Challenges of Traditional and Agile Software Processes

Jacqueline Marín Sánchez
Computer Science Department, University of Chile.
jamarin@dcc.uchile.cl

Abstract— **Software processes have evolved significantly since software engineers started to follow a disciplined flow of activities to gain quality and productivity in development. Several process models, methodologies, methods and/or software development practices have been proposed, adopted or rejected that differ at the ceremony level. For many years, there have been conflicts over whether to follow a totally traditional approach ("classical") or to become more agile. Each approach has its strengths and weaknesses, and each has followers and critics. However, the ongoing diversity of software projects and the advancement of technology has led to debates about what kinds of software process approaches are more context-effective. This paper surveys existing traditional and agile processes and discusses their challenges.**

I. INTRODUCTION

The question of how software development should be organized has been debated by the software engineering community for decades. Improving the ways in which software products are developed is a challenge that must be addressed carefully [1] due to the complexity of the software [2]. The software development industry recognizes the need to define and manage processes in order to gain quality and productivity, as well as to apply practices that allow them to make the best use of the available resources [3].

Software development has adopted different forms during its evolution. It has followed highly formalized or structured approaches (high ceremony) with a close management supervision, and also by other more unstructured approaches (low ceremony). Many proposals for processes, models, methods, tools, techniques and concrete practices have been proposed to this end. However, evidence suggests that "there is no unique approach to software development" [4].

Traditional processes follow methods based on the quality of the artifacts, on the predictability of their processes and on architectural designs that can adapt the change before it has an adverse impact on the system [5]. The first proposal that followed this philosophy was the waterfall model created by Winston Royce in 1970 [6]. Waterfall model established an era in which software development is recognized as a complex activity, centered on people, which has many difficulties if it is not properly organized [7].

Soon problems arose: long development time, involvement of the client only at the beginning of the project, costly changes, difficulties for innovation and excessive documentation that these rigorous processes implied. As a consequence, around 1990 a community that defends change from a radically different perspective has emerged. This new way of addressing software development, called "agile" by its defenders, is ruled by 12 principles explained in the "Agile Manifesto" [8]. In this environment individuals and interactions are valued over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation and responding to change over following a plan.

Agile methods and practices aim to simplify the software process by avoiding "bureaucracy" and advocate short time cycles, close involvement of the client and an adaptive rather than predictive strategy [9]. Many researches defend that agile methods provide a better way to address people's needs, accelerate software development and improve quality and customer satisfaction. These and other reasons make the industry to advocate for the agile philosophy. However, agile methods can fail when they are applied in the wrong context [10] and are also rarely used in their "pure" form [11].

The history of software development shows that the choice of a process is not a simple deterministic exercise, but depends on the situational characteristics of individual development environments [7]. Therefore, it is necessary to consider the application domain, as well as organizational, project and team characteristics, among others [12].

This research seeks to gain a better understanding of how software development has evolved and the challenges involved in following a traditional or agile software process depending on the context. The results of this study can help organizations, projects and software teams get a better idea of which software processes fit their needs.

II. BACKGROUND

According to Cugola and Ghezzi [13], the interest towards processes arises at the beginning of the 70s. Software engineers realized that desired qualities such as reliability, efficiency, evolution capacity, ease of use, etc., could only be achieved by following a disciplined flow of activities. However, only at the end of the 1980s software processes were recognized as a subject that deserves attention. Thereafter, efforts were made to understand its fundamentals, develop useful models, identify methods, provide tool support and help manage its progress.

According to Kroll and Kruchten [14] a process "describes who is doing what, how, and when". Feiler and Humphrey [15] emphasize that a process is a set of partially ordered steps intended for achieving a goal that can be associated with the production or improvement of software products, or the provision of services.

In [2], Fuggetta states in a comprehensive way, that "a software process can be defined as the coherent set of policies, organizational structures, technologies, procedures, and artifacts that are needed to conceive, develop, deploy, and maintain a software product". From his perspective, Pressman [16] emphasizes that it is necessary to see the software process as a guide for the activities, actions and tasks that are required to obtain high quality software. In a more general way, Münch et. al. [17] argue that "a software process is a goal-oriented activity in the context of engineering-style software development", where "software process" not only applies to software development, but also to planning, coding, testing and maintaining software.

Paraphrasing Osterweil [18], software processes must be treated in the same way we treat software. These must be carefully designed and constructed in addition to being tested, executed, maintained and reused. Seeing software development as a process has significantly helped to identify the different dimensions of development and the elements necessary to propose effective methods and practices for each context [2].

III. An Overview of Software Process Approaches

Since the beginning, software processes have been used to collect and organize knowledge about software development and, since then, a large number of approaches compete for "the users' favor" [11]. According the literature [5], [11], [19], [20], [21], [22], [23], there are in practice two trends that guide software development, one following a "rigid plan-driven approach" and the other a set of "slim agile methods".

Processes that follow a more rigid approach are known as traditional [16], plan-driven [5], classic [11], structured [24], rich [19], and heavyweight [25], among others. For this research, the term traditional will be used. This approach aims to address the entire life cycle of the software project. On the other hand, the agile methods or more precisely "agile software development methods or processes" [26], intend to simplify the software process to the minimum to avoid "bureaucracy".

Each of the approaches, agile or traditional, emerged at a critical moment in the history of software development with well-defined purposes and contexts.

A. *Traditional processes*

Traditional processes intended to bring some order to the so-called "chaos of software development" [16], providing structure to the work and a roadmap for software teams. The traditional development assumes that all the desired requirements and properties of the final product can be known and specified with precision before beginning the construction of the final product [23]. This philosophy also supports the need for a detailed plan from the beginning to the end of the project, where a deviation from this plan is a sign of bad work in the early stages of the project [19].

Boehm and Turner [5] state that traditional development is desirable when requirements are stable and predictable, as well as when the project is large, critical and complex. In general, it can be said that traditional processes are characterized by: predictive approach, integral planning, process orientation, exhaustive documentation, continuous learning during development and directed to large projects [5], [11], [16], [17], [23], [24].

Among traditional processes there are the so-called "life-cycle process models" such as waterfall model [6], iterative enhancement model [27], spiral model [28], evolutionary prototyping model [29], etc. A software life cycle defines the skeleton and philosophy according to which the software process must be carried out [2]. However, they generally explain in a high level of abstraction "what" to do and not "how" to do it [17].

These processes include similar activities for software development, but each one places different emphasis on them. Some examples are:

- Waterfall model [6] prescribes a systematic and sequential approach to software development that begins with the specification of requirements and follows through analysis, design, coding, testing and operations. Each phase consists of a defined set of activities and deliverables that must be completed before the next phase can begin. Even so, the strict sequence allows for controlled iterations. The use of waterfall requires familiarity with the domain, methods, techniques, tools, as well as having stable requirements and a good understanding of them. This model is often referred to, but it is rarely applied in its strict form [17].
- Iterative enhancement model [27] proposes to implement part of the complete system first, and then, add functionality in a series of iterations. Each iteration is completed following a waterfall model, that is, the requirements for the respective iteration are analyzed, designed, implemented and so on. The result of each iteration is integrated with the system developed so far. Developing software iteratively when the requirements are not completely clear or are still unstable, allows developers to deliver the most important functionalities, and thus, clients can decide their priorities by setting the requirements for the next iteration.
- Spiral model [28] represents a risk-based approach where risk assessment determines the next phase of the project. Spiral model combines aspects of waterfall, iterative enhancement and prototyping model. It allows choosing the best approach for each iteration in order to address the identified risks. A prerequisite to apply the spiral process is specialized knowledge on risk management and assessment, so a poor risk assessment can lead to the selection of wrong alternatives or development approaches [17].

The Unified Process (UP) [30] was proposed in an attempt to take advantage of the best features of life cycle processes and to include some agile development principles. UP is a generic process framework designed as a structure for the methods and tools of the Unified Modeling Language

(UML). This consists of four phases, *Inception*, *Elaboration*, *Construction*, and *Transition*, and is use-case driven, architecture-centric, iterative and incremental.

The name Unified Process is used to describe the generic process that includes the elements common to most of its refinements [31]. The most widely known and documented refinement of UP is the Rational Unified Process (RUP) [32]; others are Open Unified Process (OpenUP) [33], Agile Unified Process (AUP) [34], and Enterprise Unified Process (EUP) [35], among others. RUP provides a flexible and iterative life cycle that must be adapted to the exact characteristics of the project and a set of techniques, templates, job descriptions and tools to guide the work. In addition to UP characteristics, RUP includes three supporting disciplines: *Configuration and change management*, *Project management* and *Environment*. RUP sacrifices some speed and flexibility in the development process so that changes in the system can be explained, agreed and formally specified before implementation [20].

According to Larman [36], UP or RUP (hereinafter RUP) may or may not be considered agile. Although its creators and other defenders say that RUP is an agile software process [26], [37], many authors consider it a heavy and bureaucratic process due to the dependence on initial plans or extensive documentation to generate [11], [19], [23]. For the purposes of this research, it will be treated as a traditional process taking into account its orientation towards the process.

In the 80s, the software industry increasingly began to worry about software quality [13]. However, reports about the successful software projects completion [38] were not very encouraging. All this favored the adoption of models and quality standards under the premise that the quality of the process guarantees product quality [3]. International quality standards usually used to certify organizations started being perceived as an indirect means for guaranteeing that projects would deliver quality products. Some of these models and standards for the improvement and evaluation of processes are ISO 9001, Capability Maturity Model Integration for Development (CMMI) and ISO/IEC 12207.

- ISO 9001:2015 [39] specifies the requirements to create a quality management system in any organization, regardless of its type, size or the products and services it provides. ISO 9001 is sometimes criticized as being a general approach to product quality that is not specifically focused on software development [7].
- CMMI [40] is a collection of good practices to help organizations dedicated to the development of software products and services, in their processes evaluation and improvement. CMMI includes 22 different process areas that cover five different process maturity levels. Addressing and trying to improve performance in all these process areas requires a significant organizational effort and involves many people over a long period of time. This characteristic makes CMMI generally more attractive for large software organizations.
- ISO/IEC 12207:2008 [41] establishes a common framework for software life cycle processes from the collection of the first ideas and concepts to decommissioning obsolete systems. It organizes the activities carried out during the life cycle in seven groups of processes. Each of the 44 processes in these groups describes a purpose and shows a list of expected results, as well as the activities and tasks required. This standard has been explicitly designed to be used as a process reference model within ISO/IEC 15504 (SPICE) [42].

Numerous studies such as [43], [44], [45] suggest that higher process maturity brings higher product quality, the ability to meet deadlines and other successful indicators of organizational performance. However other publications [13], [46], [47] refer to cases in which they did not result in a better organization, but greater bureaucracy, more time than expected and greater effort. In the opinion of Münch et al. [17], people often tend to reject processes perceived as boring or unnecessarily complicated, especially if they demand work that does not seem to directly benefit the worker per se. According to Pressman [16], despite the number of mechanisms that exist to determine process maturity, the quality, opportunity and long-term viability of a product are the best indicators of the effectiveness of the applied process.

B. Agile Philosophy

The Agile philosophy emerges as a reaction to the traditional processes [25], as well as to cope with the new dynamics of the market and the client's changing needs [22]. According to Highsmith [48], the interest for agile methods or ecosystems (as he prefers to call them) appeared at the beginning of the 2000, although their roots go back more than a decade before. In this time some projects were successful using the first versions of some agile methods such as Scrum [49], Dynamic Software Development Methodology (DSDM) [50] and Adaptive Software Development (ASD) [51].

According to Abrahamson et al. [52], "agile denotes the quality of being agile; readiness for motion; nimbleness, activity, dexterity in motion". However, they argue that there is no consensus as to what is exactly agile and what it means in different contexts [1], [53]. In the opinion of Kruchten [10], agility is "the ability of an organization to react to changes in its environment faster than the rate of these changes", beyond the application of a set of practices (Scrum, XP, etc) or properties (The Agile Manifesto). Dingsøyr et al., show in [54] a set of opinions about what is meant by "agile" according to various authors.

The development of agile software is generally incremental, cooperative, straightforward, and adaptive [52]. According to Highsmith and Cockburn [55] the novelty of agile methods is not the practices that they use, but the recognition of people as the main drivers of project success, together with a focus on effectiveness and maneuverability. This vision of the agile world is described through a set of values and the principles in The Agile Manifesto [8].

Several publications [25], [48], [56], [57] refer to the term "agile" as a general way of calling methods such as Scrum, Extreme Programming (XP) [58], Lean Software Development (Lean) [59], Crystal [60], Feature Driven Development (FDD) [61], DSDM, ASD, Kanban [62] and others. Although the individual practices are varied, they can be classified into six general categories [48]:

- Visioning
- Project initiation
- Short, iterative, feature-driven, timeboxed development cycles
- Constant feedback
- Customer involvement
- Technical excellence

All methods address the six key practice areas, but some focus more on collaborative practices and project management (ASD, Scrum and DSDM), while others like XP focus on software development practices. Some of the most wide used in industry are described below.

- Scrum [49] is a process framework used for managing product development and other knowledge work in a volatile environment. It is an empirical approach based on transparency, inspection and adaptation. Scrum is structured in a way that allows for choosing techniques, methods and practices for software development specific for the context of the team or project. It includes frequent management activities with the aim of consistently identifying any deficiency or impediment in the development process, as well as in the practices that are used. Scrum is more appropriate in case where multifunctional teams work in a product development configuration where there is a non-trivial amount of work that can be divided into a set of iterations of 2 to 4 weeks [63].
- XP [58] is the most specific of the agile methods regarding appropriate engineering practices for software development. According to Highsmith [48], "in XP you only do what you need to create value for the customer". XP intends to enable the development of successful software despite vague or constantly changing software requirements. Some of the main features of XP are short iterations with small releases and rapid feedback, close customer participation, constant communication and coordination, continuous refactoring, continuous integration and testing, collective code ownership and pair programming. Due to its specificity, its use is not recommended in some situations[1] such as the development of critical systems for security where the change should be handled very carefully, to give an example.
- The first version of the DSDM method [50] was developed in 1994 and is considered the first truly agile software method. Its creators and defenders call it method or framework, that is the reason why the term process was not used in this case. DSDM covers all aspects of software development from project startup to support and maintenance as well as traditional software processes. The fundamental idea of DSDM is that instead of setting the amount of functionality in a product and then adjusting the time and resources to achieve that functionality, it is preferred to set the time and resources, and adjust then the amount of functionality accordingly.

For a better understanding of these methods, [64] shows a summary table of the practices and the corresponding agile values of six of the agile methods mentioned above. Similarly, in [65], the agile software development practices and the agile principles they support are shown. Also, in [52] each method is evaluated according to software life cycle coverage, project management support, type of guidance, situation appropriateness and the level of empirical support.

Agile development status

According to results of the study "Status Quo Agile 2016/17" [66], in which more than 1000 people participated with a representation of more than 30 countries (mostly European), 85% of the users of Agile methods surveyed agree that the most used method is Scrum and then follow Kanban, Lean and DevOps [67]. More than 50% of the participants started using Scrum in 2014 or later, and 70% declare that they started using Kanban in the last 3 years.

The "11th Annual State of Agile Report" [68] with representation from all continents (mostly North America), confirms that the adoption of Agile continues to grow. The 94% of the respondents claimed that their organizations practice agile, although they also say that more than half of the teams in their organizations still do not practice agile. Scrum and Scrum/XP Hybrid are the agile methodologies most commonly used by organizations, while the use of XP as an independent methodology decreases, even though the practices associated with XP are still frequent. The report states that 71% of organizations surveyed have current or planned DevOps initiatives.

Both surveys agree that the main reasons to work with agile methods are: shorten time to market, improve the quality, reduce project risk, better change management, higher team productivity and better visibility of the project.

The results of [66] lead the authors to conclude that the success rate of agile methods is still higher than that of traditional project management. According to the data of [66], [68] the improvements due to the implementation of agile methods are greater than the effort involved in the implementation itself. Even so, there are challenges to agile scaling such as the disagreement of the organizational culture with agile values and the lack of skills or experience with the methods [68].

IV. Challenges of traditional and agile approaches

There is no unique process or particular approach that offers a set of principles or practices equally suitable for any purpose or be flexible enough to be applicable in any type of project [4]. According to Feiler and Humphrey [15], the

[1]http://wiki.c2.com/?WhenIsXpNotAppropriate

basic requirement of an appropriate software process is that it must fit the needs of the project. Clarke and O'Connor [7] state that the needs of a project are based on the context of the situation in which it should work and therefore, the software process depends on the context.

Research such as [5], [7], [10], establish a series of factors that characterize the software development context and that may significantly affect the adoption of a software process at the level of the organization, project or development team. Some of these factors that should be taken into account are: governance type, business domain, maturity of the organization, level of innovation, culture, size of the system, personnel, team distribution, architectural effort, rate of change, and criticality, among others.

Both, traditional and agile software development, have their strengths and limitations depending on the context, and applying them far from their "sweet spot" is always a challenge.

A. Contexts for traditional approaches

Strengths of discipline

Diebold and Zehler [19] state that counting on an initial plan such as that required in traditional software development is essential for evaluating tenders and negotiating contracts before starting the project. In this way software development supported on concrete and standardized processes favors project transparency, improves management and increases the chances of success. Other reasearch [5], [11], [22] also mention that traditional processes provide better predictability, repeatability, and optimization opportunities.

According to [69], traditional approaches stress the determinant role that design and architecture play in the case of large-scale software and systems. Also, this approach is the most appropriate when most requirements of the final product can be specified from the beginning and it is not needed to involve final users during the project [19], [21]. Projects with high criticality are another context where traditional development is the best approach, since requirements should be specified and analyzed up front [5], [21].

Several research works [5], [19], [21] agree that traditional software development allows personnel to be moved between different projects without much training. Similarly, Špundak [21] suggests that certain characteristics of the development team make this approach the most appropriate: development teams with little experience, when team members are likely to change along the project, or when managers are not in constant contact with the team. He also says that large organizations are better suited for these processes since they generally put a higher stress on control.

Difficulties of traditional processes

Counting on a contract offers a series of advantages but, according to Boehm and Turner [5], this interface between developers and clients poses the highest tension when applying traditional methods. They say that a reaching a precise contract delays the beginning of the project and makes difficult to adapt to changes. On the other hand, an imprecise contract may create false expectations and may build low confidence relationships. In any case, the worst scenario happens when managers prioritize complying with the contract over reaching good project results.

Diebold and Zehler [19] establish that client's participation only at the beginning of the project usually provides limited feedback for later development stages and also problems for dealing with changes.

A study by Petersen and Wohlin [23] about challenges of the waterfall model, indicates that the main reason for failure in this approach is that the actual needs of the clients are not achieved until the end of the project. Besides, the distance with the client causes misunderstandings that may result in modified or forgotten requirements and consequently unnecessary work and rework.

Test coverage is sometimes reduced due to multiple reasons. In general testing is addressed late in the project and, if there are any delays, it is compromised [5], [19], [23]. Moreover, the later testing is performed, the higher the amount of errors found [16]. Rework is then needed and it may be hard to correct some of these errors [5], [23].

Another common criticism to traditional processes comes from the excessive documentation they require and that most of the times is not used [5], [23], [69].

Scientific literature [5], [16], [23] suggests that traditional processes are hard to adapt to small projects. Also, small and changing applications cannot afford the investment in architectural design demanded by traditional processes [5].

Excessively bureaucratic cultures and methods can block software development [5]. According to Diebold and Zehler [19], several criticisms about traditional approaches, and specially that about inflexibility, made apparent the need for alternatives in software processes.

Research about difficulties posed by traditional processes is vast and deep, mainly leaded by agile philosophy and other approaches defenders. However, in Gregory et al.'s opinion [70], reports about the application of agility usually describe success histories of solved problems and hardly ever persistent difficulties, deterioration situations or even complete failure. This is why this research just mentions some general difficulties of traditional approaches and focuses on strengths and weaknesses of agility.

B. Contexts for agile methods

Advantages of agility

Agile software development is considered more flexible and adaptable in contexts where client's needs change frequently [5], [22], [23], [48], as well as for small collocated teams [37], [71]. According to Highsmith [48], the more volatile the requirements are and the more experimental technology is, the higher success possibilities agile methods provide.

Numerous researchers [37], [71], [72], [73], [74] agree that the main advantages of agility relate with the benefits derived from frequent communication and feedback, e.g., better learning, knowledge transfer, among others. Begel and Nagappan [74] emphasize that agile methods promote higher morale among team members. People feel more comfortable, motivated and confident working in agile environments [22], [37], [71], [74].

Clients feel more satisfied when agile methods are applied because they provide the opportunity of receiving early feedback and they can soon answer to changes [37], [71]. Similarly, developers value constant collaboration with clients these methods promote [23], [37], [71], [74]. There also exist the perception that software products exhibit higher quality and teams are more productive [37], [71], [74]. Yet other works highlight other advantages such as project transparency and less time wasted in irrelevant tasks [71], [74].

Difficulties of agile software methods

Despite all the advantages that agile methods provide, the process of adopting the approach or transforming an organization toward agility is a challenge [10]. According to [22], changes not only affect the development process itself, but also they need to take into account a whole series of other factors: communication means, client participation, requirements, change management, management style, people and current processes.

Table I summarizes some of the difficulties of agile methods reported in the literature.

• *Customer involvement*

According to Boehm and Turner [5], the success of agile development depends on having customers representatives who are collaborative, representative, authorized, committed and knowledgeable (CRACK). This is apparent in the experience on the Chrysler Comprehensive Compensation project[2] (C3), that is accepted as the first evidence of the use of XP's practices [57].

Referring this issue, Stephens and Rosenberg [79] agree that C3 is a success case of XP. However, after several pauses and delays, the project was canceled [75]. Some of the problems that lead to project failure were changing the client on site without an opportune transfer [75], and bad communication between the client on site and other stakeholders [20], among others.

Another similar case is presented in [75] where the development of a software product line called DocGen faced several challenges due to the involvement of many different representatives of the client party. There is certain consensus that client on site is one of the most stressful issue for the application of agile methods [5], [23], [37], [73].

• *Professional demands*

According to scientific literature [5], [22], [37], [68], [73], [74], [76], agile methods are very demanding when considering professional skills in order to be successful. Creating an effective agile team is a challenging task [22]. Teams require highly trained managers, high qualifications in all team members, and a steep learning curve. Dybå and Dingsøyr [37] also argue that, if team members do not count on comparable skills, development will not be effective. Similarly, the lack of experience may generate big delays when new practices are implemented [22].

• *Scalability*

Scalability is understood as the breadth of activities for which the process definition is designed. This might include the ranges in number of people, size of product, time duration, product life cycle, or development environment for which the process is fit and precise [15].

Research such as [5], [37], [71], [74] found that it is a challenge to scale agile methods. A study conducted by [22] revealed that scalability is a rather significant limitation and it is still an open issue to be addressed by researchers. According to [19], [22], agility provides limited support for distributed environments, large projects (either long in time, very expensive or highly risky) and large teams. Their application in these contexts can bring difficulties in communication and/or development delays, just to mention some. According to [5], confidence in tacit knowledge is another limit to scale agility.

• *Specific domains*

Several scientific works [5], [22], [71], [72], [74], [77], [78] agree that agile methods have serious limitations in safety-critical domains (e.g., military or health care where the software needs to be of the highest quality possible) and legacy systems mainly because of simple design and the lack of documentation. Also, testing is a bottleneck in agile projects for safety-critical systems since they must be frequent and exhaustive [71]. On the other hand, Boehm and Turner [78] state that using agile methods for legacy systems, either for maintenance or for a new development, brings problems relating refactoring.

• *Architecture focus and technical debt*

Another limitation of agile methods extensively mentioned in the literature [5], [10], [71], [74] is the lack of attention received by design and architecture. These issues bring as a consequence a grow in technical debt.

According to Kruchten et al. [76], technical, quality and maintainability debts are not only caused by pressure on schedules as other researchers argue, but carelessness, lack of education, poor processes, nonsystematic verification of quality, or basic incompetence. Other factors that contribute to technical debt in agile projects are: developing and delivering very rapidly, with no time for proper design or to reflect on the longer term, and lack of rigor or systematic testing.

• *Discarded practices*

A study by Petersen [71] suggests that the lack of professional skills for agile development results in abandoning certain agile practices as time passes. The most frequently abandoned practices are: pair programming, test-driven development, and continuous integration. Others work reporting similar results [37], [71], [74] indicate that pair programming is perceived as an exhausting task whenever partners do no count on the same qualification.

[2]http://wiki.c2.com/?ChryslerComprehensiveCompensation; Octubre 11, 2017

TABLE I

Reported difficulties of agile methods

ID	Difficulty	Reference
AD1	Customer does not keep CRACK	[5], [20], [23], [37], [73], [75]
AD2	Professional skill-specific demands	[5], [22], [37], [68], [73], [74], [76]
AD3	Agile development does not scale well	[5], [19], [22], [23], [37], [68], [71], [74], [77]
AD5	Lack of suitability for specific product domains	[5], [22], [71], [72], [74], [77], [78]
AD6	Architecture does not get enough focus	[5], [10], [71], [74]
AD7	Some practices are discarded after a period of time	[22], [37], [71], [74]
AD8	Increased testing effort	[22], [23], [71], [73]
AD9	Risks in knowledge and project management	[5], [22], [37], [71], [73], [74], [77]
AD10	Increment of technical debt	[76]

• *Testing effort*

A case study conducted by [71] concluded that in several, sometimes in small projects, independent tests are performed only partially or they are omitted completely. This makes that the burden of testing falls in the latest system version, demanding a huge effort [22], [71], [73]. A determinant factor that influences test reduction is the close relation between designers and testers, where designers sometimes press testers so that they focus only in certain parts of the system. "The close relation between testers and designers affects independent testing negatively" [71]. Another problem relating testing is that agility requires a big effort for performing continuous testing because creating an integrated testing environment is difficult for different platforms and system dependencies [23], [71].

• *Knowledge and project management*

Boehm and Turner [5] highlight the risk of completely relying on the tacit knowledge assumed by agile methods, mainly in large teams, and the situations when people abandon the team. Dybå and Dingsøyr [37] alert that in agile teams, members are not so interchangeable as needed and this issue has direct consequences on project management. Provided that the organizational environment impacts significantly in the implementation of agile project management, the organization should be prepared to accept changes this approach requires or otherwise they would face serious difficulties [5]. According to [22], managerial abilities are crucial for the success of agile philosophy. The team leader is responsible for constantly reducing risks, recording team progress, and soon reacting to any difficulty. On the other hand, teams should be able to self-organize instead of being centrally managed [22], [71], [73], [74], [77].

C. *Outside of the sweet spot*

Failure is a natural part of the application of any process [80], and understanding what went wrong is a powerful learning experience that allows us to know what went well too [70]. One of the main causes of failure in software development is the application of processes in contexts that are, at least in some dimensions, far from the contexts for which they were created [10], [47].

In [47], Glass reflects on the causes that made a company try to implement CMMI for three years without being able to surpass level 1. The company develops web applications from scratch, to respond to the specific needs of diverse clients. This scenario suggests an agile development and not a traditional approach such as that implied by CMMI. The author concludes that they made a big mistake trying to apply a process -CMMI in this case- in a context far from its application "sweet spot".

Similar situations are described in [10], where Kruchten explains the failure of projects involving legacy software, safety-critical systems, low requirements change, and nobody playing the role of client to interact with, among others.

According to [10], [26], [56], the contextual factors that more frequently make agile projects to fail are: size, large systems with a lack of architectural focus, software development not driven by customer demand, lack of support from surrounding stakeholders, novice team, very high constraints of some quality attributes (safety-critical system, real-time constraints). Similarly, a study conducted by [81] about agile methods adoption for large-scale development concluded that one of the most determinant challenges is the cultural transformation of the organization, mainly at the middle management level.

V. Conclusions

This study provides an overview of the evolution of the software development approaches. It provides some definitions that are sometimes vague in literature such as what a software process actually is. A summary is provided about weaknesses of traditional approaches that have led to the proposal of agile methodologies. Strengths and weaknesses of traditional and agile processes are surveyed, allowing us to understand the challenges faced when applying one or the other approach outside the sweet spot.

According to Adolph and Kruchten [82], "failure is the dark family secret in our industry". Therefore, there is poor published evidence about persistent difficulties or failure situations in practice that would allow us to learn from errors.

There is no unique way to develop software. The best way depends on specific factors relating the context of the organization, the project and the development team. Therefore, a large number of researchers [5], [11], [12], [20], [21], [72] advocate for a balance between discipline and agility in order to take advantage of the best of both worlds.

References

[1] Marco Kuhrmann, Jürgen Münch, Ita Richardson, Andreas Rausch, and He Zhang (eds.). *Managing Software Process Evolution: Traditional, Agile and Beyond–How to Handle Process Change*. Springer, 2016.

[2] Alfonso Fuggetta. Software process: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 25–34, New York, NY, USA, 2000. ACM.

[3] Watts S. Humphrey, Terry R. Snyder, and Ronald R. Willis. Software process improvement at hughes aircraft. *IEEE software*, 8(4):11–23, 1991.

[4] F Brooks and HJ Kugler. *No silver bullet*. April, 1987.

[5] Barry Boehm and Richard Turner. *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley Professional, 2003.

[6] Winston W Royce. Managing the development of large software systems. In *Proceedings of the 9th international conference on Software Engineering*, pages 328–338. IEEE Computer Society Press, 1970.

[7] Paul Clarke and Rory V O'Connor. The situational factors that affect the software development process: Towards a comprehensive reference framework. *Information and Software Technology*, 54(5):433–447, 2012.

[8] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. Manifesto for agile software development, 2001.

[9] Sanjiv Augustine. *Managing agile projects*. Prentice Hall PTR, 2005.

[10] Philippe Kruchten. Contextualizing agile software development. *Journal of Software: Evolution and Process*, 25(4):351–361, 2013.

[11] Georgios Theocharis, Marco Kuhrmann, Jürgen Münch, and Philipp Diebold. Is water-scrum-fall reality? on the use of agile and traditional development practices. In *International Conference on Product-Focused Software Process Improvement*, pages 149–166. Springer, 2015.

[12] Leo R Vijayasarathy and Charles W Butler. Choice of software development methodologies: Do organizational, project, and team characteristics matter? *IEEE Software*, 33(5):86–94, 2016.

[13] Gianpaolo Cugola and Carlo Ghezzi. Software processes: a retrospective and a path to the future. *Software Process: Improvement and Practice*, 4(3):101–123, 1998.

[14] Per Kroll and Philippe Kruchten. *The rational unified process made easy: a practitioner's guide to the RUP*. Addison-Wesley Professional, 2003.

[15] Peter H Feiler and Watts S Humphrey. Software process development and enactment: Concepts and definitions. In *Software Process, 1993. Continuous Software Process Improvement, Second International Conference on the*, pages 28–40. IEEE, 1993.

[16] R.S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill higher education. McGraw-Hill Education, 2010.

[17] Jürgen Münch, Ove Armbrust, Martin Kowalczyk, and Martin Soto. *Software process definition and management*. Springer Science & Business Media, 2012.

[18] Leon Osterweil. Software processes are software too. In *Proceedings of the 9th international conference on Software Engineering*, pages 2–13. IEEE Computer Society Press, 1987.

[19] Philipp Diebold and Thomas Zehler. *The Right Degree of Agility in Rich Processes*, pages 15–37. Springer International Publishing, Cham, 2016.

[20] Vishnu Vinekar, Craig W. Slinkman, and Sridhar P. Nerur. Can agile and traditional systems development approaches coexist? an ambidextrous view. *IS Management*, 23(3):31–42, 2006.

[21] Mario Špundak. Mixed agile/traditional project management methodology–reality or illusion? *Procedia-Social and Behavioral Sciences*, 119:939–948, 2014.

[22] Adam Solinski and Kai Petersen. Prioritizing agile benefits and limitations in relation to practice usage. *Software quality journal*, 24(2):447–482, 2016.

[23] Kai Petersen and Claes Wohlin. The effect of moving from a plan-driven to an incremental software development approach with agile practices. *Empirical Software Engineering*, 15(6):654–693, 2010.

[24] Hans-Christian Estler, Martin Nordio, Carlo A Furia, Bertrand Meyer, and Johannes Schneider. Agile vs. structured distributed software development: A case study. *Empirical Software Engineering*, 19(5):1197–1224, 2014.

[25] Kevin Aguanno. *Managing agile projects*. Multi-Media Publications Inc., Lakefield, Ontario, Canada, 2005.

[26] Philippe Kruchten. Voyage in the agile memeplex. *Queue*, 5(5):1, 2007.

[27] Victor R Basil and Albert J Turner. Iterative enhancement: A practical technique for software development. *IEEE Transactions on Software Engineering*, (4):390–396, 1975.

[28] Barry W. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61–72, 1988.

[29] Christiane Floyd. A systematic look at prototyping. In *Approaches to prototyping*, pages 1–18. Springer, 1984.

[30] Ivar Jacobson, Grady Booch, James Rumbaugh, James Rumbaugh, and Grady Booch. *The unified software development process*, volume 1. Addison-wesley Reading, 1999.

[31] Kendall Scott. *The unified process explained*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[32] Philippe Kruchten. *The rational unified process: an introduction*. Addison-Wesley Professional, 2004.

[33] Bjorn Gustafsson. Openup–the best of two worlds. *Methods & Tools*, 16(1):21–32, 2008.

[34] Scott Ambler. *Agile modeling: effective practices for extreme programming and the unified process*. John Wiley & Sons, 2002.

[35] Scott Ambler, John Nalbone, and Michael Vizdos. *Enterprise unified process, the: extending the rational unified process*. Prentice Hall Press, 2005.

[36] Craig Larman. *Agile and iterative development: a manager's guide*. Addison-Wesley Professional, 2004.

[37] Tore Dybå and Torgeir Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and software technology*, 50(9):833–859, 2008.

[38] Standish Group et al. The chaos report. *Capturado em: http://www. standishgroup. com*, 1995.

[39] ISO/TC 176/SC 2 Quality systems. Iso 9001:2015 quality management systems – requirements). Technical report, International Organization for Standardization, 2015.

[40] CMMI Product Team. Cmmi for development (cmmi-dev). Technical report, Version 1.3, Technical Report, CMU/SEI-2010-TR-033, Software Engineering Institute, 2010.

[41] ISO/IEC JTC 1/SC 7 Software and systems engineering. Iso/iec 12207:2008 systems and software engineering – software life cycle processes. Technical report, International Organization for Standardization, 2008.

[42] Han Van Loon. *Process Assessment and ISO/IEC 15504: a reference book*, volume 775. Springer Science & Business Media, 2004.

[43] Dennis Goldenson and James D Herbsleb. After the appraisal: A systematic survey of process improvement, its benefits, and factors that influence success. Technical Report CMU/SEI-95-TR-009, CMU, 1995.

[44] B Clark. Quantifying the effects on effort of software process maturity. *IEEE Software Journal*, 17(6):65–70, 2000.

[45] Donald E Harter, Mayuram S Krishnan, and Sandra A Slaughter. Effects of process maturity on quality, cycle time, and effort in software product development. *Management Science*, 46(4):451–466, 2000.

[46] Mark C Paulk. Surviving the quagmire of process models, integrated models, and standards. In *ASQ World Conference on Quality and Improvement Proceedings*, volume 58, page 429. American Society for Quality, 2004.

[47] Robert L Glass. Some heresy regarding software engineering. *IEEE Software*, 21(4):104–103, 2004.

[48] Jim Highsmith. What is agile software development? *CROSSTALK The Journal of Defense Software Engineering*, 15(10):4–9, 2002.

[49] Ken Schwaber. *Agile project management with Scrum*. Microsoft press, 2004.

[50] Jennifer Stapleton. *DSDM, dynamic systems development method: the method in practice*. Cambridge University Press, 1997.

[51] Jim Highsmith. *Adaptive software development: a collaborative approach to managing complex systems*. Addison-Wesley, 2013.

[52] Pekka Abrahamsson, Juhani Warsta, Mikko T Siponen, and Jussi Ronkainen. New directions on agile methods: a comparative analysis. In *Software Engineering, 2003. Proceedings. 25th International Conference on*, pages 244–254. Ieee, 2003.

[53] Pekka Abrahamsson, Kieran Conboy, and Xiaofeng Wang. 'lots done, more to do': the current state of agile systems development research, 2009.

[54] Torgeir Dingsøyr, Sridhar Nerur, VenuGopal Balijepally, and Nils Brede Moe. A decade of agile methodologies: Towards explaining agile software development, 2012.

[55] Jim Highsmith and Alistair Cockburn. Agile software development: The business of innovation. *Computer*, 34(9):120–127, 2001.

[56] Rashina Hoda, Philippe Kruchten, James Noble, and Stuart Marshall. Agility in context. *ACM Sigplan Notices*, 45(10):74–88, 2010.

[57] Alan Moran. *Managing Agile*. Springer, 2016.

[58] Kent Beck. *Extreme programming explained: embrace change*. addison-wesley professional, 2000.

[59] Poppendieck Mary and Poppendieck Tom. Lean software development: an agile toolkit, 2003.

[60] Alistair Cockburn. *Crystal clear: a human-powered methodology for small teams*. Pearson Education, 2004.

[61] Steve R Palmer and Mac Felsing. *A practical guide to feature-driven development*. Pearson Education, 2001.

[62] David J Anderson and Andy Carmichael. *Essential Kanban Condensed*. Blue Hole Press, 2016.

[63] Mike Cohn. *Succeeding with agile: software development using Scrum*. Pearson Education, 2010.

[64] Asif Qumer Gill, Brian Henderson-Sellers, and Mahmood Niazi. Scaling for agility: A reference model for hybrid traditional-agile software development methodologies. *Information Systems Frontiers*, pages 1–27, 2016.

[65] Laurie Williams. Agile software development methodologies and practices. *Advances in Computers*, 80:1–44, 2010.

[66] Ayelt et al. Komus. Study report: Status quo agile 2016/2017. Technical report, Hochschule Koblenz University of Applied Sciences, http://www.status-quo-agile.de/, 2017.

[67] Michael Httermann. *DevOps for developers*. Apress, 2012.

[68] VersionOne. 11th annual state of agile report. Technical report, Technical report, VersionOne, http://stateofagile.versionone.com/, 2016.

[69] Kai Petersen, Claes Wohlin, and Dejan Baca. The waterfall model in large-scale development. In *PROFES*, pages 386–400. Springer, 2009.

[70] Peggy Gregory, Leonor Barroca, Katie Taylor, Dina Salah, and Helen Sharp. Agile challenges in practice: a thematic analysis. In *International Conference on Agile Software Development*, pages 64–80. Springer, 2015.

[71] Kai Petersen and Claes Wohlin. A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case. *Journal of systems and software*, 82(9):1479–1490, 2009.

[72] Marco Kuhrmann, Philipp Diebold, Jürgen Münch, Paolo Tell, Vahid Garousi, Michael Felderer, Kitija Trektere, Fergal McCaffery, Oliver Linssen, Eckhart Hanser, and Christian R. Prause. Hybrid software and system development in practice: Waterfall, scrum, and beyond. In *Proceedings of the 2017 International Conference on Software and System Process*, ICSSP 2017, pages 30–39, New York, NY, USA, 2017. ACM.

[73] Minna Pikkarainen, Jukka Haikara, Outi Salo, Pekka Abrahamsson, and Jari Still. The impact of agile practices on communication in software development. *Empirical Software Engineering*, 13(3):303–337, 2008.

[74] Andrew Begel and Nachiappan Nagappan. Usage and perceptions of agile software development in an industrial context: An exploratory study. In *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, pages 255–264. IEEE, 2007.

[75] Arie Van Deursen. Customer involvement in extreme programming. *ACM SIGSOFT Software Engineering Notes*, 26(6):70, 2001.

[76] Philippe Kruchten, Robert L Nord, and Ipek Ozkaya. Technical debt: From metaphor to theory and practice. *Ieee software*, 29(6):18–21, 2012.

[77] Leo R Vijayasarathy and Dan Turk. Agile software development: A survey of early adopters. *Journal of Information Technology Management*, 19(2):1–8, 2008.

[78] Barry Boehm and Richard Turner. Management challenges to implementing agile processes in traditional development organizations. *IEEE software*, 22(5):30–39, 2005.

[79] Matt Stephens and Doug Rosenberg. *Where Did XP Come From? (Chrysler Knows It Ain't Easy.)*, pages 31–56. Apress, Berkeley, CA, 2003.

[80] Pekka Abrahamsson, Muhammad Ali Babar, and Philippe Kruchten. Agility and architecture: Can they coexist? *IEEE Software*, 27(2), 2010.

[81] Kim Dikert, Maria Paasivaara, and Casper Lassenius. Challenges and success factors for large-scale agile transformations: A systematic literature review. *Journal of Systems and Software*, 119:87–108, 2016.

[82] Steve Adolph and Philippe Kruchten. Summary for scrutinizing agile practices or shoot-out at process corral! In *Companion of the 30th international conference on Software engineering*, pages 1031–1032. ACM, 2008.