electronics

Article

Guidelines for Future Agile Methodologies and Architecture Reconciliation for Software-Intensive Systems

Fábio Gomes Rocha [1], Sanjay Misra [2] and Michel S. Soares [1,*]

[1] Department of Computing, Federal University of Sergipe, São Cristóvão 49100000, Brazil
[2] Institute for Energy Technology, 1777 Halden, Norway
* Correspondence: michel@dcomp.ufs.br

Abstract: Background: Several methodologies have been proposed since the first days of software development, from what is now named traditional/heavy methodologies, and later their counterpart, the agile methodologies. The whole idea behind agile methodologies is to produce software at a faster pace than what was considered with plan-based methodologies, which had a greater focus on documenting all tasks and activities before starting the proper software development. **Problem:** One issue here is that strict agilists are often against fully documenting the software architecture in the first phases of a software process development. However, architectural documentation cannot be neglected, given the well-known importance of software architecture to the success of a software project. **Proposed Solution:** In this article, we describe the past and current situation of agile methodologies and their relation to architecture description, as well as guidelines for future Agile Methodologies and Architecture Reconciliation. **Method:** We propose a literature review to understand how agile methodologies and architecture reconciliation can help in providing trends towards the success of a software project and supporting software development at a faster pace. This work was grounded in General Systems Theory as we describe the past, present, and future trends for rapid systems development through the integration of organizations, stakeholders, processes, and systems for software development. **Summary of results:** As extensively discussed in the literature, we found that there is a false dichotomy between agility and software architecture, and then we describe guidelines for future trends in agile methodologies and reconciliation of architecture to document agile architectures with both architectural decisions and agile processes for any system, as well as future trends to support organizations, stakeholders, processes, and systems.

Keywords: software architecture; agile development; software process

Citation: Rocha, F.G.; Misra, S.; Soares, M.S. Guidelines for Future Agile Methodologies and Architecture Reconciliation for Software-Intensive Systems. *Electronics* **2023**, *12*, 1582. https://doi.org/10.3390/electronics12071582

Academic Editor: Claus Pahl

Received: 14 February 2023
Revised: 14 March 2023
Accepted: 15 March 2023
Published: 28 March 2023

1. Introduction

Software development has been recognised by many researchers and practitioners as a challenge for society in the past decades [1–3]. Software is used for controlling infrastructures such as water, waste, and electricity distribution, as well as means of transport, telecommunications, industry, entertainment, and so on, in such a way that software makes our daily life easier and safer. However, there are also difficulties and concerns related to activities of software development and evolution, such as balancing the necessary effort for activities of Requirements Engineering and designing the Software Architecture.

Barry Boehm [4] shows his software experiences through the stages of the evolution of Software Engineering practices in the 20th and 21st centuries, also, the strategies used to evaluate and adapt software changes. Boehm used the philosopher Hegel's hypothesis divided into three stages: thesis (things happen the way they do), antithesis (the thesis fails in some aspects, but there is a better explanation), and synthesis (rejection of the original thesis with the antithesis, but there is a similar improved one). In this way, he pointed out the evolution of Software Engineering from the 1950s to 2006, as well as a few topics for the next decades.

As software is considered a crucial element of software-intensive systems [5] and systems of systems, failures are studied by many researchers [3,6]. Failures in software development and deployment may have consequences of loss of lives, may jeopardize human life, and may have financial and even moral consequences. Given the many issues involved with software development, including delays for product delivery [7], lack of communication [8], low-quality [9], and delivery of a product that does not reflect the necessities of the organization [10], many methodologies have been proposed for establishing some systematic order for activities and tasks of software development [11]. Therefore, these methodologies have been proposed to organise the overall software development process, trying to provide structure to the activities of software development.

Software development methodologies can be classified as heavy/traditional/plan based [12] (heavy in the remainder of the article), or agile [12,13]. The first type relies on specific strict rules, well-defined activities, and tasks to describe a systematic process that can be repeated and followed by a development team. Heavy methodologies rely on documenting each development step in such a way that the team can store and, in the future, retrieve information at a higher level of abstraction than source code. The whole idea behind agile methodologies is to produce software at a faster pace than what was considered heavy by plan-based methodologies, which had a greater focus on documenting all tasks and activities.

Agile architecture is the result of the transformation from traditional and agile approaches to software development, e.g., the architecture uses agile techniques to integrate people, processes, and technologies to ensure that the architecture is flexible, adaptable, and tolerant to changing demands through the agile iterative–incremental development process.

The need for agile architecture happened through the emergence of modern software platforms and frameworks, such as microservices, cloud services, IoT, mobile devices, artificial intelligence, and smart machines, as well as the need for solutions with a high response speed to changes and facilities. Therefore, documenting this kind of architecture with both architectural decisions and agile processes for any system is a challenge [14].

This article is organized as follows. Section 2 describes a review of items related to Software and Systems Architecture and Agile Development, including Architecture-Based Development and Agile-Based Development. Section 3 describes two main phases that the community of Software Engineering, both working in academia and in the software industry, followed for architecture and agile reconciliation. Section 4 brings the Guidelines for future Agile Methodologies and Architecture Reconciliation, provided in seven steps. Section 5 is about the discussion of related works and threads for validity, and the article ends with a section on the conclusion and future works.

2. A Review on Items about Architecture and Agile Development

Since the mid-1990s, new software development processes have been categorised according to the principles that they embody, which can be heavy (or plan-driven) and agile processes. While software process development based on the traditional paradigm is recommended for risky and large-scale projects, agile processes are more suitable for small and low-risk projects. Critical software projects may be negatively affected by the lack of rigour and predictability of agile methods, while small and low-risk projects may be jeopardised by the lack of simplicity and flexibility of heavy methodologies [15].

2.1. Agile Methods

By the end of the 1990s, the advent of the Internet as a platform for the execution of all kinds of software and information systems became a reality. The idea of a more agile way of software development in this highly changing environment seemed necessary. Therefore, the Agile Manifesto was proposed, inspired by researchers and practitioners who were leading this important field.

The agile manifesto demanded the use of iterative methods for product development based on four principles:

(i) individuals and interactions over processes and tools, (ii) working software with comprehensive documentation, (iii) customer collaboration in contract negotiation, and (iv) responding to changes following a plan.

Extreme Programming (XP), Scrum, Lean, Kanban, and other agile methodologies were introduced in practice in the past two decades. These methodologies reduce the risks related to both the uncertainties and the unforeseen aspects of projects more naturally and rapidly, aiming at better communication among stakeholders and customer satisfaction. In addition, these methodologies emphasize qualities and standards to make the relationship between people, communications, software, and cooperation with the customer and the reactions to changes more direct [16].

2.2. Software Architecture

The importance of Software architecture has been discussed by many researchers and practitioners since the 1970s, when Djikstra [17] and Parnas [18] first mentioned the importance of the field for software development. Their first attempts at defining software architecture relied on organizing the structure of large-scale software systems at the time. Their approach was centred on documenting the structure of software systems by using then-well-known concepts such as separation of concerns, modularity and hierarchical decomposition.

Then, in the 1990s, other conceptions and ideas on software architecture were introduced by Royce [19]and Garlan and Schaw [20]. Many definitions for software architecture were proposed as well. For Kruchten [21], software architecture deals with the design and implementation of the high-level structure of the software, as it is the result of assembling architectural elements to satisfy the main functionality and performance requirements of the system, as well as some non-functional requirements. Clements et al. [22] assert that the software architecture of a system is the structure or structures of the system, which comprise the software components, the externally visible properties of these components, and the relationships between them. According to the standard ISO/IEC/IEEE 42010:2011, software architecture is the fundamental concept or properties of a system in its environment embodied in its elements, relationships, and the principles of its design and evolution. More than 100 additional definitions for software architecture are presented on the Software Engineering Institute website (SEI).

Modern definitions of Software architectures have a greater focus on (i) software decisions to guide software composition (constraints and goals) and (ii) key components (relationships, interactions, principles, and structures). Software architecture is considered a crucial technical element that facilitates communication among stakeholders, determines which change paths have the least risk, assesses the consequences of changes, and decides both sequences and priorities for changes by looking at their relationships. Important quality constraints such as performance, security, safety, modularity, and many others are the direct consequence of a well-defined software architecture.

2.3. Architecture-Based Development

Architectural documentation cannot be neglected, given the well-known importance of software architecture for the success of a software project [23,24]. A well-defined architecture helps in new developments from existing processes and systems, as well as in proposing trade-offs and analysing and making changes.

Examples of concerns related to software architecture include separation of interests, coupling, cohesion, encapsulation, modularity, change implementation (learnability, instability, testability, and manageability), interoperability, compliance, and reusability, among others. The software architecture is documented through the different scenarios related to the properties and, consequently, by evaluating the impact throughout the organization.

Table 1 summarises a non-exhaustive list of seminal articles, models, books, methods and standards related to software architecture from the first years of the 1990s up to 2020.

Enterprise architecture models, such as Zachman and TOGAF, are not mentioned in the table because our approach here is to include architecture elements for software/systems.

Table 1. Seminal works related to software architecture, from the first years of the 1990s up to 2020.

Name	Year	Type
Systems Architecting: Creating and Building Complex Systems [25]	1990	Book
Foundations for the Study of Software Architecture [26]	1992	Article
Architecture Modeling Language (ADLs)	1992–2005	Languages
The 4+1 View Model of Software Architecture [21]	1995	Model
Siemens' 4 Views (S4V) [27]	1995	Model
Software architecture - perspectives on an emerging discipline. [20]	1996	Book
RM-ODP - ISO/IEC 10746	1998	Model
IEEE 1471	2000	Standard
Software architecture analysis method (SAAM)	1995	Method
Architecture Tradeoff Analysis Method (ATAM)	2000	Method
Model Driven Architecture (MDA)	2001	Method
ISO/IEC/IEEE 42010	2011	Standard
ISO/IEC/IEEE 42020	2019	Standard
ISO/IEC/IEEE 42030	2019	Standard
Architectural Decisions	2000	Model

2.4. *Agile-Based Development*

Scrum and XP are currently two of the most widely used agile methodologies [28].

The main concept of Scrum is the Sprint, a time-boxed iteration of two to four weeks. The Scrum methodology has a special focus on management practices and consists of a set of Scrum teams and their associated roles, artefacts, rules and time-boxed events such as Release Planning Meetings, Sprint Planning Meetings, Sprint, Daily Scrum, Sprints, Sprint Reviews and Sprint Retrospective.

Common practices in XP are Test-Driven Development (TDD), pair programming, continuous integration, the planning game, metaphor, small release, open workspace, and refactoring [29].

The first results with agile methodologies were linked to cost and return of investment (ROI), productivity, response to change, client participation, and individuals and their daily working tasks in an agile environment [30,31]. Most recently, the outcomes of agile methodologies were customer-focused, customer satisfaction, flexibility, productivity, collaboration, multiple deliverables, product quality, and reduction of problems and failures [32,33].

In agile development, data, information, processes, requirements, and solutions evolve through the participation and planning of teams and organizations. In this way, it promotes adaptive planning, evolutionary development, early delivery, and continuous improvement, and encourages a fast and flexible response to change [34].

Table 2 summarises a non-exhaustive list of seminal books, processes, practices, and guides related to agile methods and agile architecture from the first years of the 1990s up to 2011.

Table 2. Seminal works related to agile methods, from the first years of the 1990s up to 2018.

Name	Year	Type
Dynamic systems development method (DSDM)	1994	Process
Rapid application development (RAD)	1994	Process
Scrum	1995	Process
User Story	1997	Practice
Feature-driven development (FDD)	1999	Process
XP	1999	Process
Test-driven development (TDD)	1999	Practice
Agile manifesto	2001	Manifesto
Agile unified process (AUP)	2001	Process
Behavior-driven development (BDD)	2003	Practice
KANBAN	2004	Process
Domain-driven design (DDD)	2004	Practice
Guide to Agile Practices	2011	Guide
The Lean Startup	2011	Book
Disciplined Agile Toolkit	2018	Process

3. Attempts for Agile-Architecture Reconciliation

Considering the introduction of agile methodologies by the middle of the 1990s, and the situation of architecture knowledge during the same period, we can divide the reconciliation into two attempts.

3.1. First Attempts for Agile-Architecture Reconciliation

Starting from the first years of the 1990s up until around 2005, from the platform point of view, client-server architectures were commonly deployed in the software industry. Internet and web-based systems passed from infancy (simple websites, static Internet pages) to complex e-commerce systems (business to consumer, business to business), to Intranet systems coordinating many applications and enabling agility for a company's business processes.

From 1992 to 2005, a variety of agile methods were proposed, including XP, Scrum, FDD, DSDM, RAD, and others [35]. Initially, the research and industry work in this field involved many aspects, including the efficiency of pair programming [36], final quality of systems developed using agile methods [37], size of the team [38], and costs and schedule of projects [39].

However, in the 1990s, the relationship between architecture and agility was hardly mentioned. There is no surprise about this result, as research in these first years was mostly about evaluating what agility could bring to the industry in terms of better quality, lower costs, improvement in project management, and faster delivery of software products.

This period brought another "flame war" on agility versus heavy processes, and many introduced the idea that a software architecture would emerge naturally [40,41], instead of being a specific activity to be performed throughout the software life cycle. Therefore, agilists were against the idea of establishing a software architecture before design and implementation, which they called "Big Design Up-Front". In their opinion, considering that agile methods support the idea of embracing changes to requirements, defining a whole big picture of software architecture would be a waste of time, as the requirements were most certainly changing a few times during software development. Even design diagrams were seen as not important or even counterproductive, as the teams did not have time to spend designing diagrams considering the fast pace of software delivery.

A report from SEI [42] describes a method for capturing architectural information in a manner consistent with agile philosophies. The report compares the Software Engineering Institute's Views and Beyond approach for documenting software architectures with the documentation philosophy embodied in agile methods.

Later, another report from SEI [43] proposes two versions of a design of an ATM: the first one from the eXtreme Programming perspective and then the second using an Architecture Centric approach from SEI. As a result, the authors found that the architecture-centric methods help to fill gaps in the XP design process, for instance, by eliciting and documenting quality attribute requirements and also by understanding and predicting the consequences of the design decisions in terms of risks and tradeoffs. In the end, there are guidelines on how to integrate architecture-centric methods from SEI into the XP software development process.

In 2000, Kent Beck, the proponent of XP, published the book *Test-Driven Development: By Example* [44] presenting TDD as a way to teach developers to communicate and to adopt constant feedback. Additionally, it promotes a continuous improvement of the source code through refactoring. The TDD method was born inside XP as one of the practices of that process. However, it was widely adopted by agile developers following other agile methods. In 2009, Freeman and Pryce published the book "Growing object-oriented software, guided by tests," bringing a practical vision applied to projects, demonstrating that TDD helps in the code quality, aiming for better integration of the components.

In 2002, with the expansion of the adoption of Agile, Ambler published the book *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process* [45]. In this book, the author presents a way to model software to better align with agile principles. He presents a model with alignment between eXtreme Programming (XP) and the Unified

Process (UP) methodologies. As a result, the author presents a lighter form of modelling but with some values such as communication, simplicity, constant feedback, and courage.

The need for strategies and models that allow software architecture planning in an agile process led Evans, in 2004, to publish the book *Domain-driven design: tackling complexity in the heart of the software*, presenting DDD. This methodology proposes a domain-oriented software design. A book regarding balancing agility and discipline brings six results in the end, one concluding that in the future, we will need both agility and discipline [13].

One can see the evolution in Figure 1. This idea was not usual at the time, which is why the authors would expect that in the near future. Thus, this balancing would be more common in the next years, as described in the following section.

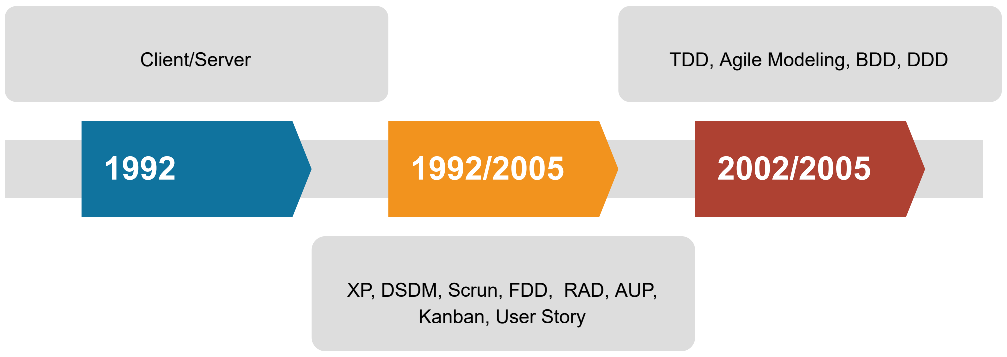
Client/Server
TDD, Agile Modeling, BDD, DDD
1992
1992/2005
2002/2005
XP, DSDM, Scrun, FDD, RAD, AUP, Kanban, User Story

Figure 1. Agile-Architecture Reconciliation.

3.2. Maturity—Second Decade for Agile-Architecture Reconciliation

From the execution platforms point of view, the mid-2000s bring innovations and evolution in platforms such as highly distributed systems, improved middleware and publish–subscribe technologies, and high demand for business-to-business applications, including e-commerce, e-government and so on.

From 2006 to 2016, agile methods reached a maturity level in the software industry. Agile methods were commonly applied to a variety of domains and types of software, including critical software such as railway signalling [46], embedded systems [47], and medical device software [48]. Regarding the size of teams, although agile methods were first mentioned as suitable only for small teams [49,50], as they present difficulties in scaling and coordination of large teams [51], other results have applied agile methods to larger teams. An example is a study that presents an agile project following the SCRUM method involving 70 people (including 30 developers) [52].

The software industry suggested in the 2000s that agile methods presented many challenges regarding the adoption of safety-critical systems. The most mentioned issues include communication and scalability [53], difficulties in coordinating the team [54], and too short a time for delivery of complex components [55]. In addition, one cannot guarantee that practices used in an agile environment for developing an information system for business activities can be successfully replicated for highly demanded flight control systems. In another example [56], agile methods, and more specifically, the economics of pair programming, was evaluated for NASA projects, and the result is that this practice will only be demonstrably useful for a very small number of cases. Software development approaches that do not precisely define the requirements are not indicated for NASA projects with significant or hard risks. However, even for safety critical systems, there are good examples of success [46,57,58].

Relationships with architecture in this decade also evolved. Instead of a flame war between agilists on one side and software architects on the other side, new ideas and research were introduced on combining and reconciling the two worlds. One issue for agilists is that they believe that software architecture is about creating "lots of documents" that are not useful. On the other hand, software architects cannot agree on a software project in which interfaces between components, important decisions, and styles are not documented in detail. Some researchers and practitioners believe that the truth is somewhere in the middle [13,59–61]. Agile methods and software architecture can coexist and support each other where appropriate.

In 2012, Robert C. Martin published an article on his website called "The Clean Architecture" bringing essential points related to software architecture when adopting agile methods. After that, in 2018, the author, together with other experts, published the book *Clean Architecture: A Craftsman's Guide to Software Structure and Design* [62] bringing debate points about software architecture focused on the agile model.

Dan North, seeking to facilitate the adoption and explanation of TDD, presents the Behaviour-Driven Development (BDD) [63], a way to guide the developer in the elaboration of unit tests, being guided by the acceptance criteria. Along with the methodology, Dan North created JBehave, a framework that allowed the automation and integration of user stories and unit tests. Subsequently, the book *The Cucumber Book: Behaviour-Driven Development for Testers and Developers* [64] presented Cucumber, a tool that became a standard in the adoption of BDD to help testers and developers.

DDD had advanced in adoption, especially after 2013 with the publication of the book *Implementing Domain-Driven Design* by Vernon [65]. In this book, the author brings a realistic view of the adoption of DDD. In addition, works such as [66,67] point to Domain-driven design as a form of architecture planning when adopting microservices.

In addition to DDD, two strategies for the discovery and documentation of the architecture stand out. Event Storming, created by Alberto Brandolini [68], widely disseminated in the DDD community, has been gaining space for allowing more visual and agile documentation that involves the stakeholders in the process. Another model that aims to support the documentation based on DDD is Domain Storytelling, proposed by Stefan Hofer and Henning Schewentner [69], which introduces a model based on languages with figures to model usage scenarios, having events in a workshop format to integrate the stakeholders and to elaborate, step by step, a visual model of the domain. Both techniques are complementary ways to DDD, working as a way to create a model that can guide the development.

4. Guidelines for Future Agile Methodologies and Architecture Reconciliation

The relationship between Architecture and Agility was also studied in [70], in which the authors argue that instead of another debate on more agility in architecture, or more architecture in agility, the correct would be rephrasing the question in more general terms, by addressing the relationship between architecture and processes in general.

We describe guidelines for future trends in agile methodologies and reconciliation of architecture to document agile architectures with both architectural decisions and agile processes for any system, as well as future trends to support organizations, stakeholders, processes, and systems.

Considering the new platforms, including IoT and Cloud Computing, and the high complexity of current software systems, with novelties such as Self-healing systems, we propose guidelines for future agile methodologies and architecture reconciliation.

Our assumption is that architecture and agile methods are compatible, and it is not at all impossible to consider them together for software development. To define a good architecture in agile systems, it is necessary to design for important factors such as agility and quality and maintain the balance between stability, flexibility, and business changes [71].

Therefore, instead of discussing agility or architecture, the solution would be to discuss how much architecture is good enough after considering important aspects of the problem and the proposed solution, such as system complexity, number of modules and their interactions, and quality characteristics that are mandatory for the solution.

Architecture reconciliation ensures the identification of requirements, the integrity between system functionalities, and the control of information for the documentation process through the use of agile methods. It identifies and corrects architectural deviations in the implementation of the systems to monitor quality [72]. Thus, methods like BDD [63], and DDD [73] bring architectural planning and documentation into the agile world in a lightweight way that enriches teams with information. However, this is not enough, and we need to move forward.

It is essential to perform (i) a plan to define the architectural directions and structure the business needs. All stakeholders (and here one has to understand that all really means developers, users, managers, and architects, i.e., the group of stakeholders related to the problem and solution to be developed and deployed) should bring their necessities, constraints, beliefs and ideas for the system.

After understanding the context in which the stakeholders are involved, then (ii) divide the architectural project into small teams to identify architectural concerns to balance business priorities. For instance, it makes no sense at all to spend 15 months defining all aspects of a software architecture for an e-commerce solution to sell laptops and mobile phones, as the time to market is an issue here. New improved products are introduced by these industries in less than 12 months.

Once the ecosystem that includes business requirements and architectural constraints is well-understood, then the stakeholders can (iii) establish time intervals for building functionality, which includes defining release dates, time to market, and the main functionalities of products to be developed, followed by (iv) measuring the value and state of the architecture, including quality concerns such as maintainability, scalability, and extensibility, and (v) outline the structures to be designed and developed by other teams. This fifth step is facilitated by understanding the proposed architecture in such a way that it is clear what is expected by each part of the architecture, as well as the interfaces between the architectural elements and the environment.

To support the first five items of the guide, it is still worth highlighting that it is necessary to (vi) analyze the organization's culture transition process so that they can adapt and adopt more transparent and updated work models, i.e., agile models and agile architectures, as well as (vii) identify and analyze the stakeholders' behaviours, feelings, and difficulties during the transition process between traditional and agile methods (see Figure 2). These two activities must occur in parallel with the other five, as they are useful as a support and control of the main five activities.

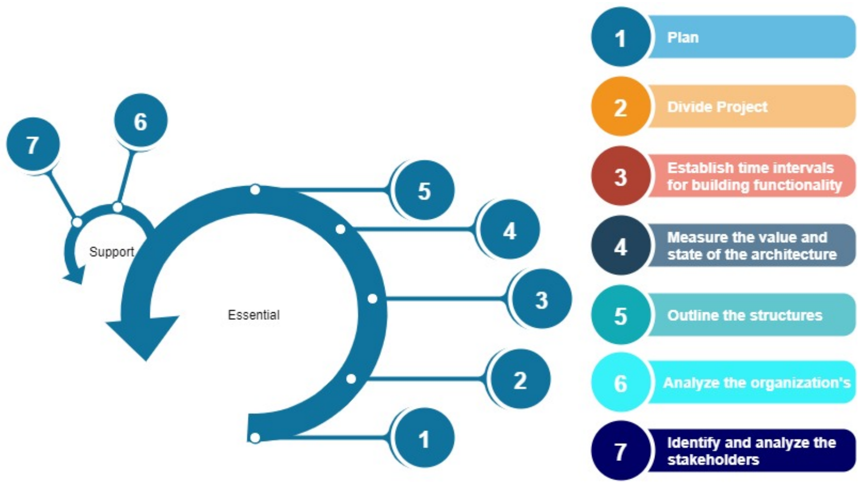
7
6
Support
5
4
3
2
1
Essential
1 Plan
2 Divide Project
3 Establish time intervals for building functionality
4 Measure the value and state of the architecture
5 Outline the structures
6 Analyze the organization's
7 Identify and analyze the stakeholders

Figure 2. Architecture Guide.

5. Discussion

Software Engineering is a recent discipline of great importance in modern life. One has to understand that there are many types of Software Engineering (embedded or iterative, large-scale or small, mission-critical or casual use) [4], and it is unlikely that a one-size-fits-all software process will ever exist. Thus, Software Development processes and methodologies have been introduced, studied, applied, and evaluated since the first days of modern software development in the 1950s.

In the first decades of software development, hardware to execute the software developed was expensive and not easily available. Computer Time-sharing was crucial, as large computers were rented by the hour of actual execution. Therefore, this hardware orientation led to the idea that Software Engineering was like Hardware Engineering [4]. Fortunately, hardware evolution brought cheaper machines in the following decades, and Software Engineering emerged as an important discipline with its own processes and methodologies.

First, these Software Engineering methodologies followed step-by-step planning. Then, iterative and incremental processes emerged as a number of so-called agile methodologies. Nowadays, there are supporters for both approaches.

Those that prefer planned-based methodologies, have arguments that include the importance of documenting all steps at a higher level of abstraction than source code. As software will inevitably be part of a future software maintenance and evolution process, understanding a number of documents is easier than reading source code written by others who may even not be part of the development team anymore. Then, reading well-described software architecture documents will enable improved software evolution. Those who support agile methods have their arguments as well. For instance, as the software will be changed in the future, then spending too much time creating documents that will be different from the actual source code is a waste of time.

However, what if these heavy methodologies and agile methodologies could be combined? An example of this approach is described in [74], which proposes a new software development methodology called a waterative model, with the purpose of integrating the waterfall and iterative development paradigms. In this waterative model, the iterative model is embedded into the waterfall model to use the advantages of both models as an integrated one.

Another example is provided in [75], whose objective is to investigate the evolution of hybrid software development methods. According to the authors, modern software development is neither pure linear phase progression nor agile. A challenge arises with regard to selecting the appropriate combination of approaches that serve to reach goals and assure value creation for organizations.

Strict agilists are often against documenting the software architecture [40]. They argue that it is better to spend time on a computer keyboard using a text editor to write source code than creating high-level models. In addition, they mention the well-known difficulty of synchronising source code and design models [76].

Considering the software life cycle, activities related to Requirements Engineering (for instance, Requirements Elicitation, Prioritization, and Documentation) are often considered too heavy-weight by agile practitioners. Agilists often consider that requirements volatility (constant modifications to requirements) is a major issue in software development, causing problems such as higher defect density, project delays, and cost overruns [77]. Given the strict relation between Software Requirements and Software Architecture, this requirements volatility will certainly result in significant changes in the software architecture.

However, "a healthy focus on architecture is not antithetic" to agility [78], and late discovery of quality requirements may lead to the wrong architecture being implemented and, consequently, the need for costly refactoring. Concerns about the long-term deterioration of quality in large-scale agile projects have renewed the agile community's interest in software architecture [79]. One known challenge in the software industry in the past two decades is to find a suitable balance between up-front software design and emergent software architecture [40].

Due to Software Architecture's importance, the ISO/IEC/IEEE 42010:2011 standard was published in 2011 with the purpose of establishing a coherent practice for developing architecture descriptions, architecture frameworks, and architecture description languages within the context of a software life cycle and its processes. However, parts of the standard are still not considered in practice.

Industry and academia can benefit from learning and improving the use of ISO/IEC/IEEE 42010:2011 [80], even for agile methods, as a sound software architecture is crucial for software evolution (as the XP book's subtitle notes: "Embrace Change" [81]).

Given the aspects mentioned above, it can be seen that the union between architecture and agile methodology is not only possible but a good and reliable idea. Agility and architecture are compatible, and architecture is not at all an impediment to agility. A proper reconciliation between architecture and agility improves both the speed of development and the final quality of the software during the entire life cycle.

The validity of this study may be threatened due to some limitations. The threads in this review may be related to the primary study selection process and data extraction process.

During the study selection process, the relevance in relation to the research topic and theme was analysed. We cannot completely guarantee that we found all studies related to agile and software architecture.

Regarding the data extraction process, there may be imprecision, and this may be another threat to the validity of this research. There may be two reasons that cause this inaccuracy: the data is not extracted systematically, and its classification may be invalid. To reduce inaccurate data, we focus only on the data collected from the selected articles and books.

6. Conclusions and Future Work

The correct use of agility and the introduction of software architecture into agile environments can control complexity, eliminate long requirements specifications, speed up decision-making, and improve communication and problem-solving. In addition, one can expect to improve development and, consequently, the quality of the code and models, and as a consequence, reinforcement of best practices, providing consistency and uniformity, reducing risks, and allowing for higher levels of reuse and improved maintenance.

Therefore, there is a false dichotomy between agility and software architecture, as discussed in this article. Actually, software architecture is an enabler for agility, not an impediment, as agile methods rely on embracing change. A well-defined software architecture describes, for instance, the stakeholders and their concerns, as well as the most important components, their relationships (interfaces, APIs, and so on), and a variety of views and models. In addition, the software architecture documents important decisions and defines the general structure of source code using layers, modules, libraries, header files and configuration files.

Activities such as embracing changes to the software project, including inserting and deleting requirements and scenarios, from the agile point of view, become natural when the important elements of the software are defined, as well as when their relationships are described at a high level of abstraction. All these elements are described in documents of software architecture. Therefore, the software architecture helps the stakeholders to identify items and elements to be changed, which facilitates software maintenance.

Future research will focus on creating a software tool to help the software architect with heuristics about how much architecture is good enough after considering important aspects of the problem and the proposed solution, such as system complexity, number of modules and their interactions, and quality constraints that are mandatory for the solution. Then, the software tool will be applied to describe the software architecture in a software development process of a software-intensive system.

Author Contributions: Conceptualization, M.S.S.; methodology, M.S.S. and F.G.R.; validation, M.S.S. and F.G.R. and S.M.; investigation, M.S.S. and F.G.R.; data curation, M.S.S. and F.G.R. and S.M.; writing—original draft preparation, M.S.S. and F.G.R.; writing—review and editing, M.S.S.; visualization, M.S.S., F.G.R. and S.M.; supervision, M.S.S.; project administration, M.S.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Chapman, W.L.; Rozenblit, J.; Bahill, A.T. System Design is an NP-Complete Problem. *Syst. Eng.* **2001**, *4*, 222–228. [CrossRef]
2. Berry, D.M. The Inevitable Pain of Software Development: Why There Is No Silver Bullet. In Proceedings of the Radical Innovations of Software and Systems Engineering in the Future, 9th International Workshop, RISSEF 2002, Venice, Italy, 7–11 October 2002; Revised Papers; Lecture Notes in Computer Science; Wirsing, M., Knapp, A., Balsamo, S., Eds.; Springer: Berlin/Heidelberg, Germany, 2002; Volume 2941, pp. 50–74.
3. Charette, R.N. IT's Fatal Amnesia. *Computer* **2017**, *50*, 86–91. [CrossRef]
4. Boehm, B. A View of 20th and 21st Century Software Engineering. In Proceedings of the 28th International Conference on Software Engineering, ICSE '06, Shanghai, China, 20–28 May 2006; Association for Computing Machinery: New York, NY, USA, 2006; pp. 12–29.
5. Soares, M.S.; Vrancken, J.; Verbraeck, A. User Requirements Modeling and Analysis of Software-Intensive Systems. *J. Syst. Softw.* **2011**, *84*, 328–339. [CrossRef]
6. Hanagal, D.D.; Bhalerao, N.N. Introduction to Software Reliability Models. In *Software Reliability Growth Models*; Springer: Singapore, 2021; pp. 1–12.
7. Quiña-Mera, A.; Chamorro Andrade, L.; Montaluisa Yugla, J.; Chicaiza Angamarca, D.; Guevara-Vega, C.P. Improving Software Project Management by Applying Agile Methodologies: A Case Study. In *Proceedings of the Applied Technologies*; Botto-Tobar, M., Montes León, S., Camacho, O., Chávez, D., Torres-Carrión, P., Zambrano Vizuete, M., Eds.; Springer International Publishing: Cham, Switzerland, 2021; pp. 672–685.
8. Niazi, M.; Mahmood, S.; Alshayeb, M.; Riaz, M.R.; Faisal, K.; Cerpa, N. Challenges of Project Management in Global Software Development: Initial Results. In Proceedings of the 2013 Science and Information Conference, London, UK, 7–9 October 2013; pp. 202–206.
9. Stamelos, I. Software Project Management Anti-Patterns. *J. Syst. Softw.* **2010**, *83*, 52–59. [CrossRef]
10. Ruk, S.A.; Khan, M.F.; Khan, S.G.; Zia, S.M. A survey on Adopting Agile Software Development: Issues amp; Its impact on Software Quality. In Proceedings of the 2019 IEEE 6th International Conference on Engineering Technologies and Applied Sciences (ICETAS), Kuala Lumpur, Malaysia, 20–21 December 2019; pp. 1–5. [CrossRef]
11. Sharon, I.; Soares, M.S.; Barjis, J.; van den Berg, J.; Vrancken, J. A Decision Framework for Selecting a Suitable Software Development Process. In Proceedings of the ICEIS 2010—Proceedings of the 12th International Conference on Enterprise Information Systems, Volume 3, ISAS, Funchal, Madeira, Portugal, 8–12 June 2010; Filipe, J., Cordeiro, J., Eds.; SciTePress: Vienna, Austria, 2010; pp. 34–43.
12. Gheorghe, A.M.; Gheorghe, I.D.; Iatan, I.L. Agile Software Development. *Inform. Econ.* **2020**, *24*. [CrossRef]
13. Boehm, B.; Turner, R. *Balancing Agility and Discipline: A Guide for the Perplexed*; Addison-Wesley Longman Publishing Co., Inc.: Boston, MA, USA, 2003.
14. Maric, M.; Matkovic, P.; Tumbas, P.; Pavlicevic, V. Documenting Agile Architecture: Practices and Recommendations. In Proceedings of the EuroSymposium on Systems Analysis and Design, Gdansk, Poland, 29 September 2016; pp. 56–71.
15. Carvalho, W.C.d.S.; Rosa, P.F.; Soares, M.S.S.; Cunha, M.A.T.d., Jr.; Buiatte, L.C. A Comparative Analysis of the Agile and Traditional Software Development Processes Productivity. In Proceedings of the 2011 30th International Conference of the Chilean Computer Science Society, Washington, DC, USA, 9–11 November 2011; pp. 74–82. [CrossRef]
16. Saleh, S.M.; Huq, S.M.; Rahman, M.A. Comparative Study within Scrum, Kanban, XP Focused on Their Practices. In Proceedings of the 2019 International Conference on Electrical, Computer and Communication Engineering (ECCE), Cox's Bazar, Bangladesh, 7–9 February 2019; pp. 1–6.
17. Dijkstra, E.W. The Structure of "THE"-Multiprogramming System. *Commun. ACM* **1968**, *11*, 341–346. [CrossRef]
18. Parnas, D.L. On the Criteria To Be Used in Decomposing Systems into Modules. *Commun. ACM* **1972**, *15*, 1053–1058. [CrossRef]
19. Royce, W.E.; Royce, W. Software Architecture: Integrating Process and Technology. *Quest* **1991**, *14*, 2–15.
20. Shaw, M.; Garlan, D. *Software Architecture: Perspectives on an Emerging Discipline*; Prentice-Hall, Inc.: Upper Saddle River, NJ, USA, 1996.
21. Kruchten, P. The 4+1 View Model of Architecture. *IEEE Softw.* **1995**, *12*, 42–50. [CrossRef]
22. Clements, P.C. Coming Attractions in Software Architecture. In Proceedings of the 5th International Workshop on Parallel and Distributed Real-Time Systems and 3rd Workshop on Object-Oriented Real-Time Systems, Geneva, Switzerland, 3 April 1997; pp. 2–9.
23. Kouroshfar, E.; Mirakhorli, M.; Bagheri, H.; Xiao, L.; Malek, S.; Cai, Y. A Study on the Role of Software Architecture in the Evolution and Quality of Software. In Proceedings of the 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, Florence, Italy, 16–17 May 2015; pp. 246–257. [CrossRef]
24. Whiting, E.; Andrews, S. Drift and Erosion in Software Architecture: Summary and Prevention Strategies. In Proceedings of the 2020 the 4th International Conference on Information System and Data Mining, ICISDM 2020, Hawaii, HI, USA, 15–17 May 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 132–138.
25. Rechtin, E. *Systems Architecting: Creating and Building Complex Systems*; Prentice Hall: Upper Saddle River, NJ, USA, 1990.
26. Perry, D.E.; Wolf, A.L. Foundations for the Study of Software Architecture. *ACM SIGSOFT Softw. Eng. Notes* **1992**, *17*, 40–52. [CrossRef]
27. Soni, D.; Nord, R.L.; Hofmeister, C. Software Architecture in Industrial Applications. In Proceedings of the 17th International Conference on Software Engineering, Seattle, WA, USA, 23–30 April 1995; pp. 196–207.

28. Herdika, H.R.; Budiardjo, E.K. Variability and Commonality Requirement Specification on Agile Software Development: Scrum, XP, Lean, and Kanban. In Proceedings of the 2020 3rd International Conference on Computer and Informatics Engineering (IC2IE), Yogyakarta, Indonesia, 15–16 September 2020; pp. 323–329.
29. Beck, K. *Extreme Programming Explained: Embrace Change*; Addison-Wesley Professional: Boston, MA, USA, 2000.
30. Abrahamsson. Extreme Programming: First Results from a Controlled Case Study. In Proceedings of the 2003 Proceedings 29th Euromicro Conference, Antalya, Turkey, 1–6 September 2003; pp. 259–266. [CrossRef]
31. Kahkonen, T. Agile Methods for Large Organizations—Building Communities of Practice. In Proceedings of the Agile Development Conference, Salt Lake City, UT, USA, 22–26 June 2004; pp. 2–10. [CrossRef]
32. Jorgensen, M. Relationships Between Project Size, Agile Practices, and Successful Software Development: Results and Analysis. *IEEE Softw.* **2019**, *36*, 39–43. [CrossRef]
33. Vithana, V.N.; Asirvatham, D.; Johar, M. An Empirical Study on Using Agile Methods in Global Software Development. In Proceedings of the 2018 18th International Conference on Advances in ICT for Emerging Regions (ICTer), Colombo, Sri Lanka, 26–29 September 2018; pp. 150–156. [CrossRef]
34. Pang, C.Y. An Agile Architecture for a Legacy Enterprise IT System. *Int. J. Organ. Collect. Intell.* **2016**, *6*, 65–97. [CrossRef]
35. Kumar, R.; Maheshwary, P.; Malche, T. Inside agile family software development methodologies. *Int. J. Comput. Sci. Eng.* **2019**, *7*, 650–660. [CrossRef]
36. Canfora, G.; Cimitile, A.; Visaggio, C.A. Empirical Study on the Productivity of the Pair Programming. In Proceedings of the Extreme Programming and Agile Processes in Software Engineering, Sheffield, UK, 18–23 June 2005; Baumeister, H., Marchesi, M., Holcombe, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2005; pp. 92–99.
37. Huo, M.; Verner, J.; Zhu, L.; Babar, M. Software Quality and Agile Methods. In Proceedings of the Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004 COMPSAC 2004, Hong Kong, China, 28–30 September 2004; Volume 1, pp. 520–525.
38. Reifer, D.; Maurer, F.; Erdogmus, H. Scaling Agile Methods. *IEEE Softw.* **2003**, *20*, 12–14. [CrossRef]
39. Coram, M.; Bohner, S. The Impact of Agile Methods on Software Project Management. In Proceedings of the 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'05), Greenbelt, MD, USA, 4–7 April 2005; pp. 363–370.
40. Booch, G. The Economics of Architecture-First. *IEEE Softw.* **2007**, *24*, 18–20. [CrossRef]
41. Garlan, D. Software Architecture: A Travelogue. In Proceedings of the Future of Software Engineering Proceedings, FOSE 2014, Hyderabad, India, 31 May–7 June 2014; Association for Computing Machinery: New York, NY, USA, 2014; pp. 29–39.
42. Clements, P.C.; Ivers, J.; Little, R.; Nord, R.; Stafford, J.A. *Documenting Software Architectures in an Agile World*; Technical Note CMU/SEI-2003-TN-023; Software Engineering Institute: Pittsburgh, PA, USA, 2003.
43. Nord, R.L.; Tomayko, J.E.; Wojcik, R. *Integrating Software-Architecture-Centric Methods into Extreme Programming (XP)*; Technical Note ADA431084; Software Engineering Institute: Pittsburgh, PA, USA, 2004.
44. Beck, K. *Test-Driven Development: By Example*; Addison-Wesley Professional: Boston, MA, USA, 2000.
45. Ambler, S. *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*; John Wiley & Sons: Hoboken, NJ, USA, 2002.
46. Jonsson, H.; Larsson, S.; Punnekkat, S. Agile Practices in Regulated Railway Software Development. In Proceedings of the 2012 IEEE 23rd International Symposium on Software Reliability Engineering Workshops, Washington, DC, USA, 27–30 November 2012; pp. 355–360. [CrossRef]
47. Kaisti, M.; Rantala, V.; Mujunen, T.; Hyrynsalmi, S.; Könnölä, K.; Mäkilä, T.; Lehtonen, T. Agile Methods for Embedded Systems Development—A Literature Review and a Mapping Study. *EURASIP J. Embed. Syst.* **2013**, *2013*, 15. [CrossRef]
48. McHugh, M.; McCaffery, F.; Casey, V. Adopting Agile Practices when Developing Software for Use in the Medical Domain. *J. Softw. Evol. Process.* **2014**, *26*, 504–512. [CrossRef]
49. Lindvall, M.; Basili, V.; Boehm, B.; Costa, P.; Dangle, K.; Shull, F.; Tesoriero, R.; Williams, L.; Zelkowitz, M. Empirical Findings in Agile Methods. In Proceedings of the Extreme Programming and Agile Methods — XP/Agile Universe 2002, Chicago, IL, USA, 4–7 August 2002; Wells, D., Williams, L., Eds.; Springer: Berlin/Heidelberg, Germany, 2002; pp. 197–207.
50. Papatheocharous, E.; Andreou, A.S. Evidence of Agile Adoption in Software Organizations: An Empirical Survey. In *Proceedings of the Systems, Software and Services Process Improvement*; McCaffery, F., O'Connor, R.V., Messnarz, R., Eds.; Springer: Berlin/Heidelberg, Germany, 2013; pp. 237–246.
51. Turk, D.; France, R.B.; Rumpe, B. Limitations of Agile Software Processes. *CoRR* **2014**, abs/1409.6600.
52. Tessem, B.; Maurer, F. Job Satisfaction and Motivation in a Large Agile Team. In *Proceedings of the Agile Processes in Software Engineering and Extreme Programming*; Concas, G., Damiani, E., Scotto, M., Succi, G., Eds.; Springer: Berlin/Heidelberg, Germany, 2007; pp. 54–61.
53. Paige, R.F.; Charalambous, R.; Ge, X.; Brooke, P.J. Towards Agile Engineering of High-Integrity Systems. In *Proceedings of the Computer Safety, Reliability, and Security*; Harrison, M.D., Sujan, M.A., Eds.; Springer: Berlin/Heidelberg, Germany, 2008; pp. 30–43.
54. Rohunen, A.; Rodriguez, P.; Kuvaja, P.; Krzanik, L.; Markkula, J. Approaches to Agile Adoption in Large Settings: A Comparison of the Results from a Literature Analysis and an Industrial Inventory. In *Proceedings of the Product-Focused Software Process Improvement*; Ali Babar, M., Vierimaa, M., Oivo, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 77–91.
55. Ge, X.; Paige, R.F.; McDermid, J.A. An Iterative Approach for Development of Safety-Critical Software and Safety Arguments. In Proceedings of the 2010 Agile Conference, Nashville, TN, USA, 9–13 August 2010; pp. 35–43. [CrossRef]
56. Smith, J.; Menzies, T. *Should NASA Embrace Agile Processes*; Technical Report; NASA: Washington, DC, USA, 2002.

57. Gary, K.; Enquobahrie, A.; Ibáñez, L.; Cheng, P.; Yaniv, Z.; Cleary, K.; Kokoori, S.; Muffih, B.; Heidenreich, J. Agile Methods for Open Source Safety-Critical Software. *Softw. Pract. Exp.* **2011**, *41*, 945–962. [CrossRef]
58. Mishra, D.; Mishra, A. Complex Software Project Development: Agile Methods Adoption. *J. Softw. Maintenance Res. Pract.* **2011**, *23*, 549–564. [CrossRef]
59. Boehm, B.; Lane, J.A.; Koolmanojwong, S.; Turner, R. Architected Agile Solutions for Software-Reliant Systems. In *Agile Software Development—Current Research and Future Directions*; Dingsøyr, T.; Dybå, T.; Moe, N.B., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 165–184.
60. Kruchten, P. Software Architecture and Agile Software Development: a Clash of Two Cultures? In *Proceedings of the 2010 32nd International Conference on Software Engineering (ICSE)*; IEEE Computer Society: Los Alamitos, CA, USA, 2010; Volume 2, pp. 497–498.
61. Nord, R.L.; Ozkaya, I.; Kruchten, P. Agile in Distress: Architecture to the Rescue. In Proceedings of the Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation - XP 2014 International Workshops, Rome, Italy, 26–30 May 2014, Revised Selected Papers; Lecture Notes in Business Information Processing; Dingsøyr, T., Moe, N.B., Tonelli, R., Counsell, S., Gencel, Ç., Petersen, K., Eds.; Springer: Berlin/Heidelberg, Germany, 2014; Volume 199, pp. 43–57.
62. Martin, R.C.; Grenning, J.; Brown, S.; Henney, K.; Gorman, J. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*; Number s 31; Prentice Hall: Hoboken, NJ, USA, 2018.
63. North, D. Introducing bdd. *Better Softw.* **2006**, *12*.
64. Wynne, M.; Hellesoy, A.; Tooke, S. *The Cucumber Book: Behaviour-Driven Development for Testers and Developers*; Pragmatic Bookshelf: Raleigh, NC, USA, 2017.
65. Vernon, V. *Implementing Domain-Driven Design*; Addison-Wesley: Boston, MA, USA, 2013.
66. Balalaie, A.; Heydarnoori, A.; Jamshidi, P. Microservices architecture enables devops: Migration to a cloud-native architecture. *IEEE Softw.* **2016**, *33*, 42–52. [CrossRef]
67. Cerny, T.; Donahoo, M.J.; Trnka, M. Contextual understanding of microservice architecture: current and future directions. *ACM SIGAPP Appl. Comput. Rev.* **2018**, *17*, 29–45. [CrossRef]
68. Brandolini, A. *Introducing EventStorming: An act of Deliberate Collective Learning*; Leanpub: Victoria, BC, Canada, 2018.
69. Hofer, S.; Schwentner, H. *Domain Storytelling: A Collaborative, Visual, and Agile Way to Build Domain-Driven Software*; Addison-Wesley: Boston, MA, USA, 2022.
70. Buschmann, F.; Henney, K. Architecture and Agility: Married, Divorced, or Just Good Friends? *IEEE Softw.* **2013**, *30*, 80–82. [CrossRef]
71. Lankhorst, M.M.; Proper, H.A. Agile architecture. In *Agile Service Development*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 41–57.
72. Maier, M.W. System and Software Architecture Reconciliation. *Syst. Eng.* **2006**, *9*, 146–159. [CrossRef]
73. Evans, E.; Evans, E.J. *Domain-Driven Design: Tackling Complexity in the Heart of Software*; Addison-Wesley: Boston, MA, USA, 2004.
74. Gharajeh, M. Waterative Model: An Integration of the Waterfall and Iterative Software Development Paradigms. *Database Syst. J.* **2019**, *10*, 75–81.
75. Kirpitsas, I.; Pachidis, T. Evolution towards Hybrid Software Development Methods and Information Systems Audit Challenges. *Software* **2022**, *1*, 316–363. [CrossRef]
76. Badreddin, O.; Khandoker, R.; Forward, A.; Masmali, O.; Lethbridge, T.C. A Decade of Software Design and Modeling: A Survey to Uncover Trends of the Practice. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS '18*; Association for Computing Machinery: New York, NY, USA, 2018; pp. 245–255. [CrossRef]
77. Dasanayake, S.; Aaramaa, S.; Markkula, J.; Oivo, M. Impact of Requirements Volatility on Software Architecture: How do Software Teams Keep Up with Ever-Changing Requirements? *CoRR* **2019**, abs/1904.08164. [CrossRef]
78. Mohagheghi, P.; Aparicio, M.E. An Industry Experience Report on Managing Product Quality Requirements in a Large Organization. *Inf. Softw. Technol.* **2017**, *88*, 96–109. [CrossRef]
79. Bellomo, S.; Gorton, I.; Kazman, R. Toward Agile Architecture: Insights from 15 Years of ATAM Data. *IEEE Softw.* **2015**, *32*, 38–45. [CrossRef]
80. da Costa Junior, A.A.; Misra, S.; Soares, M.S. A Systematic Mapping Study on Software Architectures Description Based on ISO/IEC/IEEE 42010: 2011. In Proceedings of the Computational Science and Its Applications—ICCSA 2019—19th International Conference, Saint Petersburg, Russia, 1–4 July 2019, Proceedings, Part V; Lecture Notes in Computer Science; Misra, S., Gervasi, O., Murgante, B., Stankova, E.N., Korkhov, V., Torre, C.M., Rocha, A.M.A.C., Taniar, D., Apduhan, B.O., Tarantino, E., Eds.; Springer: Berlin/Heidelberg, Germany, 2019; Volume 11623, pp. 17–30.
81. Beck, K.; Andres, C. *Extreme Programming Explained: Embrace Change,* 2nd ed.; Addison-Wesley: Boston, MA, USA, 2004.