The Waterfall Model and the Agile Methodologies : A comparison by project characteristics

Wilfred van Casteren

Academic Competences in the Bachelor 2

assignment: Write a scientific article on 2 Software Development Models

March 7, 2017

***Abstract*—Software development projects have long been plan-driven processes, but the coming of age of Agile Methodologies has given rise to a more adaptive approach to software/system development. The purpose of this paper is to give a short presentation of two Software Development Models (SDMs), The Waterfall Model and the Agile Methodologies (AM), and offer for both SDMs issues and typical project characteristics.**

Keywords: Software/System Development Model (SDM), Waterfall Methodology, Agile Methodologies, plan-driven approach, adaptive approach, comparison of SDMs, software engineering history, archetypal projects, project characteristics

I. Introduction

Developing complex software systems has been documented and described in depth for the first time by Royce [14] in 1970. Besides a general idea on software development, several software development concepts, still in practice today, were introduced in this writing. This paper will elaborate on this general idea on software development in the form of the Waterfall Model, and on its issues. It will also introduce the Agile Methodologies as a natural counterpart for the Waterfall Model, and the issues related to this family of SDMs. Generally spoken, the Waterfall Model adheres to a plan-driven approach while Agility pursues an adaptive approach. This paper will offer an limited insight in the aspects determining a project's suitability for an Agile or a Waterfall approach.

In Section II, a short explanation for the Waterfall Model as an SDM, a history of its inception and some related issues will be given. Section III on the Agile Methodologies will offer an overview of Agile characteristics, practices and issues. Section IV will be a starting point for deciding which development methodology best to use for a particular project. This section mentions some typical Agile projects and one that is typical Waterfall and offers a limited insight in typical characteristics for plan-driven and Agile projects. Section V adds references to articles on varying Agile and Waterfall methodologies, to a quantifying article on the popularity of SDMs, to an article on deciding which software methodology to use and finally to an article on Agile processes in a distributed environment. Section VI outlines the merits of this paper, by resuming the projects eligible for an Agile or Waterfall approach and issues to be foreseen for both approaches.

II. The Waterfall Model: An introduction to the history of software engineering

A. Software engineering: a short history

According to Ganis [7], the term software engineering can be traced back to a set of historical conferences hosted by NATO in the late 1960's. At the time, the term software engineering was not in general use, nor widely accepted [7]. Complex software projects were troublesome and costly to complete and the prevailing assumption at the time was that it would be beneficial to consider software development as engineering [7]. These NATO conferences played a major role in gaining acceptance for the term software engineering and had a profound effect on how programmers produced code [7]. As programmers had their roots in the engineering discipline and as the cost of computing was high, they practiced the hardware concepts as 'measure twice, cut once' [7]. This cautious nature caused the development of methodologies that enabled project teams to slowly and methodically create their plans for the creation of software systems [7].

B. The Waterfall Model

At this juncture disclosed by Ganis [7], Dr. Winston Royce wrote an article on managing large and complex software developments (see Royce [14]). In this article, based on experiences in developing software for planning spacecraft missions, commanding and post-flight analysis, W. Royce describes the fundamentals of software/system development. Part of these fundamentals are still applicable today. In an intermediate

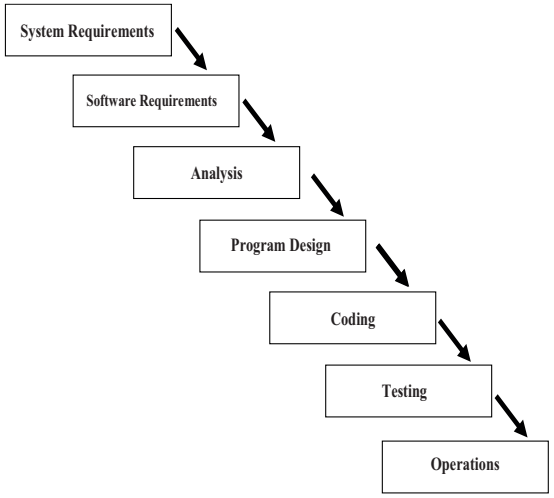

Fig. 1: The Waterfall Model

phase of his elaboration Royce presents a sequence of phases which form a software development sequence, illustrated by Figure 1. Bell and Thayer [1] will later refer to this sequence as the concept of the "waterfall" of development activities and will be the originators of the designation for the first SDM, the

Waterfall Model. In Figure 1 the analysis phase and the coding

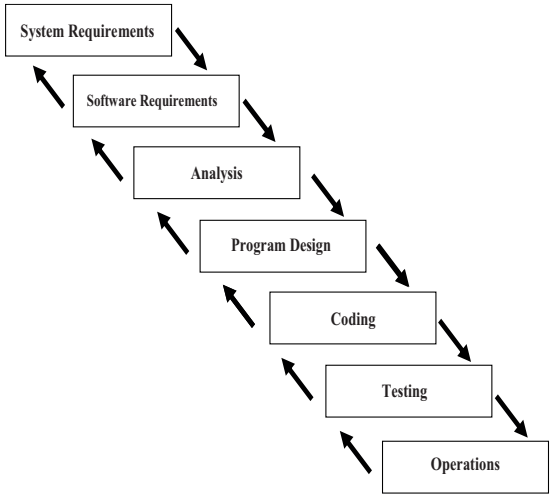

Fig. 2: The Waterfall Model: iterative relationship between successive phases

phase deliver the most direct value to the final product, the other phases are to be staffed and planned differently for the best utilization of program resources [14]. Concerning this Waterfall Model, Unhelkar [17] emphasizes the sequential dependability on the previous deliverable. A dependability which for example holds back system design when the analysis model is still to be signed off, and holds back coding if the design is still to be signed off.

The next step in Royce's disquisition covered the iterative relationship between successive development phases. Figure 2 depicts this step in his elaboration. Royce [14] believes that as each development step progresses, and the design is further detailed, there is an iteration with the preceding and succeeding steps, but rarely with the more remote steps. Royce [14] sees virtue in such a scoped down change process. As at any point in the design process after the requirements analysis is completed there exists a firm and closeup moving baseline to which to return in the event of unforeseen design difficulties [14].

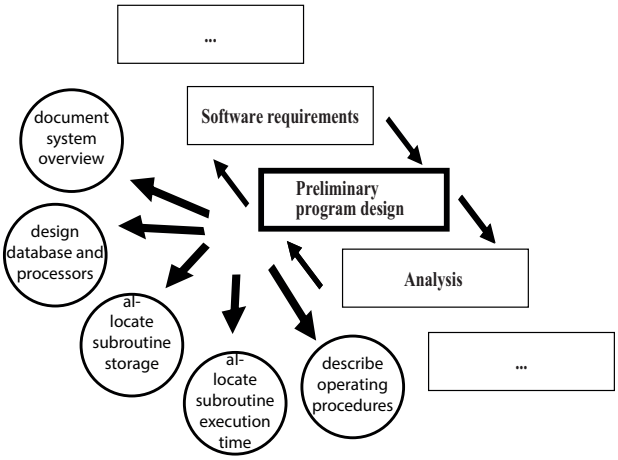

Fig. 3: The Waterfall Model: an extra preliminary program design phase

In a following step Royce [14] adds the preliminary program design phase between the software requirements phase and analysis. This phase falls apart into five components, see Figure 3. By this technique the program designer assures that the software will not fail because of storage, timing, and data flux reasons. This phase illustrates the era in which this software development sequence was conceived, and also offers an insight in the complexity of problems which the engineers had to tackle. To conclude, by the end of his elaboration, Royce's vision consists of documentation for every development phase, a shorter pilot phase comprising all phases, customer involvement both during and after the program design phase and after the testing phase.

C. Waterfall Issues

Ganis [7] has the opinion the Waterfall Model predominately emphasizes on the freezing of requirement specifications or the high-level design very early in the development life-cycle, prior to engaging in more thorough design and implementation work. So the Waterfall model is likely to be unsuitable if requirements are not well understood/defined or are likely to change in the course of the project. Petersen et al. [12] associate the Waterfall Model with high costs and efforts. They feel confirmed in their belief by the number of documents to be approved in every phase, by the difficulty to make changes, by the difficulty iterations take to initiate and achieve and by problems that arise only in later phases. Consequences of these issues have been that the customers' current needs are not addressed, resulting in implemented but unused features. To conclude his section, Munassar and Govardhan [9] consider the Waterfall Model (1) as a baseline for many -more contemporary- Software Development Life-cycle Models, while Dr. Winston Royce can be seen as the founding father of plan-driven software development methodologies.

III. Agile Methodologies: Motivation, principles and practice

According to http://www.thefreedictionary.com/agile agile means ready for motion, nimble, quick in movement.

The Agile Manifesto by Fowler and Highsmith [6], officially announced the foundation of modern Agility. The founders and subscribers of Agility find motivation in uncovering better ways of developing software by doing it and helping others do it. Though *The Agile Manifesto*'s principal concern is developing valuable software, Unhelkar [17, chap. 1] emphasizes the vital, strategic role Agility plays across all dimensions of business. The close intertwining of almost all business functions with corresponding underlying software systems causes Agile businesses to require corresponding Agile software systems [17, chap. 1].

A. Agile Principles and practices

According to Cohen et al. [5], Agile techniques vary in practices and emphasis, they share common characteristics, including iterative development and a focus on interaction, communication, and the reduction of resource-intensive intermediate artifacts. Furthermore, Agile approaches combine short iterative cycles with feature planning and dynamic prioritization. state Highsmith and Cockburn [8]. Agility requires face-to-face communication, which

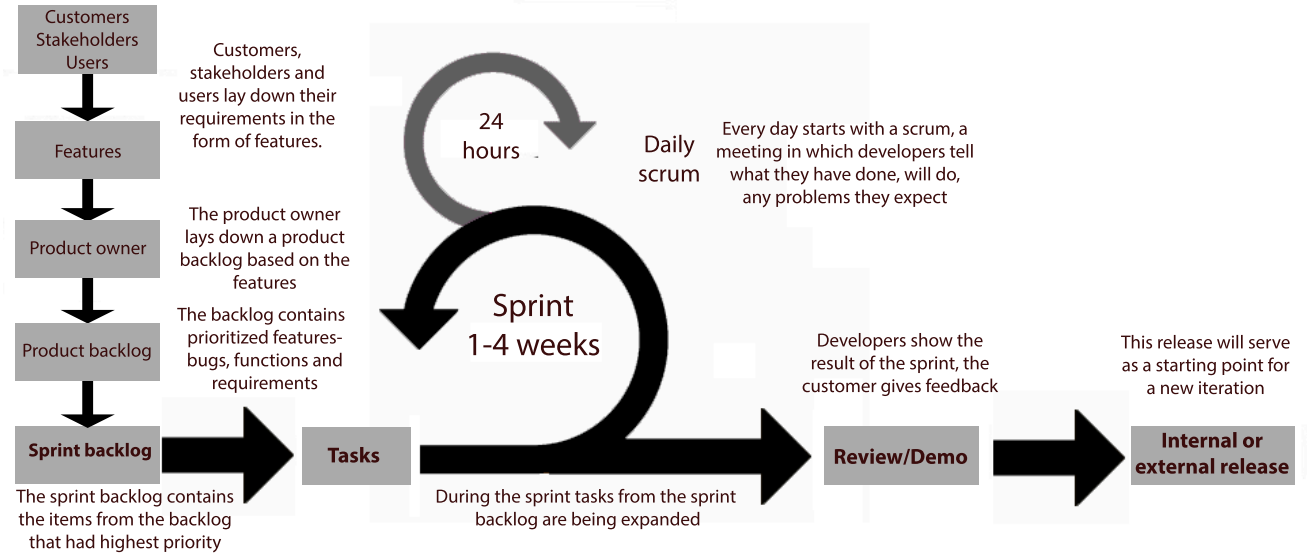

Fig. 4: Scrum: a schematic overview

in turn implies working in close location, facilitating teams to make decisions and act on them immediately rather than wait on correspondence [5]. Agile development methods requires close customer partnerships, state Highsmith and Cockburn [8]. For customers to assess an increment and give feedback, they need a visible result like a user interface or in case of an embedded system a software simulation [16].

B. Agile Iterations

Williams [20] explains that each iteration in an iterative product, is a self-contained, mini-project with activities that span requirements analysis, design, implementation, and test. Williams [20] sets forth that each iteration leads to an iteration release, which may be only an internal release, that integrates all software across the team and is a growing and evolving subset of the final system. The customer adaptively specifies his or her requirements for the next release based on the observation of the current release, rather than speculating at the start of the project [20]. Evidence exists of frequent deadlines reducing the variance of a software process and, thus, possibly increasing its predictability and efficiency [20]. The predetermined iteration length used when developing Agile serves as a time-box for the team [20]. Thus scope is chosen for each iteration to fill the iteration length. So, rather than to increase the iteration length to fit the chosen scope, the scope is reduced to fit the iteration length Williams [20].

The key difference between Agile methods and past iterative methods is the length of each iteration [20]. In the past, iterations might have been three or six months long [20]. With Agility, iteration lengths vary between one to four weeks, and intentionally do not exceed 30 days [20]. Moreover, research has shown that shorter iterations imply lower complexity and risk, better feedback, and higher productivity and success rates [20].

C. Agile: facilitating tools and techniques

Given the code-centric focus of Agility, the use of several coding tools can not be seen as a surprise. Model Driven Architecture, where models are the central artifacts, speed up development through automated generation of significant portions of application and middle-ware code and by raising the level of abstraction at which developers work, is such a tool, according to Turk et al. [16]. Furthermore, integrated development environments, systematic improvement of code through refactoring and automated test suites can help manage the cost of detecting and removing errors even late in an Agile process [16]. Version control and change tracking, very important in a iterative environment, can now be found in the toolbox of any active programmer, add Beznosov and Kruchten [2]. Other techniques adapted for Agility are fault injection and automatic test generation techniques, which can be integrated into dynamic security testing tools [2].

D. An Agile Methodology example: Scrum

In 1995 Schwaber [15] presented Scrum. Nowadays it is a widely accepted Agile Methodology. So it makes a good candidate to be described in some detail.

On their website Rawsthorne and Shimp [13] claim the goal of a Scrum team (fewer than 10 people) is to produce and release results that meet the goals and priorities that have been set down by the product owner in the product backlog (see Figure 4), typically as a result of working with stakeholders. Normally, they add, before the Scrum team can start producing results, there is a visioning phase (this could also be the first phase in a project), during which the business owner, product owner, and the stakeholders produce a product vision and a product road map [13]. The vision provides the overall focus for the project, while the road map gives guidance about releases and their goals [13].

Paetsch et al. [10] state that the main Scrum techniques are the product backlog, sprints, and daily scrums (a scrum is daily team meeting of 15 minutes). With regard to requirements engineering, the product backlog plays a special role in Scrum as all requirements regarded as necessary or useful for the product are listed in the product backlog [10]. Furthermore it contains a prioritized list of all features, functions, enhancements and bugs [10]. Paetsch et al. [10] claim the product backlog to be

an incomplete and changing requirements document containing information needed for development. The sprint (see Figure 4) is a 30 day development iteration. Paetsch et al. [10] describe the sprint as follows, for each sprint, the highest priority tasks from the backlog are moved to the sprint backlog (see Figure 4). The requirements to be fulfilled during a sprint are not to be changed after a sprint has started [10]. At the end of a sprint, a sprint review meeting is held that demonstrates the new functionality to the customer and solicits feedback [10]. Between two sprints the customer has absolute flexibility re-prioritizing the requirements for the next sprint [10].

E. Agile Issues

Concerning architectural design, which is not a key value in Agility, flaws that seriously compromise the integrity of the design, or errors that require corrective actions that have a wide impact, are more costly to correct when detected late in the development process, Turk et al. [16] affirm. Critics have argued that the focus on tacit knowledge makes projects that use agile processes dependent on experts [16]. It seems that the informal evaluation techniques of agile processes may not be sufficient for establishing the quality of safety-critical systems [16]. Unhelkar [17, chap. 3] finds it difficult to see frequent releases in short development cycles, pair programming, starting development before completing requirements and an ever-evolving project direction applicable to large-scale software projects, The coordination of a higher number of projects and tests on system levels, the complex decision processes, the creation of baselines for a high number of internal releases and the management overhead all form obstacles for large-scale, multi-team Agile development, Petersen and Wohlin [11] add.

IV. Agile Methodologies and the Waterfall Model : A comparison by project characteristics

Unhelkar [17, chap. 2] describes the traditional waterfall life cycle as a linear, phased approach to software development, while he sees an Agile approach which seems to be diametrically opposed to this traditional development life cycle. The following sections will shed some light on deciding how projects should be developed, Agile or Waterfall.

A. Archetypal projects

According to Unhelkar [17], projects that benefit the most from an Agile approach are "greenfield" development projects. A greenfield project being a project that aims to develop a system for a totally new environment. Furthermore, Agile approaches to software development are most conducive in these projects wherein coding is at the center of activities [17]. Typically, a small project comprising five programmers and lasting for a period of about 6 months would be ideally placed to make extensive use of Agile principles and practices, [17]. An experimental or pilot project in the small category is also likely to benefit with Agility. Such a project can be used to experiment with Agile itself, or may be a part of a new, major development [17]. According to Petersen et al. [12] the Waterfall Model is predictable and pays attention to planning the architecture and structure of the software system in detail which is especially important when dealing with large systems.

B. Agile and plan-driven project characteristics

How about other project types, which project characteristics will make a difference when it comes to choosing the right development methodology?

1) Primary goals: Boehm [3] states a major set of objectives for the more traditional plan-driven methods, has been predictability, repeatability, and optimization. This while Agility focuses on rapid value and responding to change [6].

2) Size: Plan-driven methods scale better to large projects as opposed to Agile projects. A bureaucratic, plan-driven organization that requires an average of a person-month just to get a project authorized and started won't be very efficient on small projects though [3].

3) Customer relations: Agile methods work best when customers operate in dedicated mode with the development team, and when their tacit knowledge is sufficient for the full span of the application [3]. This method risks tacit knowledge shortfalls, which the plan-driven methods reduce via documentation and the use of architecture review boards and independent expert project reviews to compensate for on-site customer shortfalls [3].

4) Planning and control: According to Unhelkar [17] formal project management plays an important role in successful completion of a software project. Project management requires careful planning, estimation, coordination, tracking, and control. Aspects formally covered in The Waterfall Model [17]. Agility places more value on the planning process than on the resulting documentation [3].

5) Communications: Agility advocates face-to-face communication, the Waterfall Model requires explicit documented knowledge.

6) Requirements: Formal and solution-independent, up-front requirements engineering is not directly used by pure Agile practitioners [17]. The heavyweight, formal Waterfall Model will encounter problems keeping up with rapidly changing requirements [3]. On the other hand, if the architecture anticipates and accommodates requirements changes, plan-driven methods can keep even million-line applications within budget and schedule [3].

7) Development: Agility values working software over comprehensive documentation, and emphasizes simplicity, maximizing the amount of work not done [6]. This while the Waterfall Model relies heavily upon software architecture, as it is part of the development sequence, see [14].

8) Test: According to Beznosov and Kruchten [2], conventional assurance methods involve design and architectural principles, dynamic testing, static analysis and internal and third-party reviews, evaluation, and vulnerability testing. These methods are much more adapted to Waterfall development which are well-documented and focused on architecture [2]. Agile Methodologies, on the other hand, facilitate internal design and code review, and motivate developers to adopt coding standards [2], but lack the focus on architecture or documentation.

9) Customers: Agility requires dedicated, co-located, knowledgeable customers [3]. The Waterfall Model requires, adequately skilled knowledgeable customers [3].

10) Developers: Developers in an Agile project are to be agile, knowledgeable, collocated, and collaborative and should be amicable, talented, skillful and communicative [3]. Agile approaches

TABLE I: Agile and plan-driven home grounds.

Project characteristics	Agile home ground	Plan-driven home ground
Application		
Primary goals	Rapid value, responding to change	Predictability, stability, high assurance
Size	Smaller teams and projects	Larger teams and projects
Management		
Customer relations	Dedicated onsite customers, focused on prioritized increments	As-needed customer interactions, focused on contract provisions
Planning and control	Internalized plans, qualitative control	Documented plans, quantitative control
Communications	Tacit interpersonal knowledge	Explicit documented knowledge
Technical		
Requirements	Prioritized informal stories and test cases, undergoing unforeseeable change	Formalized project, capability, interface, quality, foreseeable evolution requirements
Development	Simple design, short increments, refactoring assumed inexpensive	Extensive design, longer increments, refactoring assumed expensive
Test	Executable test cases define requirements, testing	Documented test plans and procedures
Personnel		
Customers	Dedicated, colocated Crack[1] performers	Crack[2] performers, not always colocated
Developers	At least 30% full-time Level 2 and 3 experts; no Level 1B or Level –1 personnel [3]	50% Level 3s early; 10% throughout; 30% Level 1B's workable; no Level -1s [4]
Culture	Comfort and empowerment via many degrees of freedom (thriving on chaos)	Comfort and empowerment via framework of policies and procedures (thriving on order)

[1 and 2] Collaborative, Representative, Authorized, Committed, and Knowledgeable.
[3 and 4] See Table II 'Staff software skill level'.

emphasize cross-functional teams of developers, testers, subject matter experts, and architects [17]. Waterfall developers are to be plan-oriented, adequately skilled with access to external knowledge [3].

11) Culture: Agile methods will succeed better in a culture that "thrives on chaos" than in one that "thrives on order," while the opposite is true for the Waterfall Model, this according to Boehm and Turner [4].

Table I, drawn from an article, by Boehm and Turner [4], is a summary of the aspects above. It offers an overview of typical characteristics in terms of project goals and size, project management style, project tools usage, personnel and culture, customers for both Agile projects and plan-driven projects. This table refers to Table II when it comes to defining staff developers' skill levels.

TABLE II: Staff software skill level

Level	Charasteristics
3	Able to revise a method, breaking its rules to fit an unprecedented new situation.
2	Able to tailor a method to fit a precedented new situation.
1A	With training, able to perform discretionary method steps such as sizing stories to fit increments, composing patterns, compound refactoring, or complex COTS integration. With experience, can become Level 2.
1B	With training, able to perform procedural method steps such as coding a simple method, simple refactoring, following coding standards and CM procedures, or running tests. With experience, can master some Level 1A skills.
-1	May have technical skills, but unable or unwilling to collaborate or follow shared methods.

V. Related work

In her surveys Williams [19, 20] introduces some prominent plan-driven Software Development Methodologies and offers a detailed insight in the different flavors of Agile development. Refer to West et al. [18] to get ideas on which Agile or plan-driven methodologies are popular, how methodologies are perceived by developers, how methods could be combined. A framework for deciding whether a project should be developed Agile or in a Waterfall Model or through a model somewhere in between is presented by Boehm and Turner [4] and draws the lines along which to think to balance Agile and plan-driven methods. An article on adapting Agile processes to a distributed environment has been published by Young and Terashima [21].

VI. Conclusion

In this paper two SDMs were introduced, a plan-driven Waterfall Model and the adaptive Agile Methodologies. Both models have their uses, advantages and disadvantages. Small projects are almost always suitable for an Agile approach and almost never for a Waterfall approach. Both the Waterfall Model and the Agile Methodologies have their issues when tackling medium-sized projects. The demanding Waterfall Model could add too much overhead to a fairly easy project, while a flexible Agile Methodology could be too easy going for the same project. A project type that is hard to be tackled in an Agile way is a big and complex project with multiple teams working simultaneously on different parts of the application. This kind of project is almost always a Waterfall project. For projects hard to classify as being either Waterfall or Agile other SDMs need to be investigated.

References

[1] Thomas E Bell and Thomas A Thayer. Software requirements: Are they really a problem? In *Proceedings of the 2nd international conference on Software engineering*, pages 61–68. IEEE Computer Society Press, 1976.

[2] Konstantin Beznosov and Philippe Kruchten. Towards agile security assurance. In *Proceedings of the 2004 workshop on New security paradigms*, pages 47–54. ACM, 2004.

[3] Barry Boehm. Get ready for agile methods, with care. *Computer*, 35(1):64–69, 2002.

[4] Barry Boehm and Richard Turner. Using risk to balance agile and plan-driven methods. *IEE Computer Science*, 2003.

[5] David Cohen, Mikael Lindvall, and Patricia Costa. Agile software development. *DACS SOAR Report*, 11, 2003.

[6] Martin Fowler and Jim Highsmith. The agile manifesto. *Software Development*, 9(8):28–35, 2001.

[7] Matt Ganis. Agile methods: Fact or fiction, 2010.

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

[9] Nabil Mohammed Ali Munassar and A Govardhan. A comparison between five models of software engineering. *IJCSI*, 5:95–101, 2010.

[10] Frauke Paetsch, Armin Eberlein, and Frank Maurer. Requirements engineering and agile software development. In *WETICE*, volume 3, page 308, 2003.

[11] 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.

[12] Kai Petersen, Claes Wohlin, and Dejan Baca. The waterfall model in large-scale development. In *International Conference on Product-Focused Software Process Improvement*, pages 386–400. Springer, 2009.

[13] Dan Rawsthorne and Doug Shimp. Scrum in a nutshell. https://www.scrumalliance.org/community/articles/2009/december/scrum-in-a-nutshell, 2009.

[14] Winston W Royce. Managing the development of large software systems. In *proceedings of IEEE WESCON*, number 8, pages 328–338. Los Angeles, 1970.

[15] Ken Schwaber. Scrum development process, oopsla'95 workshop on business object design and implementation. *Austin, USA*, 1995.

[16] Dan Turk, Robert France, and Bernhard Rumpe. Assumptions underlying agile software development processes. *arXiv preprint arXiv:1409.6610*, 2014.

[17] Bhuvan Unhelkar. *The Art of Agile Practice: A Composite Approach for Projects and Organizations*. CRC Press, 2016.

[18] Dave West, Tom Grant, M Gerush, and D D'silva. Agile development: Mainstream adoption has changed agility. *Forrester Research*, 2(1):41, 2010.

[19] Laurie Williams. A survey of plan-driven development methodologies, 2004.

[20] Laurie Williams. A survey of agile development methodologies, 2007.

[21] Cynick Young and Hiroki Terashima. How did we adapt agile processes to our distributed development? In *Agile, 2008. AGILE'08. Conference*, pages 304–309. IEEE, 2008.