the process for extracting and coding data. For overcoming this threat, the data extraction and codification has been performed the most strictly possible according to the Inclusion Criteria defined in the Systematic Review Protocol approved by the supervisor.

7.1.2.2 Dependability

This threat to validity is related with the stability of the data over time. For different circumstances, the duration of the study has extended during one year, during this lapse of time it is possible that new contributions have been published and that are not included in the Systematic Literature Review.

7.2 Interviews Threats

7.2.1 Construct Validity

According to Wohlin et. al [53], construct validity is concerned to check if the measures are right for the concept that is studied. In this study this has been addressed by making two decisions about the research methodology chosen for this stage of the thesis.

The first one was to decide conducting Interviews where the profile of the interviewee is known, and his "authority" on the topic is proved. And the second one was to conduct Semi-Structured Interviews, creating an interview guide that drives the conversation, but allowing the formulation of additional questions for clarifying the information provided to ensure its relation with the concepts studies.

7.2.2 Interviewees Shortage

One of the biggest problems in the conduction of the interviews was to find Volunteers that participate in the Study. The causes of such shortage are originated in the dates of conduction of the interviews, July 2013, a holiday period for the majority of the workers that could be included in the study.

This problem questions the reliability of the empirical evidences extracted, however due to the common trends shown in the interviewees' testimonies, the information obtained from the Semi-Structured interviews has been considered as useful for supporting and complementing the information extracted from the Systematic Literature Review.

8 CONCLUSIONS

This section includes the summary of the whole findings of this Thesis Work by revisiting the research questions and establishing an implicit comparison between the findings obtained from the Systematic Literature Review and the Semi-Structured Interviews. Additionally, possible future work is also presented.

8.1 Research Questions Revisited

8.1.1 Research Question 1 – What do agile practitioners understand by architecture in the context of an agile project?

There is no special concept about it in the literature. Software Architecture is a very tricky issue to define; it provides such a broad amount of mechanisms to describe a system that it is usually understood differently depending on the needs of the project. But this is a general reflection that can be done for any project, not just agile projects. A deeper discussion on the concept of architecture in general can be found at Concept of Software Architecture section.

Focusing in the *Agile Conception of Software Architecture*, Literature shows that most of the agile practitioners understand Software Architecture as the structural description of the system. Agile practitioners' position towards the documentation of Software Architecture is quite similar to the one found in the Literature, but with small variations.

The majority of the sources included in the SLR coincide in establishing an equivalence between Architecture and Structure; but the most radical agile proponents openly reject the reflection of this structure in a written document, instead they state that all the architecture a system needs is an *executable architecture*. This *Executable Architecture* is composed by the set of tests created for the system, those practitioners who propose this kind of architecture claim that it is enough with taking a look on the source code of the tests to understand how the system is built.

This position seems not feasible, as it is not realistic that every time a system must be changed, it requires reading the existing source of code, especially if the developer that is going to modify the system is not the same person who built it.

On the other hand, more moderated agile proponents propose a different approach, to create *Software Architecture On-Demand*, this is, create what is required when it is required. This is especially aimed to avoid the waste of time in the creation of useless documents.

In the literature it is observed that some Agile Methodologies (specially Extreme Programming) state some proposals that are aimed to substitute Software Architecture. Almost every source in literature, it does not matter if it is an empirical study or an expert opinion contribution, claims that these agile architectural artifacts are useless. Agile proponents did not mention that they had ever used such artifacts, they did not even mention anything about them; this suggests the low importance and usage that this kind of approaches have in Agile World.

8.1.2 Research Question 2 – Which challenges regarding Software Architecture are found in agile projects?

For answering this question the first issue to be addressed is the problem of the suitability of the project. Where is it appropriate to use Agile Software Development? The Systematic Literature Review findings depict a set of conditions that delimit the applicability of Agile Development. These limitations are related to the size of the project and the team, the complexity of the project and its criticality, the mastery of the team in the problem they are dealing with, and the stability of the environment.

Small teams, medium-complex, non-critical projects and medium-size projects are the optimal conditions to choose Agile Software Development; these characteristics are generally supported both by SLR findings and practitioners' testimonies. However, it is needed to remark that some divergences have been found between the two sources of information concerning the question of the team members' location. In SLR findings is pointed out that implementing agile in distributed teams may be an unavoidable obstacle, but agile practitioners provided evidence that this is not, strictly speaking, true; this obstacle is avoidable by using the appropriate communication tool, so Agile Development is feasible in distributed environments.

The empirical contributions included in the SLR put the stress on the size of the projects and also enhance the importance that experience has for a successful usage of the Agile Principles. It is needed to remark that there are two aspects concerning experience that are important for the project, a good one and a bad one.

The experience on the domain, understood as mastery, is desirable (even required) in an agile team for making quicker and easier decisions (argument from authority). On the other hand, the 'bad' aspect of the experience has to do with the reliance that the organization has on the experience of the team members who built the system to reach a deep understanding of the system. This kind of situation may lead to a dependency on the individuals, and this is not desirable by anyone. This duality towards experience in agile projects is frequently pointed out both in SLR and interview findings.

A related issue concerning the experience of the team is how the agile philosophy was adopted; in the opinion of several interviewees, it is better that the team is born as agile. Moving from traditional approaches towards agile might be confusing for the team members and the performance of the team is worse. This challenge was pointed out by several interviewees, however it was not found in the literature included in the Systematic Literature Review.

Among all the challenges that derive from the lack of Software Architecture that are pointed out in the SLR findings, the interviews have stressed those ones related with the long-term life of the project. Specifically the lack of scalability and flexibility has been considered by Agile Practitioners as the most important problem that Agile Software involves.

Many different types of challenges can be abstracted from the literature, the whole set of difficulties can be consulted at SLR Findings section.

8.1.3 Research Question 3 – Which practices are followed to overcome these challenges?

The Systematic Literature Review has produced a large amount of practices that have been used for solving the issue of combining Agile Software Development and Software Architecture.

The first important conclusion obtained is that there are several approaches for doing such combination, these are: adapting agile methodologies to enable the conduction of architectural tasks (see Agile with Architecture Approach section); import agile principles into traditional development phases, so the architectural tasks become more agile (see Agile Architecting Approach section); or create a new methodology by combining the two philosophies from scratch (see Mixed Methodology Approach section).

The abstracted practices can be used in the context of these three approaches, but flexibility should be the key at the time of solving each particular situation. There are no absolute solutions, only different adaptations that suit a particular context. In Practices Definition section all the practices found during the Systematic Literature Review are shown.

Despite this, there are some specific practices that are more relevant and that are supported by the empirical evidence. Participants in the Semi-Structured in this Thesis Work agreed in indicating that the key for creating architecture in agile is doing Iterative and Incrementally.

For that purpose, it has been also remarked as very useful to include an Iteration Zero for setting up a general context, including a preliminary model of the system, that drives the execution of the project.

From these two practices, it is possible to abstract that in agile, the architecture should evolve over time, as the project life goes on. For evolving the system and its architecture is very important to keep in mind customer's feedback, what provides a deeper knowledge of the system and helps in fixing problems that the team might not have detected.

Communication is essential for enabling the decision-making; some useful practices that have been remarked both by the interviewees and the empirical sources included in the SLR are introducing design meetings (joint decisions) and using communication tools, especially in those agile teams which work distributed.

However, these practices cannot be categorized as useful or useless practices, an interesting classification is given attending to their nature. Some practices enhance the agility, others enable architecture, a different group of them is concerned about improving the communication or the quality of the project and others require a costly effort to be implemented.

The key for combining these two paradigms (Agile Software Development and Software Architecture) is to evaluate which are the characteristics of each project, and which approach and techniques suit better the challenges detected in it.

8.1.4 Research Question 4 – Which effects did the practices that overcome these challenges provoke on the project?

The first important conclusion concerning this research question is that it is not possible to strictly speak about effects, as far as evaluating an effect would require to analyze specific metrics after and before taking specific measures. The information found concerning the consequences of the proposed approaches in each case has been mainly random and more based in perceptions and feelings instead of empirical studies.

But despite that, it has been possible to find evidences in literature that support that there are two types of benefits that correspond to the shortages covered by the techniques adopted from the previous section, depending on its aim (enabling architectural design or enhancing agility in the project). These benefits can be deeply explained in Enhanced agile principles and Architectural benefits sections.

However, Negative Effects can be caused as well. Introducing practices that enable architecture are extra tasks that, at minimum, will delay the project. This issue has been also pointed out by one of the participants in the Semi-Structured Interviews; although in some cases "the juice is worth the squeeze", as it is commonly said.

A notion of Software Architecture, even if it consists in a light model, can provide guidance to the team and make them feel more relaxed a confident on the work they are doing.

8.2 Future Work

In the light of the Systematic Literature Review results, it seems clear that much more empirical work on the issue is required. Only 6 out of the 42 included sources were empirical studies, what means a 14% of the total.

It seems advisable to conduct controlled experiments in order to evaluate the effects caused by the proposed techniques and along with the perceptions of the team. The main aim should be to create adaptive practices that could be used depending on the project under certain conditions.

Some of the interesting metrics that could be used for the evaluation of the practices could be: time (this could be measure different processes, or sprints, or the whole development process), lines of code (and other source code metrics like number of classes created), effort required for fixing defects, feelings of the team members (stress, pressure, happiness, etc.), deviations on the budget, customer satisfaction, etc.

Of course, these metrics should be taken both after and before introducing the practices proposed in order to measure their impact, and in those cases where this is not possible (e.g. measuring the effort devoted to add new functionality to the system), the measures should be taken through comparison between projects with similar characteristics.

From the industrial point of view, creating a body of knowledge that assess the selection of a specific methodology depending on the team, project and customer characteristics could mean huge savings in terms of time and cost.

9 LIST OF REFERENCES

[1]. N. Abbas, A. Gravell, and G. Wills, "Historical roots of Agile methods: where did "Agile thinking" come from?," *Agile Processes and eXtreme programming in Software Engineering, Limerick, IE,* 10 - 14 Jun 2008, pp. 94-103, 2008.

[2]. W. Royce, "Managing the Development of Large Software Systems," *Proceedings of the IEEE WESCON*, pp. 1-9, August 1970

[3]. IABG, "Das V-Modell." [Online] Available: http://v-modell.iabg.de/ [Accessed: 5-Aug-2013].

[4]. B. W. Boehm, "A Spiral Model of Software Development and Enhancement," *Computer*, vol. 21, no.5, pp. 61-72, 1988.

[5]. Rational Software Corporation. "Rational Unified Process: Best Practices for Software Development Teams," 1998.

[6]. P. Abrahamsson, M. A. Babar, and P. Kruchten, "Agility and architecture: can they coexist?," *IEEE Software*, vol. 27, no. 2, pp. 16-22, March-April 2010.

[7]. C. Larman, and V. R. Basili, "Iterative and Incremental Development: A Brief History," *IEEE Computer Society, vol.* 36, no. 6 pp. 47-56, 2003.

[8]. T. Gilb, "Evolutionary Delivery versus the "Waterfall model"," *ACM SIGSOFT Software Engineering Notes*, vol. 10, no. 3, pp. 49-61, 1985.

[9]. K. Schwaber, "Scrum development process," *Business Object Design and Implementation*, pp. 117-134, 1997.

[10]. A. Cockburn, "Agile Software Development," *The Agile Software Development Series, A. Cockburn and J. Highsmith (Eds.)*, Boston: Addison Wesley Longman, 2001.

[11]. K. Beck, "Extreme Programming Explained: Embrace Change", *Addison-Wesley*, Reading, MA, 1999.

[12]. J. A. Highsmith, "Adaptive Software Development: A Collaborative Approach to Managing Complex Systems," *New York: Dorset House*, 392pp, 2000. ISBN 0-932633-40-4.

[13]. S. R. Palmer, and J. M. Felsing, "A practical guide to Feature-Driven Development," *Prentice-Hall*, Upper Saddle River, NJ, 2002.

[14]. DSDM Consortium, "DSDM Atern: the Agile Project Delivery Framework." [Online]. Available: http://www.dsdm.org/. [Accessed: 5-Aug-2013].

[15]. Kent Beck et al., "Agile Manifesto" [Online]. Available: http://agilemanifesto.org/. [Accessed: 5-Aug-2013].

[16]. T. Dybå and T. Dingsøyr, "Empirical studies of agile Software Architecture: A systematic review", *Information and Software Technology*, 2008.

[17]. VersionOne, "7th Annual State of Agile Development Survey 2012," VersionOne, 2013.

[18]. D. Falessi, G. Cantone, S. A. Sarcia', G. Calavaro, P. Subiaco, and C. D'Amore, "Peaceful Coexistence: Agile Developer Perspectives on Software Architecture," *IEEE Software*, vol. 27, no. 2, pp. 23-25, March-April 2010.

[19]. H. P. Breivold, D. Sundmark, P. Wallin, and S. Larsson, "What Does Research Say About Agile and Architecture?," *Fifth International Conference on Software Engineering Advances (ICSEA 2010), 22-27 Aug 2010*, Los Alamitos, CA, USA, pp. 32-37, 2010.

[20]. K. Forsberg, and M. Harold, "System engineering for faster, cheaper, better." *1999 Ninth annual international symposium (INCOSE)*, Brighton, England. 1999.

[21]. eWeek "IBM Acquires Rational" [Online]. Available: http://www.eweek.com/c/a/Desktops-and-Notebooks/IBM-Acquires-Rational/. [Accessed: 5-Aug-2013].

[22]. K. Beck, A. Cynthia. "Extreme programming explained: embrace change," *Addison-Wesley Professional*, 2004.

[23]. P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, "Agile Software Development Methods: Review and Analysis," *VTT Electronics*, 2002.

[24]. H. Takeuchi, and I. Nonaka. "The new new product development game," *Harvard business review*, vol. 64, no. 1, pp. 137-146, 1986.

[25]. K. Schwaber, and J. Sutherland, "The Scrum Guide – The Definitive Guide to Scrum: The Rules of the Game," [Online] Available: http://www.scrumguides.org/. [Accessed: 5-Aug-2013].

[26]. Alistair.Cockburn.us "Crystal Methodologies" [Online]. Available: http://alistair.cockburn.us/Crystal+methodologies/. [Accessed: 5-Aug-2013].

[27]. A. Cockburn, "Crystal Clear: A Human-Powered Software Development Methodology for Small Teams," *Addison-Wesley*, 2001.

[28]. The Agile Unified Process (AUP) Home Page [Online]. Available: http://www.ambysoft.com/unifiedprocess/agileUP.html. [Accessed: 5-Aug-2013].

[29]. K. Beck, "Test-driven development: by example," *Addison-Wesley Professional*, 2003.

[30]. Nebulon Pty, Ltd. Feature Driven Development (FDD) [Online]. Available: http://www.nebulon.com/fdd/index.html. [Accessed: 5-Aug-2013].

[31]. D. Garlan, and M. Shaw, "An Introduction to Software Architecture," *Computer Science Department*, paper 724, 1994. Available at: http://repository.cmu.edu/compsci/724

[32]. D. E. Perry, and A. L. Wolf, "Foundations for the study of Software Architecture," *ACM SIGSOFT Software Engineering Notes*, vol. 17, no. 4, pp. 40-52. 1992.

[33]. D. Garlan, and D. E. Perry. "Introduction to the special issue on Software Architecture," *IEEE Trans. Software Eng*, vol. 21, no. 4, pp. 269-274, 1995.

[34]. P. B. Kruchten, "The 4+ 1 view model of architecture." *IEEE Software*, vol. 12, no. 6, pp. 42-50, 1995.

[35]. C. Hofmeister, P. Kruchten, R. L. Nord, H. Obbink, A. Ran, and P. America, "A general model of Software Architecture design derived from five industrial approaches," *Journal of Systems and Software*, vol. 80, no.1, pp. 106-126, 2007.

[36]. R. E. Jeffries, A. Anderson, and C. Hendrickson, foreword by K. Beck "Extreme Programming Installed," *Addison-Wesley Professional*, 2001.

[37]. B. Kitchenham, and Keele, Staffs, "Guidelines for performing Systematic Literature Reviews in Software Engineering". *EBSE Technical Report* EBSE-2007-01, 2007.

[38]. S. E. Hove, and B. Anda, "Experiences from Conducting Semi-Structured Interviews in Empirical Software Engineering Research," *IEEE International Symposium on Software Metrics*, vol. 0, p. 23, 2005.

[39]. B. Kitchenham, "Procedures for Performing Systematic Reviews," *Keele University Technical Report*, TR/SE-0401, 2004.

[40]. T. Greenhalgh, and R. Peacock, "Effectiveness and efficiency of search methods in systematic reviews of complex evidence: audit of primary sources," *BMJ: British Medical Journal*, vol. 331, pp. 1064-1065, 2005.

[41]. W. Afzal, R. Torkar, and R. Feldt, "A systematic review of search-based testing for non-functional system properties," *Information and Software Technology*, 2009.

[42]. D. S. Cruzes and T. Dybå, "Recommended Steps for Thematic Synthesis in Software Engineering," *International Symposium on Empirical Software Engineering and Measurement (ESEM) 22-23 September 2011*, pp. 275-284, 2011.

[43]. D. Cruzes, M. Mendonça, V. Basili, F. Shull, and M. Jino, "Extracting information from Experimental Software Engineering Papers," *XXVI International Conference of the Chilean Society of Computer Science 2007 (SCCC'07),* pp. 105-114, 2007.

[44]. D. S. Cruzes and T. Dybå, "Research synthesis in software engineering: a tertiary study", *IST: Information and Software Technology*, vol. 53, no. 5, pp. 440-455, 2011.

[45]. J. Thomas, and A. Harden, "Methods for the thematic synthesis of qualitative research in systematic reviews," *BMC medical research methodology*, vol. 8, no. 1, pp. 45, 2008.

[46]. S. Kvale, "Interviews: an introduction to qualitative research interviewing," *Sage Publications*, Thousand Oaks, CA, 1996.

[47]. H. Rubin, and I. Rubin, "Qualitative interviewing: the art of hearing data," *Sage Publications*, Thousand Oaks, CA, 1995.

[48]. C. Robson, "Real world research," *Blackwell*, 2nd Edition, 2002.

[49]. C. Cook, F. Heath, and R. L. Thompson, "A Meta-Analysis of Response Rates in Web- or Internet-Based Surveys," *Educational and Psychological Measurement*, vol. 60, no. 6, pp. 821-836, 2000.

[50]. Sun Tzu, "The Art of War". Fifth century B.C. approximately.

[51]. C. B. Seaman, "Qualitative Methods in Empirical Studies of Software Engineering," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 557-572, 1999.

[52]. DDDCommunity. [Online]. Available at: http://www.domaindrivendesign.org/. [Accessed: 7-Aug-2013].

[53]. C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslen, "Experimentation in Software Engineering: an Introduction," *Kluwer Academic Publishers*, 2000.

[54]. J. Pérez, J. Díaz, J. Garbajosa, P. Alarcón, "Flexible Working Architectures: Agile Architecting Using PPCs," *M. Ali Babar and I. Gorton (Eds.): ECSA 2010*, LNCS 6285, pp. 102-117, 2010.

[55]. J. Madison, "Agile Architecture Interactions," *IEEE Software*, vol. 27, no. 2, pp.41-48, March-April 2010.

[56]. M. A. Babar, "An exploratory study of architectural practices and challenges in using Agile Software Development approaches," *2009 Joint Working IEEEIFIP Conference on Software Architecture European Conference on Software Architecture*, pp. 81–90, 2009.

[57]. D. Mancl, S. Fraser, B. Opdyke, E. Hadar, I. Hadar, and G. Miller, "Architecture in an agile world," *Proceedings of the 24th ACM SIGPLAN conference companion on Object Oriented Programming System Languages and Applications.* ACM, pp. 719-720, 2009.

[58]. P. Kruchten, "Software Architecture and Agile Software Development: A Clash of Two Cultures?," 32nd *International Conference on Software Engineering, 2010 ACM/IEEE.* Vol. 2, pp. 497-498, 2010.

[59]. M. A. Babar, "Agility and Architecture: Why and how they can coexist?,".

10 APPENDIX A – SOURCES INCLUDED IN THE SYSTEMATIC LITERATURE REVIEW

[S1]. B. Boehm, “Get ready for agile methods, with care,” IEEE Computer, vol. 35, no.1, pp. 64-69, Jan 2002.

[S2]. P. Clements, J. Ivers, M. Little, and J. Stafford, "Documenting Software Architectures in an Agile World," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, Technical Note CMU/SEI-2003-TN-023, 2003.

[S3]. E. Hadar, and G. M. Silberman, “Agile architecture methodology: long term strategy interleaved with short term tactics,” *Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications* (OOPSLA Companion '08). ACM, New York, NY, USA, pp. 641-652, 2008.

[S4]. B. Boehm, and R. Turner, "Observations on balancing discipline and agility," *Proceedings of the Agile Development Conference, 2003. ADC 2003,* pp.32-39, 25-28 June 2003.

[S5]. K. Mohan, B. Ramesh, and V. Sugumaran, "Integrating Software Product Line Engineering and Agile Development," *IEEE Software*, vol.27, no.3, pp.48-55, May-June 2010.

[S6]. R. Carbon, M. Lindvall, D. Muthig, and P. Costa, “ Integrating Product Line Engineering and Agile Methods: Flexible Design Up-Front vs. Incremental Design”, *First International Workshop on Agile Product Line Engineering*, 2006.

[S7]. P.E. McMahon, “Extending Agile Methods: A Distributed Project and Organizational Improvement Perspective,” *CrossTalk, The Journal of Defense Software Engineering*, vol. 18, issue 5, pp. 16-19, 2005.

[S8]. V. Rahimian, and R. Ramsin, “Designing an agile methodology for mobile Software Development: A hybrid method engineering approach,” *2008 Second International Conference on Research Challenges in Information Science*, pp. 337–342, 2008.

[S9]. M. A. Babar, “An exploratory study of architectural practices and challenges in using Agile Software Development approaches,” *2009 Joint Working IEEEIFIP Conference on Software Architecture European Conference on Software Architecture*, pp. 81–90, 2009.

[S10]. T.J. Lehman, and A. Sharma, “Software Development as a Service: Agile Experiences,” *2011 Annual SRII Global Conference*, pp. 749–758, 2011.

[S11]. M. Waterman, J. Noble, and G. Allan, “How Much Architecture? Reducing the Up-Front Effort,” *2012 Agile India*, pp. 56–59, 2012

[S12]. T. Silva, M. Selbach, F. Maurer, and T. Hellmann, “User Experience Design and Agile Development: From Theory to Practice”, *Journal of software Engineering and Applications*, 5, pp. 745-751, 2012.

[S13]. K.M. Zaki, and R. Moawad, “A hybrid disciplined Agile software process model,” *Informatics and Systems INFOS 2010 The 7th International Conference on*, pp. 1–8, 2010.

[S14]. A. Abouzekry, and R. Hassan, “Software Product Line Agility,” *The Sixth International Conference on Software Engineering Advances (ICSEA 2011)*, pp. 1-7, 2011.

[S15]. B. Boehm, and R. Turner, “Using Risk to Balance Agile and Plan-Driven Methods,” *IEEE Computer*, vol. 36, no.6, pp.57-66, June 2003.

[S16]. C.R. Prause, and Z. Durdik, "Architectural design and documentation: Waste in agile development?," *International Conference on Software and System Process (ICSSP)*, pp.130-134, 2-3 June 2012.

[S17]. J.D. Kiper, and M. S. Feather, “From requirements through risks to Software Architecture for plan-based and agile processes,” *Proceedings of the Workshop on Requirements Engineering for Adaptive Architectures, Monterey Bay, CA*.

[S18]. R.L. Nord, and J. E. Tomayko, "Software Architecture-centric methods and agile development," *IEEE Software*, vol.23, no.2, pp.47-53, March-April 2006.

[S19]. A. A. Sharifloo, A. S. Saffarian, AND F. Shams, "Embedding Architectural Practices into Extreme Programming," *19th Australian Conference on Software Engineering, 2008. ASWEC 2008*, pp.310-319, 26-28 March 2008.

[S20]. R. Juric, "Extreme programming and its development practices," *Proceedings of the 22nd International Conference on Information Technology Interfaces, 2000. ITI 2000*, pp.97-104, 13-16 June 2000.

[S21]. H. Obendorf, and M. Finck, “Scenario-based usability engineering techniques in agile development processes,” *CHI '08 Extended Abstracts on Human Factors in Computing Systems* (CHI EA '08). ACM, New York, NY, USA, pp. 2159-2166, 2008.

[S22]. H. Obendorf, A. Schmolitzky, and M. Fink, “XPnUE – Defining and Teaching a Fusion of eXtreme Programming & Usability Engineering”, *HCI educators workshop*, 2006.

[S23]. J. Herbsleb, D. Root, and J. E. Tomayko, “The eXtreme programming (XP) metaphor and Software Architecture,” *Computer Science Department, Carnegie Mellon University*. Paper 2173, 2003.

[S24]. R. L. Nord, J. E. Tomayko, and R. Wojcik, “Integrating Software-Architecture-Centric Methods into Extreme Programming (XP),” *Software Engineering Institute, Carnegie Mellon University*, (CMU/SEI-2004-TN-036). 2004.

[S25]. P. Guha, K. Shah, S. S. P. Shukla, and S. Singh, “Incorporating Agile with MDA Case Study: Online Polling System,” *International Journal of Software Engineering & Applications (IJSEA)*, vol. 2., no.4, pp. 83-95, 2011.

[S26]. R. N. Jensen, T. Møller, and P. Sönder, “Architecture and Design in eXtreme Programming: The Architectural Game, introducing a new practice**,”** *Proceedings of the 7th international conference on Extreme Programming and Agile Processes in Software Engineering* (XP'06), Springer-Verlag, Berlin, Heidelberg, pp. 133-142, 2006.

[S27]. J. Manzo, “Odyssey and other code success stories,” *Crosstalk,* pp. 19-21, 30. October 2002.

[S28]. L. Cao, K. Mohan, P. Xu, and B. Ramesh, “How Extreme does Extreme Programming Have to be? Adapting XP Practices to Large-scale Projects,” *Proceedings of the 37th Hawaii International Conference on System Sciences,* pp. 1-9, 5-8 January 2004.

[S29]. S. Jeon, M. Han, E. Lee, and K. Lee, “Quality Attribute driven Agile Development,” *Ninth International Conference on Software Engineering Research, Management and Applications 2011,* pp.203-210, 10-12 Aug. 2011.

[S30]. J. Diaz, J. Pérez, A. Yagüe, and J. Garbajosa, “Tailoring the Scrum Development Process to Address Agile Product Line Engineering,” *Proceedings of Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2011),* 2011.

[S31]. Z. Durdik, “Towards a Process for Architectural Modelling in Agile Software Development,” *Proceedings of the joint ACM SIGSOFT conference -- QoSA and ACM SIGSOFT symposium -- ISARCS on Quality of Software Architectures -- QoSA and architecting critical systems -- ISARCS* (QoSA-ISARCS '11). ACM, New York, NY, USA, pp. 183-192, 2011.

[S32]. J. Cho, “A hybrid Software Development method for large-scale projects: Rational Unified Process with Scrum,” *Issues in Information Systems (Journal IACIS),* vol. 10, no. 2, pp. 340-348, 2009.

[S33]. P. Abrahamsson, A. Hanhineva, H. Hulkko, T. Ihme, J. Jäälinoja, M. Korkala, J. Koskela, P. Kyllönen, and O. Salo, “Mobile-D: An Agile Approach for Mobile Application Development,” *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications* (OOPSLA '04). ACM, New York, NY, USA, pp. 174-175, 2004.

[S34]. L. Cordeiro, C. Mar, E. Valentin, F. Cruz, D. Patrick, R. Barreto, and V. Lucena, “An Agile Development Methodology Applied to Embedded Control Software under Stringent Hardware Constraints.” *ACM SIGSOFT Softw. Eng. Notes*, 33, (1), pp. 1-10. 2008.

[S35]. M. R. J. Qureshi, “Empirical Evaluation of the Proposed eXSCRUM Model: Results of a Case Study,” *IJCSI International Journal of Computer Science Issues,* vol. 8, issue 3, no. 2, pp. 150-156, May 2011.

[S36]. J. Madison, "Agile Architecture Interactions," *IEEE Software*, vol. 27, no. 2, pp.41-48, March-April 2010.

[S37]. F. Chitforoush, M. Yazdandoost, and R. Ramsin, "Methodology Support for the Model Driven Architecture," *Software Engineering Conference, 2007. APSEC 2007. 14th Asia-Pacific*, pp.454-461, 4-7 Dec. 2007.

[S38]. M. Gerami, R. Ramsin, "A framework for extending agile methodologies with aspect-oriented features," *Fifth International Conference on Research Challenges in Information Science (RCIS), 2011*, pp.1-6, 19-21 May 2011.

[S39]. J. Pérez, J. Díaz, J. Garbajosa, P. Alarcón, "Flexible Working Architectures: Agile Architecting Using PPCs," *M. Ali Babar and I. Gorton (Eds.): ECSA 2010*, LNCS 6285, pp. 102-117, 2010.

[S40]. S. Farhan, H. Tauseef, and M.A. Fahiem, "Adding Agility to Architecture Tradeoff Analysis Method for Mapping on Crystal," *WRI World Congress on Software Engineering, 2009. WCSE '09*, vol. 4, pp.121-125, 19-21 May 2009.

[S41]. F. Kanwal, K. Junaid, and M. A. Fahiem, "A Hybrid Software Architecture Evaluation Method for FDD - An Agile Process Model," *2010 International Conference on Computational Intelligence and Software Engineering (CiSE)*, pp.1-5, 10-12 Dec. 2010.

[S42]. N. Nagappan, E. M. Maximilien, T. Bhat, and L. Williams, "Realizing quality improvement through test driven development: results and experiences of four industrial teams," *Empirical Software Engineering*. Vol.13, pp. 289-302, 2008.

11 APPENDIX B – INTERVIEW GUIDE

11.1 Section A. Characterization of the Interviewee

1. How many years have you been working in Software Engineering?
2. How many years have you been working in agile projects?
3. Which agile methodologies have you work with…
    a. In the past?
    b. Nowadays?
4. Which is your role in the team?

11.2 Section B. Characterization of the Project

5. Project characteristics
    a. How big is the project?
    b. How critical is the project?
    c. How complex is the project?
6. Is the company/team experienced in similar projects in the same area?
7. Which is the agile methodology/approach used?
8. How was the adoption of agile?
9. Team
    a. How experienced is the team in...
        i. Agile methodologies?
        ii. Software Architecture?
    b. Which is the organization of the team? (Roles, responsibilities…)
    c. How is enabled the communication among the team?
10. Customer
    a. Which are customer's demands concerning Software Architecture documentation?
    b. Which is the role of the customer in the agile team?
    c. How is enabled the communication with the customer?

11.3 Section C. Research Questions

11. Research Question 1
    a. How important is Software Architecture in the agile project?
        i. Usefulness
        ii. Perception of the team
    b. For the team, in the agile context; what is Software Architecture?
12. Research Question 2
    a. Which are the main problems designing Software Architecture in your team?
    b. Which problems derived from Software Architecture in the project?
    c. Why do you consider that you need Software Architecture in the project?
13. Research Question 3
    a. How is the agile methodology applied?[1]
    b. How do you make the architectural design more agile?[2]
14. Research Question 4
    a. Which are the benefits or drawbacks of the approach you are following?

[1] In case a specific agile methodology is used.

[2] In case agile architecting approach is followed

12 APPENDIX C – OBJECTIVES – RESEARCH QUESTIONS – RESEARCH METHODOLOGIES MAPPING

Thesis Aim	Thesis Objectives		Research Questions		Research Methodologies	
"To Study and characterize how agile methodologies and architectural design can be combined in industrial projects"	**Objective 1.** Collect data	**Sub-objective 1.1.** Architectural Concept & Challenges in Agile	**RQ 1.** How can Software Architecture be designed in agile projects in industry?	**RQ 1.** What do agile practitioners understand by architecture in the context of an agile project	Systematic Literature Review	Semi-Structured Interviews
				RQ 2. Which challenges regarding Software Architecture are found in agile projects?		
		Sub-objective 1.2. Approaches & Effects		**RQ 3** Which practices/approaches are followed to overcome these challenges?		
				RQ 4 Which effects did these practices/approaches provoked on the project?		

13 APPENDIX D – DATA CODING TABLES

This annex contains the assignation of labels to segments of text in the different sources that have been selected for the study. The different tables also contain information about the type of study that each source is, and a descriptive sentence that summarizes the approach followed in the source.

The *Recommended Steps for Thematic Synthesis in Software Engineering* (Cruzes and Dybå) [42] state that it is unlikely to be practical to perform a line-by-line coding, although in the original definition of the Thematic Synthesis method by Thomas and Harden [45] they recommend so. In this case, agreeing with Cruzes and Dybå, line-by-line coding has been considered as an unapproachable task due to the large amount of information found.

Instead, it has been assigned labels to portions of text, identified by the page, column and paragraph that contain it. It should be considered that, in the same section of a text, an author might discuss about issues that are related to different Research Questions of the Systematic Literature Review; in this cases it is possible to find two different labels assigned to the same portion of text.

Cruzes and Dybå in [42] state that there are three approaches that can be followed for coding data:

- *Deductive or A Priori Approach:* It starts with creating a list of codes and assign them to different portions of text.
- *Inductive or Grounded Theory Approach:* Here, the codes arise by reading the different segments of text and, constantly comparing the segments of text it is decided whether they refer to the same concept or not.
- *Integrated Approach:* This approach employs both *inductive* and *deductive* approaches. First a preliminary set of codes is created, and then the codes are refined and new codes arise progressively.

In this study, the *Integrated Approach* has been the chosen one. Starting from a list created based on the deep read of the papers, and then refining it and adding new codes at reviewing the each segment of text.

However, this approach was considered as unfeasible, due to the amount of data to be processed. Consequently it was modified in the following way: the codes created were descriptive sentences that summarize the concept of each segment of text, and the next level of abstraction (and grouping) was performed at the time of assigning themes to each code.

One of the objectives of this study is to maintain a full traceability of the steps taken, so each label is directly related to the Research Question that answers.

Finally, this section also contains the thematic translation of the codes; the approach for this translation has been to relate methods that follow the same principle to deal with the issue; for example: if an author defines architecture as a set of modules and its components and another author defines it as the specification of classes and objects, it is possible to abstract that they both share a concept of structural architecture.

The form use for coding data is the following:

Paper	Number of reference (SXX format) – Title of the source				
Type	Expert opinion, empirical study, mixed contribution		**Approach**	Descriptive label of the approach followed	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
Page number	Column number	Paragraph number	Descriptive label	Corresponding theme	Related RQ

Paper	S01 – Get Ready for Agile Methods, with Care				
Type	Expert Opinion		**Approach**	Balance agility and discipline attending to risk evaluation	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
66	2nd	2nd	Agile throws away architectural support and can create problems with customers	Maintenance problems	2
				Customer communication problems	2
67	2nd	1st	Risk management is used for balancing agility and discipline	Risk driven methodology	3

Paper	S02 – Documenting Software Architectures in an Agile World				
Type	Expert Opinion		**Approach**	Agile Architecting	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
9	1st	1st	Simplicity is the key principle	Simple design (KISS principle)	3
9	1st	5th	Create initial skeleton	(Method explanation)	3
9	1st	6th	Choose views to be developed in the V&B approach	Structural architecture	1
10	1st	1st-2nd	Identify stakeholders	Customer involved in architectural tasks	3
10	1st	3rd-5th	Prioritize sections to be completed	Prioritization of sections	3

Paper	S03 – Agile Architecture Methodology: Long Term Strategy Interleaved with Short Term Tactics				
Type	Mixed Contribution		**Approach**	Agile Architecting	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
643	2nd	1st	Architecture structured in two levels: Modules and Components	Structural architecture	1
643-646 (1st col)					
645	1st	7th	Reference architecture definition	Structural architecture	1
648	1st Col.		Implementation architecture definition	Structural architecture	1

649	1st -2nd Col.		Divide ‘architecting’ process into two cycles for developing minor and major releases	Two-cycle method	3
650	1st	9th	Effect: must – architectures as simple as possible	Simpler design	4
650	2nd	1st	Effect: danger – focus of attention	Lack of conceptual integrity	2
650	2nd	3rd	Effect: variable time to reach maturity	Team experience dependency	4

Paper	S04 – Observations on Balancing Discipline and Agility				
Type	Expert Opinion		**Approach**	Hybrid approach agile and disciplined methods	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
1	2nd	3rd	Lack of scalability and failure in handling complexity and somehow conformity in agile projects	Scalability problems	2
				Agile is not suitable for complex projects	2
3	1st	2nd	The degree of agility/discipline in the methodology is obtained by rating a project in five axes (Personnel, dynamism, culture, size and criticality)	Project characterization	3
5	2nd	2nd	Assessment of the project		3
6	1st	1st			
6	1st	2nd	Create a strategy for development attending the characteristics of the project assessed	Develop a hybrid methodology based on project characterization	3
6	1st	6th			
6	2nd	5th			
7	1st	2nd	Complete the agile/disciplined strategy design	Monitor and readjust methodology	3

Paper	S05 – Integrating Software Product Line Engineering and Agile Development				
Type	Expert Opinion		**Approach**	Apply agile practices integrated in PLE	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
52	Table		Applies practices to integrate agile and SPLE according to Complex Adaptive Systems (CAS) principles	Customer involved in architectural tasks	3
				Simple design (KISS principle)	3

			Refactoring	3
			Incremental process	3
			Experience-based architecture	3
			Risk driven methodology	3

Paper	S06 – Integrating Product Line Engineering and Agile Methods: Flexible Design Up-Front vs. Incremental Design				
Type	Mixed Contribution		**Approach**	Agile architecting in PLE	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
3	2nd	2nd	Challenge: As it does not anticipate changes, Agile fails at reusability	Lack of reusability	2
3	2nd	4th	Pulse-I methodology for PLE: produces Family Engineering aspects and Application Engineering aspects	(Method explanation)	3
4	1st Col.				
4	2nd Col.		Apply Pulse-I iteratively and apply agile specific practices are used in that context	Iterative process	3
5	1st	1st		Two-cycle method	3
5	1st	3rd	Effect: better reuse rate and reduced time to market	Enhanced reusability	4
				Reduced time to market	4
5	1st	4th	Effect: the up-front design enables flexibility for the future changes	Flexibility enabled	4

Paper	S07 – Extending Agile Methods: A Distributed Project and Organizational Improvement Perspective				
Type	Expert Opinion		**Approach**	Embed architectural practices into agile	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
5	1st	2nd a)	Set up agile architecture team	Devoted roles	3
5	1st	2nd b)	Create thin, complete high-level architecture	Preliminary high-level design	3
5	1st	3rd-4th	Solve architectural issues separately and iteratively	Iterative and incremental process	3
5	1st	5th	XP metaphor is too week for complex systems	XP proposals for architecture do not address the problem	2
6	1st	2nd	Include “super-lead” role to	Architect as a	3

			oversee agile teams	communication enabler	
Paper	S08 – Designing an Agile Methodology for Mobile Software Development: A Hybrid Method Engineering Approach				
Type	Expert Opinion		**Approach**	Hybrid approach	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
4	2nd	2nd	Prioritization of requirements	Prioritization of requirements	3
4	2nd	3rd	Iterative and incremental design	Iterative and incremental process	3
5	Image		Architectural design phase	Preliminary high-level design	3
			Detailed design inside iterations	Iterative process	3

Paper	S09 – An Exploratory Study of Architectural Practices and Challenges in Using Agile Software Development Approaches				
Type	Empirical Study		**Approach**	No specific agile method related	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
85	1st	2nd	Architect's role in agile: software architects (interact with customers), solution architects (manager role) and implementation architects (technical mentor)	Devoted roles	3
85	1st	3rd	Customer involvement into architectural processes as a solution for architectural analysis tasks	Customer involved in architectural tasks	3
85	2nd	1st a)	Architectural synthesis: two step-process: software architects perform high level design with customers and solution and implementation architects take design decisions	Customer involved in architectural tasks	3
85	2nd	1st b)	Amount of deliverables reduced and available for all the team	Less documentation	4
85	2nd	2nd	Architecture is evaluated at very high level	Architectural evaluation at high-level	3
85	2nd	3rd	Refactoring as a mean to achieve quality	Refactoring	3
86	1st	2nd	Functionality, budget and	Lack of quality	2

			time capture the main interest. Quality is a maintenance matter		
86	2nd	1st	Software Architectural Overall Plan (SAOP) guide architectural decisions	Preliminary high-level design	3
86	2nd	2nd	Reduction of architectural documentation perceived as a benefit	Less documentation	4
86	Table		Agile Artifacts: Architectural Infrastructure Plan (AIP) and User Stories	Agile architectural artifacts	1
87	1st	1st	Team communication enabled through Wiki	Communication tools usage	3
87	1st	3rd	Challenge: priorization of user stories	Reliance on Experience	2
87	2nd	1st	Proposal of solution: Feature Analysis Workshop (FAW) for prioritization of user stories	Embed architectural practices into agile	3
87	2nd	2nd	Challenge: time and budget limitations constraint the consideration of multiple design options	No alternative design solutions considered	2
87	2nd	3rd	Proposal of solution: Including Zero Iteration	Iteration Zero	3
87	2nd	4th	Challenge: unsuitability of agile in unknown domains	Agile is not suitable for unknown domains	2
88	1st	1st	Proposal of solution: Apply hybrid agile and Plan-Driven approach as development methodology	Hybrid approach	3
88	1st	2nd	Challenge: Lack of focus on quality attributes	Lack of quality	2
88	1st	3rd	Proposal of solution: Enhance quality as a measure of success	Enhance quality	3
88	2nd	2nd a)	Challenge: lack of skill of the team	Reliance on Experience	2
88	2nd	2nd b)	Challenge: difficulties in finding unplanned architectural documentation	Team communication problems	2
88	Table – Adv.		Developers take part of the	Enhanced communication	4

		design		
		Less documentation	Less documentation	4
		Less time in architectural and design activities	Simpler design	4
		Easy sharing decisions	Enhanced communication	4
88	Table – Dis.	Design is carried out based on the experience of individuals	Reliance on Experience	2
		Design alternatives are not considered	No alternative design solutions considered	
		No focus in quality aspects	Lack of quality	

Paper	S10 – Software Development as a Service: Agile Experiences				
Type	Expert Opinion		**Approach**	Hybrid approach	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
750	2nd	1st	Agile doesn't work for large teams, stable requirements and where high assurance is needed.	Non-reliable software is produced	2
				Agile is not suitable for large projects	2
				Agile is not suitable for working with large teams	2
					2
756-757	1st-1st	6th-3rd	Two parallel tracks: release and prototype (agile)	Devoted roles	3
756	2nd	1st	Foundation release: high level decisions and up front design	Iteration Zero	3

Paper	S11 – How much architecture? Reducing the up-front effort				
Type	Empirical Study		**Approach**	No specific agile method - Embed architectural practices into agile	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
56	2nd	2nd	Lack of architectural design causes failure of the projects	Non-reliable software is produced	2
57	2nd	6th	Practice: Using predefined architecture	Experience-based architecture	3
58	1st	7th	Practice: Intuitive architecture		3
58	1st	9th	Take decisions based on		3

			architectural experience		
58	2nd	5th	Practice: Being familiar with the architecture		3

Paper	S12 – User Experience Design and Agile Development: From Theory to Practice				
Type	Mixed Contribution		**Approach**	Agile applied to User Experience Design (practices abstracted). No method – but similarities with Scrum	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
745	1st	1st	Practice: Sprint 0 adoption	Iteration Zero	3
745	1st	2nd	Sprint 0 consists in creating a preliminary design to be refined later	Preliminary high-level design	3
745	2nd	1st	Design must be one iteration ahead of implementation	(Method explanation)	3
746	1st	4th	Daily meetings are useful to analyze and report problems or modifications in the design	Design meetings	3
746	1st	5th	Evaluations must be performed at the end of each sprint	Architectural evaluation at the end of each sprint	3

Paper	S13 – A Hybrid Disciplined Agile Software Process Model				
Type	Mixed Contribution		**Approach**	Hybrid approach (combine agile and traditional). The 'agile-side' of the approach is mainly based in XP.	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
1	2nd	3rd a)	Inception phase similar to Iteration/Sprint Zero to set up a context for the project including architectural activities	Iteration Zero	3
2	1st-2nd			Prioritization of user stories	3
1	2nd	3rd b)	Planning phase develops the high-level plan	Preliminary high-level design	3
3	1st				
2	1st	1st	Iterative assessment phase customizes the actions to be performed after a evaluation of the agile concerns	(Method explanation)	3
3	2nd				

2	1st	2nd	Building phase is the phase where the building of each feature is carried out		3
6	1st-2nd	5th-3rd			
2	1st	3rd	In the Production phase the product faces its users		3
6	2nd	4th			
2	1st	4th	The closure phase is the death phase that finalizes every activity as in XP		3
6	2nd	5th			
5	1st	5th	Challenge: Lack of structure generates unstructured, non reusable and not reliable products	Structural problems	2
				Lack of reusability	2
				Non-reliable software is produced	2
			Solution: Insert a initial design phase to determine the level of effort and details needed and divide the application using SOA	Preliminary high-level design	3
5	2nd	5th	Challenge: Lack of up-front planning could result in company loses and lack of risk prevention	Non-reliable software is produced	2
			Solution: perform impact analysis on the next iterations and use statistical reporting tools	Perform impact analysis every iteration	3
6	1st	2nd	Challenge: Limited support for developing large projects	Agile is not suitable for large projects	2
			Solution: divide teams into smaller teams and make testers perform stress and integration tests	Divide teams	3
				Integration effort	3

Paper	S14 – Software Product Line Agility				
Type	Mixed Contribution		**Approach**	Hybrid approach (EUP+AUP) in the context of SPL	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
3	1st – 2nd		The EPLSP proposes to split the production in two tasks: Core Assets development and Product Development	Two-cycle method	3
3	2nd	3rd	EUP (traditional approach) is		3

			used for CA Development		
4	1st	1st	Product Development needs faster responses and early deliveries, so AUP is recommended here.		3

Paper	S15 – Using Risk to Balance Agile and Plan-Driven Methods				
Type	Mixed Contribution		**Approach**	Hybrid approach, combining agile and Plan-Driven based in risk evaluation	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
57	2nd	2nd-3rd	Perform risk evaluation in areas associated with agile and Plan-Driven methods	Risk driven methodology	3
57	2nd	5th	Evaluate the risk analysis to determine the project characteristics	Risk assessment	3
58	1st	3rd	If possible, develop an agile architecture where suitable (according to the previous risk evaluation) and apply Plan-Driven methods in the remainder of the work.	Develop a hybrid methodology addressing risks	3
58	2nd	1st	Develop an overall strategy for the project addressing the identified risks		3
59	1st	3rd-4th	Monitor, reevaluate and adjust the levels of agile and Plan-Driven established initially.	Monitor and readjust methodology	3

Paper	S16 – Architectural Design and Documentation: Waste in Agile Development?				
Type	Empirical Study		**Approach**	No specific agile method – embed architectural practices into agile	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
130	1st	2nd a)	Architecture perceived as a waste, not contributive work	Perceived irrelevance of the architectural work	2
130	1st	2nd b)	Architecture contributes to improve quality, reusability and maintenance.	Lack of quality	2
				Lack of reusability	2
				Maintenance problems	2
131	1st	7th	Unsuitability of agile for	Agile is not suitable for large	2

			general projects	projects	
			Lack of documentation	Lack of documentation	2
131	2nd	3rd	Relevance of experience of the team	Reliance on Experience	2
				Experience-based architecture	3
132	1st	4th	Perceived irrelevance of architectural work	Perceived irrelevance of the architectural work	2
			Lack of quality goals	Lack of quality	2
			Lack of professional management	Lack of professional management	2
133	Table		Reviewing practice	Reviewing	3
			Create separated quality control roles	Enhance quality	3
				Devoted roles	3
133	2nd	3rd	Iteration Zero and Continuous iterative design approaches	Iteration Zero	3
133	2nd	4th	Continuous iterative design	Iterative process	3
			Initial design	Preliminary high-level design	3

Paper	S17 – From Requirements through Risks to Software Architecture for Plan-based and Agile Processes				
Type	Expert Opinion		**Approach**	Hybrid approach guide by risks	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
1	2nd	2nd	Problems with scalability	Scalability problems	2
2	1st	1st	Evaluate architectures that mitigate risks	Risk driven methodology	3
2	2nd	2nd	Iteratively refine architecture	Iterative process	3
				Refinement	3

Paper	S18 – Software Architecture-Centric Methods and Agile Development				
Type	Expert Opinion		**Approach**	Embed ‘architecting’ practices into XP	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
47	2nd	2nd	Architectures in XP depends on the development team’s experience	Reliance on Experience	2
49	1st	2nd	Identify key requirements:	Iteration Zero	3

			QAW practice	Enhance quality	3
				Customer involved in architectural tasks	3
50	2nd	3rd	Early architectural design: ADD practice	Quality attributes enable architecture	3
				Design patterns usage	3
51	1st	2nd	Views: module decomposition view, concurrency view and deployment view	Structural architecture	1
51	1st	3rd	Assess architecture: ATAM practice	Risk assessment	3
51	1st	5th	Analyze architecture: CBAM practice	Perform impact analysis every iteration	3

Paper	S19 – Embedding Architectural Practices into Extreme Programming				
Type	Expert Opinion		**Approach**	Embed 'architecting' practices into XP	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
310	2nd	3rd	Lack of structure and quality	Structural problems	2
				Lack of quality	2
312	1st	5th	Metaphors are useless for designing architecture	XP proposals for architecture do not address the problem	2
314	2nd	2nd	Continuous Architectural Refactoring (CAR) practice introduced in parallel with the development process of XP	Refactoring	3
				Devoted roles	3
315	1st	1st	Iterative and incremental process	Iterative and incremental process	3
315	2nd	2nd	Real Architecture Qualification (RAQ) practice for evaluation of the architectural decisions made	Architectural evaluation at the end of each iteration	3

Paper	S20 – Extreme programming and its development practices				
Type	Expert Opinion		**Approach**	Hybrid approach (XP+RUP)	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
101	1st	3rd	XP metaphor is not enough to replace Software Architecture	XP proposals for architecture do not address the problem	2
102	1st	7th	Software Architecture is the biggest weaknesses of XP	XP proposals for architecture do not address the problem	2
103	1st	8th	Merge XP and RUP	Embed XP practices into RUP iterations	3

Paper	S21 – Scenario-Based Usability Engineering Techniques in Agile Development Processes				
Type	Expert Opinion		**Approach**	Embed Usability Techniques into XP	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
2162	1st	1st	XP does not answer the question regarding design	XP proposals for architecture do not address the problem	2
2162	1st	2nd	Introduce techniques from Scenario-Based Eng. And Rapid Contextual Design	Preliminary high-level design	3
				(Re)-Design	3
				Iterative process	3
				Integration effort	3

Paper	S22 – XPnUE – Defining and Teaching a Fusion of eXtreme Programming & Usability Engineering				
Type	Expert Opinion		**Approach**	Embed Usability Techniques into XP	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
5	1st	5th	Benefit: focus on the architecture	Improved quality	4
5	1st	7th	Benefit: prototypes narrow down design	Simpler design	4
5	1st	9th	Benefit: continuous design under pressure	Scalability enabled	4

Paper	S23 - The eXtreme programming (XP) metaphor and Software Architecture				
Type	Empirical Study		**Approach**	XP architectural proposals	
Page	**Col.**	**Par.**	**Code**	**Translation**	**RQ**
5	1st	4th	XP Metaphors are useless either to communicate among team members or to design architecture.	XP proposals for architecture do not address the problem	2

Paper	S24 – Integrating Software-Architecture-Centric Methods into Extreme Programming (XP)				
Type	Expert Opinion		**Approach**	Embed 'architecting' practices into XP	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
6	1st	4th	XP understands architecture as system metaphors	Agile architectural artifacts	1
6	1st	5th	System metaphors are not used	XP proposals for architecture do not address the problem	2
6	1st	7th	System metaphors does not address important views	XP proposals for architecture do not address the problem	2
7	1st	1st	XP proposes Simple Design practice for architecture	Agile architectural artifacts	1
8	1st	2nd	XP does not care about design evaluation	Lack of architectural evaluation	2

Paper	S25 – Incorporating Agile with MDA Case Study: Online Polling System				
Type	Mixed Contribution		**Approach**	Hybrid MDA-XP	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
88	1st	7th-9th	XP is best suited for single projects and doesn't work where a customer insists in a complete specification or design	Agile is not suitable for large projects	2
89	1st	1st	Analyze requirements with customer and create domains	Customer involved in architectural tasks	3
89	1st	2nd	Create a general meta-model confirmed by the customer		3
90	1st	1st-2nd	The different domains work	Iterative process	3

			iteratively in XP way		
90	1st	3rd	The integration phase is the last one, where all the domain applications are integrated and testing is done	Integration effort	3
92	1st	1st	Advantage: applicable to big and small projects	Scalability enabled	4
92	1st	2nd	Advantage: the customer is involved in the whole lifecycle	Enhanced customer involvement	4
92	1st	3rd	Advantage: division in domains shortens time for completion	Reduced time to market	4

Paper	S26 – Architecture and Design in eXtreme Programming: The Architectural Game, introducing a new practice			
Type	Mixed Contribution	**Approach**	Embed architectural practice into Agile Methodology (XP)	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
7	1st	9th	Lack of usability	Usability Problems	2
8	1st	1st	Lack of flexibility	Lack of flexibility	2
8	1st	2nd	High-coupled resulting software	Structural problems	2
8	1st	5th	XP does not produce adequate architecture	XP proposals for architecture do not address the problem	2
9	1st	7th	The architectural game takes place at the beginning of the project	Iteration Zero	3
9	1st	9th	Involves an architect role	Devoted roles	3
9	1st	10th	The architectural game takes place every time a subset of the project is finished	Iterative process	3
10	1st	2nd	Prioritization of requirements	Prioritization of requirements	3
10	1st	1st	Simplicity is the key	Simple design (KISS principle)	3
10	1st	2nd	Create architecture based in a subset of requirements	Reviewing	3

			reviewing it until it is considered as sustainable		
10	1st	3rd	Involves all the developers, joint practice	Design Meetings	3
10	1st	4th	Minimal documentation (e.g. UML diagrams) can be useful in order to enable team communication	Communication tools usage	3
11	1st	2nd-6th	Architectural game improve communication, simplicity, feedback, courage and respect	Enhanced communication	4
				Enhanced customer involvement	4
				Simpler design	4
13	1st	2nd	The architectural game pauses all activities of XP - Flow	Architectural work reduces agility	4

Paper	S27 – Odyssey and Other Code Science Success Stories				
Type	Expert Opinion		**Approach**	Variation of XP	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
20	2nd	2nd	XP lacks architectural design and that causes problems of integrity and scalability	Lack of conceptual integrity	2
				Scalability problems	
20	2nd	3rd	Architecture consists in identify components and its relations	Structural architecture	1
21	1st	5th	XP Metaphors are useless	XP proposals for architecture do not address the problem	2
21	3rd	2nd a)	First iteration: high-level architecture	Iteration Zero	3
				Preliminary high-level design	
21	3rd	2nd b)	Iterative and incremental process	Iterative and incremental process	3

Paper	S28 – How Extreme does Extreme Programming Have to be? Adapting XP Practices to Large-scale Projects				
Type	Empirical Study		**Approach**	Variation of XP	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
1	2nd	2nd	Agile's lack of architecture makes it difficult to scale up projects	Scalability problems	2

1	2nd	3rd	Agile methods are not suitable to large, complex projects	Agile is not suitable for large projects	2
2	2nd	4th			
3	1st	2nd	Lack of design and rely in tacit knowledge is a high risk practice	Reliance on Experience	2
3	1st	3rd	Standing-meetings and metaphors are not suitable to describe architectures	XP proposals for architecture do not address the problem	2
4	1st	3rd	Designing up front can be introduced as a practice under certain conditions	Design upfront	3
7	1st	4th	Architecture supports rest of XP practices	Agility maintained	4
7	1st	5th	Effect: reduced development time and supports rapid development	Reduced development time	4
7	2nd	1st	Effect: reuse by refactoring enhanced	Enhanced reusability	4

Paper	S29 – Quality Attribute driven Agile Development				
Type	Mixed Contribution		**Approach**	Modification of SCRUM	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
203	2nd	2nd	Challenges of SCRUM: lack of quality, difficult maintenance, difficulties in multiple cooperating teams	Lack of quality	2
				Maintenance problems	2
				Team communication problems	2
206	1st	3rd	Insertion of three previous activities: AQUA, RAM and VAQ	Embed architectural practices into agile	3
206	2nd	2nd	Analyze quality attributes (AQUA practice)	Enhance quality	3
207	1st	2nd-3rd	Software Architecture arises from the analysis of the quality attributes	Quality attributes enable architecture	3
207	2nd	4th	Map functional requirements to quality requirements (RAM practice)	(Method explanation)	3
208	1st	3rd	Validation of the achievement of the quality	Architectural evaluation at the end of each iteration	3

			(VAQ practice)		
209	2nd	2nd	Effect: keep agility intact (Case study)	Agility maintained	4
209	2nd	2nd	Effect: quicker development (Case study)	Reduced development time	4
209	2nd	3rd	Effect: improved quality. 26% less time fixing defects	Improved quality	4

Paper	S30 – Tailoring the Scrum Development Process to Address Agile Product Line Engineering				
Type	Mixed Contribution		**Approach**	Modification of SCRUM to achieve Agile Product Line Engineering	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
2	1st	2nd	Challenge: Scalability and reusability.	Scalability problems	2
				Lack of reusability	2
4	1st	5th	PL-architecture: designed incrementally and iteratively	Iterative and incremental process	3
8	1st	4th	Pregame and Release Definition phases: analysis and prioritization of User Stories	Prioritization of user stories	3
9	1st	4th	In the Domain Engineering phase the architecture is consolidated	Two-cycle method	3
9	1st	5th	In the Application Engineering phase the architecture is refined for the working product		3

Paper	S31 – Towards a Process for Architectural Modelling in Agile Software Development				
Type	Mixed Contribution		**Approach**	Agile Architectural Modeling (Scrum)	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
183	2nd	2nd	Lack of reuse of software, product lines not supported	Lack of reusability	2
185	1st	8th	Iterative and incremental process	Iterative and incremental process	3
185	2nd	5th	Step 0: gather information and prioritize requirements	Prioritization of requirements	3

185	2nd	6th	Step 1: architectural design through patterns	Design patterns usage	3
186	1st	4th	Step 2: component design for the pattern chosen		3
189	1st	11th	Architectural sprints into scrum	Architectural Scrum sprints	3

Paper	S32 – A Hybrid Software Development Method for Large-Scale Projects: Rational Unified Process with SCRUM				
Type	Expert Opinion		**Approach**	Hybrid model (SCRUM + RUP)	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
340	2nd	4th	Challenges: rely on tacit knowledge, not suitable for critical systems, not suitable for stable projects, no large-scale projects	Reliance on Experience	2
				Lack of reliability	2
				Agile is not suitable for stable environments	2
				Agile is not suitable for large projects	2
344	2nd	2nd	Allocates RUP practices into SCRUM lifecycle	Embed architectural practices into agile	3
346	Image			Design meetings	3

Paper	S33 – Mobile-D: An Agile Approach for Mobile Application Development				
Type	Expert Opinion		**Approach**	Hybrid model (XP + Crystal + RUP)	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
175	1st	3rd	Iterative process	Iterative process	3
175	1st	4th	Capture architectural knowledge to be reused	Usage of knowledge	3
175	1st	5th	Architecture is build incrementally	Incremental process	3

Paper	S34 – An Agile Development Methodology Applied to Embedded control Software under Stringent Hardware Constraints				
Type	Mixed Contribution		**Approach**	Architectural practices applied in Scrum Context	
Page	**Col.**	**Par.**	**Code**	**Translation**	**RQ**
3	2nd	3rd	Iterative and incremental process that refines the	Iterative and incremental process	3

			architectural choices	Refinement	3

Paper	S35 – Empirical Evaluation of the Proposed eXSCRUM Model: Results of a Case Study				
Type	Mixed Contribution		**Approach**	Combination of XP and SCRUM	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
152	2nd	2nd a)	Design oriented by KISS principle (simple) focused on sprint requirements	Simple design (KISS principle)	3
				Iterative process	3
152	2nd	2nd b)	Design is focused on two types of diagrams: class and object diagrams.	Structural architecture	1
153	Image		Design activities inside Sprint Development Cycle (Inner cycle)	Architectural Scrum sprints	3
153	Image		Design iterates refining in daily scrum meetings		3

Paper	S36 – Agile-Architecture Interactions				
Type	Expert Opinion		**Approach**	Hybrid approach of SCRUM + XP along with architectural practices	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
41	2nd	1st	Practice: up-front planning: High-level architectural tasks	Preliminary high-level design	3
42	2nd	3rd	Practice: storyboarding: attend to assure architecture compliance	Refactoring	3
				Devoted roles	3
43	2nd	2nd	Practice: Sprint: heavy participation of the architect to ensure the architecture is being produced	Architectural evaluation at the end of each iteration	3
44	1st	3rd	Practice: working software: Review the software against the architecture		3

Paper	S37 – Methodology Support for the Model Driven Architecture			
Type	Expert Opinion	**Approach**	Agile architecting MDA and agile development	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
458	2nd	3rd	Initiation phase: identification of high-level requirements and overall architecture	Iteration Zero	3
458	2nd	4th	Component identification and plan development	Preliminary high-level design	3
458	2nd	5th	Iterative development process	Iterative process	3
				Architectural evaluation at the end of each iteration	3

Paper	S38 – A Framework for Extending Agile Methodologies with Aspect-Oriented Features			
Type	Expert Opinion	**Approach**	Hybrid approach Agile extended with Aspect-Oriented Software Development	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
3-4	2nd – 2nd		Performance of traditional requirements, high-level design, detailed design phases in several continuous iterations	Prioritization of requirements	3
				Iterative process	3
				Architectural evaluation at the end of each iteration	3
5	Table		Effect: increased reusability	Enhanced reusability	4
			Effect: quality increased as it is taken into account	Improved quality	4

Paper	S39 – Flexible Working Architectures: Agile Architecting using PPCs			
Type	Mixed Contribution	**Approach**	Architectural practices embedded in agile iterations	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
109	1st	7th a)	Architecture is developed incrementally and iteratively.	Iterative and incremental process	3
109	1st	7th b)	Architecture team is integrated in the agile team	Devoted roles	3
110	1st	2nd	Priorization of user stories to start architecting	Prioritization of user stories	3
				Customer involved in architectural tasks	3

110	1st	6th	The architecture previously created guides the next iterations	Preliminary high-level design	3
111	1st	5th	Agile principles enhanced	Agility maintained	4

Paper	S40 – Adding Agility to Architecture Tradeoff Analysis Method for Mapping on Crystal				
Type	Expert Opinion		**Approach**	Embed architectural practices (ATAM) into Agile Methodologies (Crystal)	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
122	2nd	8th	Challenge: Lack of Software Architecture evaluation method	Lack of architectural evaluation	2
123	1st	Image	Software Architecture arises in reflective workshops where team discusses and with the customers	Design meetings	3
				Customer involved in architectural tasks	3
			Incremental re-architecture	Incremental process	3
				Refactoring	3
General			Map ATAM to Crystal phases for enabling architectural evaluation	Embed architectural practices into agile	3

Paper	S41 – A Hybrid Software Architecture Evaluation Method for FDD – An Agile Process Model				
Type	Expert Opinion		**Approach**	Embed architectural practices (QAW, ATAM & ARID) into Agile Methodologies (FDD)	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
3	2nd	1st	Initial model, identifying requirements with QAW	Embed architectural practices into agile	3
			Architecture is designed in the initial phase concurrently	Preliminary high-level design	3
3	2nd	3rd	ATAM is used for evaluating features	Embed architectural practices into agile	3
			ARID is used for the verifications of the architecture	Embed architectural practices into agile	3

Paper	S42 – Realizing quality improvement through test driven development: results and experiences of four industrial teams				
Type	Empirical Study		**Approach**	Pre-defined architecture and TDD	
Page	**Col.**	**Par.**	**Code**	**Thematic Translation**	**RQ**
296	1	4	Hybrid-TDD approach that involves design	Design upfront	3
297	Table		TDD reduces defect density	Fewer defects	4
			TDD increases development time	Increased development time	4
			TDD with design phase is more effective in reducing defect density	Hybrid approach	3

- **(Method Explanation):** This indicates that the associated code refers to a description of the method. It has been considered as interesting to assign a code to those sections as far as those sections are very helpful for understanding the method.

14 APPENDIX E – LIST OF THEMES & HIGH-ORDER THEMES

During the Thematic Synthesis [42] process conducted, the following themes and high-order themes have been created. They are presented in relation their respective Research Question.

Research Question 1 – Concept of Software Architecture

- Agile Concept of Software Architecture
    - Agile architectural artifacts
    - Structural architecture

Research Question 2 – Challenges

- Agile suitability constraints
    - Agile is not suitable for large projects
    - Agile is not suitable for unknown domains
    - Agile is not suitable for complex projects
    - Agile is not suitable for working with large teams
    - Agile is not suitable for stable environments
- Project challenges
    - Lack of quality
    - No alternative design solutions considered
    - Lack of architectural evaluation
    - Lack of professional management
    - Lack of conceptual integrity
    - Non-reliable software is produced
    - XP proposals for architecture do not address the problem
- Team challenges
    - Reliance on Experience
    - Perceived irrelevance of the architectural work
    - Team communication problems
- Customer challenges
    - Customer communication problems
    - Usability Problems
    - Lack of documentation
- Long-Term challenges
    - Lack of reusability
    - Maintenance problems
    - Scalability problems
    - Structural problems
    - Lack of flexibility

Research Question 3 – Practices

- Agile with Architecture Approach
    - Embed architectural practices into agile
    - Architectural evaluation

    - Preliminary high-level design
    - Design upfront
    - Design patterns usage
    - Perform impact analysis every iteration
    - Iteration Zero
    - Refactoring techniques:
        - Refactoring
        - Refinement
        - Reviewing
        - (Re)-Design
    - Simple Design (KISS principle)
    - Architectural Scrum Sprints.
    - Design meetings.
    - Customer involved in architectural tasks.
    - Devoted roles.
    - Usage of knowledge.
    - Experience-based architecture.
    - Enhance Quality.
    - Quality attributes enable architecture.
- Agile Architecting Approach
    - Prioritization of input:
        - User Stories
        - Sections
        - Requirements.
    - Iterative & Incremental process.
    - Embed XP practices into RUP iterations.
    - Communication tools usage.
    - Architect as communication enabler.
    - Divide teams.
    - Integration effort.
    - Two-cycle methods.
- Mixed Methodology Approach
    - Hybrid approach
    - Risk driven methodology
    - Project characterization
    - Monitor and readjust methodology

Research Question 4 – Effects

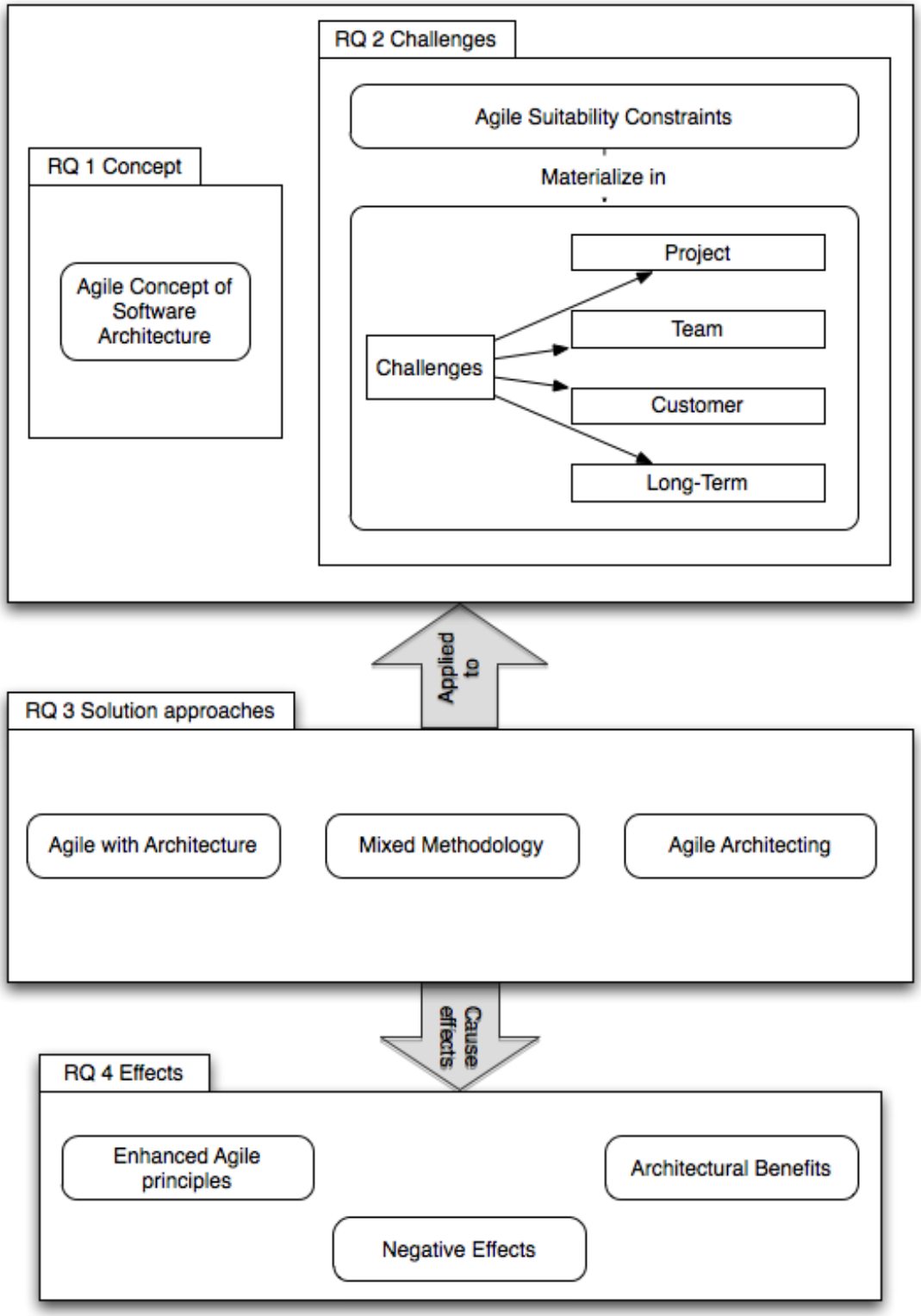
- Enhanced agile principles
    - Less documentation
    - Enhanced communication
    - Simpler design
    - Agility maintained
    - Enhanced customer involvement
    - Reduced time to market
    - Reduced development time
- Architectural benefits
    - Flexibility enabled

    - Scalability enabled
    - Improved quality
    - Fewer defects
    - Enhanced reusability
- Negative effects
    - Architectural work reduces agility
    - Increased development time
    - Team experience dependency

15 APPENDIX F – SLR PRACTICES (RQ 3) DUAL CLASSIFICATION

	Architecture-inspired practices	Agile-inspired practices	Communication practices	Quality practices	Costly practices
Agile with Architecture Approach	Embed architectural practices into agile Architectural evaluation Preliminary high-level design Design upfront Design patterns usage Perform impact analysis every iteration	Iteration Zero Refactoring techniques Simple Design (KISS principle) Architectural Scrum Sprints	Design meetings Customer involved in architectural tasks	Enhance quality Quality attributes enable architecture	Devoted roles Usage of knowledge Experience-based architecture
Agile Architecting Approach		Prioritization of Input Iterative & Incremental process Embed XP practices into RUP	Communication tools usage Architect as communication enabler		Divide teams Integration effort Two-cycle methods
Mixed Methodology Approach	Risk driven methodology	Project characterization Monitor & readjust methodology			Hybrid approach

16 APPENDIX G – SLR THEMATIC MAP



17 APPENDIX H – COMPARATIVE: FINDINGS SLR – INTERVIEWS

The following list shows which of the findings of the Systematic Literature Review (expressed through the themes and high-order themes created in the *Thematic Synthesis* conducted) are supported by empirical evidences obtained from the conduction of Semi-Structured Interviews with Agile Practitioners.

Research Question 1 – Concept of Software Architecture

- Agile Concept of Software Architecture
    - Structural architecture

Research Question 2 – Challenges

- Agile suitability constraints
    - Agile is not suitable for large projects
    - Agile is not suitable for unknown domains
- Team challenges
    - Reliance on Experience
- Long-Term challenges
    - Scalability problems
    - Lack of flexibility

Research Question 3 – Practices

- Agile with Architecture Approach
    - Embed architectural practices into agile
    - Preliminary high-level design
    - Design patterns usage
    - Iteration Zero
    - Refactoring
    - Design meetings.
    - Customer involved in architectural tasks.
    - Devoted roles.
    - Experience-based architecture.
- Agile Architecting Approach
    - Prioritization of input:
        - User Stories
        - Sections
        - Requirements.
    - Iterative & Incremental process.
    - Communication tools usage.

Research Question 4 – Effects

- Enhanced agile principles
    - Less documentation
    - Enhanced customer involvement
    - Reduced time to market
- Negative effects
    - Architectural work reduces agility

- Team experience dependency

There were also some other findings that arose during the interviews that had not been previously obtained in the Systematic Review, these are:

Research Question 1 – Concept of Software Architecture

- Executable Architecture
- Architecture on-demand

Research Question 4 – Effects

- Teams feels relaxed and confident
- Customer exceeds his tasks

The absence of some of the themes obtained from the Systematic Literature Review (see full list at Appendix E – List of Themes & High-Order Themes) does not mean that that information is not valid, only that they have less empirical support; they are theories yet to be empirically tested, and they should be taken for what they are.