OULU UNIVERSITY OF
APPLIED SCIENCES

Janne Kauppi

EXPERIENCES OF NEW APPROACHES TO UI SOFTWARE DEVELOPMENT: COMPARING THE WATERFALL AND AGILE PROCESSES

EXPERIENCES OF NEW APPROACHES TO UI SOFTWARE DEVELOPMENT: COMPARING THE WATERFALL AND AGILE PROCESSES

Janne Kauppi
Bachelor's thesis
Spring 2013
Degree Programme in Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology

Author: Janne Kauppi
Title of Bachelor's thesis: Experiences of New Approaches to UI Software Development: Comparing the Waterfall and Agile Processes
Supervisor: Lauri Pirttiaho
Term and year of completion: Spring 2013 Number of pages: 30

New, light-weight agile software development methods are gaining ground over the more traditional, heavy-weight sequential software development methods. The objective of this Bachelor's thesis was to amend personal experiences of using both a waterfall-like sequential model and an agile Scrum model in a UI software development environment and to compare them with each other. The work was done for School of Engineering at Oulu University of Applied Sciences.

This thesis is based on literature and personal experience of the methodologies in question. My own experiences were analysed and compared to literature, and based on this comparison the results were collected for this work. The practical experiences of software development environment were emphasised in the comparison.

As the result of this work it can be stated that the sequential waterfall still has its uses, but it would make sense for the software development industry to switch using agile development methods due to their versatility.

Keywords: Software development, Agile, Scrum, Waterfall, Software development processes

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Degree Programme in Information Technology

Tekijä: Janne Kauppi
Opinnäytetyön nimi: Experiences of New Approaches to UI Software Development: Comparing the Waterfall and Agile Processes
Työn ohjaaja: Lauri Pirttiaho
Työn valmistumislukukausi ja -vuosi: Kevät 2013 Sivumäärä: 30

Uudet, ketterät ohjelmistokehitysmenetelmät valtaavat alaa vanhemmilta vaiheisiin perustuvilta ohjelmistokehitysmenetelmiltä. Tämän opinnäytetyön tavoite oli käyttää omia henkilökohtaisia kokemuksia vaiheellisista ja ketteristä ohjelmistokehitysmenetelmistä käyttöliittymäohjelmointiympäristössä hyödyksi ja verrata menetelmiä toisiinsa. Työ tehtiin Oulun seudun ammattikorkeakoulun Tekniikan yksikölle.

Tämä opinnäytetyö pohjautuu kirjallisuuteen ja omiin kokemuksiin molemmista ohjelmistokehitysmenetelmistä. Omia kokemuksia analysoitiin ja verrattiin kirjallisuuteen, ja vertailun perusteella koottiin tulokset tähän työhön. Vertailussa painotettiin käytännön kokemuksia ohjelmistokehitysympäristössä.

Työn tuloksena voidaan todeta, että vesiputousmallille on vielä käyttökohteensa, mutta ohjelmistokehitysalan on järkevä siirtyä käyttämään ketteriä ohjelmistokehitysmenetelmiä niiden monipuolisuuden vuoksi.

Asiasanat: Ohjelmistokehitys, ketterät menetelmät, Scrum, vesiputousmalli, ohjelmistokehitysprosessit

CONTENTS

1 INTRODUCTION

Heavyweight waterfall methodologies of developing software are losing ground in SW development domain to newer lightweight agile and lean software developing methodologies: Scrum, Kanban and XP to name a few. Agile methodologies are better suited to many software development projects, as they are not as constraint but more versatile than waterfall methodologies.

This thesis will focus on Scrum, comparing it to the traditional sequential waterfall model. Both, Scrum and waterfall, will be reflected based on experiences and literature.

2 SOFTWARE DEVELOPMENT MODELS

There are many different software development processes available for software practitioners to choose from, of which Waterfall and Scrum are discussed in more detail in this paper.

2.1 Waterfall

After the software industry's inception in the 1950s and 1960s, the industry advanced quickly. With that, a need to better predict and control ever larger software projects led somehow to sequential waterfall model. (Leffingwell 2011, 5.)

The waterfall model is a sequential design process, often used in software development processes, in which the progress flows steadily downwards (like a waterfall) through the defined phases (Wikipedia 2013, date of retrieval 8.5.2013). Requirements are agreed upon, design is created and the code follows. Then the software is tested to verify its conformance to requirements and design. (Leffingwell 2011, 5.)

Herbert D. Benington was the first to describe a sequential software development model (FIGURE 1) in 1956. The model consisted of nine phases that were used in preparing a large system program. These nine phases were Operational plan, Machine / Operational specifications, Program Specifications, Coding Specifications, Coding, Parameter Testing, Assembly testing, Shakedown and System evaluation. (Benington 1956.)

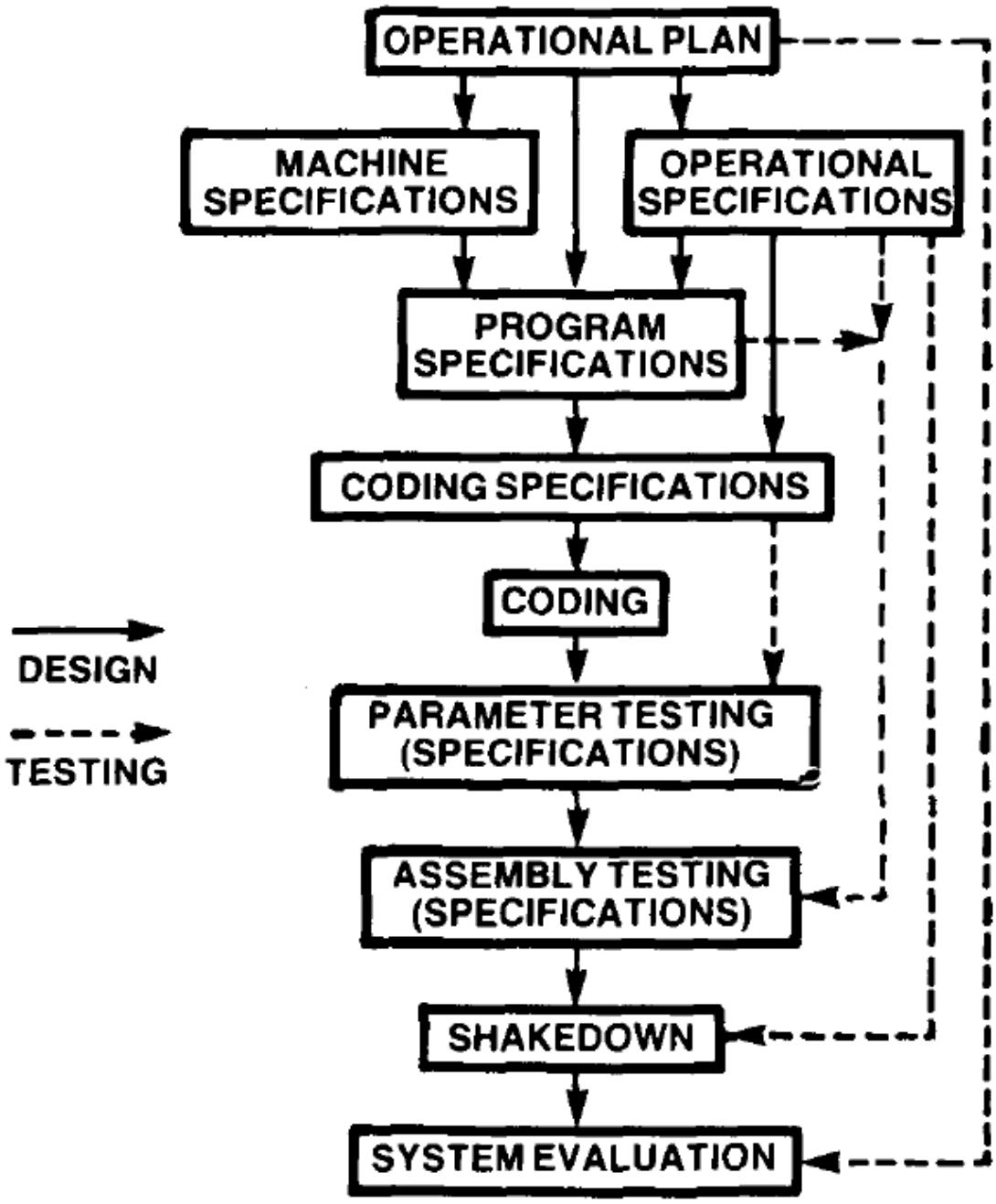

FIGURE 1. Benington's sequential model (1956)

Winston Royce simplified Benington's sequential model in 1970 (FIGURE 2) down to seven steps, where one step would be completed before the next would begin. (Royce 1970.)

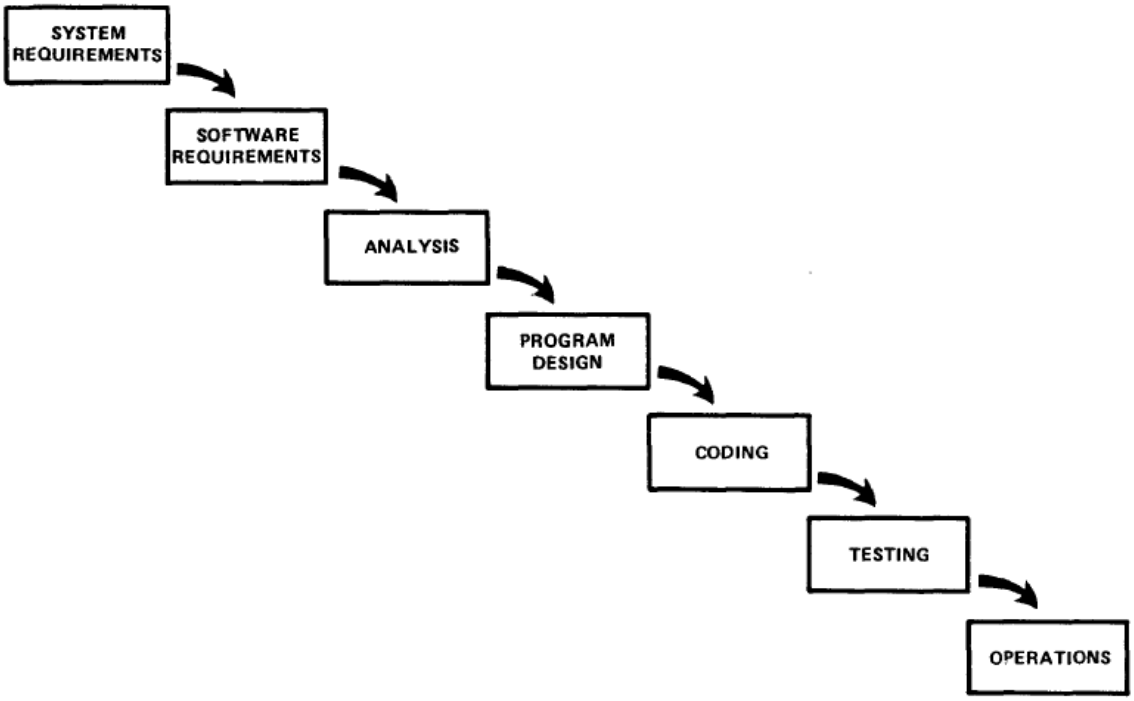

FIGURE 2. Waterfall model (Royce 1970)

However, he saw that this model was flawed by design, as the steps are not really separate – they have interactions with each other. If not before, in the testing step at the latest. Therefore, he iterated the waterfall model in the same paper, and introduced iterative model (FIGURE 3) amended with five additional features needed to eliminate most of the development risk, which he believed to be a working software development model. (Royce 1970.)

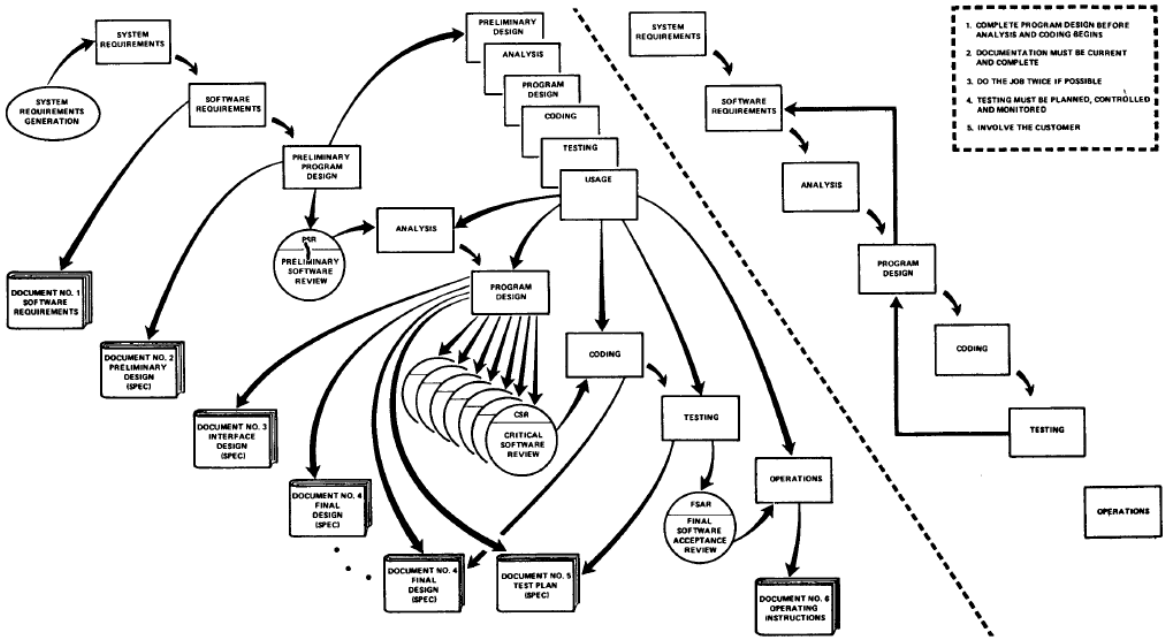

FIGURE 3. Iterative model with additional features (Royce 1970)

Unfortunately, the improved iterative model was outshined by the simple sequential waterfall model, which against Royce's intention, started to gain ground in the industry. As to why this happened, we might never know.

However, in practical software development environment, work is always more or less iterative, following thus the iterative model, actually.

2.2 Scrum

Scrum is one of the agile methodologies currently in use and it follows the Agile Manifesto:

> We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
>
> **Individuals and interactions** over processes and tools
> **Working software** over comprehensive documentation
> **Customer collaboration** over contract negotiation
> **Responding to change** over following a plan
>
> That is, while there is value in the items on the right, we value the items on the left more. (Agile Manifesto, date of retrieval 19.5.2013.)

Scrum Alliance defines Scrum as follows:

> Scrum is an agile framework for completing complex projects. Scrum originally was formalised for software development projects, but works well for any complex, innovative scope of work. (Scrum Alliance 2013, date of retrieval 8.5.2013.)

Scrum, therefore, is not a process or a technique, but a framework within which various processes and techniques can be employed, in order to complete a project or a product. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

The following figure (FIGURE 4) depicts the overview of Scrum framework.

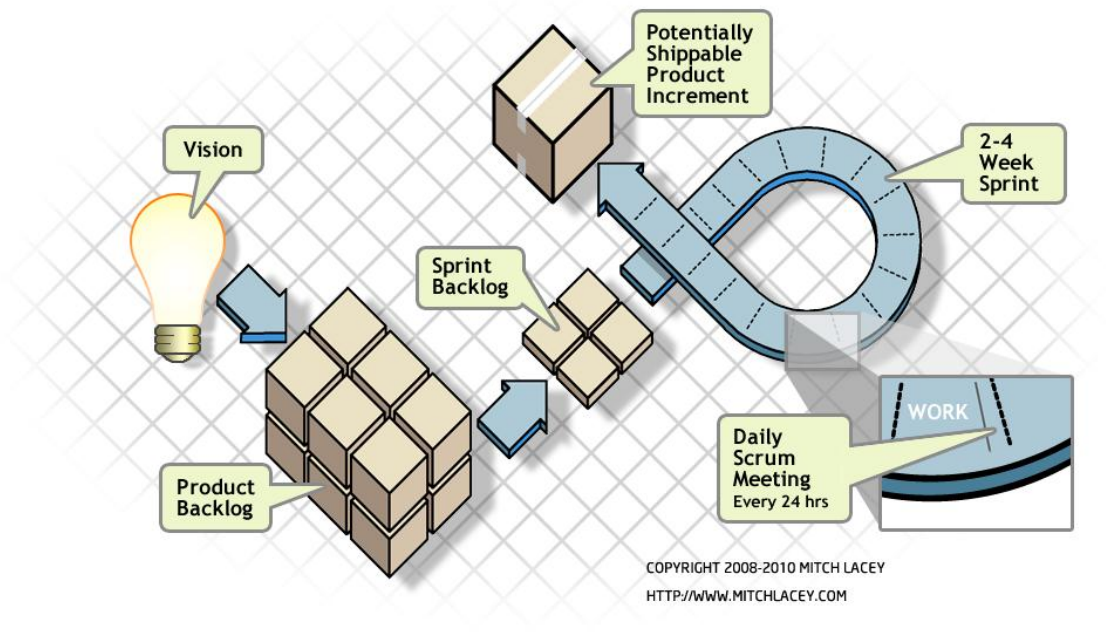

FIGURE 4. Scrum Framework Flow (Lacey 2013)

2.2.1 Scrum Theory

Schwaber states that:

> Scrum is based on an empirical process control model rather than the traditional defined process control model (Schwaber & Beedle 2002, 89).
>
> When activities are so complicated and complex that they can't be defined in advance and aren't repeatable, they require the empirical process model (Schwaber & Beedle 2002, 100).

Also in a later publication, Schwaber & Sutherland state:

> Scrum is founded on empirical process control theory, or empiricism. Empiricism asserts that knowledge comes from experience and making decisions based on what is known. Scrum employs an iterative, incremental approach to optimise predictability and control risk. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

So, Scrum is an empirical framework for software development. In addition, theory-wise, there are three very important aspects in empirical process control: transparency, inspection and adaptation. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

Transparency requires establishing a common standard for the process and the outcome, so that all relevant parties share a common understanding of what is being done. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

Inspections need be frequently held to Scrum artifacts and progress to detect undesirable functionality or direction, however these inspections must not get in the way of actual work. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

Adaptation must be done as soon as possible, if an inspection detects deviation in functionality or direction resulting in unacceptable results. Scrum defines four formal points for inspection and adaptation: Sprint Planning, Daily Scrum, Sprint Review and Sprint Retrospective. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

2.2.2 Scrum in Practice: Team, Events and Artifacts

Scrum is all about the Scrum Team and roles, events, artifacts and rules linked to it. All of the components serve a specific purpose, and are needed in the successful instrumentation of Scrum. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

The Scrum Team

The Scrum Team consists of a Product Owner, Scrum Master and Development Team. Scrum Teams are cross-functional and self-organising, meaning that there are multiple competencies

best suited for the task at hand available, and that teams can structure themselves the way they see fit to accomplish the task. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

The **Product Owner** is responsible for maximising the value of the product and the work of the Development Team, through Product Backlog. The Product Owners job is to manage the Product Backlog. The Product Backlog needs to be clear, visible to all relevant parties and up-to-date. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

The Product Owner interfaces with management and both internal and external stakeholders, who want to influence to the Product Backlog. The Product Owner has to be convinced, if a priority of an item needs to be changed, or a new item added. No one else, but the Product Owner is allowed to change the priorities on items the Development Team is working on. (Schwaber & Beedle 2002, 34.)

The **Scrum Master** is responsible for the success of Scrum and ensuring that Scrum values, practices and rules are enacted and enforced (Schwaber & Beedle 2002, 31).

The Scrum Master has a three-way role; to the Development Team, to the Product Owner and to the organisation. Scrum Master serves the Development Team by coaching, facilitating and organising Scrum Events and removing impediments. Scrum Master serves the Product Owner by helping to manage the Product Backlog, communicating the vision and goals to the Development Team and teaching the Development Team in creating clear and concise Backlog items. Scrum Master serves the organisation by leading and coaching in Scrum adoption, planning Scrum implementations and working with other Scrum Masters to make Scrum more effective in the organisation. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

The **Development Team** is responsible for delivering a “Potentially Shippable Increment” of the project or product at the end of each Sprint. Development teams are structured and empowered by the organisation to organise and manage their own work for efficiency and effectiveness. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

Development Teams are self-organising, and they figure out themselves how to turn a set of tasks from the Sprint Backlog to a working product. They are cross-functional, having members

from different fields with different backgrounds come together as one team with no titles. Everybody does everything, regardless of the background. The optimal team size is seven, plus or minus two. (Schwaber & Beedle 2002, 35-38.)

Scrum Events

Sprint is the core in Scrum. It's a time-box of a month or less, during which a Potentially Shippable Increment of the product needs to be completed. A new sprint starts right after the previous has ended, and they have consistent durations throughout the development effort. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

The sprint contains and consists of the following: Sprint Planning Meeting, Daily Scrums, the development work, Sprint Review and Sprint Retrospective. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

When the sprint starts, the development team is on its own. The Sprint Backlog agreed upon in the Sprint Planning Meeting is what guides the team for the duration of the sprint and no one has the authority to order them to do something else. The goal is set, and the team has the authority to organise itself and the work the way they best see fit, in order to accomplish the goal, as long as they hold Daily Scrums and keep the Sprint Backlog up-to-date. (Schwaber & Beedle 2002, 50-53.)

The Product Owner has the authority to cancel an on-going sprint – this is however done only in exceptional circumstances. And since the sprints are short in nature, cancelling a sprint is rarely a necessity. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

A sprint starts with a **Sprint Planning Meeting**, where the work to be done during a sprint is discussed and agreed upon. This plan is devised collaboratively by the entire Scrum Team. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

The Sprint Planning Meeting consists of two parts, where the first part agrees what is the increment to be delivered during the sprint, and the second part discusses how the increment will be delivered. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

In the first part of the meeting, the Development Team tries to forecast what it is able to deliver during the sprint. The Product Owner presents the Product Backlog as a prioritised list and the whole Scrum Team (and possibly other stakeholders) discusses what items from the Product Backlog seem feasible for this sprint. After the team has selected the items from the Product Backlog, a Sprint Goal is crafted – which is basically an objective, fulfilled by the implementation of the selected Product Backlog items. (Schwaber & Beedle 2002, 48-49.)

In the second part of the meeting, the Development Team decides how it will implement the selected Product Backlog items, or the Sprint Goal. This is done by breaking the Sprint Goal down to smaller stories or tasks, which are then compiled in a list, called the Sprint Backlog. (Schwaber & Beedle 2002, 49.)

During the sprint, 15-minute **Daily Scrum** meetings are held every 24 hours (weekends and holidays excluded), where all the Development Team members synchronise their activities and plans for the day. The Scrum Master queries the status and plan from every team member with three formalised questions (What has been accomplished since the last meeting? What will be done before the next meeting? What obstacles are in the way?), which are answered turn-by-turn, giving the Scrum Master a good overview of the situation. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

The Daily Scrum is used to evaluate the progress toward the Sprint Goal, and to see how the progress is trending toward completing the items on the Sprint Backlog. Daily Scrum optimises the team's possibilities to reach the Sprint Goal. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

Schwaber & Sutherland wrap up the Daily Scrum as:

> Daily Scrums improve communications, eliminate other meetings, identify and remove impediments to development, highlight and promote quick decision-making, and improve the Development team's level of project knowledge. This is a key inspect and adapt meeting. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

The **Sprint Review** is held at the end of Sprint, to inspect and review the increment the team has implemented during the Sprint. This is also an opportunity to adapt the Product Backlog, if

needed. The Sprint Review is an informal meeting, and the presentation of the implemented increment should prompt discussion and feedback, fostering collaboration within the team and with possible stakeholders. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

During the review, the Product Owner identifies the items that are "Done". Then the Development Team discusses how the sprint went, in terms of what went well, what didn't work, if there were problems and how they were solved. After that the team demonstrates the work done and answers possible questions regarding the implemented increment. After the team presentation, the Product Owner discusses the Product Backlog as it stands now, one Sprint later. Then the whole group collaborates on what to do next, to provide input to the subsequent Sprint Planning Meetings. Sprint Review results in a revised Product Backlog, defining the Product Backlog items for the next Sprint. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

Finally, the last thing in a Sprint is the **Sprint Retrospective** providing opportunity to the team to inspect itself and plan for improvements in the team way of working for the future Sprints. By the end of this meeting, the team should have identified improvements, which are then carried out in the next Sprint. By implementing these changes, the team adapts the Scrum Team itself. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

Scrum Artifacts

Schwaber & Sutherland define Scrum Artifacts in general as follows:

> Scrum's artifacts represent work or value in various ways that are useful in providing transparency and opportunities for inspection and adaptation. Artifacts defined by Scrum are specifically designed to maximize transparency of key information needed to ensure Scrum Teams are successful in delivering a "Done" Increment. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

The **Product Backlog** is one of these artifacts - it's an ordered list of everything that needs to be implemented for a product, effectively a single source for requirements for a given product. The Product Owner is responsible for it, keeping it up-to-date and available. Due to the nature of Product Backlog, it's never complete, but all the time a work in progress. It evolves with the product it's related to, identifying what the product needs to be appropriate, competitive and useful. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

The Product Backlog is a priority list of items. The most important items are at the top, and lesser value items in the bottom of the list – consequently, the high priority items have more detail to them, as they need to be estimated in such detail that the Development Team can take an item and complete it during a Sprint. (Schwaber & Beedle 2002, 33.)

Every once and a while, the Development Team engages Product Backlog grooming, which means adding detail, estimates and priorities to Product Backlog items. This is a part-time activity that is done in collaboration with the Product Owner during a Sprint. The Development Team must do this, as they are the only ones who can give schedule estimates on the Product Backlog items. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

In order to get visibility to all stakeholders on the progress of the work specified on the Product Backlog, the remaining work is monitored by the Product Owner at least for every Sprint Review. Various trend charts, such as burndown and burnup, are commonly used to forecast progress. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

The **Sprint Backlog** is a subset of the Product Backlog – the items the Development Team has selected for implementation during the Sprint. It's the best estimate of what functionality is going to be implemented in the next Increment and what work it requires. The Sprint Backlog makes the work needed to realise the Sprint Goal visible. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

The Sprint Backlog is maintained by the Development Team during the Sprint. When new tasks emerge, they are added to the Sprint Backlog. As tasks are done, the remaining work on the Sprint Backlog is updated correspondingly. If some tasks become obsolete, they are removed. As the Sprint Backlog is highly visible, the real-time depiction of the work done and planned to be done by the Development Team, only the team itself can update the Sprint Backlog. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

The remaining work is tracked at least for every Daily Scrum, to project the reachability of the Sprint Goal. The Development Team can manage its progress by tracking the remaining work throughout the Sprint. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

Product Increment is the sum of all the completed Product Backlog items, during this Sprint and before. At the end of each Sprint, a new Increment must be “Done”, meaning it must pass the team-specified Definition of Done. It is transparent quality criteria the team has set on its deliverables. (Schwaber & Sutherland 2011, date of retrieval 8.5.2013.)

3 EXPERIENCES & COMPARISON OF WATERFALL vs. SCRUM

3.1 Experiences of Waterfall

I have personally worked following the Waterfall model (or better, the improved iterative model depicted earlier) for many years and during that time discovered it to be rather cumbersome method for developing UI software in large scale. I'll discuss my experiences step-by-step below.

3.1.1 Requirements

As the model suggests, everything starts from Requirements, which should be well understood and strictly defined by the client when delivering them onwards. They should also be constant in order for the Design and Development work to be effective. However, to my experience, this is rarely the case. Requirements change as the work progresses – if working in the waterfall model, it causes unnecessary backtracking, which can be quite costly.

3.1.2 Design

When the Requirements come through to Design step, system design and user interface design take place. If the Requirements are well defined and the changes are trivial, this is quite straight-forward activity. In large scale development projects though, the Requirements can only give some guidance to the system and user interface design, and this activity is very time consuming.

If creating something novel, both the system design and the detailed user interface design take time. The architecture of the system needs to be figured out, interfaces thought about and some documentation done, too. When designing user interfaces, the design for the system needs to be created, proposed solutions prototyped and even user tested before making the final judgment on the design. When there is a consensus of what the user interface should be like, then a go ahead to software development is given. This is also the first phase where discrepancies to Requirements are discovered, and they need to be revised.

3.1.3 Development

In the waterfall model, a step must be completed before the next begins. Therefore Development starts only after Design is completed. The actual implementation work of the software is begun and especially in large projects, problems may arise. If so, then either Design, Requirements or both need to be revised, even modified, costing valuable time (and money).

But, on the contrary to the model, software developers rarely wait for the user interface designers to complete their tasks before starting their work. This can cause extra development work, if the user interface design changes.

Also, depending on the organisational structure, development teams may have maintenance duties alongside the new development tasks, so the full potential of the workforce cannot be utilised in the new development.

3.1.4 Testing

Even though the waterfall model makes a clear separation between the steps in the model, it makes sense to start testing activities already during the code implementation, and continue them beyond the code completion. In practice, the development and testing are parallel activities, especially in large projects. For example, a unit testing (and possibly feature functionality testing) should be carried out as soon as the relevant piece of code has been implemented – otherwise there is a high risk of running into severe problems later on in the development. Also a high inflow of errors is practically guaranteed, if the unit testing is not done in due course.

During testing, different kinds of testing activities are run on the developed code, ranging from functional to non-functional, manual to automated testing. Testing activities also depend on where the software is targeted; mobile devices, computers, medical instruments, and so forth.
When the software has gone through and passed all the testing activities, sellable software package is created. This is the end result provided to the client.

3.1.5 Operations

In this phase the software product is ready, and provided to the client. Usually, especially in larger software projects, some amount of maintenance-type work is needed. Maybe an upgrade of the software is needed after initial market reactions, or customer testing fails, and some improvements are required as a result.

3.1.6 Example based on past experience

A company with a large distributed organisation received very detailed requirements from a customer, to develop a customer-spec user interface to a device. At the time, the company still utilised a waterfall-like process in software development.

The requirement was clear. System design was to develop the user interface on top of the company's proprietary software, as a delta. Creative user interface design was virtually non-existent, as the customer provided a very detailed spec coupled with the requirements, however technical writing was needed due to the nature of delta – the customer specification needed to be added to the proprietary UI specifications, too.

The development started in parallel with the technical writing, as well as testing activities – all three having interactions with each other. Testers and technical writers worked together to create a better spec, as well as better test plan. Developers worked both ways, getting feedback from testers and guidance from technical writers. A strict sequential waterfall model was not utilised, but rather the iterative version of it. Unfortunately, the project didn't get to operations phase, as it was cancelled during the development due to a change in the business focus.

Learnings from the example

Even though the organisation uses waterfall-like model in the development environment, it seems quite difficult to apply a "pure" sequential waterfall in software development environment, because of the obvious dependencies between the Design, Development and Testing phases. These three activities work well together, providing constant feedback loops to each other. Testing already during the development helps identifying the biggest blunders at an early stage, the development

and design can agree on details as the work progresses, and testing gets a good feedback from design on what to test.

3.2 Experiences of Scrum

After years of following the Waterfall model in a UI software development environment, I was introduced to Scrum. Initially, I was sceptical, but it did not take long for Scrum to win me over. This robust, light-weight, team-oriented software development model convinced me of its benefits quite fast. In essence, all the same components that are present in the waterfall model (requirement handling, system design, code development, testing and delivery) are present in the Scrum model as well, but they are managed differently making the process agile.

3.2.1 Scrum Team

As the Scrum Team is a key to all development efforts, the team composition and chemistry between team members is really important. In order to function effectively, the rules and practices for the team need to be set and enforced. Trust is also a significant factor. Team members need to trust each other, and the Scrum Master as well as the Product Owner. In addition, Scrum Master and Product Owner need to gain a level of respect from the team.

3.2.2 Scrum Events

Scrum works within a Sprint, in my experience a period of 2 weeks. Theory suggests max. 30 days per Sprint, but reasonable results can be achieved in 2 weeks. Sprint Planning starts the Sprint, where the deliverables for the current Sprint are discussed and agreed. In the Sprint Planning, it takes some time from the team to learn how to best estimate the items. Practise helps, and after a few Sprints, the team starts to understand how much they, as a unit, can deliver in the given timeframe. Daily Scrum is a very important meeting to keep, as the progress and possible impediments are checked there. Personally, I find that Daily Scrum works best if kept in the morning, as a day starter. Having the whole team there, face-to-face, sharing information is crucial. Sprint Review, is as the name suggests, a review meeting, where the delivered increment is reviewed with needed stakeholders – usually the Product Owner is

enough. Sprint Retrospective enables the team to go through its own way-of-working, team dynamics or anything related to the team itself. It enables team improvement.

3.2.3 Scrum Artifacts

The Product Backlog is a prioritised wish-list, of what a product should be. This always changing list is managed by the Product Owner. The items can be really vague, but then low in priority and vice versa, very detailed and high priority. This is because the Development Team picks items from the Product Backlog to the Sprint Backlog, which it will deliver during a Sprint. Therefore, it is in the Product Owners best interest to keep the most important items detailed enough for Sprint content selection.

3.2.4 Examples based on experiences

Design hijack

A company had internally decided to pursue a big challenge, and established a really big project allocating hundreds of developers to the project in geographically dispersed locations. If the challenge was not great enough already, the company decided to go agile with this project, and Scrum was selected as the framework for the work.

The developers were assigned to Scrum Teams and the work was begun, with some difficulties of course, as a Scrum way of working was new to most team members. Adapting to new, more collaborative working environment took some time. This became evident e.g. in the first Sprint Planning Meetings, where people were expecting the team leader, now called Scrum Master, to tell them what to do next, instead of the team working together to select their own tasks from the Product Backlog. Clearly the most difficult thing to embrace was the “we” spirit – everyone had been an individual, doing one’s thing – now it was expected that instead of an individual, there is a collective team, which is doing one’s thing. The other was the expectation, that management will still dictate what to do, when and almost even how, instead of figuring that out as a team.

Scrum Teams worked on the project for several sprints and progressed quite well against the targets, when suddenly Design Management hijacked the UI concept and redesigned it

completely. Up until this point, the user interface design and decisions were made either within the team, or locally in collaboration with other user interface designers. After the redesign, the user interface governance transferred off-site to another organisation, which made the work much more difficult than before – basically all the design decisions were either made by them, or had to be approved by them, diminishing the role of the user interface designer in the Scrum Team. By taking this much of control away from the Scrum Team, they actually effectively hindered the way of working, changing it much closer to the iterative waterfall than that of Scrum, as in the context of user interface, the team did not have any say on what they were doing – it was laid out to them by an outside party.

Product Owners were involved to re-examine the Product Backlog and reprioritise. Lots of work was done for nothing, but luckily lots of the work done could also be saved by only visual tweaks, keeping the functionality hidden behind the changing user interface. The Product Owners jotted down new items on the Product Backlog, and the Scrum Teams pushed onwards.

Collaboration channels were established and the work continued. To this day, I'm surprised that it went as well it did – and it demonstrated the Scrum's adaptability to change.

Team chemistry

Having been in a few Scrum Teams and a Scrum Master in one, I have noticed how important it is to have a good team spirit. An atmosphere of mutual trust and respect, where every member is willing to adapt for the sake of the team, is the driving force behind any team towards an environment, where collaboration thrives.

Unfortunately, not everyone wants to adapt, trust or respect their peers. Such team members are rather hard to work with, and they hinder the Scrum way-of-working. The theory suggests, that such people could be voted off the team, but that does not really suit to our working culture – such drastic measures could actually foster more distrust among other team members. Also, in practical terms, getting rid of a team member seems a long shot – management makes capacity allocations, and once allocated, it is not that easy to move around. It is better to try to find ways to cope with such people, and try to coach them as better team players.

Scrumban in a maintenance project

A team in a company was assigned to productisation phase of a product, to help in getting the product on the market. Very little new development was done on the product, but the error inflow was quite high due to lots of last mile testing conducted by internal and external resources in different parts of the world.

As new development was scarce and unknown amount of errors were due, Scrum as such did not feel a feasible solution anymore to tackle a division of labour. However, Scrum Framework contained lots of things that were seen useful in this situation, too, such as the team composition, the idea of a Backlog of items and Daily Meetings. This is a mixture of Scrum and Kanban – Scrumban.

Wikipedia defines Scrumban as follows:

> Scrum-ban is a software production model based on Scrum and Kanban. Scrum-ban is especially suited for maintenance projects or (system) projects with frequent and unexpected user stories or programming errors (Wikipedia 2013, date of retrieval 20.5.2013).

The idea was to take the incoming errors and possible new development items to the Backlog, and have a simple whiteboard with a few columns, like e.g.: Backlog, Development ongoing, Under testing, Done. Once a new error or a development item came in, it was placed on the Backlog. The priority of an item was determined by the location of it on the board, the higher the priority, the higher the location on the board. Then a developer took the item from the Backlog and placed it on the board to the column “Development ongoing”. When it was ready for testing, a tester moved the item on the board to the column “Under testing”. When the item was tested and deemed OK, it was placed on the column “Done”. And the process started again from the beginning.

Daily meetings were held to check that everyone had something to do (and that nobody had too much to do) and to identify possible impediments.

Learnings from the examples

Scrum is really fast to react to changes, adapting to changing conditions in no time. Deploying Scrum is easy on paper, but much harder to implement successfully than it seems.

Team spirit is of great importance to the success of the Scrum Team. The mutual relationships of team members make or break the team.

Scrum Framework is adaptable, and works well together with e.g. Kanban. Of course, this is not Scrum anymore, but the basic principles are solid and can be adapted to suit special needs.

3.3 Comparing Waterfall and Scrum

Sequential waterfall and Scrum are fundamentally different, yet there are lots of similarities. The same work can be done in either way, using Scrum or waterfall, but depending on the case it makes sense to see which methodology suits better to the specific needs. It is also possible to mix waterfall and Scrum, as at least three points where Scrum and sequential development might meet have been defined – waterfall-up-front, waterfall-at-end and waterfall-in-tandem (Cohn 2010, 390).

While waterfall is more biased to individuals, Scrum emphasises collaboration – the waterfall would assign certain people on certain tasks, but Scrum gives the task to the team, expecting that the team sorts out the issue as it best sees fit. As long as the end result is what is requested, means are secondary.

While waterfall deals with changes poorly, Scrum thrives on change - when changes occur in any point of the development cycle, waterfall would have to backtrack n amount of steps, to reset the situation and start over. Scrum checks the situation as it stands, and changes direction as per change requests.

While waterfall is predictable, Scrum is not – the waterfall has all the requirements in place before the development starts, so it is easy to predict when the code can be completed. Ideally yes, but practise has proven that the requirements change despite being set in the beginning, rendering

waterfall actually unpredictable. Scrum is unpredictable at first, but as the Product Backlog diminishes, Scrum actually becomes really predictable, as the progress is visible at all times via e.g. burndown charts.

While waterfall delivers value only at the end, Scrum starts delivering value almost instantly – in waterfall, the functionality is released as one big increment or release after all development activities are done. Scrum starts delivering right after the first Sprint, as every Sprint delivers a Potentially Shippable Increment, of instant value.

4 CONCLUSIONS

The sequential waterfall is actually a myth, because there is always some iterative processing. Especially, the subsequent phases might overlap rather a lot. This is not a bad thing as it provides feedback loops between the different phases and stabilises the end result – there are organisations, where waterfall, or iterative waterfall, is the right choice as a development model.

For a software development organisation, going agile makes sense, as the world we live in is rather volatile, and using agile methods ensure a better preparation to a change compared to the sequential waterfall. Agile methods are more versatile and therefore also create new opportunities for businesses. Agile advocates an iterative and incremental way of working, which also brings immediate value to the organisation, as increments of the product can be productised earlier.

In the end, it is up to the organisation to decide if it wants to go agile or remain in the traditional sequential approach.

REFERENCES

Agile Manifesto. 2001. Manifesto for Agile Software Development. Date of retrieval 19.5.2013. http://www.agilemanifesto.org/

Benington, H. 1956. Production of Large Computer Programs. Date of retrieval 13.5.2013. http://sunset.usc.edu/csse/TECHRPTS/1983/usccse83-501/usccse83-501.pdf

Cohn, M. 2010. Succeeding with Agile: Software Development Using Scrum. Upper Saddle River, NJ: Addison-Wesley.

Leffingwell, D. 2011. Agile Software Requirements: Lean requirements practices for team, programs, and the enterprise. Upper Saddle River, NJ: Addison-Wesley.

Mitch Lacey & Associates, Inc. 2013. Scrum Framework Flow Diagram. Date of retrieval 8.5.2013.
http://www.mitchlacey.com/resources/scrum-framework-flow-diagram

Royce, W. 1970. Managing the development of large software systems. Date of retrieval 23.4.2013.
http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf

Schwaber K. & Beedle M. 2002. Agile Software Development with Scrum. Upper Saddle River, NJ: Prentice Hall.

Schwaber, K. & Sutherland, J. 2011. The Scrum Guide. Date of retrieval 8.5.2013.
http://www.scrum.org/Portals/0/Documents/Scrum%20Guides/Scrum_Guide.pdf

Scrum Alliance. 2013. What is Scrum? Date of retrieval 8.5.2013.
http://www.scrumalliance.com/pages/what_is_scrum

Wikipedia. 2013. Scrum (development). Date of retrieval 20.5.2013.
http://en.wikipedia.org/wiki/Scrum_(development)#Scrum-ban

Wikipedia. 2013. Waterfall model. Date of retrieval 8.5.2013.
http://en.wikipedia.org/wiki/Waterfall_model