2013•2014
FACULTY OF BUSINESS ECONOMICS
Master of Management

Master's thesis

Case Study: Agility at Scale Wolters Kluwer Belgium

Promotor :
Prof. dr. Benoit DEPAIRE

Tom Devos

Thesis presented in fulfillment of the requirements for the degree of Master of Management

universiteit hasselt
KNOWLEDGE IN ACTION

Universiteit Hasselt | Campus Hasselt | Martelarenlaan 42 | BE-3500 Hasselt
Universiteit Hasselt | Campus Diepenbeek | Agoralaan Gebouw D | BE-3590 Diepenbeek

universiteit hasselt
KNOWLEDGE IN ACTION

2013•2014
FACULTY OF BUSINESS ECONOMICS
Master of Management

Master's thesis

Case Study: Agility at Scale Wolters Kluwer Belgium

Promotor :
Prof. dr. Benoit DEPAIRE

Tom Devos

Thesis presented in fulfillment of the requirements for the degree of Master of Management

universiteit hasselt
KNOWLEDGE IN ACTION

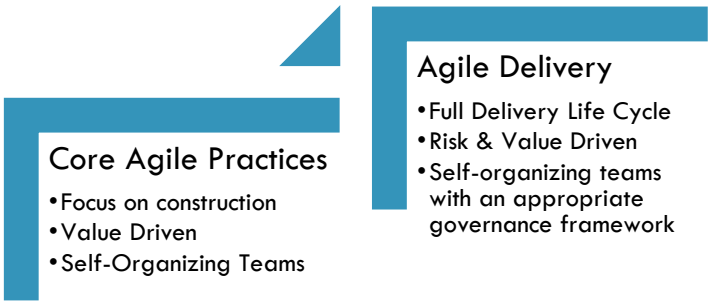
1 Contents

Executive Summary

While a lot of software startups currently are using agile software development methods to rapidly bring software to the market, larger organizations are struggling to keep up that pace. These agile methods sometimes fail in larger enterprises, and we want to know why this is happening. In order to know that, we want to know how we can define an 'Agile' organization, which challenges are ahead when wanting to become 'Agile', and what the best practices are in the market.

Becoming an Agile Enterprise

Agile enterprises have to find the true meaning behind the agile methodologies that work so well for startups, and find a way of working that supports these meanings, implemented at the scale of their organization. It is important to focus on a full delivery life cycle, talking about solutions, not about software. A solution has a vision, which is shared throughout the whole organization, it consists of software that delivers value to its stakeholders, but also minimizes the risks that are present in the organization. It also takes care of the fact that the stakeholders are prepared to receive and use the solution, by training, infrastructure, testing, etc. An agile enterprise needs an appropriate governance framework that allows their development teams to be self-organizing instead of limiting their maneuverability.



One way to assess if your enterprise is truly agile is by using the 5 criteria defined by Ambler (2009):

1. Produce working software on a regular basis
2. Do continuous regression testing, and better take a Test-Driven Development approach
3. Work closely with your stakeholders, ideally on a daily basis
4. Are self-organizing within a governance framework
5. Regularly reflect on how they work together and then act to improve their findings

When starting this research at Wolters Kluwer Belgium (WKB), I have found that only the last criteria was fulfilled within the way of working at the time. Now, two years further in their quest of becoming an agile company, WKB has managed to fulfill already 4 out of 5 criteria, leaving room for improvement on their testing strategy.

Facing the Challenges ahead

When becoming agile at scale, an organization has some challenges it must face and mitigate in order to book success, this was also the fact within Wolters Kluwer Belgium. Eight of these possible challenges have been defined by Ambler (2010) in its Agile Scaling Model. By researching these scaling factors within the context of the delivery of the project 'Online Invoicing' at WKB, I was able to validate 6 of the total of 8, being Geographic Distribution, Technical Complexity, Regulatory Compliance, Organizational Distribution, Organizational Complexity & Enterprise Discipline. The latter two, Team Size & Domain Complexity challenges, weren't present in the organization, but this doesn't mean they can't be a challenge in other enterprises, here is some room for further research. Another point for further research is the fact that this list might seem exhaustive, but it isn't, as this was out of scope of this thesis. The most important thing to remember here is that you better can have a look at the chart below and think about whether these challenges might be present in your organization.

Good Practices for Agility at Scale

The logical next question when you have identified that there are indeed challenges that you have to face in your organization when becoming agile at scale is to look for good practices. Further in this thesis you can find a summary of three frameworks that can be used as an inspiration, being the **Disciplined Agile Delivery** (DAD) approach from Scott Ambler & Mark Lines, the **Large Scale Scrum** (LeSS) method by Craig Larman and Bass Vodde and the **Scaled Agile Framework** (SAFe) by Dean Leffingwell. As said, these approaches are to be seen as an inspiration, as I wasn't able to find the one silver bullet that makes every company an agile company. At Wolters Kluwer Belgium, we have been inspired by DAD, focusing on team level, creating a clear vision throughout the whole organization, investing in a software delivery life cycle with clear roles & responsibilities and shifting from a top-down to a more bottom-up culture where enterprise stimulates teams to be self-organizing and are in charge of their technology.

Preface

While working at AE nv/sa as a consultant, the agile software delivery method was introduced to me, based on the practical technique of Scrum. I was able to put this knowledge into practice as a business analyst within the ETNIA.be program at Wolters Kluwer Belgium. At the moment of conducting this thesis, WKB was going through a large organizational and technological change. The organization was attempting to adapt an agile delivery method, whilst migrating their broad legacy application landscape towards the cloud, being innovative at the same time.

As we were facing many challenges in mastering this agile method, especially when it came to scaling, the opportunity arose for this master thesis, since AE nv/sa and WKB are both searching for ways of improving their success by delivering agile software.

I want to thank the management of WKB, in particular Mark Caelen & Lars Van Sweevelt, for the opportunity to put the gathered theoretical research in to practice within the ETNIA.be program. Also the help of my colleagues within the ETNIA.be program team were compulsory for this master thesis to succeed. A special word of gratitude goes to Hans Saenen, Joris Dresselaers, Mark Caelen, Tom Princen, David Van Steenkiste, Sophie Traen & Michaël Mertens for their willingness to assist in the interviews. I hope that this master thesis will add explicit value towards our ambitious projects within the ETNIA.be program, allowing us to deliver them faster with a higher quality level than before, allowing us to quickly adapt to the changing market of tax & accounting software.

During the creation of this master thesis, I was able to count on the large network of AE consultants with extensive theoretical knowledge combined with experience in the field of software delivery. Thank you Ralf Geenen, Guido Van Humbeeck, Steven Arnauts, Kris Coenen & Danielle Glassée for the ideas, guidance & the necessary chasing that I needed. Also I want to thank Thomas De Vries, Arnout Mertens, David Van Steenkiste & Vincent Guelinckx for the frequent advice, brainstorm sessions and the proofreading of this master thesis.

Whilst creating a practical master thesis, it is important to keep in mind the academic character of this document. My promoter Prof. dr. Benoît Depaire kept a close eye guarding this, therefore I would like to thank him. Also, I'm very thankful to Prof. Kris Pattyn and Prof. em. dr. Jeanne Schreurs for helping me by offering guidance and proofreading my work.

A special word of thanks goes to my parents Mathilde & Ludo for the long years of support during my studies and for teaching me the true meaning of entrepreneurship, for one can't find better role models. And off course not to forget my beloved Caroline, who has supported me from the start till finish, keeping an eye on the planning & making our home the perfect environment for productive thesis writing.

Part 1: Master thesis context

2 Why this research?

2.1 Purpose of this research

Many software houses are shifting from the waterfall method towards the agile method for delivering software, as it is probably one of the major shifts in project delivery of the past few years. The agile delivery method has proven its use in small independent development teams creating new software for a limited scope.

As this method is getting picked up in larger software developing organizations, my field experience shows that in a number of cases these organizations aren't receiving the full benefits they expected from following the Agile software development method. Software still isn't delivered on time, with the right scope and within budget, especially when building heavily integrated and complex solutions where the development work is distributed among different teams or locations spread around the world (outsourcing & offshoring).

As these problems are faced on a day-to-day basis by the consultants of AE nv/sa, an ICT consulting company in Leuven, Belgium, the initiative arose to investigate the potential problems of enrolling agility at scale in organizations. Knowing potential scaling problems can help the professional experts at hand to more easily identify good practices and possible mitigations, making the agile delivery method succeed at scale.

The goal of this master thesis is to research potential problems for agile delivery at larger organizations with multiple teams coping with governance, enterprise architecture, offshore development, SOA, etc., identifying potential scaling factors and do an investigation about known good practices or mitigations.

2.2 Central Research Question

Which potential scaling problems arise when enrolling the agile software delivery method at large scale and are there known mitigations or good practices to deal with them?

2.3 Sub Questions

Based on the central research question, three sub research questions are identified.

2.3.1 Why does agile sometimes fail in larger organizations?

What is the current status of the academic & professional research being done around agile software delivery, does this research mention any specific techniques or frameworks that can improve the success of delivering agile software at scale? How is this method evaluated in comparison with the traditional Waterfall method for delivering software?

2.3.2 Is it possible to identify potential scaling factors?

Where are the challenges in becoming agile at a large scale? Which factors does an agile enterprise has to take into account when delivering large scale solutions?

2.3.3 Are there known mitigations or good practices for enrolling the agile software delivery method at scale?

Are there known mitigations or good practices in the academic and professional field of work to cope with the found scaling factors? How can a large organization cope with agility at scale?

2.4 Research method: case study research

The case study research method lends itself as a tool for qualitative research by the following features: (Miles & Huberman, 1994)

- Conducted through an intense and/or prolonged contact with a 'field' or life situation (typically 'normal' situations, reflective of everyday life of, for example organizations);
- Researcher's role is to gain a holistic overview of the context under study;
- Researcher attempts to capture data on the perceptions of local actors 'from the inside';
- Main task: explicate the ways for managing day-to-day situations;
- Many possible interpretations of material;
- Little standardized instrumentation is used as the outset as most analysis is done with words.

As there is the possibility to gain access to a broad set of documents and to conduct several in-depth interviews with interesting people in the field at Wolters Kluwer Belgium, this qualitative data research in the form of words allows to test the conceptual framework and gain proper insight into its validity at the project at hand, which is one of the main purposes of this thesis.

The choice of doing a case study is supported by the vision (Yin, 1989) that a case study is a relevant option when you don't have control over behavioral events but you focus on contemporary events. Based on the research of the daily operations at Wolters Kluwer Belgium, the scaling factors identified within existing academic and professional literature will be tested, trying to validate that they can truly be seen as a scaling factor.

The case study research is done following the steps (de Weerd-Nederhof, 2001):

1. Case Study Design
2. Preparing for Data Collection
3. Conducting Case Study
4. Analyzing Data
5. Final Reporting

In order to **design the case study** correctly, a conceptual framework will be formed in Part 2 of this thesis, based on an in-depth literature study regarding the topics of agile software delivery, to clarify my version of the context in which this case study is to be conducted, as suggested by Miles & Huberman (1994). Part 3 of this thesis focusses on the Unit of Analysis, the Specific Case Study Design and the reasoning for selecting Wolters Kluwer Belgium as a relevant Case Study.

Relevant protocols and instruments are discussed in Chapter **Error! Reference source not found.** on how o get the relevant information and assure the quality of the **data to be collected**. Transcripts of **data sources** can be found in the appendices of this thesis, in order to build a correct chain of evidence to further assure the quality of this research.

In the **analysis** section of this thesis (Chapter 6.3) the data is be presented in a more relevant way via data displays, allowing to draw valid conclusions and create specific insight into the agile delivery method. These conclusions and insights are documented in Part 4 of this thesis, including the **final case study report** for the Wolters Kluwer Belgium case.

The major disadvantage of this single case study research is stated by both (Yin, 1989) and (Eisenhardt, 1989): “a number between 4 and 10 cases usually works well; with fewer van 4 cases it is often difficult to generate theory with much complexity, and its grounding is likely to be unconvincing … with more than 10 cases it quickly becomes difficult to cope with the complexity and volume of the data”. The reasoning behind the selection of a single case study is the time-constraint in building a thorough case study within the scope of this thesis.

Part 2: Building the conceptual framework

3 Gaining insight in software engineering methods

3.1 A brief history of software engineering

To gain a profound insight in delivering quality software, a brief history regarding software engineering is at hand. The term *software engineering* first appeared in the 1950s (Boehm, 2006), which was then still highly orientated towards hardware engineering. The most ambitious software development process was developed by Semi-Automated Ground Environment (SAGE), showing that sequential waterfall-type models have been used in software development for a long time (ISO, 1995). Boehm states that another indication of the hardware engineering orientation is in the names of the leading professional societies for software professionals: the Association for Computing Machinery and the IEEE Computer Society.

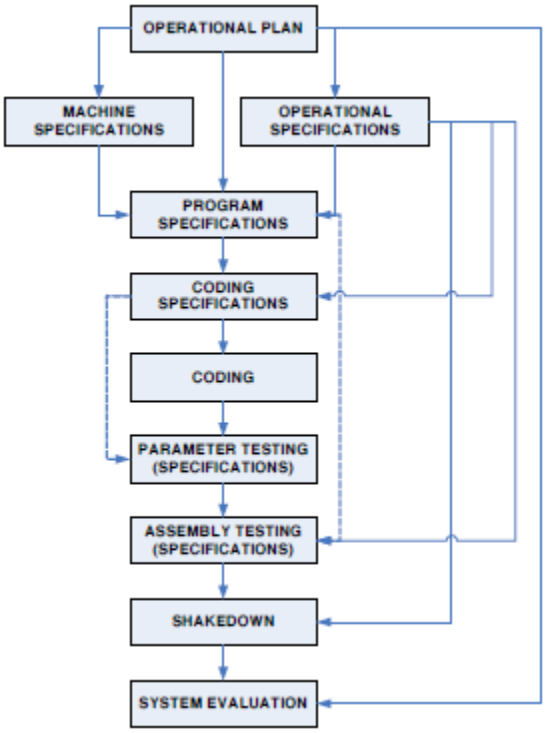


Figure 1 The SAGE Software Development Process (1956)

The 1960s brought a first paradigm shift towards the community as it became clear that software was much easier to modify than hardware. Boehm calls this the "code and fix"-approach where applications became more people-intensive than hardware-intensive. Also it seemed that software needed a lot more maintenance effort than hardware. The NATO Science Committee held two "Software Engineering" conferences in 1968 and 1969 defining a baseline that organizations and governments could use for determining and developing improvements (Boehm, A View of 20th and 21st Century Software Engineering, 2006).

In the 1970s, the need arose for more formality which led to more initiatives around structured design and programming. Together with the paradigms of 1950s and 1960s, this resulted in Royce's version (1970) of the Waterfall software delivery method (Boehm, A View of 20th and 21st Century Software Engineering, 2006).

The software engineering community aimed to create more standards in order to solve the issues from the 1970s regarding productivity and scaling problems. Results were publications like the Software Capability Maturity Model (SW-CMM) (Humphrey, 1989) and investments in integrating tools within support environments leading to Software Factories (Boehm, A View of 20th and 21st Century Software Engineering, 2006). The famous 'No Silver Bullet' paper (Brooks, 1987) consolidated other important improvement efforts regarding productivity such as expert systems, very high level languages, object orientation, powerful workstations and visual programming (Boehm, A View of 20th and 21st Century Software Engineering, 2006).

In the 1990s the first signs of agile software development were published via the Spiral model (Boehm, 1988) by emphasizing time-to-market and concurrency, rather than holding on to a sequential process. Further investments in architecture were made, focusing on usability by the end-user. Open Source software took its place in the community with major milestones as Torvalds' Linux (1991) and Berners-Lee's World Wide Web consortium (1994)

This evolution towards more agility and rapid application development continued in the 2000's with the Agile Manifesto in 2001 and agile methods such as Adaptive Software Development, Crystal, Dynamics Systems Development, eXtreme Programming (XP), Feature Driven Design and Scrum. Together with these agile methods, model-driven development offered prospects of improving compatibility by the development of domain models whose domain structure leads to architectures with high module cohesion and low inter module coupling, enabling rapid and dependable application development and evolvability within the domain (Boehm, 2006), which led in turn to the push towards integration of software and systems engineering.

Today, in the 2010's, mobile first and responsive design (Wroblewski, 2011) are major concerns in the software engineering community and agile methods are embraced and proven efficient by small teams, as many larger companies are struggling to implement the agile methods at scale (Alliance, 2001).

Interesting about this history is the trend towards agility. The next sections focus on explaining the waterfall method and compare it to this agile methodology.

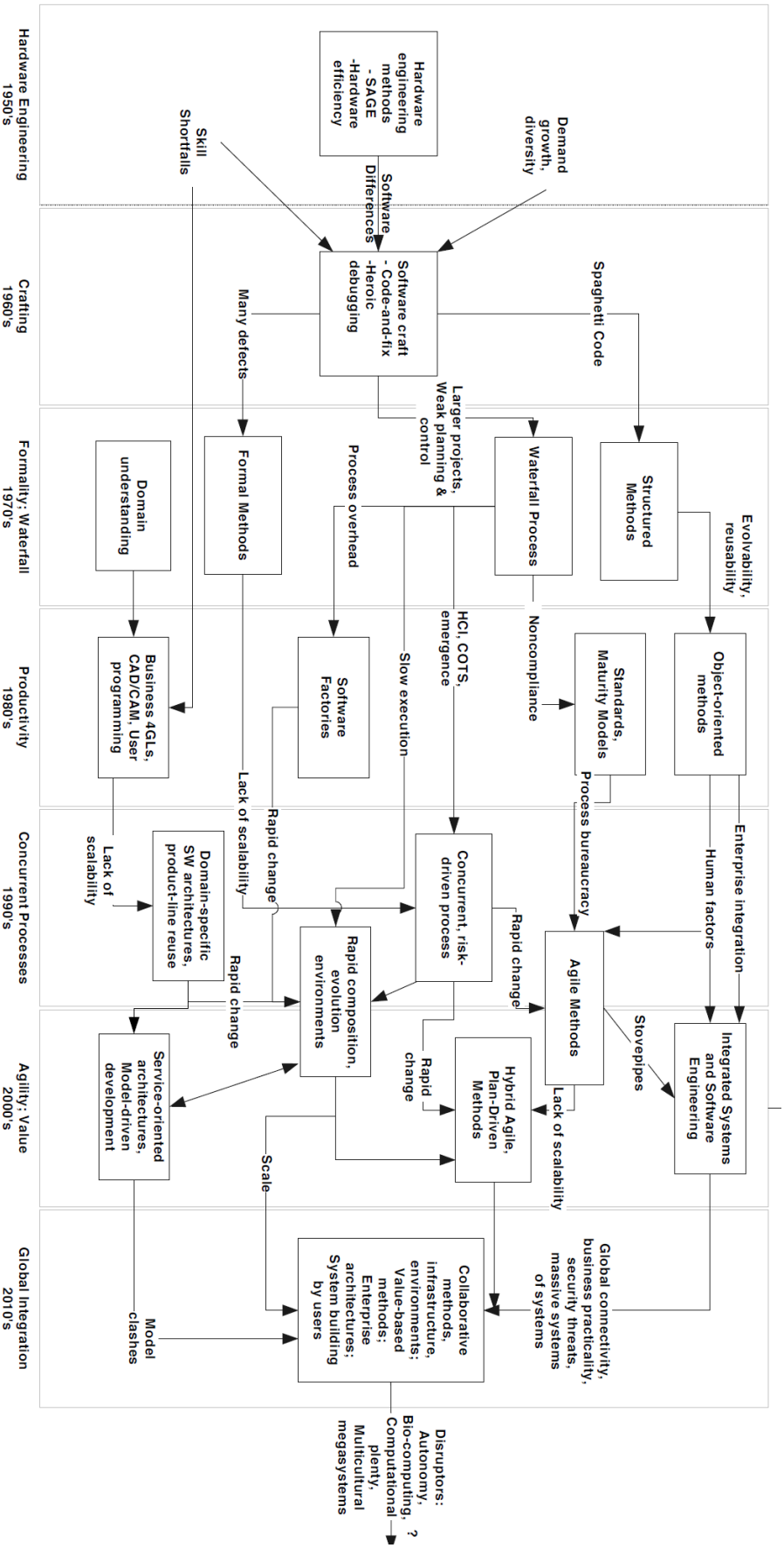


Figure 2 A full range of software engineering trends (Boehm, 2006)

3.2 So what is Waterfall, and what are its characteristics?

Also known as the 'Traditional Method', Waterfall is a sequential Software Delivery Life Cycle model first formally described in 1970 by Royce (1970) for managing large software developments. Projects are managed steadily downward (as a waterfall) through a set of phases starting from the conception phase of gathering requirements to the analysis of the project and a big design upfront, the actual coding, until the testing and maintenance phase.

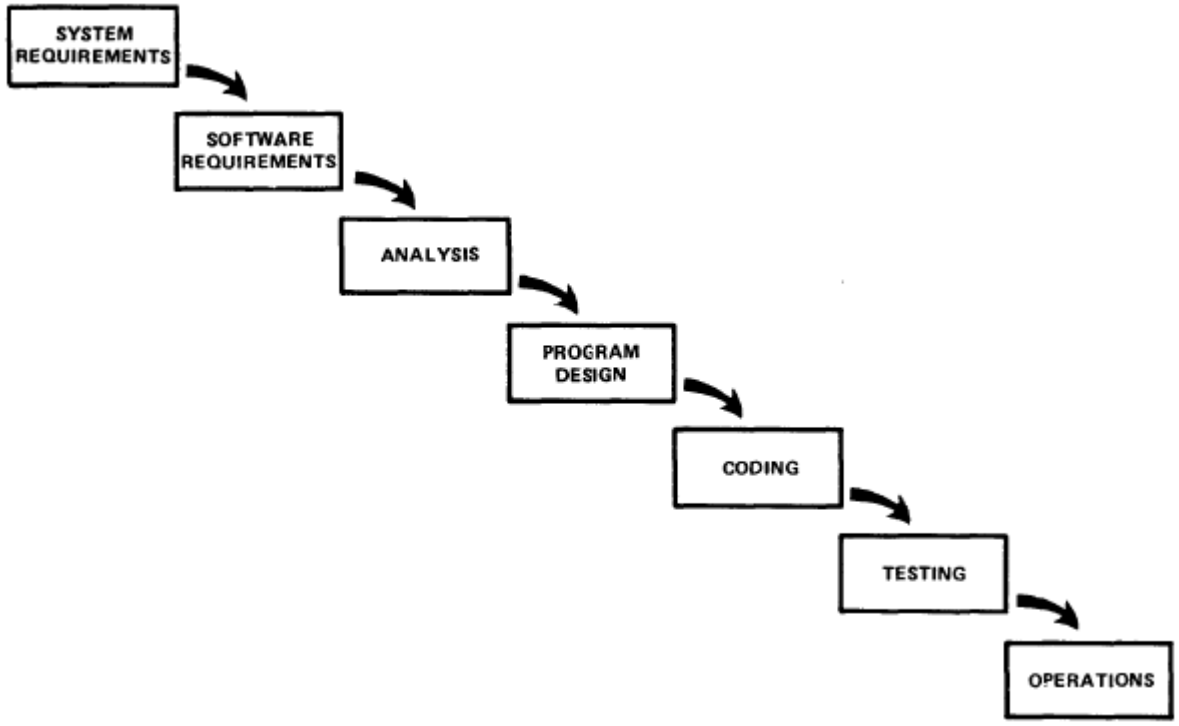


Figure 3 The Traditional Waterfall model by Royce (1970)

In the first stage of the project detailed **System Requirements** are gathered for the 'to be' system as a whole, not solely for the software to be built, but also for the hardware needed and the process to be improved. Examples are the decision about which programming language should be used, which software tools, what hardware we need to support our system…

Specific **Software Requirements** are gathered regarding the functionalities of the software to be made, it establishes which system requirements have an impact on the software. E.g. the integration needs with third-party solutions or legacy, the user interface requirements and also non-functional requirements such as performance, security…

Analysis is done in order to construct a general architectural design of how the software is to meet its requirements. The major building blocks are defined together with their interactions in between, also the main interactions with external solutions are described and clarified.

In the **Program Design** stage a more detailed specification is created about each functionality or interaction to be implemented in the system.

During the **Coding** phase, the actual implementation of the features of the solution is done, based on the component specifications and the preliminary analysis.

When all features are implemented, in the **Testing** stage it is determined whether all functionalities are implemented conform the initial specified requirements, found errors in the code are marked as bugs and subsequently solved before the launch of the project.

After the software has been released the project goes into **Operations** mode, where a follow-up is done of problems occurred during usage of the software and change request that are validated are processed as enhancements of the software.

The Traditional method is based on a **sequential model**, following a linear path (Mahalakshmi & Sundaranjan, 2013). Each phase in the project is finished before starting the next one. This makes the model very easy to understand and implement, which could explain the fact that it is widely known and implemented in the industry (Munassar & Govardhan, 2010).

The central idea of a "**Big Design Upfront**" characterizes the Waterfall method, as it emphasizes that the time spent on designing and specifying all features upfront will lead to greater economy later in the process, for a bug found in the early stages is cheaper in money, effort, and time to fix than the same bug found later on in the process (McConnel, 1996).

One of the major advantages of Waterfall stated by (Khurana & Gupta, 2012) is its **rigidity**, making it rather easy to manage. Each phase comes with prescriptive deliverables, a set of milestones and defines specific feedback loops. Munassar & Govardhan (2010) state that this approach reinforces the good habits of define-before-design and design-before-code mentality, Mahalakshmi & Sundaranjan (2010) see this document-driven approach as an advantage for assuring quality of development.

This typical characteristics make the model to have proven its value for solving many software engineering problems (Khurana & Gupta, 2012). However, it also includes features that aren't sufficient for each software engineering context (Boehm, 1988). Its main disadvantage is the sequential approach of this SDLC. No working software is gained until late in the stage of the project, early design mistakes are therefore very hard to fix. This does not only bring high amounts of risk and uncertainty, but makes it a very poor model for large complex project with ever changing requirements. Khurana & Gupta make it even more explicit by stating: "Changing the scope can kill the project". Munassar and Govardhan comply with these statements, claiming that it is unrealistic to expect accurate requirements so early in the project and that change is expensive in this model, as you have to be "swimming upstream".

Though the prescriptive nature of the Traditional method offers guidance, which can be seen as an advantage in the case of larger and mature projects, for some projects this can also bring a lot of

administrative overhead and unnecessary governance issues, making it costly for small teams and projects (Munassar & Govardhan, 2010).

The Waterfall model can be seen as a more individualistic approach, where the knowledge is centralized in the hands of the Analysts ("experts"), using structured analysis for designing software (Beaumont, 2012). This created an unnatural division between the analysis team and the development team, often leading to inadequate specifications of the to-be solution and a poor quality of the expected system, this lack of communication is also one of the major shortcomings according to Royce (1970).

According to Peter Stevens (2013) the reason why Waterfall just isn't working for producing software is because it tries to adapt the management structures that were created to manage the automobile industry in the early 20th century. "Assembly lines assemble components in steps defined by the engineers and managers who watch over unskilled and recalcitrant workers. Companies seek to maximize outputs and profits, and minimize costs", states Stevens. The underlying principles are defined by Steve Denning (2010) by the principles that managers are controllers, coordination is done through bureaucracy, utilization has to be optimized and communication is primarily top-down broadcast.

The shortcoming of the ability to rapidly respond to change made the software engineering community evolve towards new methods for dealing with these ever changing needs. This is how the agile movement arose in 2001, by a group of experts on the matter stating the Agile Manifesto.

3.3 Does Agile meet up where Waterfall fails?

This is what I see when having delivered software project for half my life. Many project teams take a lot of time, too long, building software, making it rather expensive, before validating it in the market. By the time the software was in the market no one was using it because by that time the customers' expectations had changed, the product was no longer fitting the user's needs. So how to ensure that a project is still relevant at completion, taking into account these changing needs of our valued customers?

This is where agile software engineering meets up where the Traditional method fails. During the three decennia after the first critique of Royce, the Agile Community tried to improve their software engineering methods. Methods and techniques such as Rational Unified Process, Lean Thinking and eXtreme Programming introduced new iterative and incremental ways of working. This eventually led for the Agile Community to get together in 2001 and make a bold statement in 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.

Table 1 The agile manifesto (2001)

The agile development methodology allows the direction of a project to change its course throughout the development life cycle. This is done by dividing the project into regular bits of work-to-be-done, also known as sprints or iterations. At the end of each such a sprint the project team commits itself to present a potentially shippable product increment. This is why agile development is called 'iterative' or 'incremental', instead of the sequential approach that is provided by the traditional method. By continuously looping through each phase of the development life cycle of each sprint (requirements, design, coding, testing …) it is possible to rapidly change its course every two to four weeks, i.e. the duration of a typical iteration, allowing to keep up with the changing requirements of the business.

The quality of the delivered software is ensured in the agile methodology through the close collaboration with the customer itself. By showing working software regularly and embracing a changing scope even late in the project, the customer can align the projects features with its ever changing needs, bringing the quality of the software to a higher level.

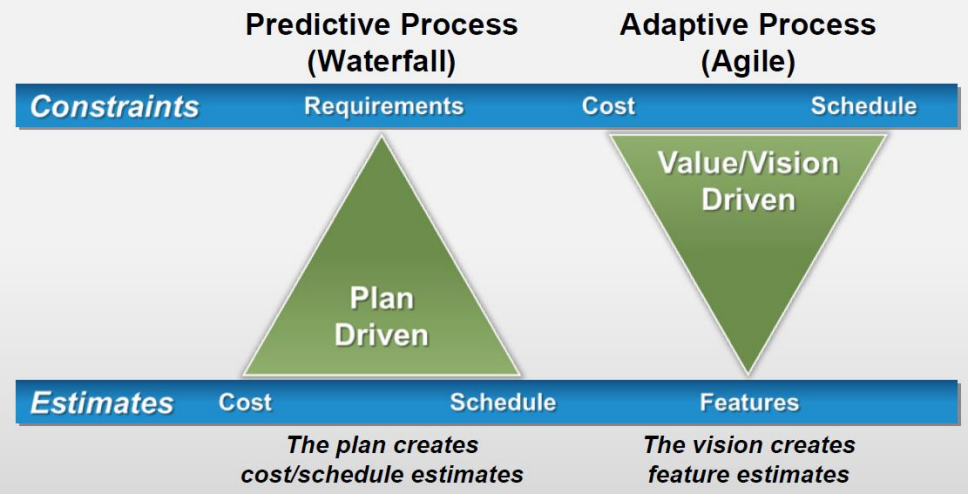


Figure 4 Agile turns Tradition Upside Down (Leffingwell, 2009)

While the waterfall method provided a very descriptive model containing a specific process with a set of tools, the agile community values a more adaptive way of working, preliminary depending on the interaction between motivated individuals in a self-organizing team, who work together towards a goal, have the ability and authority to take decisions and readily adapt to changing demands (Mittal, 2013).

This adaptive approach is also evident when it comes to documentation. Whereas the traditional approach promoted a big design upfront, the agile method values a more continuous design during each iterative product increment, where interaction is more important than thorough documentation. This is many times confused with the idea that 'no documentation is needed at all', but experience shows that this idea is far from true. However, the adaptive nature of the agile methodology can also be seen as a shortcoming, as people by nature are in need of guidance.

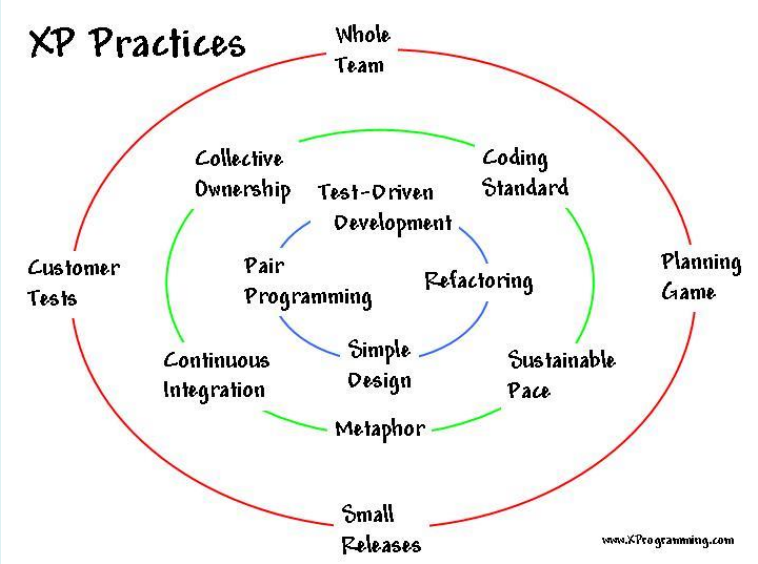
Learning is also an important aspect of Agile. The methodology recognizes the need of learning, this towards the technological skills as well as towards the domain where the software is built-in. To support these learning cycles, Scrum for instance defines the Sprint Retrospective ceremony where the team gets together after the sprint to reflect on what went well and where there is room for improvement. These 'lessons learned' moments also used in other agile practices, are key to improving the teams skills, eventually improving software quality.

Glossary of methodologies, practices & techniques

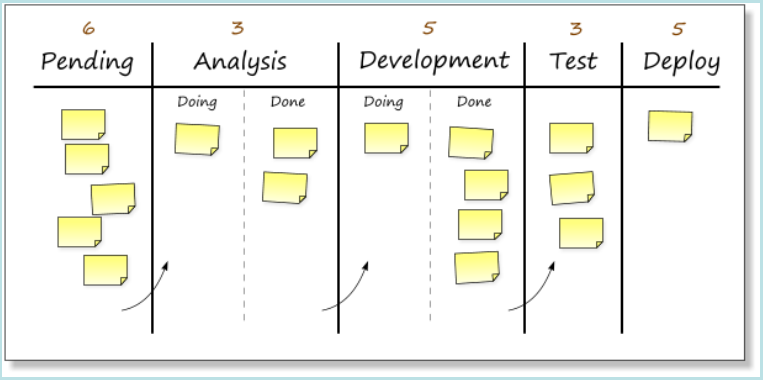
This emerged in a wide variety of methodologies from Agile Modeling to Scrumban. The table below provides a wide glossary of methodologies, practices & techniques in the industry, ordered alphabetically.

Agile Unified Process	Agile Unified Process is a simplified version of the Rational Unified Process (RUP). It describes a simple, easy to understand approach to developing business application software using agile techniques and concepts yet still remaining true to the RUP. Ambler tried to keep the Agile UP as simple as possible, both in its approach and in its description. The descriptions are simple and to the point, with links to details (on the web) if you want them. The approach applies agile techniques include test driven development (TDD), Agile Model Driven Development (AMDD), agile change management, and database refactoring to improve your productivity. Source: http://www.ambysoft.com/unifiedprocess/agileUP.html
Crystal	The Crystal methodology is one of the most lightweight, adaptable approaches to software development. Crystal is actually comprised of a family of agile methodologies such as Crystal Clear, Crystal

	Yellow, Crystal Orange and others, whose unique characteristics are driven by several factors such as team size, system criticality, and project priorities. This Crystal family addresses the realization that each project may require a slightly tailored set of policies, practices, and processes in order to meet the project's unique characteristics. Several of the key tenets of Crystal include teamwork, communication, and simplicity, as well as reflection to frequently adjust and improve the process. Like other agile methodologies, Crystal promotes early, frequent delivery of working software, high user involvement, adaptability, and the removal of bureaucracy or distractions. Alistair Cockburn, the originator of Crystal, has released a book, "Crystal Clear: A Human-Powered Methodology for Small Teams". Source: http://www.versionone.com/Agile101/Agile-Development-Methodologies-Scrum-Kanban-Lean-XP/
Dynamic Systems Development Method	DSDM is a robust Agile project management and delivery framework that delivers the right solution at the right time. DSDM has been for many years the leading, proven Agile approach, providing governance and rigor along with the agility and flexibility demanded by organizations today. The approach is the culmination of practitioners' experience drawn from a wide range of public and private sector projects over nearly two decades. The DSDM Philosophy is that any project must be aligned to clearly defined strategic goals and focus upon early delivery of real benefits to the business. DSDM is vendor-independent, covers the entire lifecycle of a project and provides good practice guidance for on time, in budget delivery of projects – with proven scalability to address projects of all sizes and for any business sector. DSDM advocates the use of several proven techniques, including: Facilitated Workshops Modelling and Iterative Development MoSCoW Prioritization Time boxing

	DSDM is designed to be easily tailored and used in conjunction with traditional methods such as PRINCE2® or to complement other Agile approaches such as Scrum. Source: http://www.dsdm.org/content/what-dsdm
Extreme Programming	Extreme Programming is a discipline of software development based on values of simplicity, communication, feedback, courage, and respect. It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situation. This allows for the development team to have a very focused goal in which they can work to, by doing small, iterative and incremental releases.  Source: http://xprogramming.com/what-is-extreme-programming/
Feature Driven Development	Feature-driven design (FDD) is an iterative and incremental software development process. It is one of a number of Agile methods for developing software and forms part of the Agile Alliance. FDD blends a number of industry-recognized good practices into a cohesive whole. These practices are all driven from a client-valued functionality (feature) perspective. Its main purpose is to deliver tangible, working software repeatedly in a timely manner. Source: http://en.wikipedia.org/wiki/Feature-driven_development
Graphical Systems Design	Competing in today's global economy requires companies to rapidly enter the market with innovative products that offer increased

	functionality and operate flawlessly. The National Instruments graphical system design approach for test, control, and embedded design meets this need by providing a unified platform for designing, prototyping, and deploying applications. The NI platform empowers engineers to integrate real-world signals sooner for earlier error detection, reuse code for maximum efficiency, benefit immediately from advances in computing technology, and optimize system performance in a way that outpaces traditional design methodologies.  Source: http://www.ni.com/company/standardize.htm
Kanban	Kanban is a new technique for managing a software development process in a highly efficient way. Kanban underpins Toyota's "just-in-time" (JIT) production system. Although producing software is a creative activity and therefore different to mass-producing cars, the underlying mechanism for managing the production line can still be applied. A software development process can be thought of as a pipeline with feature requests entering one end and improved software emerging from the other end. A bottleneck in a pipeline restricts flow. The throughput of the pipeline as a whole is limited to the throughput of the bottleneck. Kanban is incredibly simple, but at the same time incredibly powerful. In its simplest incarnation, a Kanban system consists of a big board on the wall with cards or sticky notes placed in columns with numbers at the top. The cards represent work items as they flow through the development process represented by the columns. The numbers at the top of each column are limits on the number of cards allowed in each column. The limits are the critical difference between a Kanban board and any other visual storyboard. Limiting the amount of work-in-progress (WIP), at each step in the process, prevents overproduction and reveals bottlenecks dynamically so that you can address them before they get out of hand.



Source: http://www.kanbanblog.com/explained/

Lean Software Development

Lean software development (LSD) is a translation of lean manufacturing and lean IT principles and practices to the software development domain. Adapted from the Toyota Production System, a pro-lean subculture is emerging from within the Agile community. It is based on the lean principles:

- Optimize the whole
- Energize workers
- Eliminate waste
- Learn first
- Deliver fast
- Build quality in
- Keep getting better

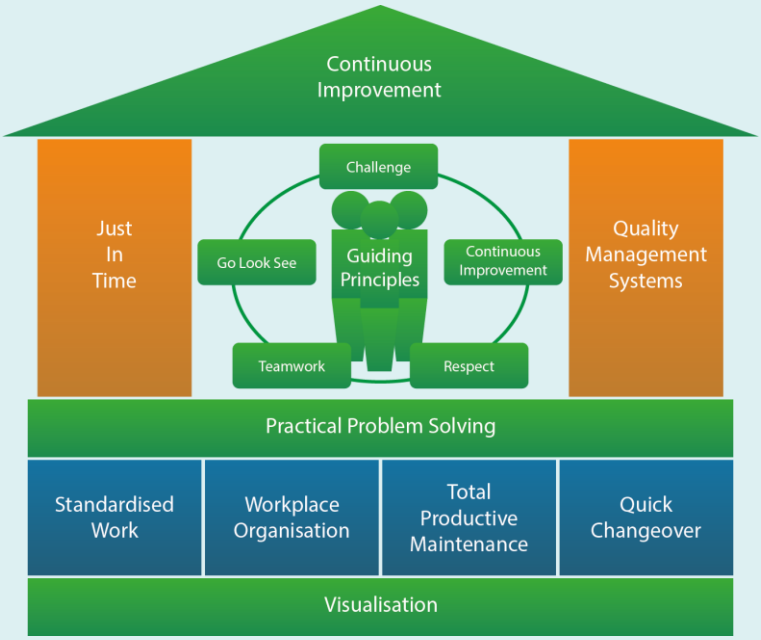
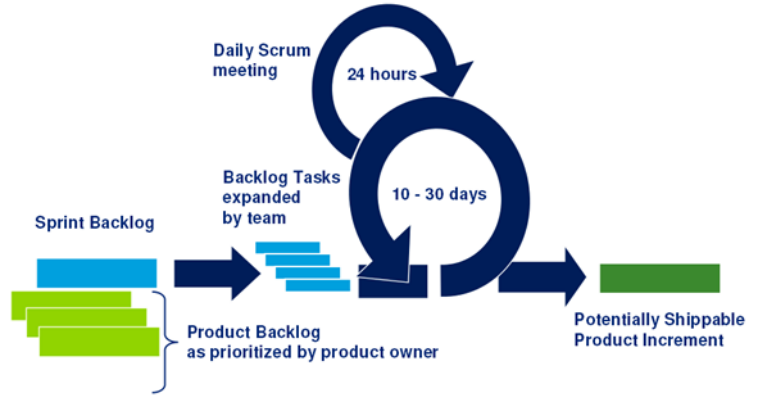


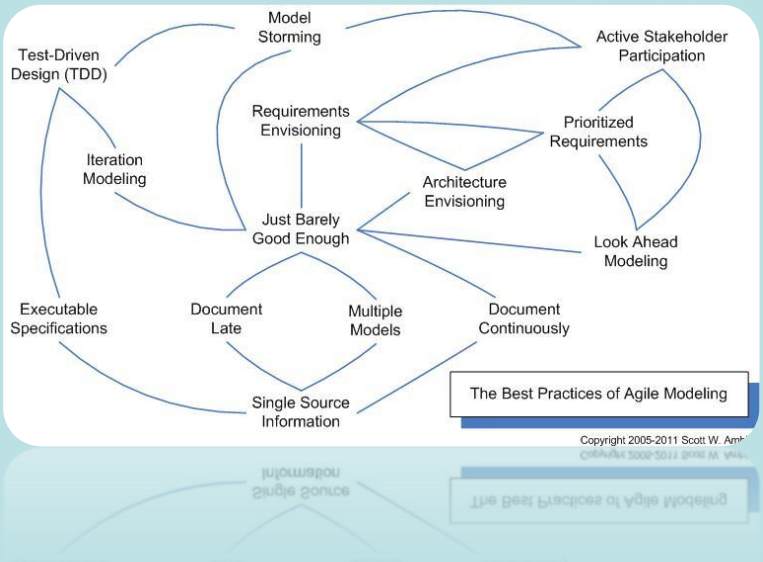
Figure 5 The Lean Temple

	Source: http://www.poppendieck.com/
Scrum	Scrum is a way for teams to work together to develop a product. Product development, using Scrum, occurs in small pieces, with each piece building upon previously created pieces. Building products one small piece at a time encourages creativity and enables teams to respond to feedback and change, to build exactly and only what is needed. Building complex products for customers is an inherently difficult task. Scrum provides structure to allow teams to deal with that difficulty. However, the fundamental process is incredibly simple, and at its core is governed by 3 primary roles. • Product Owners determine what needs to be built in the next 30 days or less. • Development Teams build what is needed in 30 days (or less), and then demonstrate what they have built. Based on this demonstration, the Product Owner determines what to build next. • Scrum Masters ensure this process happens as smoothly as possible, and continually help improve the process, the team and the product being created. While this is an incredibly simplified view of how Scrum works, it captures the essence of this highly productive approach for team collaboration and product development.  *Figure 6 The Scrum Sprint Cycle* Source: https://www.scrum.org/Resources/What-is-Scrum
Scrum-ban	We know Scrum and Kanban as flavors of Agile. Scrum is best suited for products and development projects. Kanban is best for

	production support. We use Scrumban - which combines the best features of both - for maintenance projects. Scrumban is becoming very popular these days in service industries, where we have both development and maintenance projects. • Use the prescriptive nature of Scrum to be Agile. • Use the process improvement of Kanban to allow the team to continually improve its process. Source: http://www.solutionsiq.com/resources/agileiq-blog/bid/87799/What-is-Scrumban

Table 2 Agile Process Methods

Within these agile methodologies or approaches, also a wide variety of agile practices are used, the most common practices are described in the table below.

Agile Modeling	Agile Modeling (AM) is a practice-based methodology for effective modeling and documentation of software-based systems. At a high level AM is a collection of good practices, depicted in the pattern language map below. At a more detailed level AM is a collection of values, principles, and practices for modeling software that can be applied on a software development project in an effective and light-weight manner.  Source: http://www.agilemodeling.com/
Agile Testing	Agile testing is a software testing practice that follows the rules of the agile manifesto, treating software development as the customer of testing.

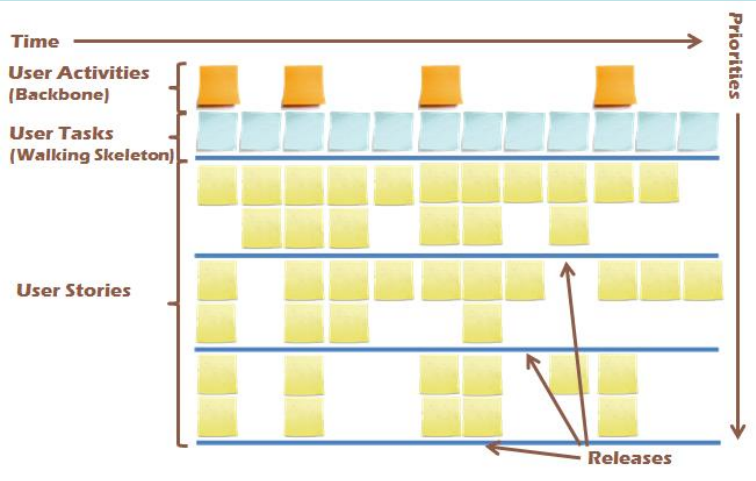
	Agile testing involves testing from the customer perspective as early as possible, testing early and often as code becomes available and stable enough from module/unit level testing. Since working increments of the software are released very often in agile software development there is also a need to test often. This is often done by using automated acceptance testing to minimize the amount of manual labor. Doing only manual testing in agile development would likely result in either buggy software or slipping schedules because it would most often not be possible to test the whole software manually before every release. Source: http://agiletesting.com.au/
Backlogs	The agile product backlog in Scrum is a prioritized features list, containing short descriptions of all functionality desired in the product. When applying Scrum, it's not necessary to start a project with a lengthy, upfront effort to document all requirements. Typically, a Scrum team and its product owner begin by writing down everything they can think of for agile backlog prioritization. This agile product backlog is almost always more than enough for a first sprint. The Scrum product backlog is then allowed to grow and change as more is learned about the product and its customers. The sprint backlog is a list of tasks identified by the Scrum team to be completed during the Scrum sprint. During the sprint planning meeting, the team selects some number of product backlog items, usually in the form of user stories, and identifies the tasks necessary to complete each user story. Most teams also estimate how many hours each task will take someone on the team to complete. Source: http://www.mountaingoatsoftware.com/agile/scrum/product-backlog
Behavior Driven Development	BDD creates more harmony between the user story practices from Scrum and the Test-Driven Development practices from XP. The user stories practices represent analysis and specification in agile projects and Test-Driven Development represents software design. The resulting increased fluidity between analysis and design is brought about by a stronger focus on acceptance criteria. Acceptance criteria are those parts of requirements that specify when a requirement has been satisfied by shippable software. Source: http://www.codemag.com/article/0805061
Burn down Charting	Regularly updated display of project progress (usually in terms of decreasing remaining effort) (Moran, 2014).

Cross-functional Team	A cross functional group of individuals that has the ability and authority to define (elaborate and prioritize requirements and design the solution), build (write the code and tests that implement the solution), and test (run the test cases and validate the solution against the defined requirements), all in a short iteration time box. Source: http://scaledagileframework.com/agile-teams/
Continuous Integration	Teams practicing continuous integration seek two objectives: minimize the duration and effort required by "each" integration episode & be able to deliver "at any moment" a product version suitable for release. In practice, this dual objective requires an integration procedure which is "reproducible" at the very least, and in fact largely "automated". This is achieved through version control tools, team policies and conventions, and tools specifically designed to help achieve continuous integration. Source: http://guide.agilealliance.org/guide/ci.html#sthash.qocrKfSZ.dpuf
Daily Standup	Daily meeting of all team members focusing on what has been achieved since the last meeting, what is planned that day and what is blocking work (Moran, 2014).
Definition of Done	The definition of done is a practice that is used by agile teams to document when a typical work items can be seen as fully completed. It typically consists of steps that the development team does for each task to assure the quality of the work delivered, such as unit testing, system testing, merging, building, staging, product owner checks, deployment, …
Domain Driven Design	Domain-driven design is not a technology or a methodology. It is a way of thinking and a set of priorities, aimed at accelerating software projects that have to deal with complicated domains. It is the practice of laying focus on the domain, and the business logic behind it, making sure that you model out more complex designs, in order to build long lasting quality. To accomplish that goal, teams need an extensive set of design practices, techniques and principles. Refining and applying these techniques will be the subject of discussion for this site, generally starting from the language of patterns laid out in Domain-Driven Design, by Eric Evans. Source: http://dddcommunity.org/learning-ddd/what_is_ddd/

Heterogeneous Small Teams	Formation of teams, with preferably five to nine co-located members (Miller, 1956) comprising of business representatives and solution developers, with a preference towards generalists over specialists. (Moran, 2014)
Information Radiators	"Information radiator" is the generic term for any of a number of handwritten, drawn, printed or electronic displays which a team places in a highly visible location, so that all team members as well as passers-by can see the latest information at a glance: count of automated tests, velocity, incident reports, continuous integration status, and so on. Examples are the Scrum Board, Kanban Board, Task Board, Burn down Chart, … Source: http://guide.agilealliance.org/guide/information-radiator.html#sthash.jlxCrcWw.dpuf
Iterative and Incremental Development	Moran (2014) defines Iterative development as the traversal of an entire SDLC within a fixed timeframe with the aim of producing tested and stable code that contributes to the evolving solution. Incremental delivery is also defined by Moran as releasing partial evolving solutions into the production environment during the project lifecycle. Nearly all Agile projects are incremental as well as iterative. However, it is possible to use iterative strategies which are not also incremental; for instance, a "build it twice" strategy in which one first creates a throwaway prototype to gather user feedback, then uses insights from that experience to build the "real thing". Prototyping is necessarily an iterative strategy, and may have been a precursor to the development of iterative software development ideas. Source: http://guide.agilealliance.org/guide/iterative.html#sthash.Y0bNUuKR.dpuf
MoSCoW Prioritization	Classification of priorities in terms of MUST, SHOULD, COULD and WON'T (in respect of the current iteration!) and the accommodation of contingency in planning (Moran, 2014).
Pair Programming	Pair programming consists of two programmers sharing a single workstation (one screen, keyboard and mouse among the pair). The programmer at the keyboard is usually called the "driver", the other, also actively involved in the programming task but focusing more on overall direction is the "navigator"; it is expected that the programmers swap roles every few minutes or so. Source: http://guide.agilealliance.org/guide/pairing.html#sthash.yoXCLb1M.dpuf

Planning Poker	A playful approach to estimation, used by many Agile teams. The team meets in presence of the customer or Product Owner. Around the table, each team member holds a set of playing cards, bearing numerical values appropriate for points estimation of a user story. The Product Owner briefly states the intent and value of a story. Each member of the development team silently picks an estimate and readies the corresponding card, face down. When everyone has taken their pick, the cards are turned face up and the estimates are read aloud. The two (or more) team members who gave the high and low estimate justify their reasoning. After brief discussion, the team may seek convergence toward a consensus estimate by playing one or more further rounds. Source: http://guide.agilealliance.org/guide/poker.html#sthash.vZxuTGYa.dpuf
Product Vision	Formulation of a simple evocative statement that clearly states the purpose of the product which the project is endeavoring to create (Moran, 2014).
Prototyping	Creating a (possible throwaway) mock to trial the design of a solution or to better understand the problem (Moran, 2014).
Refactoring	Refactoring consists of improving the internal structure of an existing program's source code, while preserving its external behavior. The noun "refactoring" refers to one particular behavior-preserving transformation, such as "Extract Method" or "Introduce Parameter". Source: http://guide.agilealliance.org/guide/refactoring.html#sthash.W1EqQdz1.dpuf
Retrospective	Gathering of all team members to discuss lesson learned during the latest iteration with a view to identify and act on possible process improvements (Moran, 2014).
Scrum Meetings	In Scrum, on each day of a sprint, the team holds a daily scrum meeting called the "daily scrum." Meetings are typically held in the same location and at the same time each day. Ideally, a daily scrum meeting is held in the morning, as it helps set the context for the coming day's work. These scrum meetings are strictly time-boxed to 15 minutes. This keeps the discussion brisk but relevant. Source: http://www.mountaingoatsoftware.com/agile/scrum/daily-scrum/
Story-driven Modeling	Current object-oriented modeling methods focus on the specification of the static structure of software objects and their interaction at runtime. A major deficiency

	of these methods is that they do not provide sufficient means to specify the behavior of object structures. In this paper we propose a novel high-level, graphical method called Story Driven modeling (SDM). SDM employs so called story boards to analyze the dynamics of object structures as sequences of graphical snap shots for sample scenarios. A major benefit of this approach is that story boards allow to develop and illustrate design and ideas of a systems object structure and central mechanisms in a simple visual notation. On the other hand, story boards have well-defined syntax and semantics that gives way to semi-automatic derivation of subsequent specifications like e.g. the static class hierarchy and dynamic operations on object structures. For the latter, SDM employs a high-level, graphical formalism called story diagrams, which is based on the theory of programmed graph rewriting systems. In this paper we illustrate SDM with a sample case study which is the development of the bus route information system of Paderborn. Source: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.21.8341
Test-driven Development	Test-driven development (TDD) (Beck 2003; Astels 2003), is an evolutionary approach to development which combines test-first development where you write a test before you write just enough production code to fulfill that test and refactoring. What is the primary goal of TDD? One view is the goal of TDD is specification and not validation (Martin, Newkirk, and Kess 2003). In other words, it's one way to think through your requirements or design before your write your functional code (implying that TDD is both an important agile requirements and agile design technique). Another view is that TDD is a programming technique. As Ron Jeffries likes to say, the goal of TDD is to write clean code that works. I think that there is merit in both arguments, although I lean towards the specification view, but I leave it for you to decide. Source: http://www.agiledata.org/essays/tdd.html#sthash.Ts4II8Mz.dpuf
Time boxing	A time box is a previously agreed period of time during which a person or a team works steadily towards completion of some goal. Rather than allow work to continue until the goal is reached, and evaluating the time taken, the time box approach consists of stopping work when the time limit is reached and evaluating what was accomplished. The critical rule of time boxed work is that work should stop at the end of the time box, and review progress: has the goal been met, or partially met if it included multiple tasks?

	Source: http://guide.agilealliance.org/guide/timebox.html#sthash.WU932Kmu.dpuf
User Story Mapping	In consultation with the customer or product owner, the team divides up the work to be done into functional increments called "user stories". Moran defines a User Story as a formulation in simple narrative form of requirements together with statements of what constitutes successful implementation (definition of done). Each user story is expected to yield, once implemented, a contribution to the value of the overall product, irrespective of the order of implementation; these and other assumptions as to the nature of user stories are captured by the INVEST formula: A good user story should be: • "I" ndependent (of all others) • "N" egotiable (not a specific contract for features) • "V" aluable (or vertical) • "E" stimable (to a good approximation) • "S" mall (so as to fit within an iteration) • "T" estable (in principle, even if there isn't a test for it yet) To make these assumptions tangible, user stores are reified into a physical form: an index card or sticky note, on which a brief descriptive sentence is written to serve as a reminder of its value. This emphasizes the "atomic" nature of user stories and encourages direct physical manipulation: for instance, decisions about scheduling are made by physically moving around these "story cards".  *Figure 7 User Story Map* Source: http://guide.agilealliance.org/guide/user-stories.html#sthash.7FYwwb6Z.dpuf

Velocity Tracking	Velocity is an extremely simple, powerful method for accurately measuring the rate at which scrum development teams consistently deliver business value. To calculate velocity of your agile team, simply add up the estimates of the features, user stories, requirements or backlog items successfully delivered in an iteration. http://www.versionone.com/Agile101/Agile-Scrum-Velocity/

Table 3 Notable Agile Practices

As these practices are built into practice every day at agile teams, many of these practices have their focus on the team itself, and it has proven to be working for small teams with a limited scope. When enrolling these practices at a larger scale, we are currently facing some problems. Therefore it is important that we get a more clear understanding about when a typical project is 'scaling up'. These scaling factors are discussed in the next chapter.

4 Agile Scaling Challenges

When it comes to discussing the challenges of delivering agile at scale, we will introduce the Agile Scaling Model (Ambler S. W., Scaling Agile: An Executive Guide, 2010). Ambler defines the Agile Scaling Model as a contextual framework for effective adoption and tailoring of agile practices to meet the unique challenges faced by a system delivery team of any size.

In the Agile Scaling Model (Figure 8 Agile Scaling Model), Ambler defines three levels of agility. The first agility level is **Core Agile Development**, it encapsulates the broad spectrum of mainstream strategies such as XP and Scrum, which many organizations have adopted and been successful with them. When looking at these present agile techniques, we find that they are based on a similar philosophy. They all focus on **delivering value**, as stated in the Agile Manifesto (in Section 3.3). They also find it important to have **self-organizing teams**. These self-organizing teams receive a prioritized work item list from the business and they are allowed to organize themselves to be effective. But Ambler states that this isn't enough, as the minimum level of agile software delivery that an organization should consider is **Agile Delivery**. Agile Delivery is defined by Ambler as: "a highly collaborative, evolutionary, self-organizing, and governed approach that regularly produces high-quality solutions in a cost-effective and timely manner via a risk and value driven lifecycle." Next to the core agile development techniques, Agile Delivery offers a project management framework with explicit **project phases** (the full delivery life cycle with an inception phase, construction & transition phase – more on that in Section 5.1), whereas within core agile development there is more focus on the development itself (which, in disciplined agile delivery is only the construction phase).

The third level of agility is Agility at Scale. When thinking about "scaling", we often think only about large development teams. Although this is off course true, it is not limited to only team size, as Ambler defines 7 extra 'scaling factors'. These in total 8 scaling factors (team size, geographic distribution, regulatory compliance, domain complexity, organizational distribution, technical complexity, organizational complexity and enterprise discipline) are discussed further in this chapter.

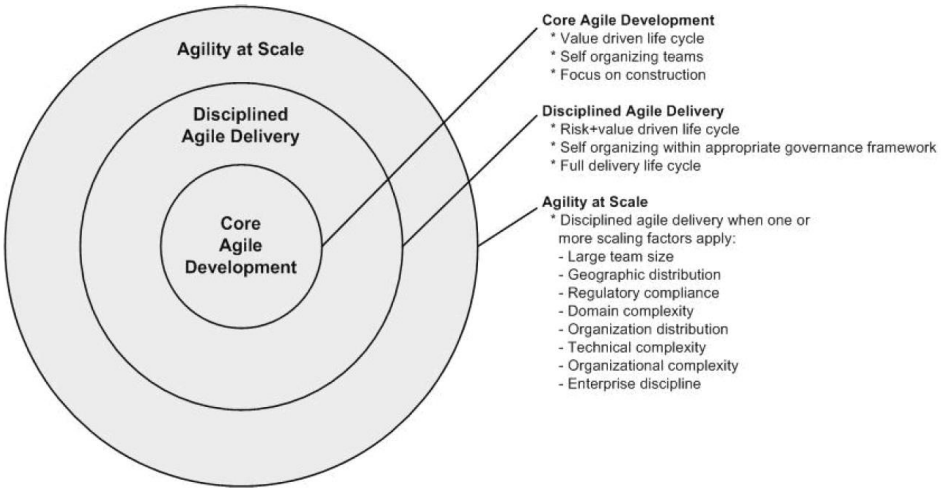


Figure 8 Agile Scaling Model (Ambler S. W., Scaling Agile: An Executive Guide, 2010)

4.1 From Core Agile Development towards Agile Delivery

4.1.1 Towards a full delivery life cycle

The agile scaling model discusses three important prerequisites that an organization should have set into place when wanting to achieve agility at scale. At first the enterprise has to evolve towards a **full delivery life cycle**. Whereas core agile practices focus purely on delivering "working software" (Agile Manifesto, 2001), Ambler states that an enterprise should deliver "working solutions", where the organization as a whole is ready to deliver the solution to the customer. This includes for instance the training of the service desk and sales teams, but also the preparation of the production environment for the solution to run on, the preparation of training material for the users, etc.

4.1.2 Risk & Value Driven Life Cycle

Ambler further addresses the need of being **risk & value-driven** as an organization, whereas core agile practices such as SCRUM and XP are more value-driven. Instead of focusing on delivering features with the highest value, you also take features into accounting which define a high risk, as you want to continuously try to reduce the project risks throughout the delivery life cycle.

A value-driven life cycle already addresses important value-related risks such as the risk of not delivering at all, the risk of delivering wrong functionality and political risks resulting from the lack of visibility into what the team is producing. These risks can be mitigated by:

- **Developing potentially consumable solutions**, extending SCRUM's potentially shippable product to address usability concerns (the consumable aspect)
- **Iteration demos**, to obtain feedback from the stakeholders and improve the solution and indicate the health of the project
- **Active stakeholder participation**, involved stakeholders within the team through the entire iteration and not only on the demo's reduces both delivery and functionality risks

Ambler & Lines (2012) further state next to addressing risk, there are some key milestones to be defined to further reduce risk:

- **Stakeholder consensus** before developing the project
- **Proven architecture** in the early development reduces the uncertainties about the project
- **Continued viability** checks of the project during its execution
- **Sufficient functionality** needed, but not more, to go to a release
- **Production readiness** needs to be evaluated by the key stakeholders prior to releasing
- **Delighted stakeholders** when the solution is running in production

4.1.3 Self-organizing teams with appropriate governance framework

The importance of having self-organizing teams is frequently expressed when talking about agile practices such as SCRUM and Kanban. If we take a typical startup company as an example, this can be surely pointed out as being true. But in the reality of larger organization, you have to look further than

your own team, there is an organizational context as well, as there are also other teams working on delivering solutions, coping with the same problems as you, trying to reach the same company goals and following the same enterprise strategy. It would be a waste if an organization shouldn't leverage these specific assets. Therefore Ambler & Lines (2012) state that there is a need of putting into place an appropriate governance framework to support the team by doing the following:

- **Leveraging enterprise assets** such as development guidelines, user interface standards, reusable components, …
- **Enhancing your organizational ecosystem**, e.g. working closely together with operations and support teams to speed up the release management.
- **Sharing learnings** across teams improves the overall quality.
- **Open and honest monitoring**, trust is an important base, but there is also a need for verification and guidance.

Once these practices are set into place, an organization is ready to face the challenges of agility at scale. When talking about scaling, it is important that we define what we understand what we mean with that. Which factors can cause a project to fail when if they exist, hence, preventing an organization to achieve agility at scale? The Agile Scaling Model (2010) addresses 8 potential scaling factors which an organization has to take into account when delivering solutions at scale, they are discussed in the following sections.

4.2 From Agile Delivery towards Agile at Scale

When talking about Agility at Scale, it is important to define what 'Scaling' means in an agile context. The first thing that comes to mind when discussing 'scaling', is off course the size of the team, as it is more straightforward for a smaller team to work together than larger teams of hundreds and even thousands of developers. But 'scale' is not only limited to the size of the teams. It is also defined by a number of other so called 'scaling factors'. Today, organizations want to work together with other companies; their teams are no longer co-located; there is a need of being compliant with governmental regulations; the technological issues are more complex; larger organizations are not so flexible as startups, they need to address these cultural issues; and they want to change their silo-mentality towards an enterprise-minded solution approach. Off course, not all factors are present to the same extent in each organization. Figure 9 shows these 8 scaling factors, where for each factor the simple and complex situation is pointed out. The more scaling factors are situated to the left, the simpler it is for a team to book successes in agile delivery. When one or more scaling factors are situated on the right, they are in the situation of agility at scale.

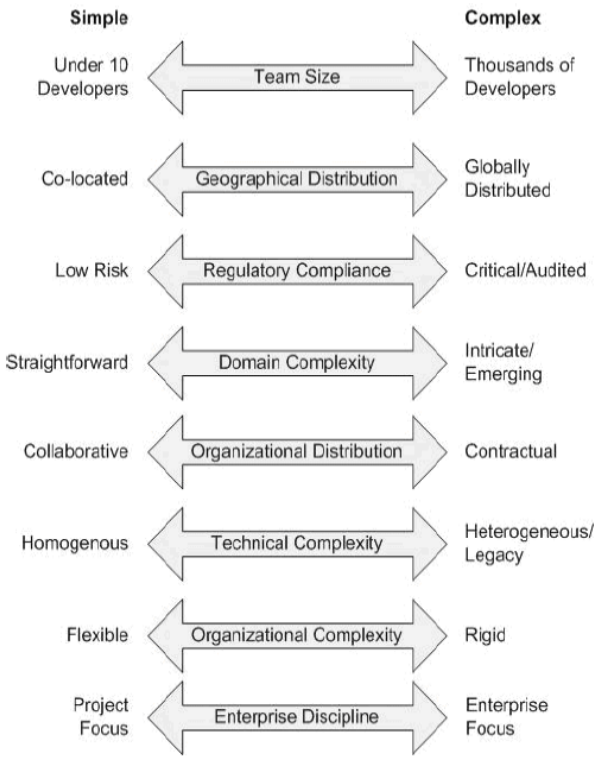


Figure 9 Potential Scaling Factors for Agile Software Delivery

In the next sections, each scaling factor is addressed into more detail, structured at three levels: the team level containing the scaling factors that can arise within the team, at product level, where scaling factors are solution or product-related, and the third part containing the enterprise-related scaling factors.

4.2.1 Team Agility

The first thing you think about when talking about 'scaling' is maybe the size of your team. Whereas agile has been proven to be working on rather small teams, agility is much harder to achieve when running an enterprise with a couple of thousand developers. Therefore (1) **Team Size** is to be seen as a scaling factor for reaching Team Agility.

Also a team that is co-located will more rapidly be able to achieve agility, as communication is a lot easier within the team. If your team is spread around the world, a lot of more effort is needed in order to collaborate effectively. This way, (2) **Geographic Distribution** can be seen as a scaling factor. A good practice is stated by Larman (Agile & Iterative Development: A Managers Guide, 2003) that you should take care that although teams are geographically distributed, all of the members within a team should be co-located.

4.2.2 Product Agility

The product you are delivering can bring along some scaling challenges as well. There is a difference in building a very straightforward solutions as a 'calculator' versus developing an enterprise-wide ERP-system. The (3) **complexity of the domain** you are working in can surely provide a scaling challenge.

Next to the complexity of the domain, we also need to have a look at the (4) **technical complexity** of the product we want to deliver. When a startup is building a new solution from scratch, this will surely be a lot easier than a large organization having to integrate with existing and evolving legacy systems.

Regulatory Compliance (5) may also cause a challenge to the product you are creating. E.g. when developing an invoicing solution for the Belgian market, you have to be compliant with the Belgian legislation when it comes to the rules for invoicing, mandatory texts when invoicing to a foreign country, tax rules to be applied for exporting to the EU, …

4.2.3 Enterprise Agility

You can't put it better than Peter Drucker when saying that "Culture eats strategy for breakfast". When you want to achieve agility on enterprise level, it is important to take a couple of challenges into account. (6) **Organizational Distribution**, of the way you compose your teams, is to be seen as one of them. When you have a very collaborative culture, it will be much easier for your enterprise to become more agile. When instead a part of the work is outsourced to external partners, or contractors are used, the distribution tends more towards a contractual culture, providing extra challenges when it comes to achieve agility at enterprise scale.

Next to the team composition, you also have to take into account the (7) **complexity of the organization**: let's talk about politics! If all your stakeholders have the same vision, shared objectives and are closely aligned, this collaborative culture will enhance the enterprise agility. When instead different subdivisions are promoting their own vision, their own way of working, this will make it a lot harder delivering software, because there might be problems regarding stakeholder consensus.

The last scaling factor that is mentioned by Ambler is (8) **Enterprise Discipline**. When teams are strongly focused on their project, they may forget to leverage the assets within the organization. Things will be built double, solutions won't be aligned, and the end-to-end customer experience may not be optimal. Instead, collaborating on a shared enterprise architecture allows teams to leverage these assets and take advantage of strategic reuse opportunities.

4.3 Is my team truly agile?

Based on the scaling factors defined in the Agile Scaling Model (Ambler S. W., Scaling Agile: An Executive Guide, 2010) and the categorization made at Team, Product & Enterprise level, we can define a visual overview of our conceptual framework towards achieving agility at scale (Figure 10).

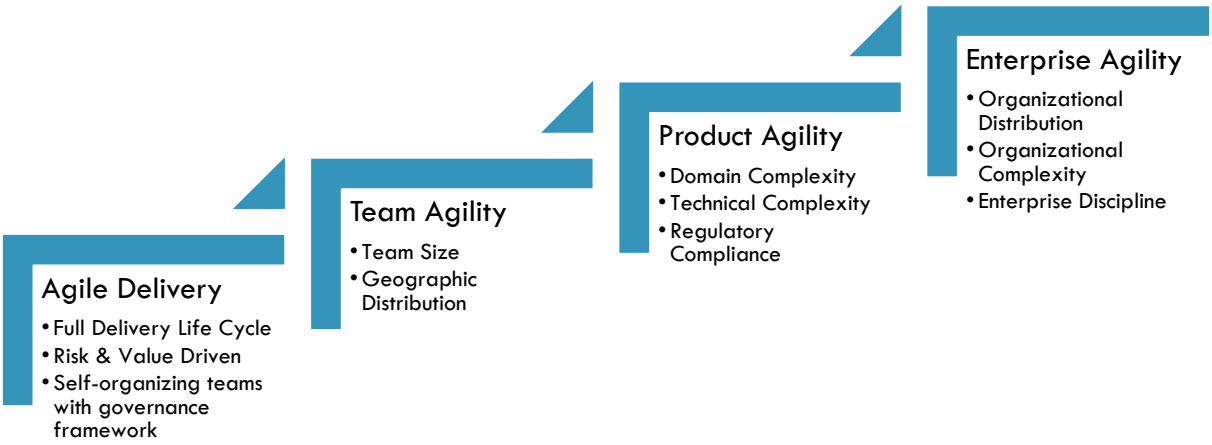


Figure 10 Agility at Scale - Conceptual Framework

Next to the fact that we will use this conceptual framework for spotting challenges in our case study later in this master thesis, we also are in need of being able to measure the 'agility' of an enterprise. Therefore Ambler (2009) defines five criteria to determine if a team is truly agile:

1. Produce working software on a regular basis
2. Do continuous regression testing, and better take a Test-Drive Development approach
3. Work closely with your stakeholders, ideally on a daily basis
4. Are self-organizing within a governance framework
5. Regularly reflect on how they work together and then act to improve their findings

Now that we have identified the challenges that an organization has to face in her quest to delivering agile solutions at a larger scale, the next chapter will discuss a set of good practices in the field of agile software delivery when it comes to scaling.

5 Agility at Scale, but how?

Many companies are facing the struggle to enroll agile delivery at a larger scale. To cope with these challenges, the agile community started developing different approaches in the form of specific frameworks that provide a set of guidelines and methods to scale agile delivery. What is interesting to know is which scaling factors do the different approaches recognize and what is their way of trying to cope with them.

This will be done for the **Disciplined Agile Delivery** (DAD) approach from Scott Ambler & Mark Lines, the **Large Scale Scrum** (LeSS) method by Craig Larman and Bass Vodde and the **Scaled Agile Framework** (SAFe) by Dean Leffingwell.

5.1 Disciplined Agile Delivery

The Disciplined Agile Delivery (DAD) process framework is a hybrid agile approach to IT solution delivery. It has a risk-value lifecycle, is goal-driven, scalable, and enterprise aware. (Ambler & Lines, 2012).

DAD doesn't provide a clear 'route to success' or a prescriptive process which an organization can follow to achieve agility at scale. Instead, Ambler & Lines focus on providing a framework of potential methods and strategies that an enterprise can put into practice when handling large scale solution delivery.

It is a hybrid framework, it adopts agile techniques and strategies such as Scrum, Extreme Programming and Kanban, allowing each organization to create a tailor made subset of the process framework to ensure a maximal fit to the situation at hand.

The DAD framework is extended to embrace a full delivery lifecycle. A phased approach is chosen, where solution delivery is subdivided into three phases: Inception, Construction and Transition. Scrum doesn't recognize the distinction between these phases, all three should be part of one sprint, delivering a potentially shippable product.

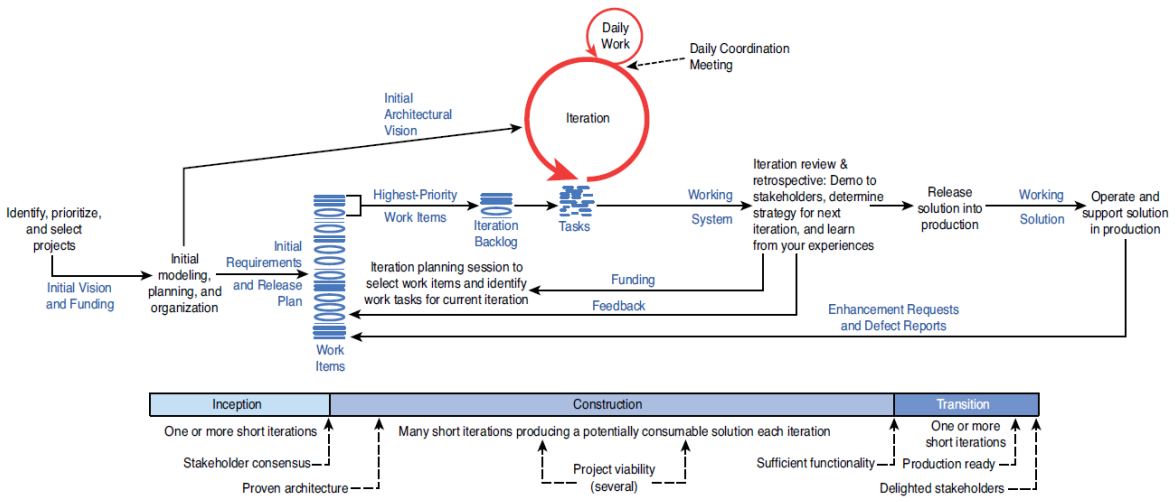


Figure 11 Scrum version of the Disciplined Agile Delivery process framework

The **Inception** phase defines goals such as the initial composing of the team, the identification of the projects vision and the alignment with the enterprise direction. Requirements are gathered and an initial technical strategy is formed. Primary risks are identified as the funding is being secured.

The actual production of a consumable solution is covered within the **Construction** phase. Changing needs of stakeholders are off course embraced at any given time, whilst the quality is maintained or improved. It is also in this phase that the development team focusses on proving the architecture as early as possible, minimizing risks, maximizing value.

The **Transition** phase covers the formalities needed to ensure that a solution is production ready and that the stakeholders are prepared to receive the solution before the actual deployment to production is done.

However the broadness of the framework leaves room for interpretation. There is no 'clear way ahead' or a defined set of practical guidelines towards agility at scale. The framework surely addresses the 8 potential scaling factors of the 'Agile Scaling Model' discussed in section 4.3 to take into account *(team size, geographical distribution, regulatory compliance, domain complexity, organization distribution, technical complexity, organizational complexity & enterprise discipline)*, but there is no clear set of good practices on how to mitigate all of them. Ambler (2010) states: "*The first four scaling factors listed – team size, geographical distribution, regulatory compliance, and organizational distribution – are relatively straightforward to address via disciplined work, adoption of appropriate technology, and tailoring of practices to reflect the realities of each scaling factor. The other four scaling factors – domain complexity, technical complexity, organizational complexity, and enterprise discipline – are more difficult to address because environmental complexity often reflects systemic challenges within your organization and enterprise discipline requires a level of maturity that many organizations struggle to achieve (although most desire such discipline).*"

Keeping that in mind, the framework has its use though when it comes to the getting the agile basics right. It also gives a clear overview on how proven lean & agile approaches can be brought into use in a full delivery lifecycle rather than focusing purely on development. Therefore I see Disciplined Agile Delivery as an interesting framework allowing a team to function in an enterprise context, an important step towards agility at scale.

5.2 LeSS

LeSS is an acronym of **L**arg**e** **S**cale **S**crum, an approach for large-scale agile development created by Larman & Vodde (2013). After Larman published Agile & Iterative Development (2003) he identified the interest of applying these techniques on a larger scale, instead of executing it in small group. Recognizing **Team Size** as a scaling factor, the LeSS framework offers a solution to cope with groups from a few hundred to a few thousand individuals. They also identify **geographical distribution** as a challenge when talking about multisite of offshore development.

In order to achieve Large Scale Scrum, Larman & Vodde tried to identify the true purpose of the single-team Scrum elements and figure out how to scale them within the constraints of the standard Scrum rules.

In the definition of their framework they mention some good practices on how to cope with **organizational complexity & distribution**. One of the good practices are to create multidisciplinary teams instead of separate analysis, testing and development teams, existing roles dissolve in the organization. Next to that the practices, roles & responsibilities should be adapted to fit the new way of working instead of being predicated on long phases. There is no longer need for a separate Research & Development department as they become an integral part of the teams in the role of the Product Owner. The role of the team lead or the project manager becomes obsolete, while there is no need for an organization-wide standard process, ultimately trying to reduce the complexity of the organization.

Off course in order to achieve this, the organization needs to go through a lot of change. Larman & Vodde state that this simply can't be achieved without the buy-in of the leaders of the organization. The **enterprise needs discipline** by changing its focus from projects towards products. LeSS provide two organizational frameworks to deal with these challenges towards large scale agility. Framework 1 provides good practices for scaling up to about 10 teams (of 7 individuals), whilst Framework 2 focusses on scaling towards a few thousand people working on one product. '10' is not a magical number here, as it is more defined by Larman & Vodde as the tipping point where a single product owner can no longer grasp an overview of the entire product.

Framework 1

There are still many resemblances with the One-Team Scrum setup: there is still one product backlog, one definition of done, one potentially shippable product increment after each sprint, one (overall) product owner, one sprint and a team is 'one team' (each team is a team, no single-specialist teams are setup).

There are no particular role or artifacts changes, but there is a difference when it comes to the behavior of the Scrum meetings, which is explained in Figure 12. The Sprint Planning meeting is split into two separate meetings. The first being done by two representatives of each team, and the second within the team. There are also representatives of other teams joining the daily scrum to ensure knowledge sharing. In addition a Scrum of Scrum meeting can be set into place. There is still one sprint review meeting, but the retrospectives are done at team level as well as at product level.

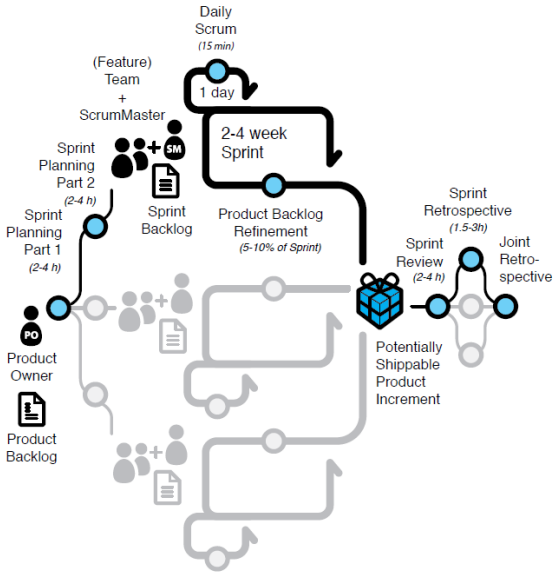


Figure 12 Large Scale Scrum - Framework 1

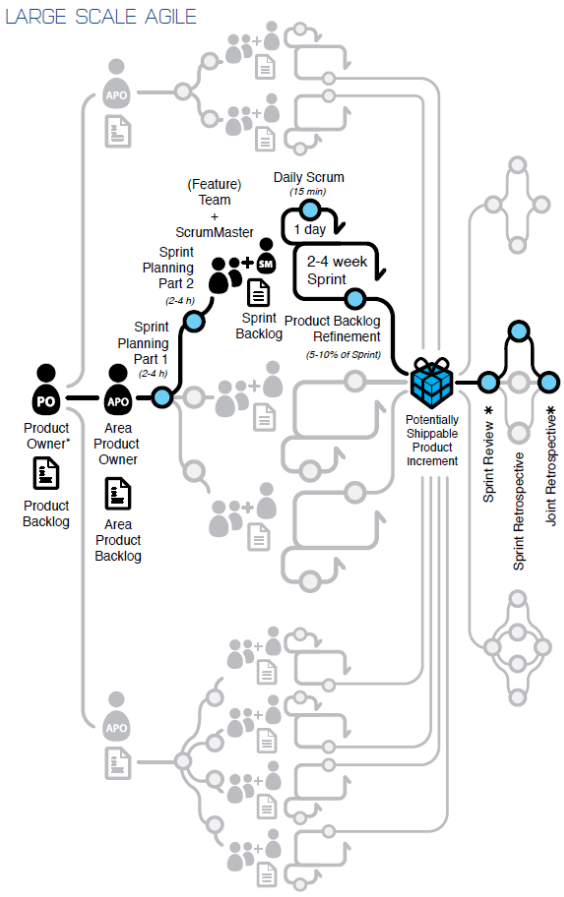


Figure 13 Large Scale Scrum - Framework 2

Framework 2

The second LeSS framework by Larman & Vodde brings a new Artifact in to place, namely the 'Requirement Area' where the Product Backlog is divided into multiple area views such as 'Performance' or 'Protocols', or 'Management', … A new Role is introduced, as each Area gets an Area Product Owner. Also the meetings change. There is now a Pre-sprint meeting of the Product Owner Team and Area-level meetings are conducted as in the first framework. There are Sprint Review meetings at Area level as well as on Product level, which is the same for the Sprint Retrospectives.

In conclusion these two frameworks offer some good practices when looking at the scaling factors of Team Size, Geographical Distribution, Organizational Complexity & Distribution and Enterprise Discipline.

5.3 Scaled Agile Framework (SAFe)

The Scaled Agile Framework (SAFe) is an interactive knowledge base for implementing agile practices at enterprise scale. The 'Big Picture' figure below highlights the individual roles, teams, activities, and artifacts necessary to scale agile from the team to program to the enterprise level, (Leffingwel, 2014). It is a rather prescriptive model where a clear link is created between the enterprise and the agile teams.

Next to the known roles as Product Owner, Scrum Master & Team Members, SAFe defines additional roles at program and enterprise level being: the Business Owner, the Systems Architect, and the User Experience lead, the Release Train Engineer, the Enterprise Architect and the Epic Owner.

Also there are other teams involved next to the development team: The Product Management, the Release Management, System Team, DevOps Team & the Program & Portfolio Management Team.

When talking about artifacts, there is the Team Backlog, but also the Program & Portfolio Backlog, with Business & Architectural Epics and a clear defined Roadmap & Vision.

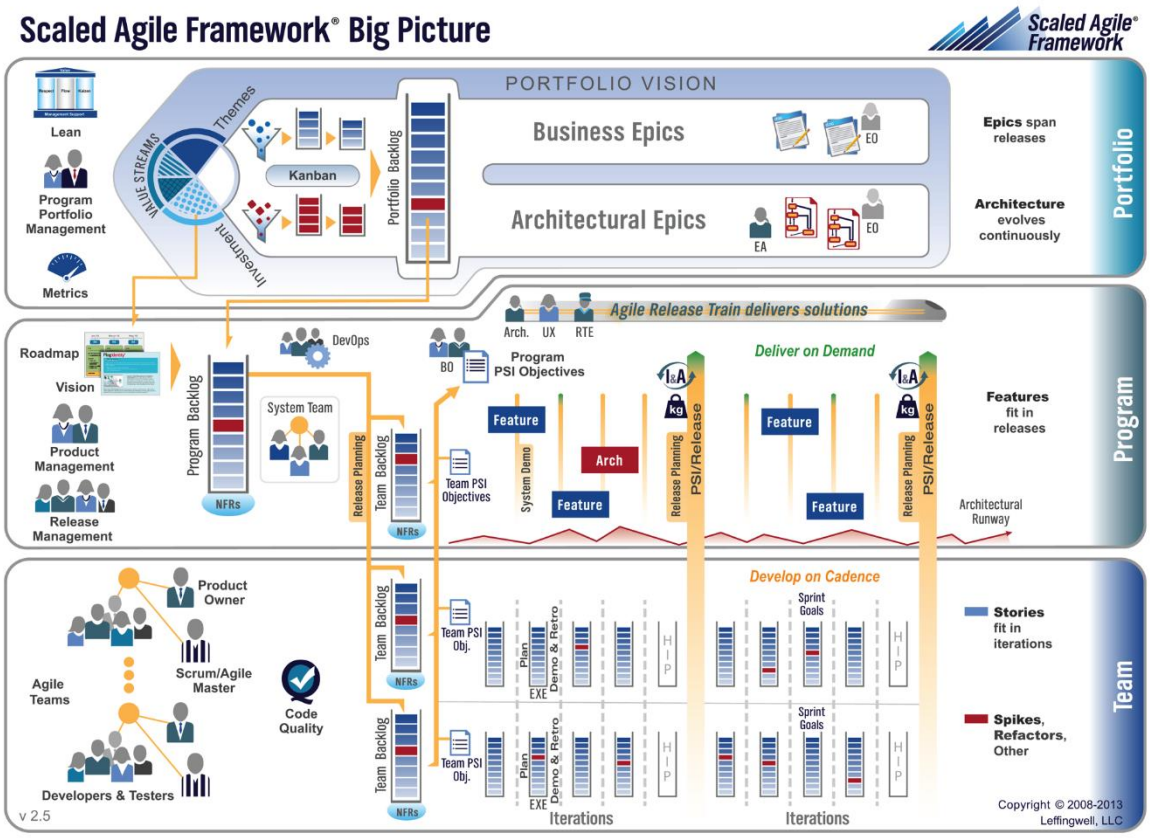


Figure 14 Scaled Agile Framework Big Picture (Leffingwel, 2014)

The SAFe knowledge base is based on a set of good practices that Leffingwell has defined in his book Scaling Software Agility (2009). There Leffingwell defines agility to be achieved at Team as well as on Enterprise level.

Team Agility

Leffingwell states Team Agility empowers teams to more rapidly deliver quality software, explicitly driven by intimate and immediate customer feedback. Therefore he has defined a disciplined set of enhanced software engineering practices, empirical software project management practices and modified social behaviors in order to achieve these goals. Core SAFe practices to achieve Team Agility include:

1. **The Define/Build/Test Team** – a multidisciplinary team optimized for communication about the matter
2. **Mastering the Iteration** – A fixed-time, fixed-resource iteration with a definition phase, development phase and acceptance phase.
3. **Smaller, more frequent Releases** – shorter release dates of 60-120 days, fixed date, theme, planned feature set & quality but with a variable scope
4. **Two-level Planning** – At system level: plan releases with a horizon of 3-6 months; at feature level plan 2-4 iterations ahead
5. **Concurrent Testing** – all code is tested code, test are written before or concurrently with the code itself & testing is a team effort
6. **Continuous Integration** – the team's ability to continuously build software
7. **Regular Reflection & Adaption** – Periodically reflection of the process & lessons learned

Enterprise Agility

Next to the measures defined at team level, Leffingwell (2009) defines also a set of organizational good practices, beliefs & core values to reach enterprise agility:

1. **Intentional Architecture –** Build the simplest architecture that can actually work;
2. **Lean Requirements at Scale –** Using a Vision, Roadmap & Just-in-time elaboration
3. **Systems of Systems and the Agile Release Train –** Managing dependencies amongst teams is important, therefore there are periodic fixed release dates and intermediary global integration milestones.
4. **Managing Highly Distributed Development** – Co-locate team often, at least at release planning, better continuously around a feature
5. **Changing the Organization –** Define change as a project, moving to agile portfolio management
6. **Impact on Customers & Operations** – More frequent releases require more collaboration and availability with customers, suppliers, marketing, sales, support, …
7. **Measuring Business Performance –** The primary metric for agile is whether or not working software actually exists and is demonstrably suitable for it intended purpose, therefor it is compulsory to have a demo at the end of each single iteration.

Whereas Disciplined Agile Delivery is more a set of good practices for getting the basics right, SAFe and LeSS leave more opportunity towards the actual scaling issues. Whereas DAD is more a framework that is intended to be tailored to the needs of the organization, LeSS and SAFe offer a more prescriptive approach for the scaling challenge. LeSS focuses more on offering a clear scalable process, going into detail about the underlying principles of agility at scale, SAFe offers a more governance oriented framework with clear roles en responsibilities linking multiple development teams with the enterprise.

Part 3: Case study: Wolters Kluwer Belgium

6 Designing & Selecting the case study

6.1 Case Study Design

In order to provide an answer to the Central Research Question as defined in Section 2.2 we have already defined our conceptual framework in Chapter 0. This set of scaling factors will be used as the **Unit of Analysis**. By conducting this case study, the goal is to find if the scaling factors are present within the organization and thereby validating their existence when it comes to reach agility at scale.

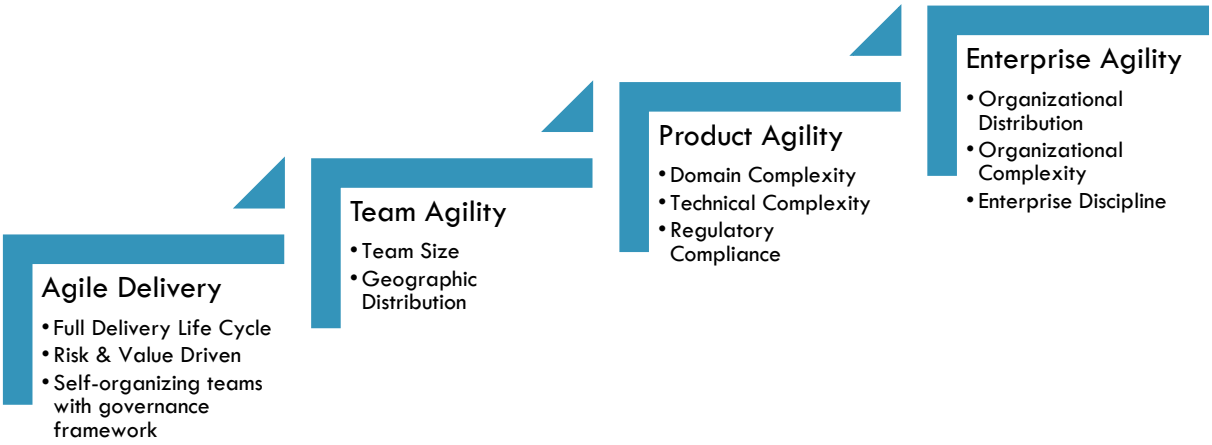


Figure 15 Unit of Analysis - Scaling Factors

I have chosen to conduct a single holistic case study (with one unit of analysis). Yin (1989) states that a single-case design is eminently justifiable where the case represents a critical test of existing theory. This is exactly what I want to achieve: testing the existence of the scaling factors defined in Chapter 0. The choice for a single case study is justified by the fact that there is a time boundary while writing this master thesis, together with the fact that there is a big opportunity to gain access to the organization of Wolters Kluwer Belgium for conducting this case study research.

6.2 Case Study Selection: Wolters Kluwer Belgium

Wolters Kluwer Belgium is the Belgian site of the international Wolters Kluwer group, specialized in business & legal software and publishing. The business unit Tax & Accounting focusses on software & publishing for Tax Advisors (accountants) & SMEs, supporting their day-to-day operations. This business unit offers a large set of legacy on-premise software packages for ERP, Accounting & Tax. Most of these software solutions are in maintenance mode and the result of bought up smaller companies and integrated in the Wolters Kluwer branding.

The ETNIA program (European Tax & Accounting New Integrated Applications) launched in 2010 has the purpose of developing a new set of on-line solutions based on the expert domain knowledge of the on-premise legacy solutions, combined with innovative business ideas & business models. Together with branches in Italy & Spain, shared general modules are created trying to reduce redundant coding between branches, together with a uniform setup of the technical architecture in the form of the CSP (Commercial Software Platform).

The first project built by the Belgian development team is called 'Online Invoicing v1.0'. OIv1.0 is a tool for the Small Business Owner to easily create his sales invoices online and send them to his customers. The development team chose to adapt the agile delivery method in the program, trying to be as responsive as possible to the changing demands of the profound customer base. However, the project didn't run as planned, as there were a lot of problems at organizational and technological levels causing serious budget overruns and crucial deadlines to be missed. This ended in the result that the developed solution was no longer in line with customer expectations, as many technological shortcuts were made, together with the fact that customer expectations changed, needing the solution to become mobile.

At the evaluation of this first project, management and the development team demanded an extensive improvement plan at both organizational and technological level. Within this assessment the scaling factors as defined in our conceptual framework were checked and if found, an action was set in to place to mitigate them. This led to the opportunity to enroll an agile framework and align the ETNIA.be software delivery method. During the third Quarter of 2013 targeted investments were made enrolling this framework, leading to the approval for the redevelopment of the first project realigned with the new business needs, described in the project plan of 'Online Invoicing v2.0'.

This chain of events is the ideal moment for this research to validate the scaling factors in our conceptual framework, resulting in agility at scale at Wolters Kluwer Belgium.

Given the above reasoning is sufficient to justify the selection of the cases, the next chapter will focus on the specific design of the case study and the methods and deliverables used for qualitative and quantitative data gathering.

6.3 Preparing for entering the field: Ensuring Quality

When preparing to conduct the case study research, it is important to keep in mind the quality of the research. In order to ensure this quality, Yin (1989) argues that there are four tests that can be done. At first it is important to construct validity in general, which can be done by using multiple sources of evidence, qualitative as well as quantitative; by establishing a chain of evidence, showing the transparency for other research; and by having key informants review the draft case study report to make sure that it is compliant with the reality. Secondly there is the need for internal validation between the multiple sources of evidence found, which can be done by pattern matching, explanation building and time-series analysis practices. Thirdly also, external validity is an important test to conduct, by using the same method in other case studies. Reliability is the fourth test that can be done by specifying a clear case study protocol allowing further research to maintain the same way of working, and by developing a case study database. All these tactics are summarized in Table 4, whilst further in this chapter for each tactic is discussed what the impact is on this case study.

Tests	Case-study tactics	Phase of research
Construct validity	Use of multiple sources of evidence	Data collection
	Establish chain of evidence	Data collection
	Have key informants review draft case study report	Composition
Internal validity	Do pattern matching	Data analysis
	Do explanation building	Data analysis
	Do time-series analysis	Data analysis
External validity	Use replication logic in multiple case studies	Research design
Reliability	Use case study protocol	Data collection
	Develop case study database	Data collection

Table 4 Case study tactics for four design tests (Yin, 1989)

Multiple sources of evidence

In the context of this case study I have chosen to use a mixed set of qualitative as well as quantitative instruments for data collection. At first a series of **interviews** are conducted with a selection of key informants from each project, performing various roles within the team, from developer to program manager. This qualitative data is gathered by going through a predefined questionnaire. The questionnaire is created with the goal to be able to identify the potential scaling factors as defined in our unit of analysis. The questionnaire is to be found in Appendix A: Template Questionnaire for Interviews. It is important to state here that in order to assure the quality of the research, for each project there will be a different set of team members, including a mix of team members who have worked only on Online Invoicing v1, only on Online Invoicing v2 and team members that have worked on both projects.

Another qualitative data gathering instrument that is used, are the **sprint retrospectives** from both projects. A sprint retrospective is a gathering of the team after each iteration to discuss lessons learned

on what went well and what do we want to improve. These retrospectives allow to triangulate the data found in the interviews with other qualitative information from the project archives.

In order to improve the quality of this research even further I chose also to add a quantitative data source, namely the **project burn down charts**. A burn down chart is an artifact used within each iteration to check whether a team is on track of 'burning down' on user stories, based on the ideal trend line and the actual effort spent and work done. This data source can assist in determining what the quality of the project is, with the scaling factors identified.

As a fourth source of data, I have done **observations** within the team for each project, by checking the 5 criteria defined by Ambler (2009) to measure if the team is truly agile, as defined in Section 4.3:

1. Produce working software on a regular basis
2. Do continuous regression testing, and better take a Test-Drive Development approach
3. Work closely with your stakeholders, ideally on a daily basis
4. Are self-organizing within a governance framework
5. Regularly reflect on how they work together and then act to improve their findings

Establishing a chain of evidence

By appending all the transcripts of the interviews conducted, together with the burn down charts and sprint retrospectives used for analysis, a thorough chain of evidence is set-up in order to ensure the quality of the data collected. This way further research is able to replicate the way of working even better.

Have key informants review the draft case study report

There were three key informants identified to review the draft case study report, being the Business Development Manager, the ETNIA.be Program Manager and the Project Manager of the Online Invoicing version 1 and version 2 projects.

Pattern-matching Analysis

The pattern matching technique will be used when analyzing the two projects separately. The scaling factors identified in the unit of analysis will be used as patterns, where the different data sources from each project will be checked on matching results. This way it will be possible to validate the presence of these scaling factors.

Explanation Building Analysis

As this special case of a pattern matching technique is more devoted towards explanatory case studies as argued by Yin (1989), this technique won't be applied in this research.

Time-series Analysis

Next to the pattern matching technique, the time-series analysis technique will be used in this case study research. Due to the fact that Online Invoicing v2 is a started as a sequel to the development of Online Invoicing v1, we will try to identify the changes made through observations in the field.

Use replication logic in multiple case studies

Due to the fact that there is only a single case study at hand, it isn't possible to use replication logic for multiple case studies, however, for further research we have included a detailed protocol for approach of conduction this case study.

Use Case Study Protocol

The protocol that is used to conduct this case study is provided below:

1. Introduction to Management about the Purpose of this Case Study containing
    a. Discussing the matters of confidentiality
    b. Discussing the value of the research for the organization
    c. Discussing the effort needed and schedule in place for the research to be done
    d. Management approval for the research
    e. Identification of key informants & interviewees of each project
    f. Identification of key data sources (sprint retrospectives, burn down charts)
2. First Interview Cycle: Online Invoice v1 (See Section 6.2 for more details regarding the project)
    a. Interview with selected key informant & interviewees
        i. Hans Saenen, Developer, only involved in Olv1
        ii. Joris Dresselaers, Lead Technical Architect in Olv1 & Olv2
        iii. Mark Caelen, Business Development Manager taking the role as one of the two Product Owners in Olv1 & Olv2
    b. Introduce the purpose of the research
    c. Ask for collaboration
    d. Schedule interview
    e. Do the actual interview by going through the template questionnaire, asking additional open questions if they arise and immediately take field notes in the form of interview transcripts
    f. Send the transcript via mail to the interviewee for approval
3. First iteration of within-case analysis (pattern matching of Online Invoicing v1)
4. Second Interview Cycle: Online Invoicing v2 (See Section 6.2 for more details about the project)
    a. Interview with selected key informant & interviewees
        i. Tom Princen, Functional Analyst, only involved in Olv2
        ii. David Van Steenkiste, Project Manager, involved in Olv1 & Olv2
        iii. Sofie Traen, Developer, only involved in Olv2

iv. Michaël Mertens, Developer, only involved in OIv2

b. Introduce the purpose of the research
c. Ask for collaboration
d. Schedule interview
e. Do the actual interview by going through the template questionnaire, asking additional open questions if they arise and immediately take field notes in the form of interview transcripts
f. Send the transcript via mail to the interviewee for approval

5. Second iteration of within-case analysis (pattern matching of Online Invoicing v2)
6. Third iteration of time-series analysis regarding the evolution of OIv1 to OIv2

Develop Case Study Database

Due to the fact that this master thesis is limited to a single case study, no database will be developed, however all the transcripts, and used data sources are included in the appendices to offer a clear chain of evidence.

With these quality assurances brought into place, the case study is conducted. In the next Chapter I will go into more detail regarding the analysis.

7 Analysis

7.1 Analysis Iteration 1 – Online Invoicing v1

7.1.1 Is the team truly agile?

In order to test if the team is truly agile, the gathered data via the interviews, sprint retrospectives and sprint burn down charts is matched with the 5 agile criteria as defined in our Unit of Analysis.

1. **Produce working software on a regular basis**

   Based on the interviews, the team was struggling to create working software on a regular basis. Hans Saenen (developer) states that 'there was a result, but it was far from what we wanted'. Also Mark Caelen (product owner) argues that it was very hard for the team to produce working software, since there was no clear goal to focus on.

   When analyzing the sprint retrospectives, there are certain points mentioned that validate the interpretation of the interviewees in the fact that there was a struggle to create working software. In multiple sprints, there is stated that there are a lot of merging problems. Next to that the issues in reaching the goals that were set at the beginning of the sprint, or the teams velocity (the amount of work done within a sprint), are clearly mentioned in sprint 5, 6 & 8. In the retrospectives of sprint 6 can be found that the functionality wasn't ready for demo. Also in Sprint 8 there were a lot of performance issues, preventing the software to be considered as working.

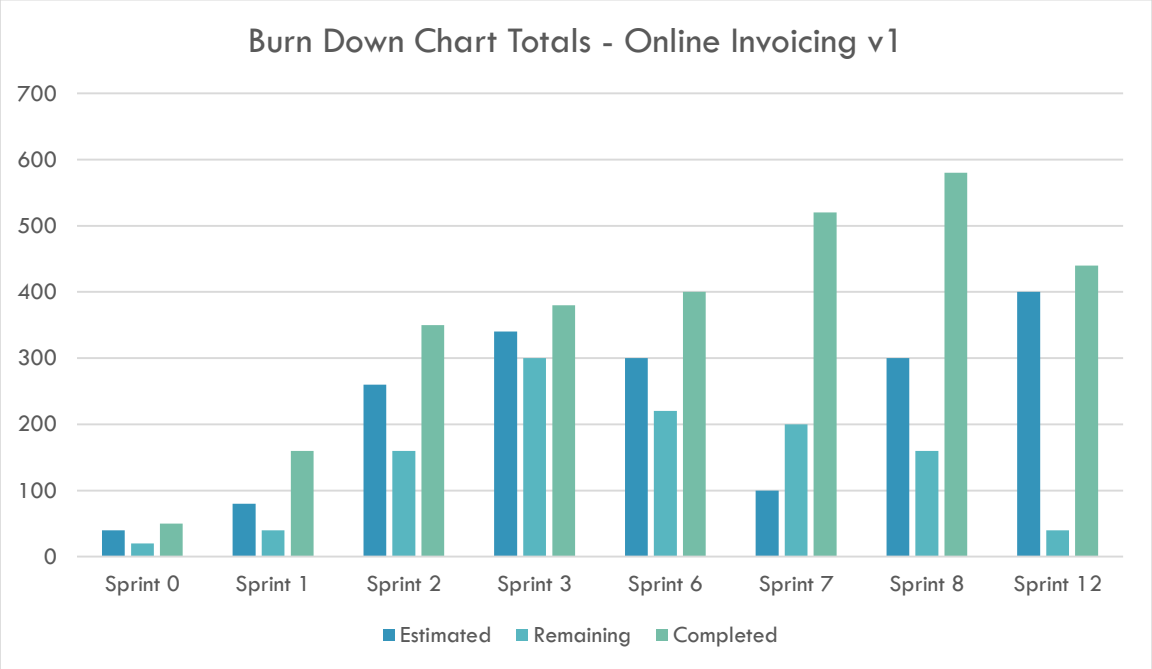


Chart 1 Burn Down Chart Totals - Invoice 1, values based on the burn down charts in appendix D.1

When looking at the burn down chart totals (Chart 1 Burn Down Chart Totals - Invoice 1), one can conclude that the team was struggling in reaching their targets. Each sprint, the completed hours is higher than the original estimated time, meaning that the team had to be working overtime in order to get the scope ready in time for the sprint demo. But each sprint you can see that there still remains a lot of remaining scope by the end of the sprint. This means that although the team was working hard in means of hours, it wasn't reaching the proposed scope, hence not producing working software on a regular basis. When looking at Sprint 0 and Sprint 12, two exceptions can be seen, there the remaining effort is very low in accordance to the original scope, and here the targets were met. Though we need to take these exceptions into account, the general overview makes me conclude that there are sufficient arguments to state that the team wasn't producing working software on a regular basis.

2. **Do continuous regression testing, and better take a Test-Driven Development Approach**
   In this project, there was no continuous regression testing, nor a test-driven development approach. This is confirmed based on all data types gathered. Joris Dresselaers states in his interview that no unit tests were done, nor user testing was performed. There were though ad hoc system tests, but this shouldn't be seen as regression testing. There was no Test-Driven Approach towards development. This can be confirmed when looking at the Sprint Retrospectives. Within Sprint 5 it is mentioned that Unit Testing is behind schedule, whereas in Sprint 6 there were a lot of problems with the demo data, and Sprint 8 Retrospectives state that the Testing process leaves a lot of room for improvement.

3. **Work closely with your stakeholders, ideally on a daily basis**
   When talking to the interviewees it became clear that there were issues when it came to close collaboration with the stakeholders. Hans Saenen tells this by saying that they only saw the product owner 'once a month', and that not a single customer was involved in the development of the project. This is confirmed by the interview with Joris Dresselaers, who adds that there was no clear vision by business for this project, and if there was a vision, it wasn't very clear to the project team. Mark Caelen, product owner, confirms this statement with the fact that though the vision was clear to the business, there was a hard time clarifying this to the dev-team. Also Mark states that the stakeholders from other departments such as Sales, Marketing and the top-level management could have been more involved in the project, making the project visible to all stakeholders. Triangulating this data with the information gathered from the sprint retrospectives, such as the fact that there were availability issues with the product owner in Sprint 0 and 8 and the missing business vision within the team, when already far in the execution of the project (Sprint 8), allows us to validate the fact that this criteria isn't fulfilled.

4. **Are self-organizing within an appropriate governance framework**

   All the interviewees confirm that there were issues when it came to the governance framework in place. Joris Dresselaers states that the Prince II governance framework was too heavy for the project at hand, Mark Caelen confirms this arguing that the traditional top-down governance framework was limiting the potential of the self-organizing agile team. Also there were no clear roles en responsibilities, as discussed earlier, the product owners didn't see themselves as part of the project team, which led to issues translating the business needs to working software. This is confirmed by the Sprint Retrospectives, where Sprint 8 mentions the lack of clear roles & responsibilities. Also in Sprint 8 it is discussed that the team motivation was rather low due to the technical complexity of the project, the contract with the other branches to deliver a proper development platform was seen as the largest issue by the interviewees. Furthermore the traditional governance framework also failed when it came to having a solid process, states Joris Dresselaers. The traditional silos were kept into place, there were business teams, analyst teams, development teams, and architecture teams, but there was no "project team". This makes me conclude that although there was a lot of motivation for the team to be self-organizing and to deliver quality, the governance structure was limiting this team from becoming agile. Also there was a need for coaching and training, in order to become a skilled team.

5. **Regularly reflect on how they work together and then act to improve their findings**

   As discussed in the previous point, the team was regularly trying to improve themselves. This can be confirmed by the fact that there are archives regarding the sprint retrospectives, which is a 'lessons learned' artifact. Joris Dresselaers confirms this by stating that the teams continuously tried to improve themselves, regardless of the big technical obstacles due to obligatory development platform. Again, here the governance framework didn't allow the full potential of the self-improvement to be used, as there was a limited budget for training and the compulsory development framework remained a steep learning curve for the development team. Nonetheless from my point of view there are sufficient efforts done to mark this criteria as fulfilled.

In conclusion we can state that only one of the five agile criteria have been fulfilled. Given the fact that based on these criteria, we have to conclude that in the view of Ambler, the team can't be seen as a true agile team, we will discuss the challenges at hand in the next section.

Agile Criteria		Fulfilled
1	Produce working software on a regular basis	No
2	Do continuous regression testing, and better take a Test-Drive Development Approach	No
3	Work closely with your stakeholders, ideally on a daily basis	No
4	Are self-organizing within an appropriate governance framework	No
5	Regularly reflect on how they work together and then act to improve their findings	Yes

Table 5 Agile Criteria Results for Online Invoicing version 1

7.1.2 Are there scaling challenges?

Based on the data gathered from the interviews I have identified some of the scaling factors as defined in our Unit of Analysis. These findings are discussed per topic.

- **Team Size**

  As the team consisted mostly of about 5-6 developers, 2 analysts, a product owner and a project manager, it is hard to identify that team size was a challenge. In the interviews nothing has popped up that could make us conclude that the team size was a challenge. The only time that the team talked about problems in the size of the team was in the retrospectives of Sprint 8, when two extra developers were added to the team to catch up lost time. This can be clarified by the fact that there was a rather small scope, as developers were running in to each other's code when developing the small scope. Although it is rather straightforward that team size can offer scaling challenges, the composition of this team doesn't allow us to validate this.

- **Geographic Distribution**

  People were working on different locations within the team. The majority of the team was located in the offices in Hasselt, whilst one developer and one analyst were locate in Ghent. However, when talking to the teams, they managed this challenge by bringing in to place a lot of tools for efficient virtual communication. Hans Saenen talks about the usage of e-mails, conference calls (skype) and screen sharing tools (join.me). Joris Dresselaers argues that for a beginning team it is not advised to be dislocated, as it brings in to place a lot of overhead. Not only the virtual communication tools, but also the regular face-2-face meetings in the Mechelen branch of the organization were rather helpful, which is confirmed in the interviews with Mark Caelen.

- **Domain Complexity**

  Due to the limited scope and the fact that the team members already had a lot of experience, made the domain to be seen as not complex, as stated by Hans Saenen and again validated by Mark Caelen. However when talking to Joris Dresselaers, he found the domain more complex than the other two interviewees, since he was rather new in the organization, not so much regarding the domain itself, but the fact that the scope wasn't very clear upfront made the

domain more complex. In the retrospectives no evidence is found on the fact that the domain was of a complex nature.

- **Technical Complexity**

  When talking to the interviewees, this scaling challenge was the biggest issue to overcome during this project. The development team was using a development platform called CSP, which was centrally developed by Kluwer development teams in Spain. The obligation of the management to use this development platform which wasn't ready is to be seen as one of the major reasons for the problems that arose during the development of the project. This is confirmed when looking at the sprint retrospectives, as in sprint 1 already there were performance issues, which only got worse by Sprint 8, whereas sprint 9 was held only for bug solving. The term 'CSP' is also frequently found in the retrospectives, allowing us to surely validate that technical complexity of the matter was a very large challenge for the team.

- **Regulatory Compliance**

  As the development team was building a solution for creating and managing invoices, there was a regulatory compliance, since an invoice is a sort of official document, which needed to be compliant with the Belgian legislation, as discussed with Hans Saenen. There were some obligatory notifications, VAT-regulations and others to keep in mind. Also there were some regulatory needs when offering a Software as a Service platform, when it came to the storage and backup of the data, which are identified within the interview with Joris Dresselaers. The requirements for the latter part were not clear in the beginning, leading to weird requirements as e.g. the decision to give the customer his data back on a cd when he should stop his subscription on the software. Mark Caelen confirms the presence of regulatory compliance, and the fact that this wasn't top of mind when starting the project, which lead to be a challenge during the development of the project. Although it was present, it isn't to be seen as the biggest challenge on the road, as it was rather straightforward to mitigate the challenge, by thorough analysis of the team.

- **Organizational Distribution**

  Where within the development team there was a rather collaborative atmosphere, the collaboration with other teams was more to be seen as contractual. For instance the collaboration with the product owning team was not always as easy, based on the retrospectives, claiming the limited availability of the product owners as a point of improvement, already in sprint 1. Also the collaboration regarding the obligatory development platform CSP was to be seen as a major issue, as pointed out by Joris Dresselaers. The very contractual nature of the delivery of the development platform lead to major issues when it came to creating a higher velocity of the team. The sprint retrospectives confirm this fact, as mentioned in Sprint 1 that the support from the CSP team could be improved.

- **Organizational Complexity**

  It can be concluded that the organization was rather complex and top-down structured. A specific example of this is the mentioning in the retrospectives of the fact that for the development team the vision was still very unclear, when the development was already in sprint 8. Also when talking to all interviewees, they confirm that the top-down structure of the organization didn't motivate the team to become more self-organizing. The decisions were made at top-level and the dev-team was only there to execute, not to 'co-create'. There was very little bottom-up feedback, as the reasoning was often neglected by the top-level. Also there was a very little engagement of other stakeholders of the project during its execution, as argued by Mark Caelen.

- **Enterprise Discipline**

  The best example of the struggle towards enterprise discipline is surely the CSP development platform. A lot of investments were made by the organization to create a reusable asset for speeding up the development. However this effort turned out to deliver exactly the opposite of what its goal was, slowing down the development team rather than offering the tools needed to create a higher velocity. This can be seen as one of the major challenges for this project, as there was consensus on different levels of the organization regarding the reusability fact, but there were a lot of different meaning when it came to the chosen approach. This lets me conclude that the reuse of enterprise assets is also to be seen as one of the major challenges within this organization.

	Scaling Factor	Hans Saenen	Joris D	Mark Caelen	Retrospectives
Team Agility	Team Size	Simple	Simple	Simple	Complex
	Geographic Distribution	Complex	Complex	Complex	N/A
Product Agility	Domain Complexity	Simple	Complex	Simple	N/A
	Technical Complexity	Complex	Complex	Complex	Complex
	Regulatory Compliance	Complex	Complex	Complex	N/A
Enterprise Agility	Organizational Distribution	Complex	Complex	Complex	Complex
	Organizational Complexity	Complex	Complex	Complex	Complex
	Enterprise Discipline	Complex	Complex	Complex	Complex

Table 6 Scaling Factors identified in Online Invoicing 1

Table 6 summarizes the above analysis, by matching the scaling factors with the gathered data. The explicit presence or absence of a scaling factor is mentioned, as well as when no information was regarding the specific topic.

7.2 Analysis Iteration 2 – Online Invoicing v2

7.2.1 Is the team truly agile

Also for the second project, the 5 criteria will be used to test if the team can be considered as truly agile.

1. **Produce working software on a regular basis**

   The team was able to produce working solutions, in the first 8 sprints the focus was more on the functionalities, where you can see in the burn down charts that there was a slight amount of remaining effort is rather low in comparison with the original estimates. Also when talking to Sophie Traen about this she confirms that the teams were reaching the predefined goals for about 90-95% in these sprints. After the launch of the minimum viable product, the team went into more stabilizing phase of the project, where the focus was more on the underlying matter and less visible on functionalities, this can clarify why the next sprints has a lot more remaining effort by the end of the sprint, as the team had a harder time predicting the work to be done.

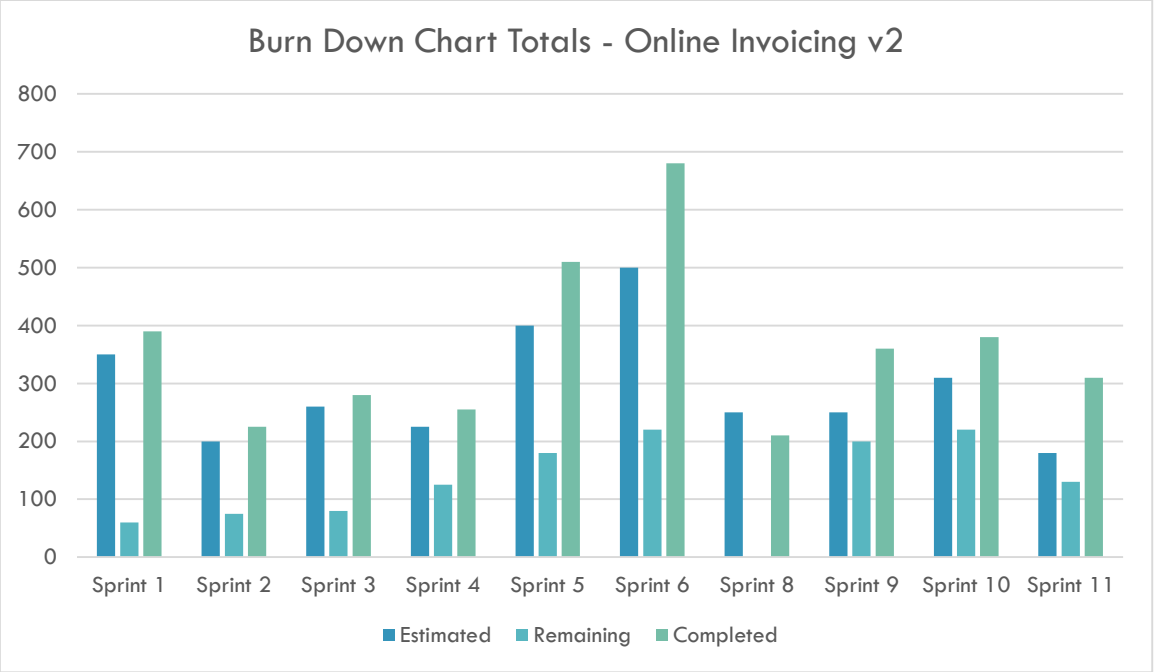


Chart 2 Burn Down Charts Online Invoicing v1 - Based on Burn Down Charts in Appendix D.2

2. **Do continuous regression testing, and better take a Test-Driven Development Approach**

   The team does unit testing, which in the previous project wasn't possible due to the technological limitations of the development platform. There is sort of test-driven approach where the analysts prepare the test scenario's upfront and discuss them with the developer, but

there is still room for improvement when it comes to continuous regression testing, argues David Van Steenkiste. Sophie confirms this, but states that improvements are made, as there recently has been added a dedicated tester to the team. Michaël concludes there is also room for improvement as the process is not clear to all team members, we do unit testing, system testing etc. but it is not automated. These issues are confirmed by the retrospectives, regarding the issues with the acceptance criteria in sprint 1 & 4 and with the unit testing in sprint 11. Therefor we can conclude that already the team has made progress, but there is still room for improvement when it comes to automated testing & regression testing.

3. **Work closely with your stakeholders, ideally on a daily basis**

   To all interviewees, the target customer is known to them, the small independent entrepreneur, this is clearly top of mind. The customers are frequently involved in the process, by up front user testing with mood boards & mockups and also during the development with the working software.

   Also there is a dedicated product owner within the team who has the mandate to take the decisions for the team. There is also a product development team involved which is used by the product owner as a sound board and who are visiting the demo's regularly, the last part is still room for improvement as there sometimes were issues with the availability of the product developers.

   Next to that there is close collaboration with other teams, such as the operations team GSS for the infrastructure, the marketing & sales teams for the go-to-market preparation and the legal team when it comes to compliance questions.

4. **Are self-organizing within an appropriate governance framework**

   Based on the interview with Tom Princen, the team is collaborating well, as there is solid communication. Sophie & Michaël confirm this as they both state that the team drive is very good. The team members are not only developing, they are also coaching each other, argues Michaël Mertens. Regarding the governance framework, a lot of improvement has been made by the implementation of a solid Software Delivery Life Cycle framework, which allowed the team to rely on a process with clear roles & responsibilities for the project to succeed. This can be seen as one of the major points of improvement, states David Van Steenkiste. Also the assignment of a dedicated program manager who has control over the process & coaches the people has improved the governance structure a lot.

5. **Regularly reflect on how they work together and then act to improve their findings**

   As a starting point, David states that the team was composed with skilled people, who had experience in building web software, which allowed a higher velocity together with the new technology that was set into place. Next to that each sprint, retrospectives were done and actions were defined and followed-up by the project manager. The analysts were trained, and

UX-expertise was sourced in from a third-party expert, states Tom Princen. Sophie Traen discussed the fact that not only the experienced team, but also the internal coaching of each other was an important factor. Also efforts are made to share the knowledge within the team with the broader organization, by means of brown bag sessions, which is stated by both developers. These Brown Bag Sessions are short sessions by the development team towards other teams during the lunch break (brown bag), where they share knowledge about how they handled specific topics.

In my opinion, we can conclude that 4 out of 5 criteria are now fulfilled, with the exception of the testing approach. The team has surely evolved since the previous project. It is very interesting to know what lead to these changes. This will be discussed in section **Error! Reference source not found.**, first we will have a ook towards the scaling challenges present in the new project, in the next section.

Agile Criteria		Fulfilled
1	Produce working software on a regular basis	Yes
2	Do continuous regression testing, and better take a Test-Drive Development Approach	No
3	Work closely with your stakeholders, ideally on a daily basis	Yes
4	Are self-organizing within an appropriate governance framework	Yes
5	Regularly reflect on how they work together and then act to improve their findings	Yes

Table 7 Agile Criteria Results - Invoice v2.0

7.2.2 Are there scaling challenges?

Based on the progress made by the team, it is interesting to have a view on how the scaling challenges have evolved over time, and how they were handled.

- **Team Size**

  Regarding the team size, we have to conclude that the team remained consistent during the execution of the project, and that there were no traces found that there were issues with the size of the team. This can be clarified by the fact that it indeed is a small team.

- **Geographic Distribution**

  The development team was co-located in Hasselt, but the ux-expert and the product owner were located respectively in Mechelen & Ghent. All interviewees confirm that there were challenges due to this fact. Surely tools as e-mail, teleconferencing, screen sharing & video conferencing were used, but there was a minimum of face-2-face meetings needed. The development team states the need for co-creation in the same room at the same screen, certainly in the beginning of the project. More later on in the project this challenge was managed by the tools at hand and a minimum of co-located time. This off course required some traveling.

- **Domain Complexity**

  All interviewees conclude that the domain was rather straightforward, as the scope was very clear upfront, no immediate problems arose during the development and that it was rather easy to build. So due to the scope, this challenge wasn't present in the team. But Sophie as well as

Michaël state that for the next coming projects, this might become an important factor to think about.

- **Technical Complexity**

  Following Tom Princen, the most important enhancement regarding the technical complexity is that the chosen development framework is now 'googleable' for support, instead of having to count on support from other branches. The fact that there is control over the own technology stack, which is used by the skilled team, makes it a huge productivity improvement, states Sophie, when comparing it to other web technologies and certainly the CSP platform. Michaël confirms this by stating that the new technologies offer a high ease-of-use and it is enjoyable to work with as a developer.

- **Regulatory Compliance**

  There was a need to be compliant with the Belgian legislation when it came to the product, as the invoices needed to correct, these requirements remained the same as the previous version of the project. The product owner had the basic knowledge over what was needed, and was allowed by the organization to work together closely with the legal department of the organization to clarify the requirements even further, as stated by Sophie & David. This compliance was indeed a challenge, but a minor one, thanks to the close collaboration with experts in the matter.

- **Organizational Distribution**

  The clear roles & responsibilities made it a lot easier for everyone to perform their job. This allowed us to move from a more 'contractual' culture, towards a rather collaborative culture, surely within the team, but also broader with for instance the Legal, Marketing & Operations teams, states David Van Steenkiste. The implemented process allowed a better collaboration with the product owner, argues Tom Princen. Sophie confirms this collaborative culture: "Ideas go bottom-up, top-down, left-right, everyone who has an opinion is allowed to give his feedback, and it is then taken into account, that matters". "The product developers & management who used to sit in their separate office are now sitting next to us at our desks, this makes it a lot easier to communicate with them.", adds Sophie. Michaël Mertens confirms this fact, as he discussed that the recent team event has been good to improve the alignment between management & the team, but there is still room for further action on this topic.

- **Organizational Complexity**

  The business vision is very clear to all interviewees. David Van Steenkiste clarifies that business has invested in creating a clear business vision document, and has done a lot of effort to transfer this vision throughout the organization, setting targets towards the market as well as goals for internal improvements.

- **Enterprise Discipline**

  There was a major change in the approach for mitigating this challenge. Instead of the top-down compulsory development framework, the organization has evolved towards 'reusable

components' based on the needs of the business project at hand. The new project offers a new baseline when it comes to reusable assets within the enterprise. There was surely focus on quality & reusability, states David Van Steenkiste. All interviewees agree on the fact that although that this new project is still a pioneer, sufficient efforts are done to ensure scalability & reuse on the long term, towards other projects, other teams & even other branches within the European context.

Table 8 shows a visual overview of the defined challenges, and their presence in the team by the opinion of the interviewees.

	Scaling Factor	Tom Princen	David VS	Sophie Traen	Michaël M
Team Agility	Team Size	Simple	Simple	Simple	Simple
	Geographic Distribution	Complex	Complex	Complex	Complex
Product Agility	Domain Complexity	Simple	Simple	Simple	Simple
	Technical Complexity	Complex	Complex	Complex	Complex
	Regulatory Compliance	Complex	Complex	Complex	Complex
Enterprise Agility	Organizational Distribution	Complex	Complex	Complex	Complex
	Organizational Complexity	Complex	Complex	Complex	Complex
	Enterprise Discipline	Complex	Complex	Complex	Complex

Table 8 Scaling Factors identified in Online Invoicing v2

7.3 Analysis iteration 3 – Changes between Olv1 & Olv2

Based on the observations made, the interviews done, the sprint retrospectives and the burn down charts, an overview is made what has changed during and after the Olv1 project that allowed the Olv2 project to be evaluated as more successful. This will be discussed at three levels: people, process & tools.

People

In the Olv1 project, the strategy was chosen to combine team members who already had experience in the technology (and less with the domain), with team members who had experience in the domain, but less in the technology. The Olv2 project started off with a fully 'skilled' team, who can be considered as expert in the domain and were experienced users of the technology at hand.

Process

For the first project Olv1, the existing Prince II governance framework was applied, which was tailored for the maintenance-driven organization. The process was drastically changed for the Olv2 project with the introduction of the new program manager Lars Van Sweevelt. Based on the Disciplined Agile Delivery method (Ambler & Lines, 2012), combined with the Scrum methodology and the Prince II governance framework at hand in the organization, a whole new Software Delivery Life Cycle (SDLC) process was tailored to the organization, well documented and communicated to the team and the management. This SDLC defined a very clear process with clear roles & responsibilities, guiding the team through the project, and also allowing the team to become more self-organizing. Clear project phases were set, embracing the full delivery life cycle as advised by DAD (inception, construction & transition).

Tools

Next to the new SDLC process, the choice of technology is to be seen as the biggest change between the two projects. Whereas within the first project, the server-oriented architecture that was created by the European software development team was used, the Olv2 project allowed the introduction of a whole new technological framework, which consisted of a client-oriented architecture, composed by the development team itself, allowed the more experienced team to successfully deliver the solution.

Part 4: Conclusions

8 Case Study Report: Wolters Kluwer Belgium

When looking at the analysis of the Wolters Kluwer Belgium Case Study, it shows that the organization has been able to implement key changes needed to evolve towards becoming an agile software delivery company. In the first project, started in 2012, the organization was struggling in getting the basics right for becoming an agile company. Surely there was a willingness to become agile, but there were some challenges in the organization that prevented it of implementing the agile way of working. Using the scaling factors that we have defined in our Unit of Analysis, it has shown that these challenges remained consistent over time. The challenges that were in place in the first version of the project, were still there in the second run. It is the way that these challenges were faced, the approach that has changed. Table 9 shows the different manner in which these challenges were mitigated.

	Scaling Factor	Olv1	Olv2
Team Agility	Team Size	N/A	N/A
	Geographic Distribution	Minimal Face-2-face meetings; E-mail; Teleconference; Videoconference; Screen sharing	Dev-team Co-located; Minimal Face-2-face meetings; E-mail; Teleconference; Videoconference; Screen sharing
Product Agility	Domain Complexity	N/A	N/A
	Technical Complexity	European Compulsory Development Framework with support, documentation, maturity & performance issues	Own-defined cutting-edge web technology that has a broad development community, focused on reusability, prioritized on the business project
	Regulatory Compliance	In-team research	Collaboration with Legal dept.
Enterprise Agility	Organizational Distribution	Collaborative within team Contractual between business and development team, Contractual with third-party CSP-vendor	Insourced UX, analysis & development knowledge, internal coaching, collaboration between business & ict, no more external vendor, clear roles & responsibilities, clear software delivery life cycle
	Organizational Complexity	Vision not clear to the development team	Efforts in clarifying the business vision in the whole organization;

			clear vision, clear persona's, clear stakeholders structure
	Enterprise Discipline	Central development framework pushed to the development teams, no feedback loops or priorities based on team needs	Team in charge of its own technology stack, building its own features, focusing on reusability by the whole organization

Table 9 How Challenges were mitigated in OIv1 vs. OIv2

Because the fact that the organization recognized these challenges and defined proper mitigations towards them, the evolution is shown when looking at the agile criteria in Table 10. These results clearly show that the organization has succeeded in improving their 'agileness' from only one criteria to a better 4 out of 5. Off course there still remains room for improvement, certainly regarding the test approach within the organization.

Agile Criteria		OIv1	OIv2
1	Produce working software on a regular basis	No	Yes
2	Do continuous regression testing, and better take a Test-Driven Development Approach	No	No
3	Work closely with your stakeholders, ideally on a daily basis	No	Yes
4	Are self-organizing within an appropriate governance framework	No	Yes
5	Regularly reflect on how they work together and then act to improve their findings	Yes	

Table 10 Agile Criteria Results - Evolution of Online Invoicing

Also I believe that it is no 'one-time' shot of changes to be put into place, as it is more a way of continuous improvement of the organization. By using the identified challenges in this work, Wolters Kluwer Belgium was able to visualize their problems, assess the current way of working and put impediments into place that were able to cope better with the challenges that were present in the organization. Hopefully this can be useful for other organizations that are facing the same challenges to inspire and help them becoming agile at scale.

9 General conclusions & further research

It is time to provide an answer to the general research question that I have defined at the beginning of this research. This will be done by generation conclusions on each sub question that is proposed.

Why does agile sometimes fail in larger organizations?

A larger organization has another way of working compared to a startup, the bigger structure needs more governance, it has legacy software to be taken into account, multiple teams, … In order for an organization to shift towards to becoming agile, it is in need to get the basics right. These basics are defined by Ambler & Lines (2012) as moving from 'Core Agile Practices' towards 'Agile Delivery'. There is a need to work on an aligned vision with all stakeholders, the time has to be taken to not only create the software, but also for the organization to prepare itself to be ready to consume it, therefor there is a need for a Full Delivery Life Cycle. Next to the fact that off course value needs to be delivered by the team, also there needs to be significant more attention to minimizing the risks ahead in order for a project to succeed. Self-organizing teams in larger environments are in need of an appropriate governance framework that stimulates and motivates them, instead of limiting their maneuverability.

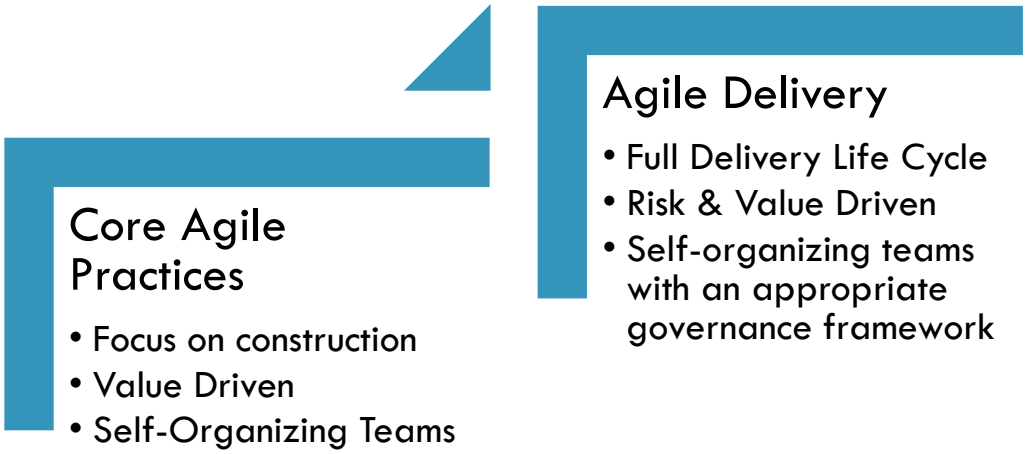


Figure 16 From Core Agile Practices towards Agile Delivery

However the way of working changes in a larger environment, the agile goals behind it remain the same, therefore I have found that the criteria defined by Ambler (2009) are usable for assessing the agile state of an enterprise:

1. Produce working software on a regular basis
2. Do continuous regression testing, and better take a Test-Drive Development approach
3. Work closely with your stakeholders, ideally on a daily basis
4. Are self-organizing within a governance framework
5. Regularly reflect on how they work together and then act to improve their findings

Is it possible to identify potential scaling factors?

Next to getting these basics right, an organization has to face a reality, and within that reality a lot of challenges are ahead for the enterprise to succeed. In the Wolters Kluwer Belgium case, the Agile Scaling Factors that were identified by Ambler in its Agile Scaling Model (2010) were used. Each scaling factor was checked on their presence as a challenge for the organization. About the 8 challenges proposed by Ambler, I was able to validate that 6 of them were present in the organization, excluding Team Size and Domain Complexity.

Figure 17 Agile Scaling Factors or Challenges

This however doesn't mean that the latter two aren't challenges when becoming agile at scale, they simply weren't present in this case. This also immediately proposes the limitations to the case at hand, as the team size was rather small and the domain wasn't perceived as complex to the team. This leaves room for further research, as other cases could be conducted in the same manner to validate if the latter two, but also the first 6 that already are validated can be seen as challenges within other organizations. In the context of this master thesis, the transparence of the way of working should provide sufficient information for further research on this topic.

Another limitation of this thesis is the fact that the challenges that an organization can face are limited to the ones defined by Ambler. In the context of this master thesis, the transparence of the way of working should provide sufficient information for further research on this topic, conducting a series of extra case studies.

None the less, the case study has proven to be useful for the Wolters Kluwer Belgium organization, as it provides a clear view on the major challenges on its road towards agility at scale, helping to define clear actions to move ahead, allowing a continuous assessment of the 'agility' of the organization.

Are there known mitigations or good practices for enrolling agile software delivery method at scale?

As discussed earlier in this master thesis, the current professional world provide a number of frameworks, methodologies, good practices, … For the Wolters Kluwer Belgium Case, the change teams have used the **Disciplined Agile Delivery** framework by Ambler & Lines as inspiration for improving their agility when defining their new Software Delivery Life Cycle process.

Personally, I think that there is no silver bullet. I have seen the way we are facing the challenges on our road to becoming agile at Wolters Kluwer Belgium, the current frameworks certainly offer guidance and inspiration, but it depends a lot on the way it is applied within the organization. Whereas DAD offers more a set of tools that can be tailored to the context at hand, LeSS and SAFe are more of a prescriptive nature, which can maybe offer even more effectiveness when combining them with DAD principles.

In my opinion, agile software delivery is currently in its puberty, small teams are using it to rapidly reach their goals, larger, rigid organizations are wanting to become young and agile again, trying to make the Agile way work for them, some are successful, others aren't. There is a lot of community work, a lot of scattered information, and we are working towards a mature process that allows scaling. An interesting further research would be to consolidate this broadly scattered offering of agile practices, methodologies, frameworks, etc. to a higher-level philosophy, which can inspire organizations to help them adapting the agile way of working, and most of all deliver qualitative software and creating delighted stakeholders.

List of tables

List of figures

References

Alliance, A. (2001). Retrieved from Agile Manifesto: http://www.agilemanifesto.org/

Ambler, S. W. (2009, 01 13). *Agility@Scale: Strategies for scaling agile software development*. Retrieved from IBM Developer Works: https://www.ibm.com/developerworks/community/blogs/ambler/entry/agile_criteria?lang=en

Ambler, S. W. (2010). *Scaling Agile: An Executive Guide*. IBM.

Ambler, S. W., & Lines, M. (2012). *Disciplined Agile Delivery*. Boston, USA: IBM Press.

Beaumont, R. (2012). *Information Systems Devemlopment Methods*. Robin Beaumont.

Berners-Lee, T., Cailliau, r., Luotonen, A., Frystyk, H., Nielsen, F., & Secret, A. (1994). The World-Wide Web. *Comm. ACM (CACM)*, 37(8), 76-82.

Boehm, B. (1988). A Spiral Model of Software Development and Enhancement. *Computer*, 61-72.

Boehm, B. (2006). A View of 20th and 21st Century Software Engineering. *ICSE*.

Brooks, F. (1975). *The Mythical Man-Month*. Addison Wesley.

Brooks, F. (1987). No silver bullet: Essence and accidents of software engineering. *IEEE Computer*, 20(4): 10-19.

de Weerd-Nederhof, P. C. (2001). Qualitative case study research. The case of a PhD research project on organising and managing new product development systems. *Management Decision 39/7*, 513-538.

Denning, S. (2010). *The Leader's Guide to Radical Management*. San Francisco, CA: Jossey-Bass.

Eisenhardt, K. (1989). Building theories for case study research. *Academy of Management Review*, Vol. 14 No. 4, pp. 532-50.

Humphrey, W. (1989). *Managing the Software Process*. Reading, MA: Addison-Wesley.

ISO. (1995). ISO/IEC 12207 - Standard for Information Technology - Software Life Cycle Processes. *ISO*.

Khurana, G., & Gupta, S. (2012). Study & Comparison of Software Development Life Cycle Models. *International Journal of Research in Engineering & Applied Sciences*, 1513-1521.

Larman, C. (2003). *Agile & Iterative Development: A Managers Guide*. Addisson-Weasly.

Larman, C., & Vodde, B. (2013). Scaling Agile Development. *CrossTalk*.

Leffingwel, D. (2014, 07 16). Retrieved from Scaled Agile Framework: http://scaledagileframework.com/

Leffingwell, D. (2009). *Scaling Software Agility Overview*. Retrieved from Agile Alliance: http://agile2009.agilealliance.org/files/session_pdfs/Scaling%20Software%20Agility%20Overview%20Agile%202009.pdf

Mahalakshmi, M., & Sundaranjan, M. (2013). Traditional SDLC vs. SCRUM Methodology - A Comparative Study. *International Journal of Emerging Technology and Advanced Engineering*, 192-196.

McConnel, S. (1996). *Rapid Development*. Microsoft Press.

Miles, M., & Huberman, A. (1994). *Qualitative Data Analysis. An Expanded Sourcebook, 2nd ed*. Thousand Oaks, CA.: Sage.

Miller, G. (1956). The magical number seven, plus or minus two: Some limits on our capacity for. *Psychological Review*, 63(2), 81-97.

Mittal, N. (2013, 01 07). *Self-Organizing Teams: What and How*. Retrieved from Scrum Alliance: http://www.scrumalliance.org/community/articles/2013/january/self-organizing-teams-what-and-how

Moran, A. (2014). *Agile Risk Management*. SpringerBriefs in Computer Science.

Munassar, N., & Govardhan, A. (2010). A Comparison Between Five Models of Software Engineering. *International Journal of Computer Science Issues*, 94-101.

Royce, W. (1970). Managing the Development of Large Software Systems: Concepts and Techniques. *Proceedings of WESCON*.

Stevens, P. (2013, 10). *Scaling Scrum, DAD, Safe or LeSS*. Retrieved from Scrum Breakfast: http://www.scrum-breakfast.com/2013/10/scaling-scrum-safe-dad-or-less.html

Sutherland, J., & Coplien, J. (2014, 07 12). Retrieved from ScrumPLoP: http://www.scrumplop.org

Weinberg, G. (1971). *The Psychology of Computer Programming*. New York: Van Nostrand Reinhold.

Wroblewski, L. (2011). *Mobile First*. Wroblewski.

Yin, K. (1989). *Case Study Research. Design and Methods (Applied Social Research Method Series, Vol. 5)*. Newbury Park, CA: Sage.

Appendices

Appendix A: Template Questionnaire for Interviews

Participant: Role: Project(s) involved:

Date:

The goal of this survey is to evaluate how the 'Online Invoicing v1.0/v2.0' project was handled in terms of software delivery process. These questions below have the goal of identifying success factors of the project, and potential improvements in the SDLC. As this is an in-depth survey, a basic set of open questions is documented that can be used as a starting point of the discussion.

Question: Can you tell about the context of the project, why was it founded, what are the drivers

Question: What was your role in the project?

Question: In your experience, which topics did you evaluate as positive during the delivery of the project?

Question: In your experience, what where the struggle points that the team tried to solve during the project? Which of these topics were solved and how, and which topic remained a problem by the delivery of the project?

Question: Who was your customer, and how was he involved in the process?

Question: How was the team organized? Can you discuss how a typical development iteration took place?

Question: Did you find that you had the right tools to use for your role?

Question: When you think about the project scope, was this clear upfront?

Question: Were there frequent changes during the project execution, and how were they handled?

Question: Was there a specific process that was followed?

Question: How many people did your team consist of, including product owners, customers, …

Question: Did you collaborate with other teams for this project?

Question: Was everyone working in the same place? Were there face-2-face meetings or was this done otherwise?

Question: In your opinion, did you find your coworkers to be working in a silo, or as a one group?

Question: Describe the company culture, how are the relations with the coworkers, the management … Is there a lot of team spirit, a top-down culture, bottom-up culture, is there a specific hierarchy?

Question: Was the project done ad-hoc, or did it reuse architecture from other projects, programs, … Was this integration easy?

Question: Talking about the project itself, did you find it was a rather easy project to build, or were the features of the project more complex?

Question: Was there a need for being compliant with a regulatory instance? How was this handled?

Question: What was the technical complexity of the project, which technologies were used, and how are they evaluated?

Appendix B: Interview Transcripts Online Invoicing v1

B.1 Hans Saenen

Participant: Hans Saenen Role: Developer Project Involved: OI v1

Date: 9 April 2014

The goal of this survey is to evaluate how the 'Online Invoicing v1.0' project was handled in terms of software delivery process. These questions below have the goal of identifying success factors of the project, and potential improvements in the SDLC. As this is an in-depth survey, a basic set of open questions is documented that can be used as a starting point of the discussion.

Question: Can you tell about the context of the project, why was it founded, what are the drivers

De reden waarom was omdat we naar een nieuw pakket te gaan, Kluwer heft altijd firma's overgenomen, nu op de market gekomen dat webondersteunend was en lopen ze achter op hun concurrenten om toch zo een project in de markt te krijgen, kijk wij zijn ook met nieuwe dingen bezig, kluwer staat niet stil.

Question: What was your role in the project?

Developer, frontend als backend, is niet echt gescheiden, zowel c-sharp als html, maakt niet uit wie wat deed, in het begin vooral html-code.

Question: In your experience, which topics did you evaluate as positive during the delivery of the project?

Ik kan me momenteel enkel slechte dingen herinneren. De teamwerking was goed, onderling nooit problemen, devvers goed overweg, de spirit was er de eerste maanden, laatste maanden minder aanwezig. Dag één al CSP, genereert code, 'wat is dit?!', blij dat we ondanks alle problemen opgeleverd hebben. We kregen twee ijsblokjes en moesten een auto bouwen, er was 'een' resultaat.

Question: In your experience, what where the struggle points that the team tried to solve during the project? Which of these topics were solved and how, and which topic remained a problem by the delivery of the project?

CSP, trok op niets, code generatie was niet wat we wilde, controls waren niet goed, management zag niet in dat dit niet goed was. Grote frustratie om controls weg te nemen, en dan lopen we maanden achter, dan nog met het CSP dat niet goed was. Plus we hadden weinig Javascript kennnis, onderschat geweest, wel cursus .NET, niet gekeken naar nieuwe technologieën, serieus misgelopen. Begonnen aan het project zonder te kijken welke technologieën er zijn, trek uwe plan maar. Ook met het gedacht van 'het zal wel loslopen', en niet met de kennis van wat er ons nog allemaal te wachten stond. Als het CSP er niet was geweest hadden we zeker goede functionaliteit kunnen opleveren.

De dingen waren in het begin niet heel duidelijk, dus dat was wel wat moeilijk, we hadden geen target, maar dat was niet de grootste bottleneck. Begon ook te verbeteren na het project verderging. Leercurve voor iedereen, voor iedereen was het 'nieuw'.

Question: Who was your customer, and how was he involved in the process?

Geen flauw idee, eindklant was de kleine KMO denk ik, maar was niet betrokken, we gaan uit van intrinsieke kennis, nooit met de klant gaan praten. Dit weet ik niet.

Question: How was the team organized? Can you discuss how a typical development iteration took place?

Goed, elke dag standups, iedereen duidelijke taken in TFS, we wisten wat we moesten doen, meedenken wat we kunnen doen, dat gebied zit het goed. In het begin de sprintplanning, taken toegewezen, taak duurt ongeveer 4u, iedereen zijn mening, tijd ingegeven, volgende twee, sprint van 4 weken (te lang), guido vond het van in het begin dat 4 weken te lang was. Voor mij waren sprints nieuw. Op sobre hebben we twee weken genomen, en dan zijn korte sprints makkelijker.

Question: Did you find that you had the right tools to use for your role?

Nee, CSP was de grote bottleneck, javascript kennis veel te laag, te moeilijk.

Question: When you think about the project scope, was this clear upfront?

Ja, online facturatietool was de scope, duidelijk genoeg. De klant moet 'factuur kunnen maken', punt.

Question: Were there frequent changes during the project execution, and how were they handled?

Ja, in het begin zat er veel meer in scope, deels door beperkingen in technologie, niet alles opleveren, vb. Mails konden we niet doen omdat technologie ons beperkte.

Controls van CSP konden we niets mee doen, door spanje gepusht en gewrapt, maar bepaalde functionaliteit maar beschikbaar, veel te weinig mogelijkheden.

Question: Was there a specific process that was followed?

Zou kunnen, maar hier heb ik geen weet van.

Question: How many people did your team consist of, including product owners, customers, …

8 à 10 personen

Question: Did you collaborate with other teams for this project?

Iedereen kwam van andere projecten.

Question: Was everyone working in the same place? Were there face-2-face meetings or was this done otherwise?

Nee, was geen probleem via skype konden we wel alles regelen. Meetings via Join.me. Soms in Mechelen wel face2face meetings, maar waren één keer in de maand, verder eens in Mechelen maar beperkt.

Question: In your opinion, did you find your coworkers to be working in a silo, or as a one group?

Nee, iedereen was team, op een uitzondering na.

Question: Describe the company culture, how are the relations with the coworkers, the management … Is there a lot of team spirit, a top-down culture, bottom-up culture, is there a specific hierarchy?

Goed binnen het team, op een uitzondering na. Met management totaal geen contact. Top-down cultuur overheerst, één keer per maand demo, en dikwijls afgekraakt over futuliteiten, eerste sprints zat het er al meteen op. De PM en de Lead Dev waren geen bazen, wel coaches. Wel vanuit het management dat er 'bazen' zijn.

Question: Was the project done ad-hoc, or did it reuse architecture from other projects, programs, … Was this integration easy?

Wederom het CSP dat gemaakt werd in Barcelona, geen documentatie of hulp beschikbaar. Plantrekkerij.

Question: Talking about the project itself, did you find it was a rather easy project to build, or were the features of the project more complex?

Nee, maar altijd op ERP gewerkt dus dat was een eitje.

Question: Was there a need for being compliant with a regulatory instance? How was this handled?

Qua wetgeving kregen we alles door van de analysten, versturen van facturen via mail moest toch wel bekeken worden of dit mocht. Mag allemaal wel. Gewoon geïmplementeerd.

Question: What was the technical complexity of the project, which technologies were used, and how are they evaluated?

CSP was zeer complex, veel overhead, niet bruikbaar, extra traag, extra moeilijk.

Algemeen besluit, als we het project zonder CSP hadden gemaakt was het 5 à 10 keer beter geweest. Het lag niet aan de mensen, maar aan de technologie.

B.2 Joris Dresselaers

Participant: Joris Dresselaers Role: Lead Technical Architect Projects Involved: OIv1/OIv2

Date: 9 april 2014

The goal of this survey is to evaluate how the 'Online Invoicing v1.0' project was handled in terms of software delivery process. These questions below have the goal of identifying success factors of the project, and potential improvements in the SDLC. As this is an in-depth survey, a basic set of open questions is documented that can be used as a starting point of the discussion.

Question: Can you tell about the context of the project, why was it founded, what are the drivers?

Vanuit business perspectief omdat er een nood was aan een kleinschalige facturatieoplossing voor SME's en dat het een gat in markt was. Kluwer vandaag niet zo een soort oplossing aanbiedt. vanuit een meer technisch standpunt als pilootproject om te kijken of we klaar zijn om zelf SaaS apps aan te bieden en te testen of het CSP kan werken, en of we onze projecten kunnen outscalen en scope verbreding en migratie van het landschap.

Question: What was your role in the project?

Niet mee in het scrumteam, van buitenaf een technische kijk er op als lead architect aansturen van het architectuurteam, lead developer een aantal developers en een architect die een ondersteunende rol en communciatie met CSP team, en daar zat ik ook wat mee op. Geen rechtstreeks deel van het core team.

Question: In your experience, which topics did you evaluate as positive during the delivery of the project?

Moelijke vraag, want niet veel dingen lopen goed. Als positief ervaren dat het team en daarmee het devteam zich bleef inzetten ondanks de vele nederlagen, een team met correcte mindset maar niet meer dan dat, er was een wil om te vechten. Ze willen leren en zichzelf naar hoger niveau tillen ondanks de obstakels.

Question: In your experience, what where the struggle points that the team tried to solve during the project? Which of these topics were solved and how, and which topic remained a problem by the delivery of the project?

Development platform in beta versie, zeer slecht idee om daarop te beginnen developen, maar moest iemand de eerste zijn, en we hadden functionele scope nodig om te starten, wij genomineerd.

Development team dat geen ervaring had op dat platform, geen ervaring in de technologieën die in dat platform verwerkt waren.

Een architectuurteam zonder ervaring met het platform dat in zijn kinderschoenen staat.

Vanuit governance standpunt de organsatie rond het project die veel te zwaar, was veel te veel mensen met verschillende rollen die involved waren in beslissingsproces. Geen duidelijke beslissingrechten, geen persoon met verantwoordelijkheid, immense scope creap, geen duidelijk project, veel te veel bewegend.

Vanuit business standpunt was er een lack of business vision, gemis aan visie, waar gaan we naartoe, wie is onze klant, wat gaan we er voor vragen, want zijn onze ciritical succes factors, dit was niet duidelijk voor het devteam, de organisatie was daar niet van doordrongen.

Geen functionele analyse, enkel business analyse, die was sterk voor interpretatie vatbaar, en vooral de happy flows behalve de exceptional flows waar het meeste tijd in kruipt om te maken, de vertaling van de business naar de developer door iemand die daar kaas van gegeten heeft en over nadenkt. De rol van functionele analyst is niet aanwezig.

User experience hebben we niet op gefocust.

Unit testing werd niet gedaan. Usertesting niet gedaan. Alles wat er tussenlicht ad hoc, system testing.

Question: Who was your customer, and how was he involved in the process?

Geen eindklant, en was niet betrokken.

Question: How was the team organized? Can you discuss how a typical development iteration took place?

Sprints van een maand, begon altijd met presprintplanning, PM, lead dev, architect & soms lead architect en analyst. Sprintplanning, waar volledig devteam en pm en lead dev en analyst. Eerste sprint maar daarna altijd retrospective en demo. In het devteam zelf zaten developers en lead dev, tech arch en PM en analyst.

Question: Did you find that you had the right tools to use for your role?

Gedistribueerd team, was een problem, dus daily standups hadden context nodig, interactief digital scrumboard is wel ok, maar geeft niet de echte situatie meer, meer administratie minder the real life situation. Rol veel communiceren met mensen uit buitenland (spanje, italie), niet de juiste communciatietools, geen zicht op andere agenda's, shedulen van meetings een ramp, 15 keer verplaatst, jezelf behelpen met commtools (office comm, lynq, niks), ...

Question: When you think about the project scope, was this clear upfront?

Nee

Question: Were there frequent changes during the project execution, and how were they handled?

Ja, heel veel, ad hoc, geen process, het is veranderd dus omgooien, niet scope herbekijken, niet herbekijken van de timeline, we nemen het mee en we zien wel.

Question: Was there a specific process that was followed?

Niet methodologie noemen, wel recurring governance meetings, over reqs praten, … geen visie achter het process, we deden zo maar iets.

Question: How many people did your team consist of, including product owners, customers, …

Question: Did you collaborate with other teams for this project?

Ja, mensen uit italië en spanje als architect, analyse voor accounting deden we ook redelijk wat workshops mee, legacy ook nog zaken.

Question: Was everyone working in the same place? Were there face-2-face meetings or was this done otherwise?

Nee, distributed teams, zowel developers als analysten. Is voor beginnend team zeker geen goed idee, ik denk dat het beter zou zijn moesten we allemaal samenzitten, overtuigd dat dit overhead is.

Question: In your opinion, did you find your coworkers to be working in a silo, or as a one group?

Silo. Business, IT zeer sterk in place, weinig samenwerking er tussen, weinig interactive, informele manier nee. ANA, DEV en ARCH binnen silo van IT organisatie meer wel ok. Over teams heen was ok.

Question: Describe the company culture, how are the relations with the coworkers, the management … Is there a lot of team spirit, a top-down culture, bottom-up culture, is there a specific hierarchy?

Binnen het projectteam goed. Legacy van bedrijfscultuur die gegroeid is uit acquisities waar verschillende mensen uit verschillende silo's worden samen gezet in één projectteam, in het begin niet evident, maar redelijk goed gegaan, de mensen zijn er wel op geselecteerd, mensen stonden er wel voor open, qua change management viel dat mee. Enkel voel je wel dat de mensen niet gewoon zijn om projectmatig te werken, bedrijfscultuur van 9-5 is zeer sterk aanwezig, flexibiliteit gaat gepaard met gemor.

Hierarchie, top down: bijvoorbeeld het meest pertinente voorbeeld waren de demo's, je voelt die topdown organisatie, gefocust op details en die worden uitvergroot en uitgelicht en gebruikt als stok om mee te slaan terwijl devteam toch zeer hard haar best doet om doelstellingen te bereiken. Weinig involvement van management in het verhaal. Weinig accountability, het is altijd iemand anders zéér hierarchisch.

Question: Was the project done ad-hoc, or did it reuse architecture from other projects, programs, … Was this integration easy?

CSP als basis, was het 'walhalla' dat uit de hemel neerdaalde, waarvan we blij van mochten zijn dat we er op mochten developen, we zaten in de structuur en architectuur van het verhaal, dit is hergebruikt en je werd op de vingers getikt als je niet volgens de regels werkte. Slecht, niet constructief, ging niet, duidelijk dat de governance niet klaar was om dit project op te leveren. Heel defensief agressief.

Question: Talking about the project itself, did you find it was a rather easy project to build, or were the features of the project more complex?

Voor de beperkte scope van het project vond ik het complex, omdat de scope nooit duidelijk was wat we gingen doen, en daarom omdat er altijd een was heeft rondgehangen wist je niet wat je moest opleveren en raak je nergens.

Question: Was there a need for being compliant with a regulatory instance? How was this handled?

We denken dat we verplicht zijn om de klant aan te bieden op de originele drager, impliceert dat we geen full SaaS kunnen doen, werkt limiterend. En we moeten de facturen x-jaar bijhouden.

Question: What was the technical complexity of the project, which technologies were used, and how are they evaluated?

CSP als dev-omgeving, .NET framework 4.0, Entity Framework 5.0, ASP .NET Webforms, Telerik Control Library, MVVM+C, Ajax, Javascript, Facade patterns, Repository Pattern, T4-templates for code generation en Enterprise Library.

Was niet het soort technologystack die dient voor het soort applicatie dat we gaan bouwen, de devomgeving is voor RAD visual studio on steroids, is niet het soort app dat we willen maken, we willen revolutie en innovatie en 'wauw'-gevoel kweken, 'is dit van Kluwer, waauw!, hier ga ik alle software van kopen voor heel mijn leven', geen software die eenheidsworst is in een software factory. Gewoon een mismatch, merkt je ook op het onthaal, niet wat we nodig hebben. Ook aan het vechten tegen de complexiteit om het simpeler te maken, alle abstractielagen hadden we niets aan, debuggen in source wat er aan de hand is. Alle abstractielagen extra bijkennen en complexiteit te leren of te omzeilen.

B.3 Marc Caelen

Participant: Mark Caelen Role: Business Development Manager Projects Involved: OIv1/OIv2

Date: 09/04/2014

The goal of this survey is to evaluate how the 'Online Invoicing v1.0' project was handled in terms of software delivery process. These questions below have the goal of identifying success factors of the project, and potential improvements in the SDLC. As this is an in-depth survey, a basic set of open questions is documented that can be used as a starting point of the discussion.

Question: Can you tell about the context of the project, why was it founded, what are the drivers

Scope beperkt houden, behapbaar houden, materie die we kenden, geen nieuwe materie, risico beperken op kanibalisatie van portefeuille, eerder een revenue generator dan migrator, niet de bedoeling om bestaande software te vervangen, maar toevoegen, vanuit eigen ervaring wisten we dat er nood was aan eenvoudige facturatie en dat huidige pakketten te veel functionaliteiten hadden obv wat we hoorden in de markt, niet zelf afgechecket. Vanuit bepaalde gewoonte gezegd voor ons zit daar potentieel in qua product en nieuwe segmenten, zijnde eenmanszaken waar we geen offering voor hadden. Rond ons ontstonden facturatiepakketjes, het was te moelijk, kijk eens naar dat... Eigen business case gemaat toch in materie waar we ons thuis in voelde. Niet de bedoeling om grote omzet te draaien, omzet om de whistespot in de offering in te vullen, marktgericht.

Question: What was your role in the project?

Business & Product Owner, requirements bepalen, scope uitzetten, executie opvolgen.

Question: In your experience, which topics did you evaluate as positive during the delivery of the project?

Niet zo veel eerlijk gezegd, sfeer van negativiteit gestart, wat positief uithalen? De ene negatieve boodschap na de andere, intern, extern, negatieve feedback, frustratie, in dusdaning negatief gesternte, als ik er iets positief uithaal, zo had het nooit mogen gestart zijn. Wel positief: in controle van de scope, niet van de requirements, ik kon bepalen wat er in moet en dat is niet zoals ik het gevraagd heb. In controle van de scope.

Question: In your experience, what where the struggle points that the team tried to solve during the project? Which of these topics were solved and how, and which topic remained a problem by the delivery of the project?

Onvoldoende structuur, onvoldoende duidelijke rollen en verantwoordelijkheden, wel gedefinieerd maar we handelen er niet naar, onbepaalde scope, van sprint tot sprint veranderen, geen doelstelling gedefinieerd, business requirements waren er niet, eenvoudig facturatiepakket veel te high level, keer op keer aanpassingen vragen. Verwachtingen matchen niet tussen business en team, veel sprinten, geen constante in progressie, geen evolutie te meten, afhankelijk van het platform, was niet matuur.

Alle dingen die we fout hebben gedaan. Er was geen teamspirit, in coachende rol dingen fout gedaan. We hebben het traditionele manier van managen toegepast 'goed is normaal, slecht is kletsen'. Naar project management toe hadden we als business owner niet altijd de juiste informatie. Het was een moving target, je kon er op schieten maar het beweegt te veel. Dus je wist niet waar het ging uitkomen, wanneer, hoewel er toch efforts gedaan werden; Sprint 5, catastrofeel, team pandoering, Sprint 6 was schitterend, boodschap begrepen, 7 weer slecht, 8 weer beter, ... geen verkeerde perceptie van het team omdat je niet kan meten, geen referentiepunten.

Question: Who was your customer, and how was he involved in the process?

De eindklant was niet betrokken, werd vertegenwoordigd door de business owner, gehandeld op basis van intrinsieke kennis, ervaring van piet, johan & guido die al jaar en dag in de erp zaten en dus worden verondersteld de klant te kennen.

Question: How was the team organized? Can you discuss how a typical development iteration took place from your point of view?

Vrijdag voor de sprint begon de sprintplanning, project manager en product owner (johan) en de architect (technisch profile) en bekeken de backlog, ok, dat zijn de topics. Dan de echte planning met het team en rond bepaalde usies kiezen, heel veel discussie over 'wat is dat nu?', reqs verheldering en planning. Dan ontwikkeld, sprinten van 4 weken, zonder echt thema, usie x,y,z, het doel of de deliverable was niet helder, geen potential shippable increment. Daarna de demo, donderdag voor de demo preparatie (was te laat in mijn ogen), vrijdagochtend de demo: team, paul en ik uit business, johan was veel feller betrokken, dagdagelijkse bezigheid. Retrospective waar alleen de eerste sprint er bij waren, daarna niet meer, was niet zo erg, retrospective is van het team, sponsor is extern, geen integraal deel uit van het team, wel het klankbord, jij zegt of het goed is of niet. Het was goed, want door het team werden we beschouwd als boemand. Je leert hier mee omgaan, je begrijpt het ook, er was een dermate frustratie, want de middelen en competentie waren er niet om verwachtingen in te vullen. In de zaagtandfase erkend, we hebben proberen te escaleren, invloed op technische luik was te beperkt, hele governance verhaal van csp 1 zat niet goed en zelfs christine ingeschakeld, exceptional steerings e.a. om ons probleem duidelijk te maken, communicatie tussen CSP en Team was verre van optimaal, meer frustratie dan communicatie, wel geprobeerd maar niet geslaagd.

Question: Did you find that you had the right tools to use for your role?

De reporting die we kregen is voldoende, maar niet altijd correct of consistent. Burndown charts waren niet uptodate, dit kan niet de realiteit zijn, en als dit de realiteit is, dan is de prognose niet goed. Je krijgt de signalen dat het niet goed gaat, maar de charts zeiden iets anders, we konder er de juiste interpretatie uithalen.

Question: When you think about the project scope, was this clear upfront?

Nee, zie eerder, was veel voor interpretative vatbaar.

Question: Were there frequent changes during the project execution, and how were they handled?

Zijn veranderingen geweest, niet alles uitgeschreven, dus als we de eerste factuur zagen en er stond geen intracomm op, hoe is het mogelijk dat we dat niet hebben, dus we zullen het meenemen in de volgende sprint, alles kwam erbij, inschattingen niet juist… Veel voortschrijdend inzicht. Veel wijzigingen in het platform waar we geen invoeld op hadden, tenzij escalatie. Centraal team voor platform werkte niet, dedication was ook minder.

Question: Was there a specific process/methodology that was followed?

Volgens het WKB Princell system gewerkt, mandate, pid, goedkgekeurd, tfs gebruikt, sprints gewerkt, artfefacten van analysemethodes.

Question: How many people did your team consist of, including product owners, customers, …

Ongeveer +10 personen.

Question: Did you collaborate with other teams for this project?

Er was een bepaalde samenwerking nodig voor het CSP, maar dit was eerder klant-leverancier.

Question: Was everyone working in the same place? Were there face-2-face meetings or was this done otherwise?

Er waren face2faces, ook virtuele meetings, niet iedereen op dezelfde plaats, Gent en Hasselt.

Question: In your opinion, did you find your coworkers to be working in a silo, or as a one group?

Binnen development team was er zeker groepsgevoel, architecten meer bezig om problemen met CSP op te lossen, hans, guido meer functionele problemen oplossen. Het reageerde wel als een team. Ik reken mezelf niet als deel van het team.

Question: Describe the company culture, how are the relations with the coworkers, the management … Is there a lot of team spirit, a top-down culture, bottom-up culture, is there a specific hierarchy?

Top-down cultuur, komt van bovenaf, naar onder toe, met opdrachten van voer het maar uit, weinig bottom-up, star in uitvoering en in mindset. Gesloten mindset. Vanuit bestaande denken en niet vanuit het nieuwe, is een kopie van een windows app, gedacht vanuit redevelopment van beperkte functionaliteit, iets kleiner maken en we denken daar omzet mee te halen. Te weinig engagement in executionphase, voorbeeld sprintdemo is niet verboden voor een bu-director, ontwikkelteam, sales marketing, management niet betrokken. Toch nog te veel silo, chinese muur, om de vier weken blokje over de muur, bedrijf heeft dat wel beetje in zich, de zuilen die zich organiseren organisatiegewijs, met horizontale lagen ertussen. Vanuit silo-development organisatie gedacht.

Question: Talking about the project itself, did you find it was a rather easy project to build, or were the features of the project more complex?

Nee, maar ja door het CSP wat niet te managen was; Geen influence. Domeingericht niet complex, maar technologisch wel, geen beïnvloeding mogelijk op technische componenten om het functionele te maken.

Question: Was there a need for being compliant with a regulatory instance? How was this handled?

Niet aan gedacht, niet aan de orde, niet op voorhand scope duidelijk, geen echte volledige aandacht voor stakeholders, puur vanuit development organisatie gedacht. 'we gaan een app maken', voor wie? Hoe gaan we naar de markt? Waar nodig? Wat moet product doen? Er was enkel aandacht voor fucntionaliteit, niet voor alles er rond.

Appendix C: Interview Transcripts Online Invoicing v2

C.1 Tom Princen

Participant: Tom Princen Role: Functional Analyst Projects Involved: OIv2

Date: 9 april 2014

The goal of this survey is to evaluate how the 'Online Invoicing v2.0' project was handled in terms of software delivery process. These questions below have the goal of identifying success factors of the project, and potential improvements in the SDLC. As this is an in-depth survey, a basic set of open questions is documented that can be used as a starting point of the discussion.

Question: Can you tell about the context of the project, why was it founded, what are the drivers

Kunnen bewijzen dat we de nieuwe technologie onder de knie hebben, mensen intern naar een hoger niveau brengen, en de nieuwe way of working in place gebracht door improvement plan testen en ondervinden aan den lijve of het beter is of niet. Project begint vruchten af te werpen, buy-in van de organisatie.

Question: What was your role in the project?

Functionele Analyst, schakel tussen business en vertaling doen naar development. Klankbord zijn voor vragen die er komen vanuit development.

Question: In your experience, which topics did you evaluate as positive during the delivery of the project?

De manier van werken op het team is heel positief, vroeger was het 'ik gooi documentatie over de muur', nu is het een 2-way communication, en development beseft dat als er code wordt aangepast dat de analyse ook moet bijgewerkt worden. Ook mijn rol was vroeger niet uitgewerkt en de bijhorende deliverables ook niet, nu wel, die zijn veel dichter bij de code.

Question: In your experience, what where the struggle points that the team tried to solve during the project? Which of these topics were solved and how, and which topic remained a problem by the delivery of the project?

De maturiteit van verschillende analysten was redelijk laag, vrij onzeker door insourcing van expertise, maar het UX-vlak blijft een moeilijk punt. Rare requirements die van business werden doorgegeven, vroeger werd het toch uitgevoerd, nu wordt er serieus op gereageerd en gemitigeerd. Er wordt nu kritisch nagedacht over toxic requirements. Van belang voor technische profielen, technologie is googlebaar, veel meer transparantie hieromtrent.

Question: Who was your customer, and how was he involved in the process?

Mijn klant is de 'loodgieter', wordt betrokken in het project door contextual design te doen, vragen wat hij er van vindt, nu vooral opgepikt via outsourcing, persona's opgesteld, gebruikt geweest als

geheugensteun tijdens het opstellen van de requirements. Er wordt nu ook nog steeds op gesprek gegaan bij de klanten voor testing, kathleen is hier mee bezig.

Question: How was the team organized? Can you discuss how a typical development iteration took place?

Typische sprint verloopt met sprintplanning, na de retrospective van de vorige sprint, voorstelling van wat er in scope genomen is, product owner stelt backlog voor, developers kijken na, schatten in en plannen de sprint in, hoe gaan we het aanpakken, elke dag scrummen, twee weken aan een stuk. Vragen beantwoorden van development, verschil tussen eerste en tweede week, tweede week vooral voorbereidingswerk, we werken wel 't een en ander op voorhand uit, maar ook tijdens de sprint zelf, nu meer analyse tijdens de sprint zelf, dat is positief, de analyse upfront verduidelijtk het beeld, de analyse tijdens de sprint laat finetuning door door development.

Question: Did you find that you had the right tools to use for your role?

Ja, behalve EA is een ambetante tool qua UX, knoeien met de riemen die we hebben, ook de laptop is zeer slecht, van moment dat we naar de nieuwe locatie gaan gaat scrumboard mss digitaal moeten, minder tevreden over dit vooruitzicht. Belangrijk is het artisanale de communicatie, zeggen wat je bezig bent. Elektrisch gaat het meer een opvolgingsformaliteit voor de PM waar hij uren op gaat checken, is minder nuttig. De keren dat ik op afstand zat is het wel moeilijk met het fysieke scrumboard. Voor piet moet het heel moeilijk zijn, je ziet het niet bewegen, wel overtuigd dat een team dicht bij elkaar moet zitten om het beter te laten gaan, dit is echter niet haalbaar.

Question: When you think about the project scope, was this clear upfront?

Ik vond het redelijk duidelijk op voorhand, algemene requirements duidelijk, op enkele rare reqs na, maar dat is altijd zo. Voldoende niveau uitgewerkt, nadien kom je in detail wel zaken. Net genoeg op voorhand gewerkt, laat u de mogelijkheid om na te denken met het devteam hoe je bepaalde zaken gaat oplossen. Je neemt al een heel pak designbeslissingen upfront die misschien tijdens de ontwikkeling beter anders aangepakt worden. Het is goed om na te denken over het wat in plaats van het hoe. Waterfall analyse zorgde voor analysis paralysis waardoor we slechte solutions maakte.

Question: Were there frequent changes during the project execution, and how were they handled?

Er waren wel wat veranderingen, afhankelijk van de boondoggle meetings omtrent, nog enkele ideeën vanuit business, de changes kwamen in een log, besproken met de product owner en indien relevant meegenomen, zeker ok. Binnen de sprint zelf worden er geen changes in de scope toegelaten en dat is goed en belangrijk.

Question: Was there a specific process that was followed?

Prince II.

Question: How many people did your team consist of, including product owners, customers, …

8 à 10.

Question: Did you collaborate with other teams for this project?

Nee, consulteren voor domeingerelateerde zaken, nu alles onder controle. Wel nog met GSS voor de infrastructuur.

Question: Was everyone working in the same place? Were there face-2-face meetings or was this done otherwise?

Nee, niet echt, piet niet, Kathleen niet, hangt er vanaf, het devteam zit wel op dezeflde plaats. Minder geleuter als iedereen op één plaats zit, belangrijk, eiland verder is soms al moeilijk qua communicatielijn. In het begin heel moeizaam, afstand zorgt voor afstand, nu meer de neiging om te skypen voor het minste, dat werkt beter.

Question: In your opinion, did you find your coworkers to be working in a silo, or as a one group?

Groep.

Question: Describe the company culture, how are the relations with the coworkers, the management … Is there a lot of team spirit, a top-down culture, bottom-up culture, is there a specific hierarchy?

Binnen het team een heel positieve cultuur, open voor vragen, open voor opmerkingen, er wordt geluisterd, niet denigrerend, andere teams hebben dit niet en dit kan wel toekomstige problemen. Tegenover management staan negatief tegenover suggesties en opmerkingen, hangt er vanaf over wat het precies gaat, als je er kan op inspreken, maar percipieert niet bereikbaar te zijn. Meer vlakke structuur binnen het team. Nog steeds meer top-down, business zegt hoe het moet en duwt er toch nog zaken door.

Question: Was the project done ad-hoc, or did it reuse architecture from other projects, programs, … Was this integration easy?

Er is architectuur hergebruikt van de proof of concept, er is wel een integratie met VIES en UBL, eerst gezegd om bestaande architectuur te hergebruiken, maar was een dependency richting slechte code, en daarom niet hergebruikt. Er wordt ook geen gebruik meer gemaakt van het CSP.

Question: Talking about the project itself, did you find it was a rather easy project to build, or were the features of the project more complex?

Er is zeer lage domeincomplexiteit.

Question: Was there a need for being compliant with a regulatory instance? How was this handled?

UBL, E-FFF, compliant want we zijn voortrekker, documentatie opzoeken. Niet echt complex, maar er was wel politiek getouwtrek.

Question: What was the technical complexity of the project, which technologies were used, and how are they evaluated?

Alles qua javascript is complex, asynchrone technologie is wel moelijk, maar niet dermate complex.

C.2 David Van Steenkiste

Participant: David Van Steenkiste

Date: 23/07/2014 Role: Project Manager Projects Involved: Olv1 & Olv2

Question: Can you tell about the context of the project, why was it founded, what are the drivers, did it change towards v2?

Olv1 drivers: de meest concrete driver was: CSP technisch platform zweeft al lang in spanje, verantwoordelijke zei dat het matuur genoeg was voor ontwikkeling, omdat onze technische architecten er op bezig waren, kleine scope, goede scope business project, marktesgment dat we nog niet hebben aangeboord (ook voor Olv2), daarom er mee gestart, klein project om eigenlijk ETNIA in execution phase te krijgen, wetende dat er een heleboel zaken nog niet in place waren.

Business drijfveer is nog steeds dezelfde: we hebben een marktsegment dat we nog niet aangeboord hebben, maar ook een PM-drijfveer: laat ons iets nemen dat we kennen; Qua andere drijfveren (CSP, interne drijfveer, zat er nu een veel bredere change management drijfveer in, we gaan toch eens zorgen dat we op vlak van people/process/management/... op de sporen krijgen, fase op programma niveau doorgemaakt om structuur, alle pijnpunten vastgepakt en te verbeteren, nieuwe start maken). Wat niet wegneemt voor den 2, dat het topmanagement daar niet meer helemaal achterstond, is wat geforceerd geweest omdat je nu eenmaal met iets moest beginnen.

Pijnpunten die er waren: skills/ervaring binnen het team;

tools (architectuur voorhanden), vroeger was dat server-oriented architectuur, vooral geënt om langs serverzijde technisch alles implementeren, was heel duidelijk dat dat niet de way to go was; Lang op gewerkt om de ogen te openen qua technische architectuur;

Process was ook een pijnpunt: maturiteit van kluwer als organisatie is nog steeds een weg af te leggen, maar op vlak van business development was er veel te weinig houvast om dat goed gemanaged te krijgen, geen SDLC, procesmatig werken niet van de grond krijgen.

De “what’s in it for me”, er waren zodanig veel problemen, de motivatie om tot een succes te komen was zeer moeilijk, met vallen en opstaan gelukt, maar bij Oiv2 was er wel structuur én architectuur, er was véél meer drive, toen legde management een aantal zaken op en was niet bereid om bijvoorbeeld externen aan te wenden.

Question: What was your role in the projects?

Project manager, maar voor de 1.0 moest ik overal mijn neus in stoppen, program manager risk & issues naar management, scrum master, … rollen en verantwoordelijkheden zaten niet goed. Enorm veel werk aan het verzetten, en dat je op vanalles afgerekend kan worden omdat je met 1001 dingen moet bezighouden, een PM zou zich vooral met PM-werk moeten kunnen bezighouden.

In de 2.0 waren de juiste mensen op de juiste plaats, veel meer focus op PM-werk. Standaardzaken: planning, communicatie, risk & issue management, quality management.

Question: In your experience, which topics did you evaluate as positive during the delivery of the first project?

Op zich in het kader van change management is het goed dat we dat project gedaan hebben, het verhaal van becoming an online company, organisatie is vandaag cd's aan het branden en per post aan het opsturen naar de klanten voor updates, maar je moet wel naar een heel nieuwe businessmodel waarbij je SaaS gaat leveren, dat betekent dat je op gigantisch veel punten, van de visie tot de infrastructuur, van project management, testen, … het eerste project heft er voor gezorgd dat we gestart zijn met zo goed mogelijk voorhanden team, met goede business reasoning (klein packet naar de markt, nog niet in vertegenwoordigd), heeft er voor kunnen zorgen dat het change management gewijs en drive gewijs bij de mensen in het team de problemen veel tastbaarder hebben kunnen maken naar het 'top'managmeent, waardoor we veel zaken echt op de rails hebben kunnen zetten, zoals het onboarden van Lars als program manager.

Het heeft een hele tijd geduurd (maart 2012 gestart met analyse, juli 2012 met development), initiaal tegen eind 2012 klaar, uiteindelijk in september 2013 de stekker er uit getrokken.

Question: In your experience, what where the struggle points that the team tried to solve during the first project? Which of these topics were solved and how, and which topic remained a problem by the delivery of the second project?

Door samen met Lars het improvement plan te doen, dat we op die aspecten de juiste zaken in place gebracht.

Er is een duidelijk process gedefinieerd (SDLC), scherp; duidelijke rollen en verantwoordelijkheden (anders is er geen proces); We hadden een goede technische architectuur om van te starten, nog niet volledig op punt, maar er was een goed startpunt; er is een goed team gedefinieerd geweest met de mensen die de skills en de ervaring hadden, ook de internen mee kunnen trekken in de goede richting; de balast die opgelegd werd vanuit CSP spanje (bug proces, ...) dat als tang op varken werkte hebben we ons eigen proces gedefinieerd, we zaten veel beter in de startblokken. De samenwerking met business zat ook al een stuk scherper, een dedicated product owner met duidelijk mandaat; paar keer serieus op moeten duwen (in the end was het Mark, maar dit was niet realistisch, nu is hij business manager, de product owner heeft mandaat). Heel duidelijke communicatie vanuit management: business visie & roadmap document: dit is etnia, daar gaan we voor, invoice v2 hangt samen, iedereen duidelijk de visie, ook bij dev'ers, de reasoning was duidelijk en iedereen geloofde bij de start dat we succesvol konden zijn, de controle was er, er zijn nog steeds risico's, maar behapbare risico's terwijl er bij Oiv1 gigantisch veel issues waren. Als je er een andere PM zou hebben opgezet, met pakken meer ervaring, zou het dan anders gelopen zijn? Ik ben iemand die open communiceert maar ook inzet om er iets van te maken.

Question: How are things running now?

Nu loopt het al seen gecontroleerd project, maar het blijft in de context van becoming an online company pionierswerk, gigantische change, veel risico's en issues, veel communiceren, maar dit kan nu op een gecontroleerde manier gebeuren, er zijn nog issues die opduiken, maar het zijn geen uitslaande branden niet meer waardoor de heleboel in de soep draait. Het blijft een eerste software delivery project binnen een zeer ambitieus programma, als we eens in de markt staan met invoiceone en de hele organisatie is doorlopen (legal, marketing, sales, ...) en niet alleen development, dan kunnen we gaan schalen, maar op dit moment kunnen we dat nog niet. We kunnen als schalen op voorstudie vlak, maar we gaan geen verschillende projecten tegelijk in execution krijgen, eerst de andere niet-delivery departementen meekrijgen. Development/GSS is ook een moeilijke affaire, maar ook stappen gezet, maar nog een weg te gaan. V1 was brandjes blussen, nu is er controle.

Question: Who was your customer, and how was he involved in the process in both projects?

Olv1: De klant was Johan & Mark: product developers; Johan was de product owner, was wel bij de daily standups, maar had weinig beschikbaarheid, op de demo's waren ze er ook bij; de knop moest doorgehakt worden door Johan, en vaak eens met Mark erbij. De eindklant was niet in de picture, intrinsiek vanuit Johan zijn rol, rekening mee gehouden in zijn reflecties, maar niet actief betrokken, we hebben één keer usertesten gedaan met de versie die we hadden tegen september 2013, waarmee we ook naar management geweest; Resultaten gebruikt om V2 mee af te toetsen dat er nu een veel beter resultaat is.

Oiv2: Effectieve eindklant hebben we in het begin, tijdens de initiatiefase betrokken bij het project, het verhaal van de persona's, moodboards, wireframes testen bij potentiële eindklant testen, was niet binnen 1.0; regelmatiger usertesten gedaan, is ook vooral omdat we nu de tools, skills en expertise aan boor om dit te doen, in de oiv1 hadden we niet eens de infrastrctuur voorhanden om dit te doen, het was niet dat we dit niet wilden doen, maar er waren technische en infrastructuurbelemmeringen om dat te doen.

Question: Did you find that you had the right tools to use for your role?

Aan de tools zal het niet gelegen hebben; Testing was wel crap om dat daar in te doen; maar dat was niet de reden dat het moeizaam verliep. Als je naar het devteam kijkt van de v1; als je dat team zou hebben laten werken op de v2 had dat ook wel een heel pak vlotter gegaan. En ik heb nu natuurlijk een project status rapport template; in de v1 had ik wit blad en mocht ik mijn zin doen, nu hebben we een program manager die duidelijk zijn rol vervult, dan moet je daar geen tijd aan verspillen. Op Program Management vlak ben je bezig over de manier waarop je rapporteert, maar op PM-vlak was het rapporteren zelf.

Testen verliep op dezelfde manier als de v2, de analist die test, maar de business value die in v1 werd opgeleverd was zéér weinig waardoor er weinig te testen viel. Je moet ook infrastructuur hebben je moet een process hebben voor een nieuwe deploy, was veel trekken en sleuren, zelfde issue als in v2.0, analist die te weinig tijd hebt om te testen, extra mensen aantrekken om te testen. We hadden niet echt veel, nu hebben we wel een testplan, nu zijn er wel al stappen vooruit, maar nog geen regressietesting

die we doen; unit testen in v1 waren zeer beperkt omdat de architectuur dit niet toeliet, codebase gaf zelf al veel warning en issues, nu gaat dit al beter. Nu nog steeds op testing vlak stappen te zetten: de dag vandaag doen we unit testen en manuele systeemtesten; ik wil sowieso naar een situatie waar we geautomatiseerd gaan testen (nu te vroeg), maar als we in de markt staan moet dat er zijn, dat we ook in een situatie zitten dat de systeemtesten ook geautomatiseerd verloopt, ook naar regressie toe, en dat we ook een doordachtere manier hebben om met onze testplannen om te gaan. Onze web applicatie moet op 5 verschillende browsers, op x-aantal devices goed werken, we kunnen dat manueel maar dat is onbegonnen werk om dit op te schalen. En dan ook de stap naar integratietesten enz. Voor één applicatie in de markt te krijgen is het niet dat er grote wielen gaan afgereden worden maar kwalitatief niveau kan nog beter.

Question: When you think about the project scope for each project, was this clear upfront?

Op zich functioneel wel. Voor OI v2, idem.

Question: Were there frequent changes during the projects execution, and how were they handled?

Veel niet nee; ook niet veel in v2; qua scope controle valt het wel goed mee. Groot verschil wel is dat het aan business is om duidelijk te maken wat de scope is die ze verlangen, zodat we een goede product backlog is, als we sprints definiëren heb je een zeer goede wisselwerking nodig tussen devlead en product owner om dat te doen, bij OI1 hadden we niet de personen die dit onder de knie hadden; op de OI2 hadden we hier wel de ervaring en skills om op verstandige manier sprintbacklogs definiëren, twee sterke figuren nodig die kunnen definiëren wat ze willen, als de ene de bovenhand haalt op de andere gaat business niet tevreden zijn wanneer er business value wordt opgeleverd dan anders; functionele versus technische drive in de knoop. Zit nu veel beter.

Question: How many people did your team consist of, including product owners, customers, … , what has changed in the second project?

Vanuit duidelijke rollen en verantwoordelijkheden vertrokken in Oiv2; plus het feit dat we skills en experience onboard hadden; in de 1.0 geen ervaring op webdevelopment en niet op CSP; volledige grip op architectuur in v2, groot verschil, vergelijking steven: “iemand vraagt om put te graven met pikhouweel zonder compressor, dat gaat, maar in de 2.0 had je een bulldozer én een plan”.

OI1.0 wisten we niet beter, we gaan er voor, we moeten en we doen ons best, maar als je niet weet wat er beter kan, dan is dat moelijk. In OI2.0 hadden we team met ervaring die verbeteringen kon uitstippelen.

En ook boondoggle aan boord op UX-stuff, dat heeft wel véél bijgedragen qua html’er, responsive, ... veel efficiënter inzetten van onze centjes.

Question: Did you collaborate with other teams for these projects?

Olv1: met GSS, maar nooit zover geraakt, ook met business maar was geen ander team, zit mee in etnia; sales/marketing nooit zover geraakt, wel opgelijst maar nooit uitgevoerd omdat het core verhaal niet van de grond is geraakt: Ook CSP-team om samen te werken in het buitenland, was zéér problematisch

Oiv2: met GSS samengewerkt, dedicated infrastructuur architect; Legal/Marketing/Sales/Backoffice, services nog niet, nog nooit zover geraakt in 1.0 om daar maar aan beginnen te denken.

Question: Was everyone working in the same place? Were there face-2-face meetings or was this done otherwise? Were there mitigations to help this?

Door het team nooit als een groot problem ervaren, minstens ééns per maand samenzitten, verder skype, maar je moet een zekere vorm van maturiteit bereikt hebben vooraleer dat dit goed werkt; (tools, process, skills, …) en dan nog verlies je sowieso een bepaald percentage aan efficientie op vlak van communicatie, zelfs al zit je altijd in videoconference naar elkaar te kijken, je verliest nog altijd non-verbale communicatie; We hebbel geleerd dat als je zo'n project opstart het beter is om ze samen te zetten.

Question: In your opinion, did you find your coworkers to be working in a silo, or as a one group? Has it evolved over time?

Binnen het devteam+johan was er zeker geen zeer opvallende tegenwerking, er werd wel samengewerkt, maar omdat je met zeer veel moeilijkheden zat zorgde dat voor veel frustraties, misverstanden, agitatie waardoor de samenwerking niet altijd even makkelijk verliep, maar het is niet zo dat er blockage was van we willen niet samenwerken, onderhuis merkte je wel soms wat tegenstrijdigheden, maar geen zeer opvallende blokkages.

Veel betere omstandigheden geschapen om als team kunnen samen te werken (process, tools, betrokkenheid management, ...), dynamiek en dedicated product owner met juiste mandaat geeft een groot verschil.

Als je het team van de 1.0 in de 2.0 en trek de skills omhoog, dan gaat dat ook een team zijn, het is niet dat de mensen aan zich niet wilden samenwerken, je had nood aan de juiste omstandigheden, en dat was in de 2.0 sterk aanwezig. Core verbeteringen: Process & skilled team! Het is echt iets nieuws, als je dat van de grond wil krijgen kost dat veel te veel als je dat doet zonder de twee.

Question: Describe the company culture, how are the relations with the coworkers, the management … Is there a lot of team spirit, a top-down culture, bottom-up culture, is there a specific hierarchy? How did this evolve over time?

Nog steeds een top-down cultuur, maar er zijn wijzigingen gebeurd om het bottom-up te krijgen, rollen en verantwoordelijkheden (ux-expert, product owner, marketing, mandaat om ervoor te zorgen dat het in orde komt, niet teruggefloten door mgmt als er iets niet volledig volgens de goesting is, ze volgen mee.)

Question: Was the project done ad-hoc, or did it reuse architecture from other projects, programs, … Was this integration easy?

In de twee het uitgangspunt te hergebruiken, CSP was bedoeld om te hergebruiken, maar was niet de way to go, voor de Olv2 is het ook bedoeld een basis te leggen voor webapps. De één is bottom-up, terwijl CSP top-down is opgelegd geweest, genre "in de 1.0 stond de fabriek om ford c-maxen te maken, maar het moest scania-trucks zijn", in de 2.0 gingen we scania truck maken, en dan gingen we de fabriek bouwen, nu vlotter en vlotter te krijgen.

Question: Was there a need for being compliant with a regulatory instance? How was this handled?

Uiteraard, houden we rekening met wetgeving: product was dit van toepassing, btw-regelgeving, in de 2.0 aangepakt omdat we de juiste spoc hebben genomen om ons te ondersteunen en de mensen bereikbaar waren, in de 1.0 nooit zover geraakt. Én qua wetgeving op ons project e.d. contractueel met NDA's e.d., maar dat is als outsourcing e.a.

C.3 Sofie Traen

Participant: Sophie

Date: 23/07/2014 Role: Developer Projects Involved: Olv2

The goal of this survey is to evaluate how the 'Online Invoicing v2.0' project was handled in terms of software delivery process. These questions below have the goal of identifying success factors of the project, and potential improvements in the SDLC. As this is an in-depth survey, a basic set of open questions is documented that can be used as a starting point of the discussion.

Question: Can you tell about the context of the project, why was it founded, what are the drivers

Basicly omdat Olv1 is foutgegaan, ze zijn begonnen met een cloudbased web oplossing te gaan, waar je een hele shift in denken en doen moet creëren, poging gedaan, mislukt, waarom: combincatie van héél veel dingen, de technologie waarme gewerkt was heeft niet bijgedragen tot een mogelijk succes, en qua team heeft het deugd gedaan dat er frisse ideeën naar boven kwamen. Martkwaarde ook sowieso, voor mij voornamelijk: moderniseren van de sofware, twee: het feit dat een gebruiker op alle devices aan zijn software kan; drie: ik denk dat het ook makkelijker is om support te leveren aangezien je een centrale oplossing hebt, dus niet langer maatwerk, service gaat verbeteren er door.

Question: What was your role in the project?

Front-end developer: ik sta in voor de implementatie van voornamelijk de frontend volledig nieuwe technologie die in staat is om veel betere ux te bieden, veel performantere software aanbieden, moderne manier van werken die aangenamer is om te gebruiken, technologie toepassen op Olv2 en de rest van de mensen in de volgende apps.

Er wordt ook testing gedaan, functionele testen door de analisten en dedicated tester sinds kort, daarnaast security & performantietesten door een externe partij, wat ik mis is gebruikerstesten en het proberen kapot te krijgen en je niet houden, de piloot testen was er voor bedoeld maar de pilootklanten gebruiken de applicatie niet. Unit testing doen we ook, het zou nog meer mogen, maar ik denk wel dat we een redelijk ok-coverage hebben, de code die testbaar is testen we wel. In het begin zaten testen zeker goed ingebakken in het proces, en de testscenario's waren duidelijk op voorhand, veel functionele code en veel ge unit-test, sinds het functionele quasi klaar is hebben we het testen wel wat naar de achtegrond geplaatst, maar nu hebben we wel een dedicated tester.

Ook de brown bags zijn een goed voorbeeld van hoe we de teams proberen te versterken.

Question: In your experience, which topics did you evaluate as positive during the delivery of the project?

Team, drive is echt geweldig, aanpakken oplossen opleveren, gaan! Algemene aanpak, demo, sprint scrum, zorgt voor hele mooie drive, de samenwerking met iedereen in het team, (ux;analist, technisch,

projectleider), heel sterke wisselwerking, niemand zit op zijn eiland, een developer mag input levern op ux, iemand mag buiten zijn grenzen treden, is niet overal het geval.

Het feit dat we de kans hebben gekregen om ons eigen ding te doen met de eigen technologie, methodologie. Kluwer was nogal gebonden aan technologie die vanuit europa werd gepusht, die heel moeilijk werkbaar is, het kiezen van de eigen technologie is veel beter. De manier van werken geeft de vrijheid om je mening te zeggen, om over alle domeinen iets te kunnen verbeteren. De hiërarchie is voor een stuk weggevallen.

Question: In your experience, what where the struggle points that the team tried to solve during the project? Which of these topics were solved and how, and which topic remained a problem by the delivery of the project?

Kennis en opleiding van andere teams, niet alleen technisch, maar ook de mindset, de nieuwe manier van werken aanleren, de drive terugvinden, uit het vastgeroeste geraken, niet iedereen is hiervoor gedreven om op deze manier te werken.

Toch nog altijd wel wat hiërarchie, er zijn nog wel situaties geweest waarin we als team een mening naar voor schoven die onderbouwd was en toch van tafel is geveegd, niet altijd even onderbouwd, gelukkig niet vaak, maar af en toe, zit er nog een beetje in. Wel een sterke evolutie in de goede richting gaandeweg het project, in het begin 'maak het maar ook al ben je niet akkoord', de laatste keer moelijk gegaan, maar toch heeft business aanvaard en met een open geest naartoe gegroeid, dat had een jaar geleden niet gebeurd.

Question: Who was your customer, and how was he involved in the process?

Ja, in dit geval wel, is niet moelijk; voor volgende boekhoudgerichte producten is dat moelijker; 'wat een boekhouder precise bezighoudt weet ik niet'; Kleine zelfstandigen, KMO's.

Question: How was the team organized? Can you discuss how a typical development iteration took place?

Voor de sprint begint is ere en meeting met analist, projectleider en lead dev om te kijken wat er mogelijks kan opgenomen worden (prioritized backlog), dan analisten gaan het voorbereiden in x-1, sprint x wordt dan een planningsmeeting gehouden, en ook een retrospective gehouden van de vorige, het werk wordt verdeeld, ad-hoc oppikken. Iedere dag een scrum, op het einde een stavaza typisch twee dagen voor einde van de sprint, en helemaal op het einde een demo en een retrospective. Vergeten, maar eigenlijk niet mijn ding, het eigenlijke requirements verzamelen, maar ik weet niet hoe dit gebeurt, maar dat zal er voorkomen.

De doelen die we vooropgesteld hebben, tot de MVP hebben we gehaald, toch voor 90à95%, daarna hebben we wat achterliggend werk gaan doen qua deployment, dan is de functionele scope wat verzacht waardoor we niet altijd hebben opgeleverd wat we als doel hebben gesteld, maar was ook niet echt een probleem, als er in de stavaza bleek dat we de doelstelling niet gingen halen, dan was het alle hens

aan dek om er voor te zorgen dat we de doelstelling haalden, ook de prioriteiten lagen goed, als we aan iets begonnen moest het passen in de sprintdoelstelling. In het begin waren de sprintdoelstelling ook steeds zeer functioneel en concreet, op het einde wat meer flou omdat je de kleine dingen of de achterliggende problematieken werkte.

Question: Did you find that you had the right tools to use for your role?

Ja, alles wat ik nodig heb, technische tools, maar ook fatsoenlijke VPN, skype, al de zaken om op afstand te kunnen werken, of van thuis, niet ste kort.

Question: When you think about the project scope, was this clear upfront?

Scope was duidelijk, document met alle use case duidelijk in beschreven, kleine applicatie, heel bevattelijk, ik wist waar ik aan begon is niet sterk gewijzigd, paar dingen bijgekomen, maar dat gaat zo.

Question: Were there frequent changes during the project execution, and how were they handled?

Ja, er zijn altijd wijzigingen, maar geen gigantische, verschillende bronnen (testers, business, pilootklanten), eerst bekeken door analist, als die het waardig vindt wordt het met het devteam een schatting gemaakt qua effort en als het dan waard was kwam het op de backlog.

Question: Was there a specific process that was followed?

Ik heb de indruk dat we voor OI2, alleszinds een heel ander process dan de rest. Ik weet niet of we dit bij andere teams wordt toegepast, binnen legacy is dat anders.

Question: How many people did your team consist of, including product owners, customers, …

Piet, Kathleen, Tom P, Joris, Steven John, Bart, David, Michaël & ikzelf. Lars hoort daar ook wel bij, weinig direct contact daarmee. Mark & Johan waarschijnlijk wel, maar op een heel ander niveau, is minder het uitvoerende team, eerder het business ownend team, in dagdagelijkse omgang niet tot het team, die ga ik niet bellen om iets aan te vragen, voor Tom Princen ligt dat anders, maar voor mij de afstand te groot. Ramon reken ik ook tot het team.

Question: Did you collaborate with other teams for this project?

Zelden, heel af en toe eens iets afgestemd met UBL, en uiteraard GSS (operations).

Question: Was everyone working in the same place? Were there face-2-face meetings or was this done otherwise?

Nee, maar het devteam zit wel bijna altijd samen, wordt wel eens van thuis gewerkt, maar in de regel zitten we samen. Precies wel belangrijk, daarom niet elke dag, maar ook wel vaak samenzitten, heeft bijgedragen tot de drive die er was en de snappyness, en het onmiddelijk kunnen sparren met iemand of een vraag stellen en de sfeer voelen. Ik denk dat het in het begin met Piet wel stroef gelopen omdat

hij op een andere locatie was, ik weet nog dat ik toen dacht 'als piet bij ons had gezeten, dan had het beter gelopen', er was wat frustratie door het feit dat hij tussen twee vuren zat, maar uiteindelijk wel opgelost.

Question: In your opinion, did you find your coworkers to be working in a silo, or as a one group?

Binnen het team, logisch dat er silo's zijn, dev en analisten is verschillend, maar voor de rest behoort iedereen even hard tot het team als de velopers. Binnen de organisatie zijn er wel wat meer isolementen en zijn er nog wel eilanden, en weinig contact met andere teams, wij eigenlijk ook niet.

Question: Describe the company culture, how are the relations with the coworkers, the management … Is there a lot of team spirit, a top-down culture, bottom-up culture, is there a specific hierarchy?

Met mijn team zeer goed, echt plezant, één van de tofste projecten die ik al heb gedaan door het feit dat we zo goed overeenkomen. Er wordt gepraat en gelachen, maar binnen andere teams niet, in het vorige gebouw kregen we mails als je te hard praatte... De eerste keer in het oude gebouw schrok ik dat er zoveel volk in een grote zaal zat en je kan een speld horen vallen, maar binnen het team geen enkel probleem in tegendeel.

Ook met management is er sterke verbetering, vroeger zaten ze apart, in hun bureau, maar nu zitten we samen aan eilanden, de barrière is zeer sterk weggevallen, er wordt geïnvesteerd in team events waardoor je naar elkaar groeit, Christine heb ik wel nog maar één keer gezien, dus eerder een gevoel van meer afstand.

De cultuur is top-down, zeker. Ik hoor niet iemand de telefoon nemen en zeggen 'hier ben ik niet mee akkoord', andere teams zijn sterk uitvoerend en zwijgen, en vooral niet protesteren. Binnen eigen team is dat anders, daar, misschien vooral ik, de de ideeën gaan top-down, bottom-up, links, rechts, alles komt van overal en alles komt feedback terug, maakt niet uit welk profiel dat je hebt, dat gevoel heb ik ook meer met Mark & Johan (in het begin eerder top-down, nu wordt er wel sterk geluisterd).

Question: Was the project done ad-hoc, or did it reuse architecture from other projects, programs, … Was this integration easy?

Het is een pioneersproject, het is nieuwe technologie en architectuur, maar er is wel veel onderzoek gebeurd wat de technische community nodig heft en er is veel geïnvesteerd om dit te kunnen herbruiken naar andere projecten, ad-hoc zou ik het niet noemen, maar we zijn wel de eerste die het gebruiken, ook niet laten leiden door andere teams.

Question: Talking about the project itself, did you find it was a rather easy project to build, or were the features of the project more complex?

Makkelijk.

Question: Was there a need for being compliant with a regulatory instance? How was this handled?

Ja, dat wel, ik denk dat Piet in samenspraak met Johan en Mark wel konden zeggen, als er twijfels waren is er aan legal hulp gevraagd, maar de product owners kennen de wetgeving er rond.

Question: What was the technical complexity of the project, which technologies were used, and how are they evaluated?

Langs serverzijde C-sharp entity framework, restful, web api, langs frontend angular-javascript-typescript, was zalig om meet e werken, entity framework durft af en toe wat tegenspruttelen, maar is nieuwe technologie die extreem productiviteitswinst in vergelijking met oudere webtechnologieën en ZEKER in vergelijking met waar we van komen.

C.4 Michaël Mertens

Participant: Michaël Mertens

Date: 23/07/2014 Role: Developer Projects Involved: OIv2

The goal of this survey is to evaluate how the 'Online Invoicing v2.0' project was handled in terms of software delivery process. These questions below have the goal of identifying success factors of the project, and potential improvements in the SDLC. As this is an in-depth survey, a basic set of open questions is documented that can be used as a starting point of the discussion.

Question: Can you tell about the context of the project, why was it founded, what are the drivers

Becoming an online company, is de slagterm van Lars, achter het project. Heel wat legacy software bij Kluwer, om meet e spelen met 'the next big thing' is er heel wat kennis nodig binnen Kluwer, online invoicing is een pilot voor toekomstige projecten, welke kennis hebben we en wat zijn onze ervaringen in becoming an online company en uitdagingen identificeren die op de weg liggen. Klein begonnen, facturatietool is vrij beperkt in scope, controleerbaar, zelfs voor zo'n project komen er toch veel dingen bij de online wereld, als je kijkt naar User Accounting. Hoofdreden is kennisopbouw. Ook echt voor in de markt te gaan zetten, Kluwer beter te positioneren naar de markt toe: signaal van de online company (SaaS-oplossing aanbieden), 'mee zijn'. Service aan lage kost voor kleine zelfstandigen.

Question: What was your role in the project?

Developer, since sprint 4, ook technische architectuur. Niet het ervaren van andere technische architecten binnen het team, maar ook in een ondersteunende rol als developer. Opnemen van nieuwe development taken, meewerken aan nieuwe features, maar ook mee nadenken over technische uitdagingen van nieuwe features, als externe consultant. Unit testen schrijven, performance & system testing. Niet echt neergeschreven, maar wordt verwacht als onderdeel van het development, ook retrospectives en demo's wordt er nadruk gelegd op de belangrijkheid van unit testen en het onder controle houden van de testen.

Question: In your experience, which topics did you evaluate as positive during the delivery of the project?

In een team terechtgekomen dat héél gevarieerd is, zeer ervaren is, groot voordeel om constant bij te leren en te sparren met andere mensen. De nieuwe technologieën die gebruikt worden en de uitdagingen die er bijkomen maar ook voordelen (vb. Met nieuwe dingen bezig zijn), meer naar de uitkijk zijn naar wat er in de IT-community gebeurt, we zijn met cutting edge technologie bezig, zeker qua frontend. Andere zaken zijn functionele analyse en project management. Als ik vergelijk met andere agile projecten waar ik in gewerkt heb, worden de regels veel beter gehandhaafd, inplanning van resources, scrum meetings zijn correct, de communicatie met de analisten werkt héél vlot, ondanks het feit dat Piet in gent zit, maar dit is geen communication block. Skype en andere tools helpen daarbij, ook in het begin veel op dezelfde plaats gezeten, de face-2-face meetings hebben zeker een belangrijke meerwaarde. Skype is een toffe tool, maar er zijn nog altijd zaken die niet mee worden overgebracht, dat zorgt al

eens voor miscommunicaties of soms wat wrevel. F2F uitleg van een functionele analist heeft zeker zijn meerwaarde, ook minder barrière om een vraag te stellen, terwijl op skype kan het zijn dat je juist pauze neemt tussen zinnen, en ook het feit dat je oogcontact kan maken is heel belangrijk. Het team is ook heel gemotiveerd, de nieuwe technologie heeft daar zeker iets mee te maken. Af en toe is er wel vanuit business wel een verplichting of beperking, maar dit valt héél goed mee.

Question: In your experience, what where the struggle points that the team tried to solve during the project? Which of these topics were solved and how, and which topic remained a problem by the delivery of the project?

Eén van de mooie dingen die we besproken op het team event, is dat we nog meer moeten inzetten op co-creation met business en ict, hoewel dit al véél beter is dan eerdere projecten die ik ooit heb gedaan.

Binnen kluwer in het algemeen is er wel nog het probleem van de verspreide kennis, voor wetgeving is het al eens moeilijk om te weten waar je moet zijn binnen de organisatie, ook naar legacy toe.

Het feit dat we Kathleen niet vaak zien, met Kathleen is het ook alleen via skype, aan de hand van screenshots (UX), dat zou beter kunnen als je dit in co-creation kan doen, of via F2F-meetings. Ze is zeker expert op haar domein, het is spijtig dat we haar niet meer op de vloer hebben om samen te zitten.

Question: Who was your customer, and how was he involved in the process?

Een loodgieter of een stukadoor hou ik als profiel voor mij, uit de persona's gekomen van Kathleen, wel voldoende beeld om in je achterhoofd te houden om geode beslissingen te nemen in functie daarvan. Ik denk wel voor nieuwe projecten dat dit moelijker gaat zijn, omdat dit specifiekere applicaties gaan worden voor een publiek waar niet mee vertrouwd ben. Johan (product owner), gaf ook aan dat als een developer nu eens een halve dag mee op de baan zou gaan, hoe klanten juist werken zou dit veel helpen. In functie van het project.

Question: How was the team organized? Can you discuss how a typical development iteration took place?

- Begint met sprintplanning, waarin de PM (David), planning van resources in gang steken adhv Piet die de scope bepaalt (Product Owner), samen met de functionele analist en de lead developer (Steven). Developers zitten er bij om input te geven, zo komen we tot een sprintplanning, op basis van prioriteiten (wat in changelog, backlog)
- Eens de sprint van start, twee weken in dagelijke scrum meetings worden de user stories in taken opgedeeld en opgenomen, in samenspraak onder de developers, natuurlijk heb je van die taakjes die niemand wil doen, maar niet het gevoel dat het top-down is
- Burn down chart wordt in het oog gehouden en rapportering van David tijdens elke scrum, hoewel dit niet altijd was, maar was niet echt gemist, developers hadden niet altijd de gewoonte om hun effort in te voeren in de software, een vorm van 'luiheid'
- Sprint eindigde met sprint demo die met alle stakeholders waar de nieuwe business value gecreëerd werd wordt gedemonstreerd. Ik had echt wel het gevoel dat tijdens de demo's ook heel zichtbaar was dat we werkende software opleverden. Soms niet helemaal zichtbaar, maar dan gaat het eerder over technische verbeteringen die evenzeer belangrijk zijn, zoals performance en herbruikbare architectuur.

- Retrospective waar werkpunten en positieve punten worden bekeken en er worden actiepunten geëxtraheerd.

Question: Did you find that you had the right tools to use for your role?

Ja, team is ervaren genoeg waardoor de tools uit zichzelf werden benut.

Question: When you think about the project scope, was this clear upfront?

Ja, ik ben aangekomen in sprint 4, toen document gekregen waar alle requirements duidelijk in opgelijst stonden, natuurlijk veranderen de verwachtingen vooral naar scalability reken je op voortschrijdend inzicht, maar dat werd gaandeweg uitdagender, andere zaken zeker binnen de verwachting. Vooral de non-functionals waren in het begin nog niet volledig duidelijk.

Question: Were there frequent changes during the project execution, and how were they handled?

Aantal infrastructuurkeuzes die achteraf gewijzigd zijn, maar qua impact redelijk gecontroleerd, zoals message queing infrastructuur is gewijzigd, maar weinig impact.

Question: Was there a specific process that was followed?

Van wat ik hoor is in de presentaties van Lars is deze sterk verfijnd voor OI, becoming. De wiki wordt eerder gebruikt als file share, is niet de eerste plaats die ik zoek om te antwoorden, eerder vragen binnen het team. Voorstel van de nieuwe SDLC voor OI als test, waarneem dat het positief ervaren wordt.

Question: How many people did your team consist of, including product owners, customers, …

Kern: Mezelf, Sophie, Steven, Joris, Bart, John, en Daarnaast: David, Tom Princen, Piet, Tom D.

Ik reken ze maar in mijn team als ze meer dan de helft van hun tijd er bij zijn, voor Johan is dat moelijker in te schatten, en Mark reken ik niet meteen tot mijn team, beschouw ik eerder als supervisor.

(Note: Ramon dus duidelijk niet)

Question: Did you collaborate with other teams for this project?

Ik zou zeggen van niet, ‘eens iets vragen’, of ‘supervisen’, is dat samenwerken? Samenwerken moet van twee kanten komen.

Question: Was everyone working in the same place? Were there face-2-face meetings or was this done otherwise?

Question: In your opinion, did you find your coworkers to be working in a silo, or as a one group?

Binnen het team als één groep gefunctioneerd, er zijn een aantal dingen waarbij je de mensen zou kunnen onderscheiden (frontend-backend, analyse-developer, member-management), maar ik had niet het gevoel dat er silo’s waren.

Question: Describe the company culture, how are the relations with the coworkers, the management … Is there a lot of team spirit, a top-down culture, bottom-up culture, is there a specific hierarchy?

Binnen het team zelf is de sfeer héél familiair, maar heel positief, vertrouwen in elkaar is heel belangrijk. Tussen dat team en andere teams is het nogal afstandelijk, daar zijn wel silo's, we weten niet wat andere teams aan het doen zijn en we weten het niet van elkaar, we hebben wel brown bags gedaan en informatiesessies, dus daar is verbetering bezig.

Management: kan het niet zo sterk vergelijken, beschouw als individuele mensen, er is niet 'business' als blok, maar eerder individu's (Mark/Christine, met de laatste minder contact), daarnaast heb je Lars, die staan wel naast elkaar, maar ze hebben wel een serieuze invloed in wat we doen, dus het is wel top-down, maar we hebben nog niet veel weerstand gehad.

Question: Was the project done ad-hoc, or did it reuse architecture from other projects, programs, … Was this integration easy?

Zeker omdat OI een soort van pilot is voor andere projecten voor de toekomst (web foundation is specifiek gericht op reuse), voldoende rekening nagedacht over herbruikbare architectuur. We hebben exportformaat voor representatie van facturen, daar hebben we de legacy niet gevolgd, maar voor de rest niet meteen samenwerking.

Question: Talking about the project itself, did you find it was a rather easy project to build, or were the features of the project more complex?

Scopegewijs vond ik OI redelijk gemakkelijk en beperkt, het is bijna niet veranderd gedurende loop van het project, als je web foundation er bijhaalt complexer. Bij web foundation zijn er héél wat uitdagingen met degelijke kwalitatieve oplossing op zoek, moesten we niet zoveel aandacht hebben besteet aan de kwaliteit, dan had het sneller in de markt gestaan, maar dan was het niet zo lang bruikbaar geweest, dat is zeer goed aan het team, hoe complex het functioneel ook is, het schaalbaar maken is zeer belangrijk.

Question: Was there a need for being compliant with a regulatory instance? How was this handled?

Er zijn een paar regels zijn die er inzitten in het product, je moest een correcte factuur kunnen maken voor de Belgische wetgeving. Er zijn enkele verplichte mededelingen, dus ja, er waren een aantal aspecten. Het gebruik van componenten, als we externe componenten zoals een pdf-writer, dat die componenten ook wel effectief mochten gebruikt worden. Voor de rest niet echt.

Question: What was the technical complexity of the project, which technologies were used, and how are they evaluated?

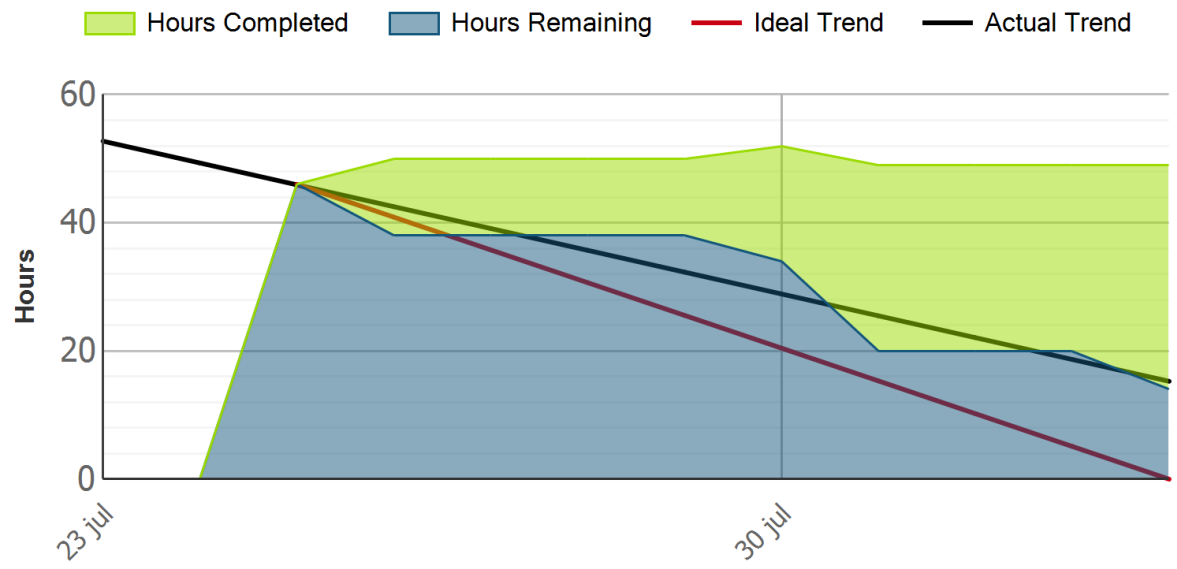
Backend: SQL Server DB, is gekend, was goede keuze in .net stack, MVC-project als representatielaag, WEB-API project om web applicatie aan te bieden, ook al ervaring mee, veel voorkomend patroon,

restful apps te maken. Ook nog wat C-Sharp windows services die background work doen, gebruik gemaakt van RapidMQ voor communiceren van jobs van web api naar background workers.

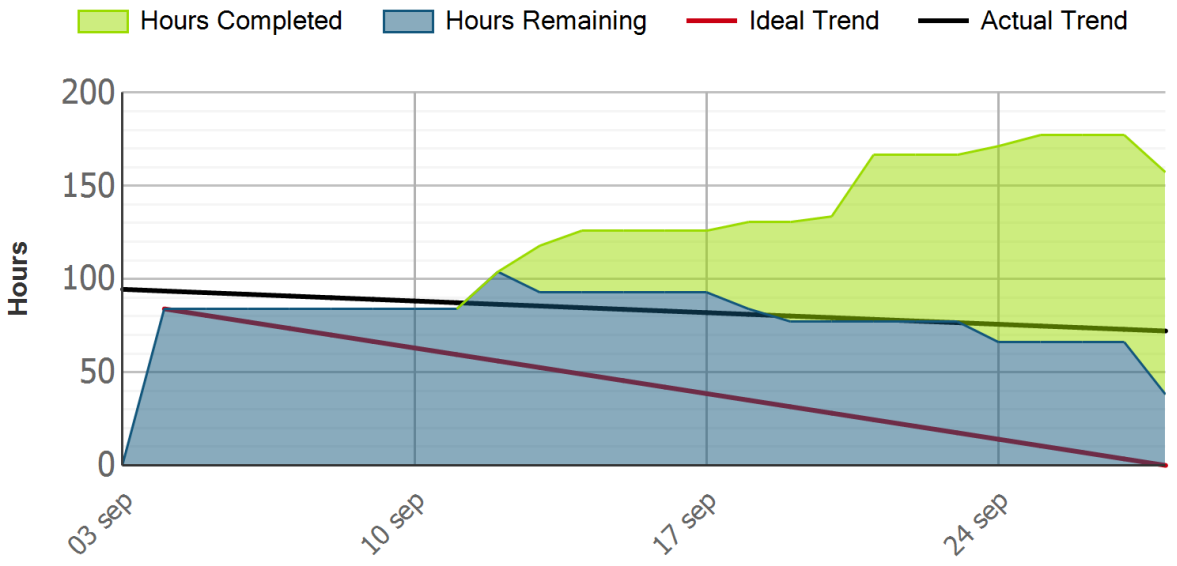
Frontend: AngularJS om applicatie rich te maken in de client, nieuwere technologie, zeer fijn om mee te werken als frontend developer, TypeScript voor het typen van de JavaScript, wat het javascriptverhaal makkelijker maakt omdat er compilatiefouten kunnen opgemerkt worden. Voor unit testen op de frontend gebruiken we Jasmin, en alles wordt frontend gebuild in een Grunt project.

Appendix D: Burn Down Charts OI v1 & v2

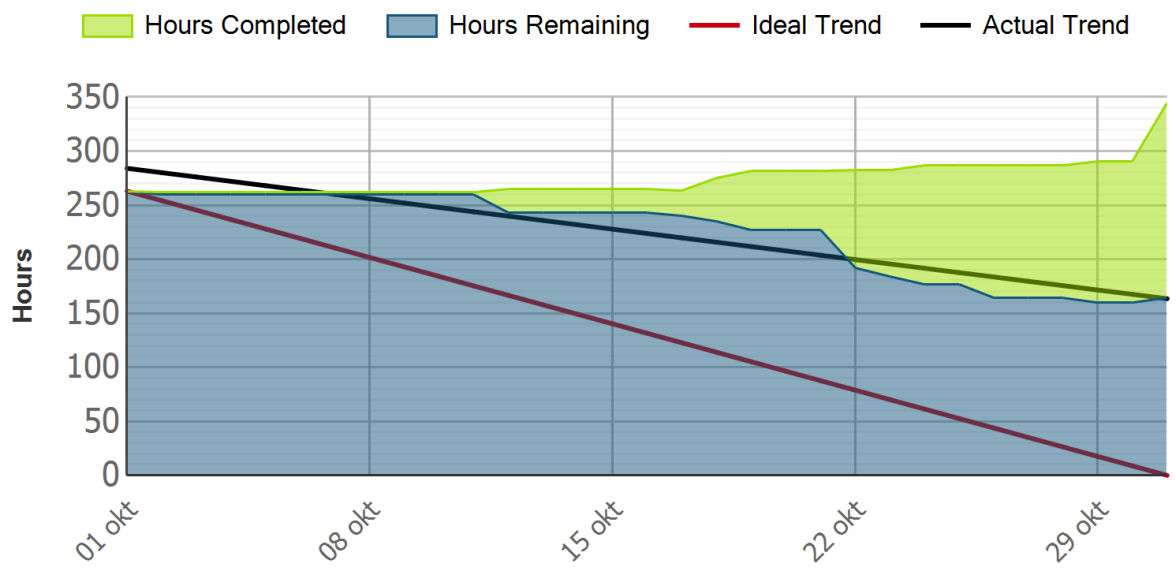
D.1 Burn Down Charts Online Invoicing version 1



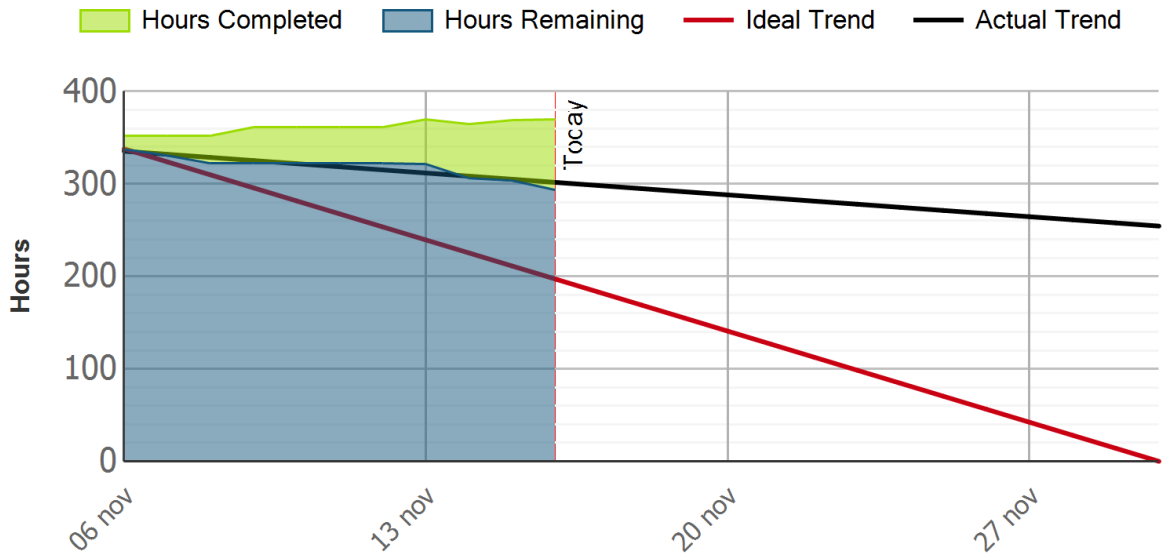
Burn Down Chart - Sprint 0 – August 2012



Burn Down Chart - Sprint 1 - September 2012

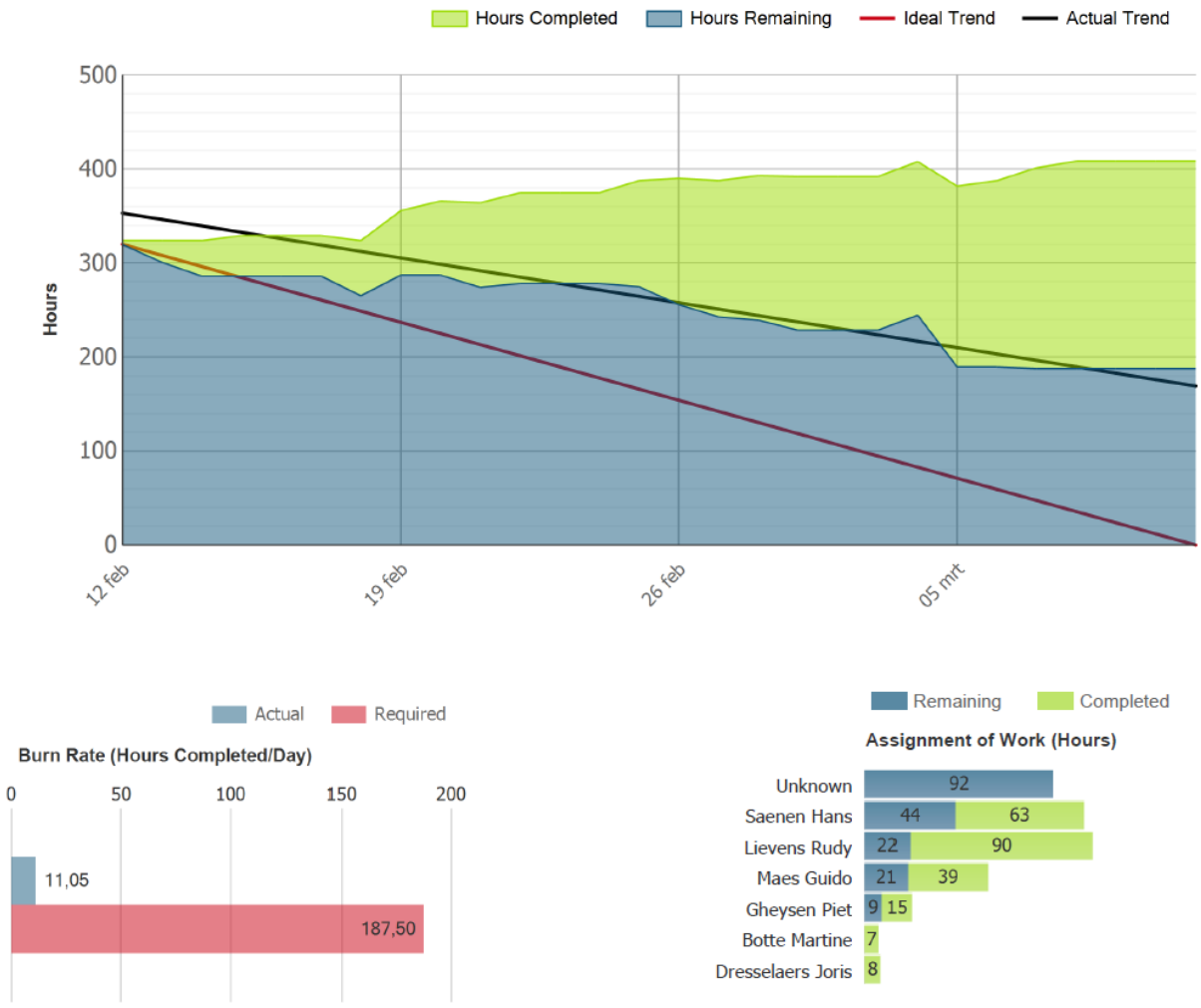


Burn Down Chart - Sprint 2 - Octobre 2012

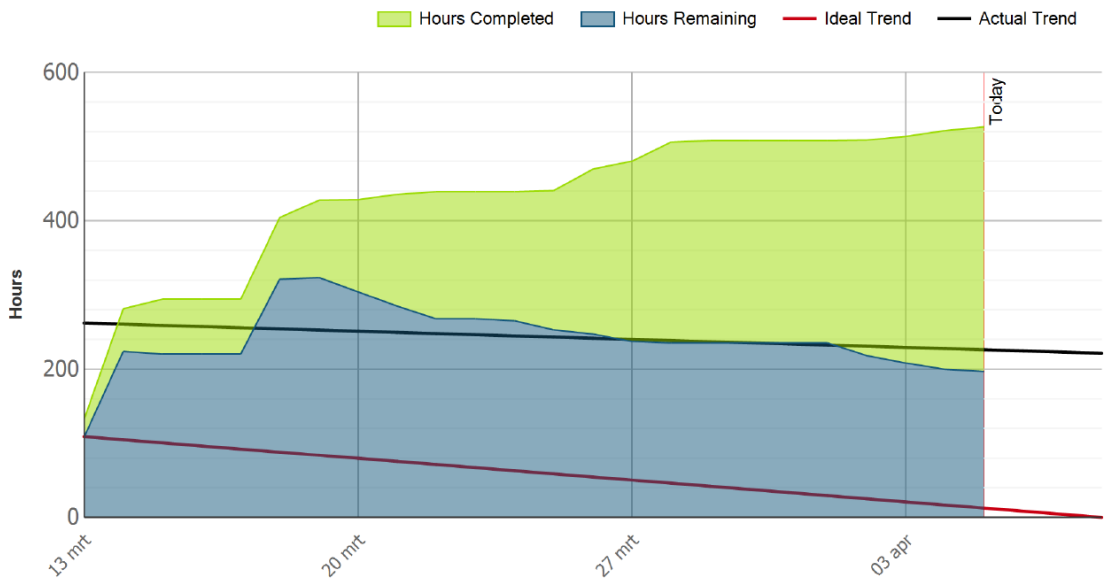


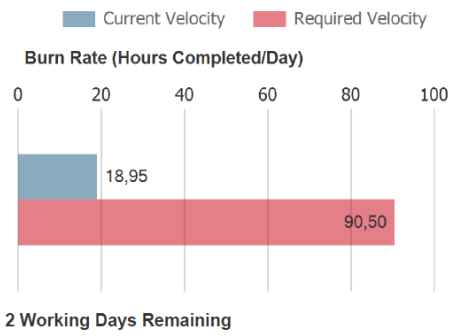
Burn Down Chart - Mid-Sprint 3 - November 2012

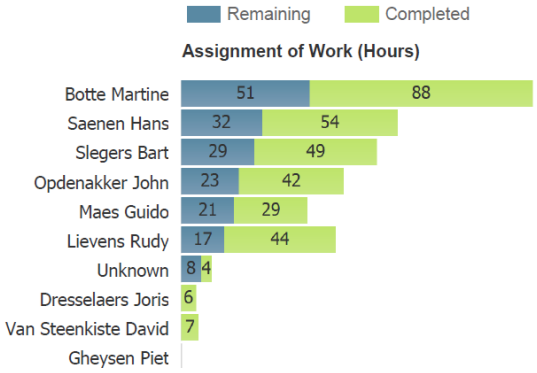
For Sprint 3 there was only a partial burn down chart available until the 15th of November 2012. The Burn Down Charts for Sprint 4 & 5 aren't found in the archives, data missing from mid-November 2012 until end of January 2013.



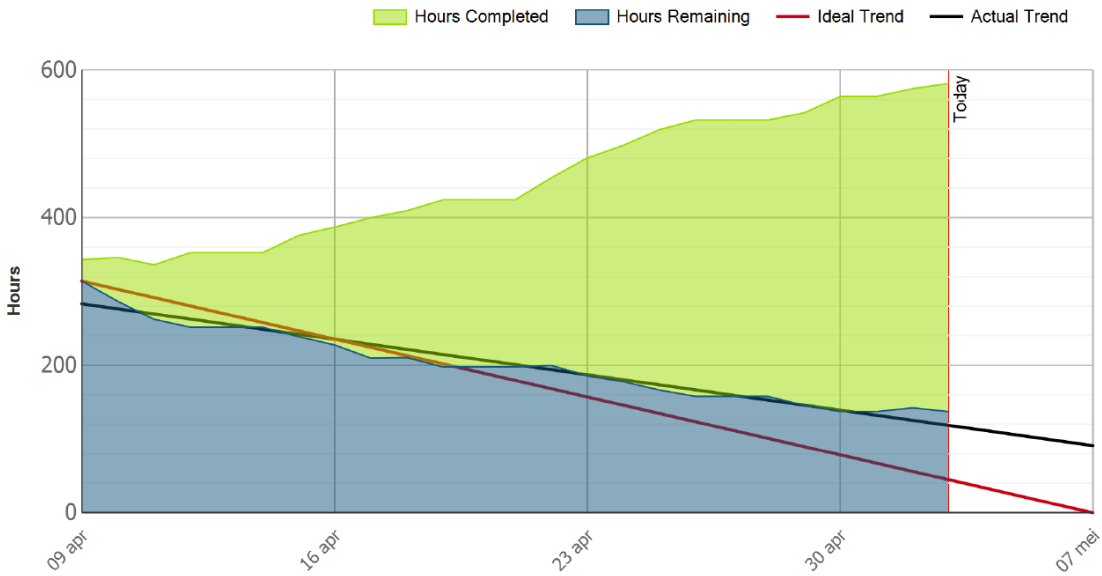
Burn Down Chart - Sprint 6 - February 2013

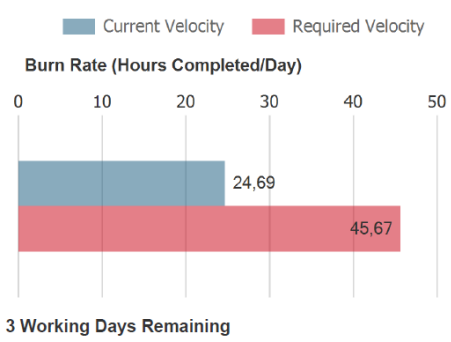


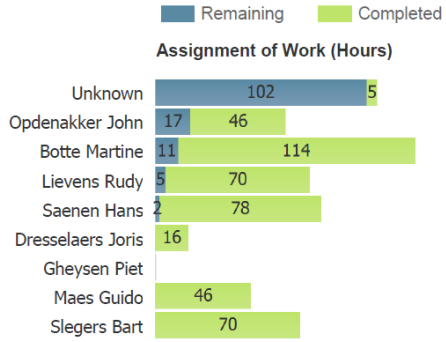




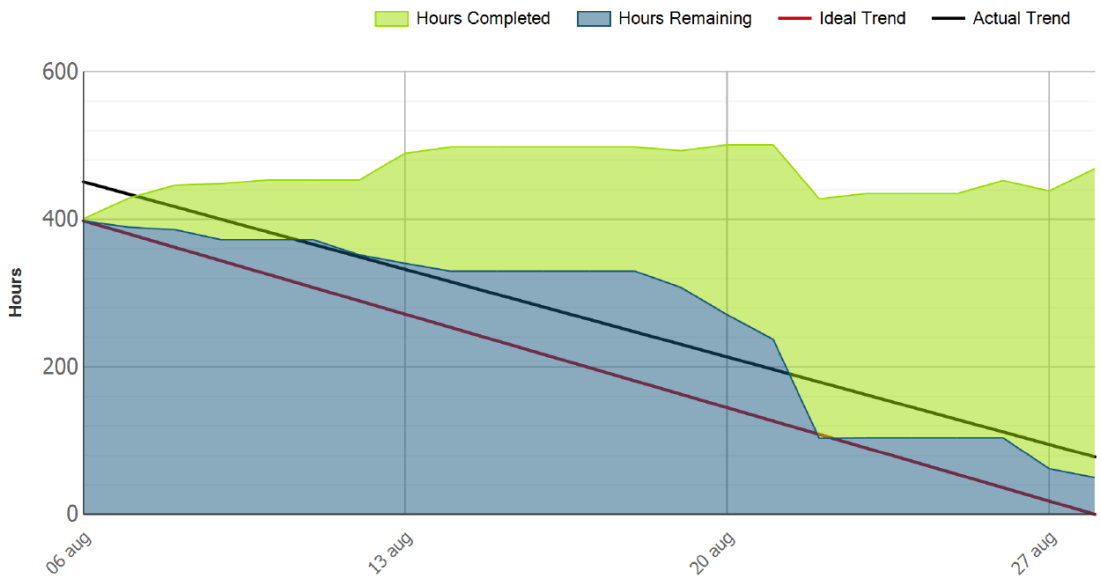
Burn Down Chart - Sprint 7 - March 2013

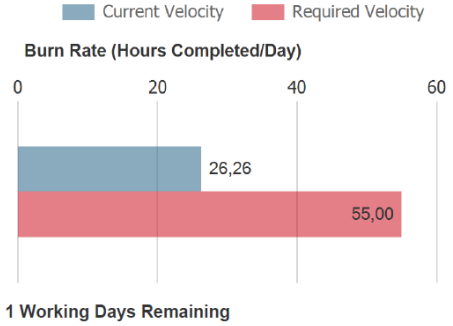


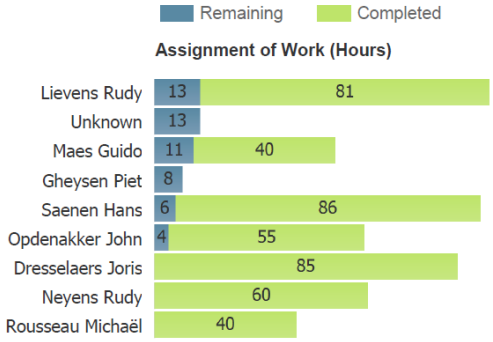




Burn Down Chart - Sprint 8 - April 2013

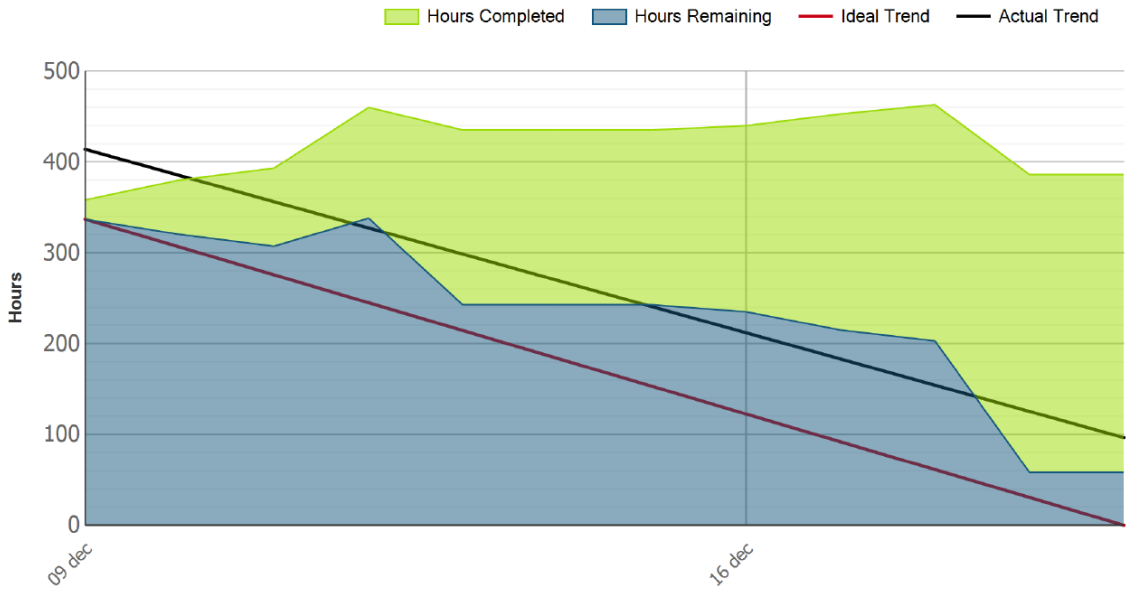


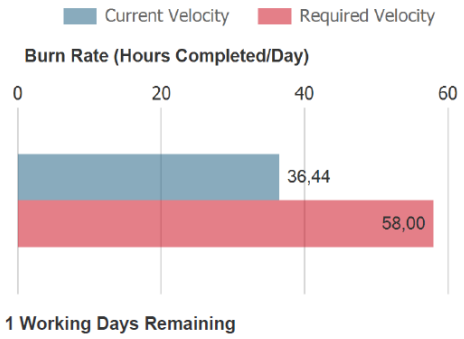


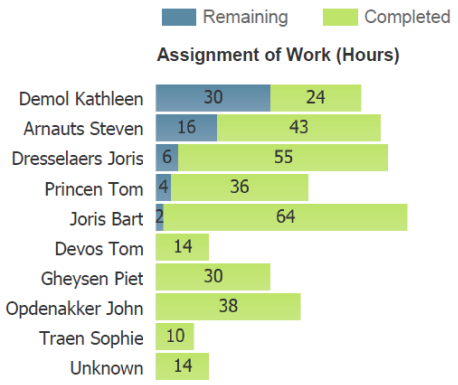


Burn Down Chart - Sprint 12 - August 2013

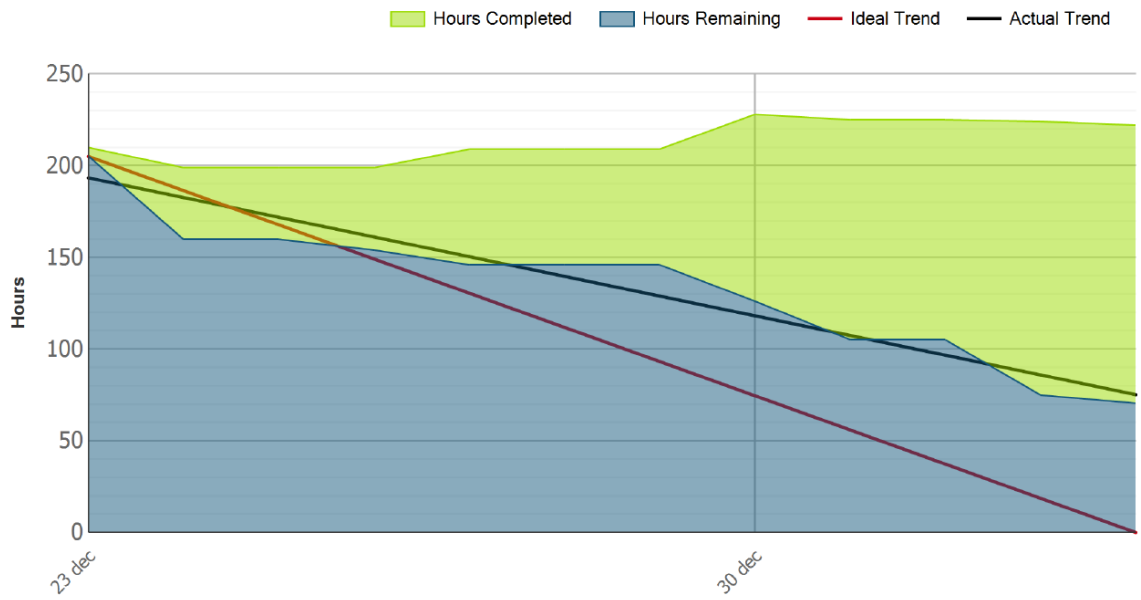
D.2 Burn Down Charts Online Invoicing version 2

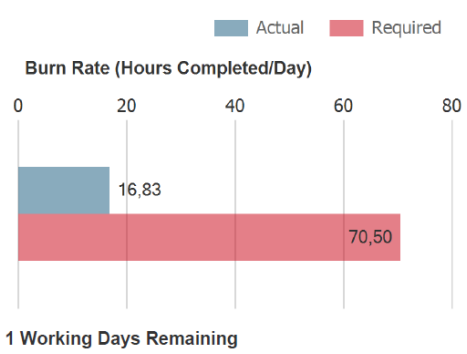


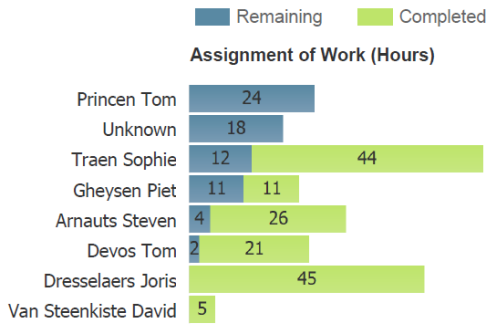




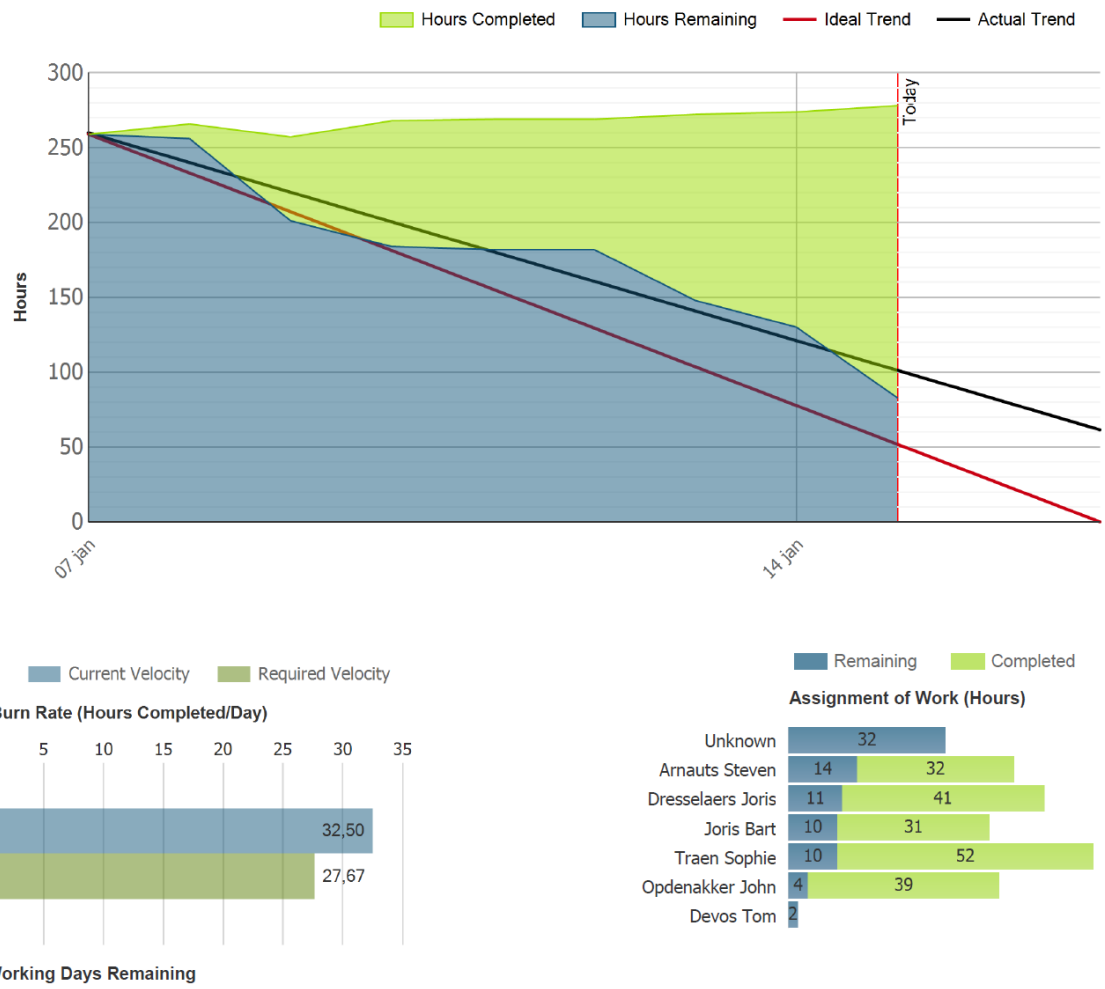
Burn Down Chart - Sprint 1 - December 9 - 20, 2013



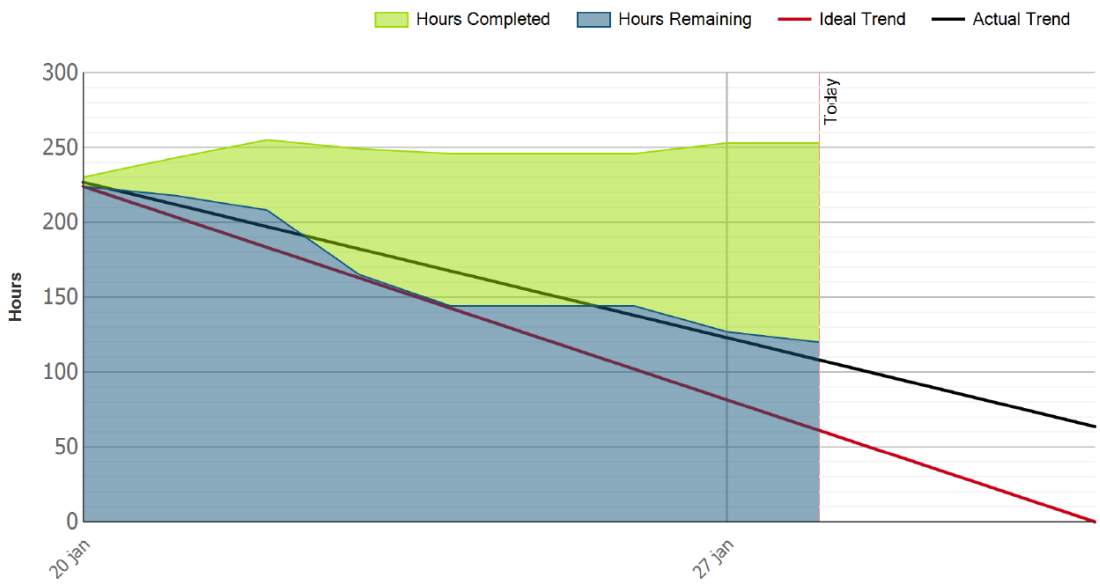


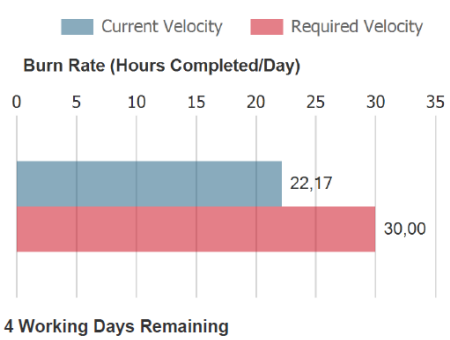


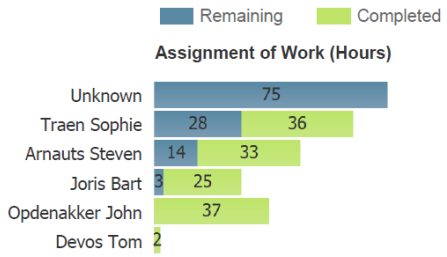
Burn Down Chart - Sprint 2 - December 23, 2013 - January 3, 2014



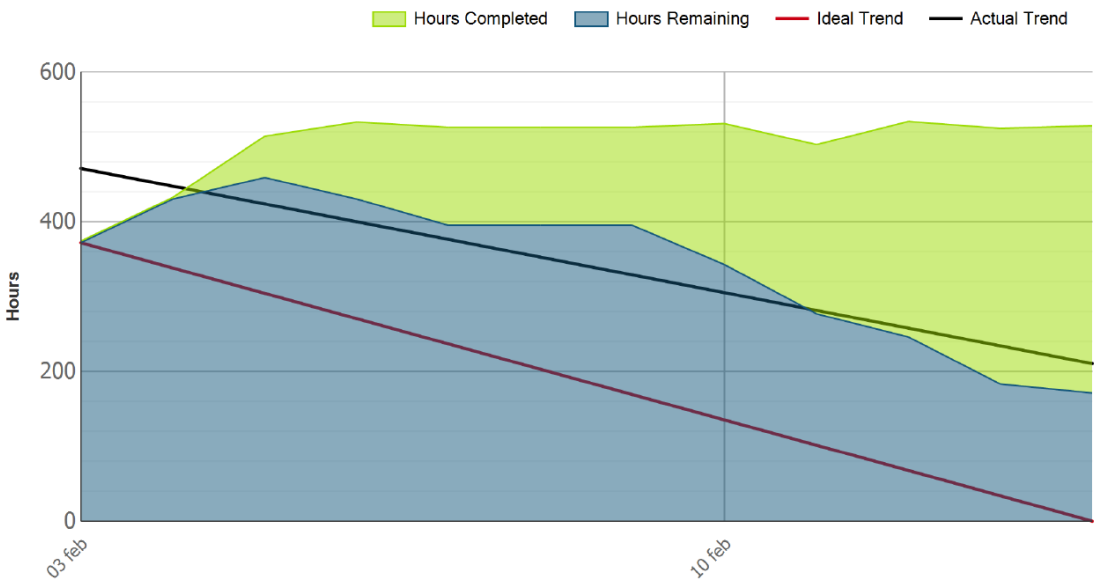
Burn Down Chart - Sprint 3 - January 07-17, 2014

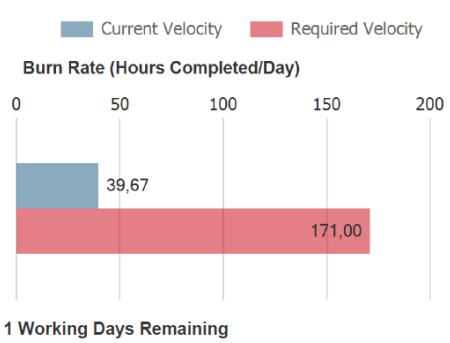


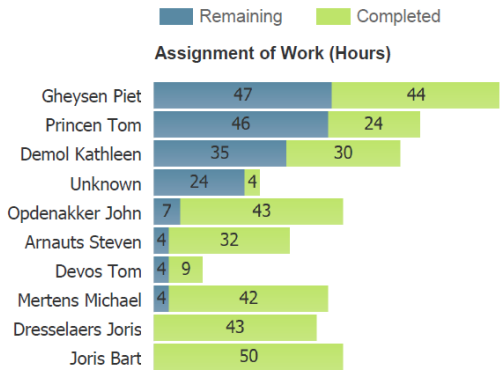




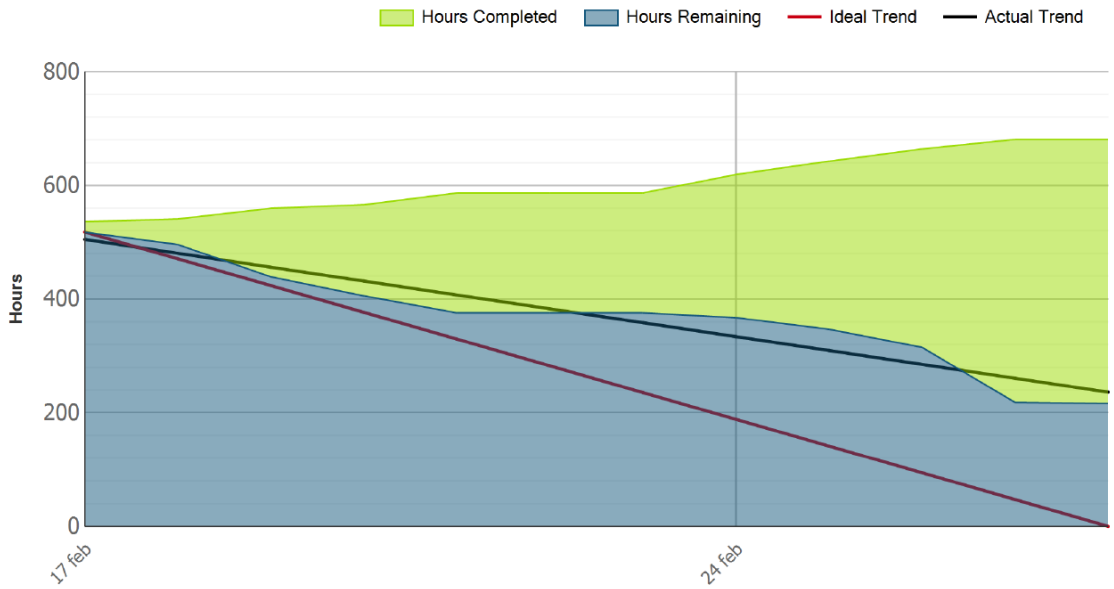
Burn Down Chart - Sprint 4 - January 20-31, 2014

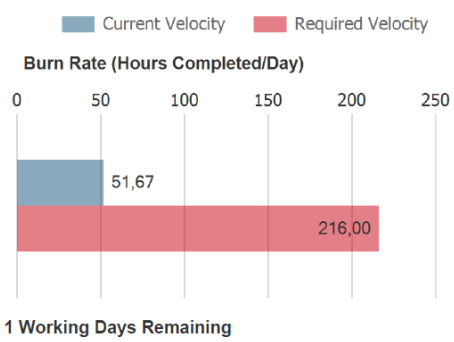


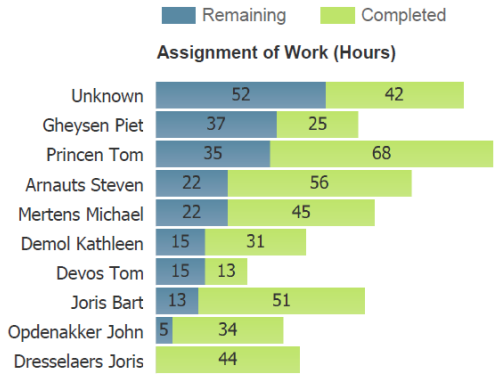




Burn Down Chart - Sprint 5 - February 3-14, 2014

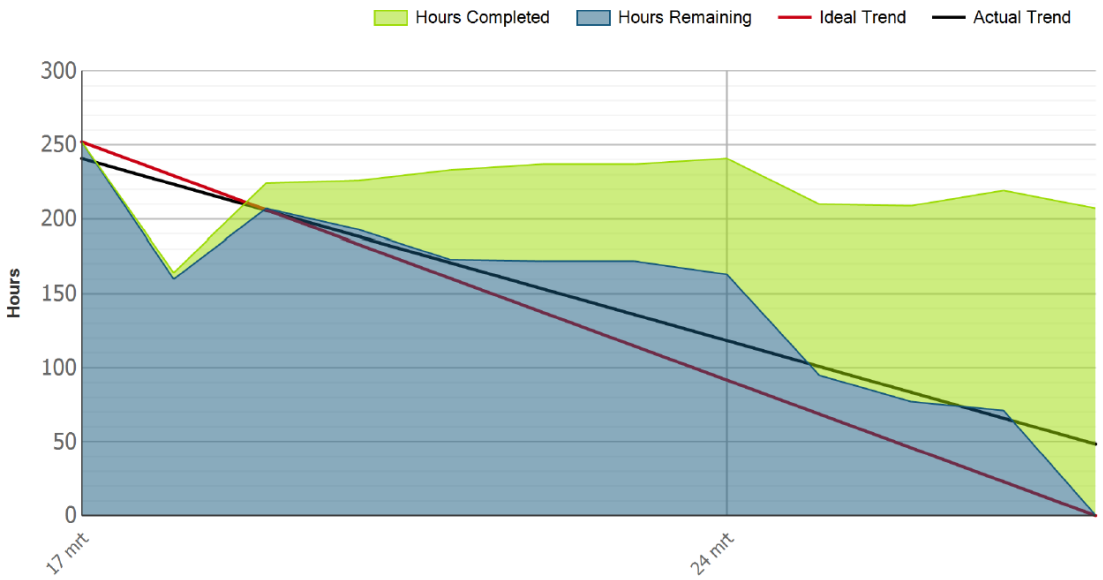


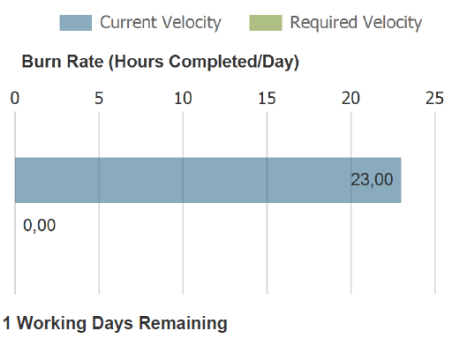


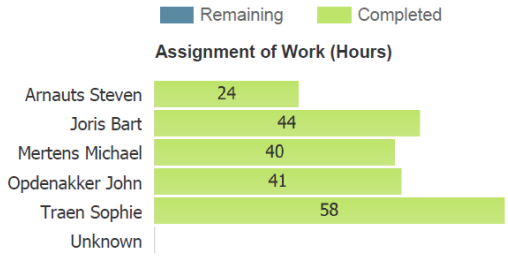


Burn Down Chart - Sprint 6 - February 17-24, 2014

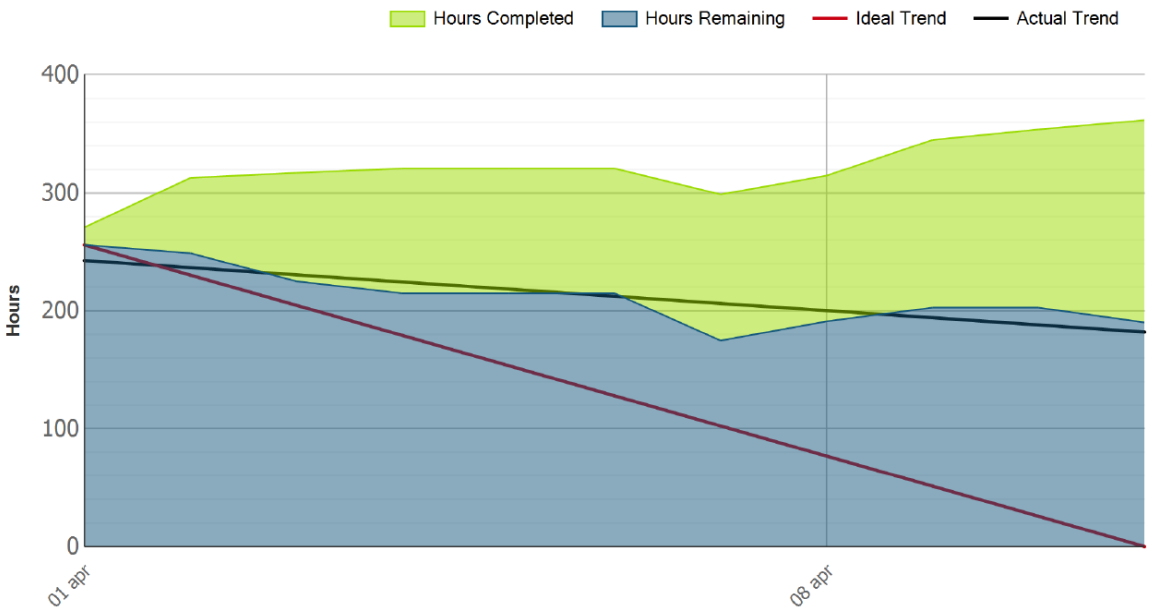
Detailed information regarding Sprint 7 is not available in the archives.

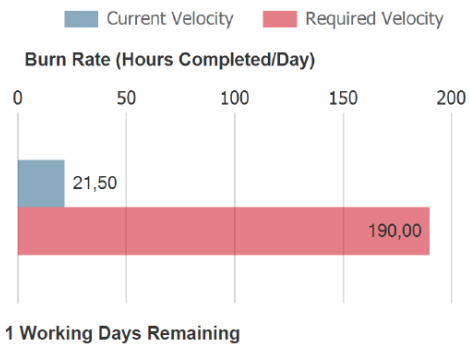


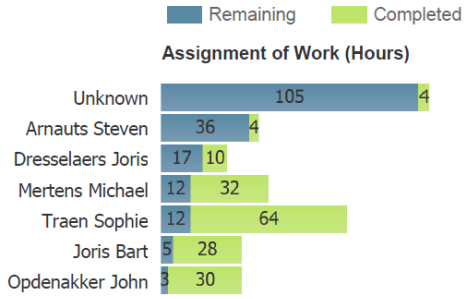




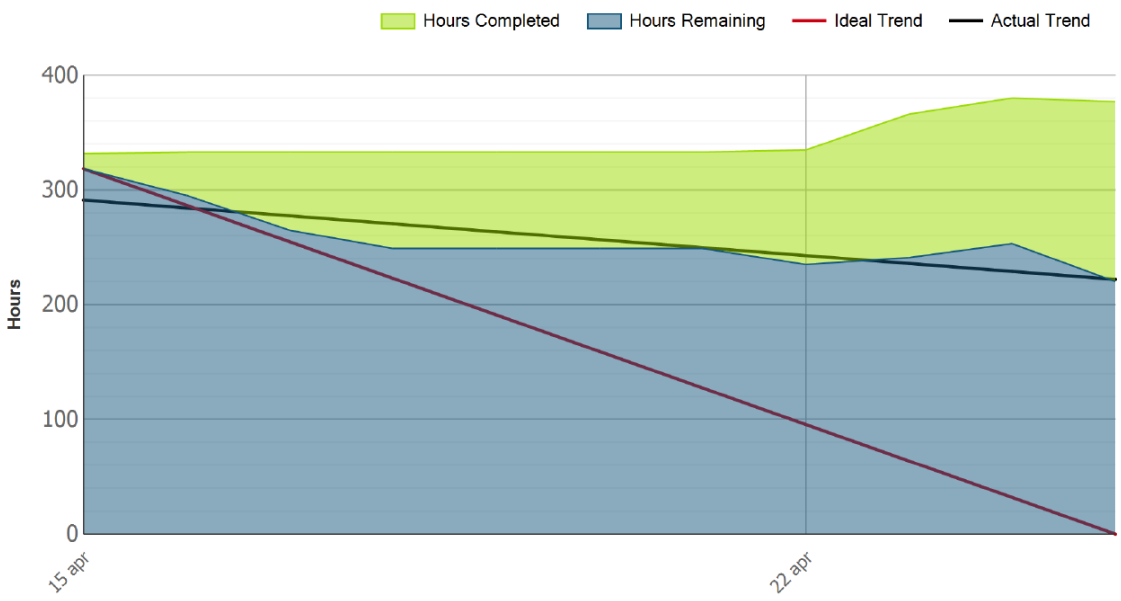
Burn Down Chart - Sprint 8 - March 17-28, 2014

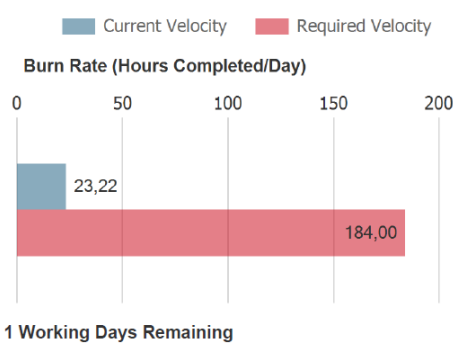


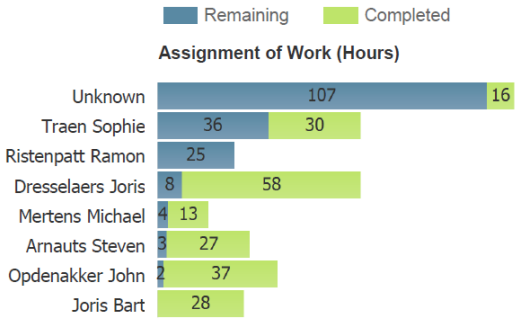




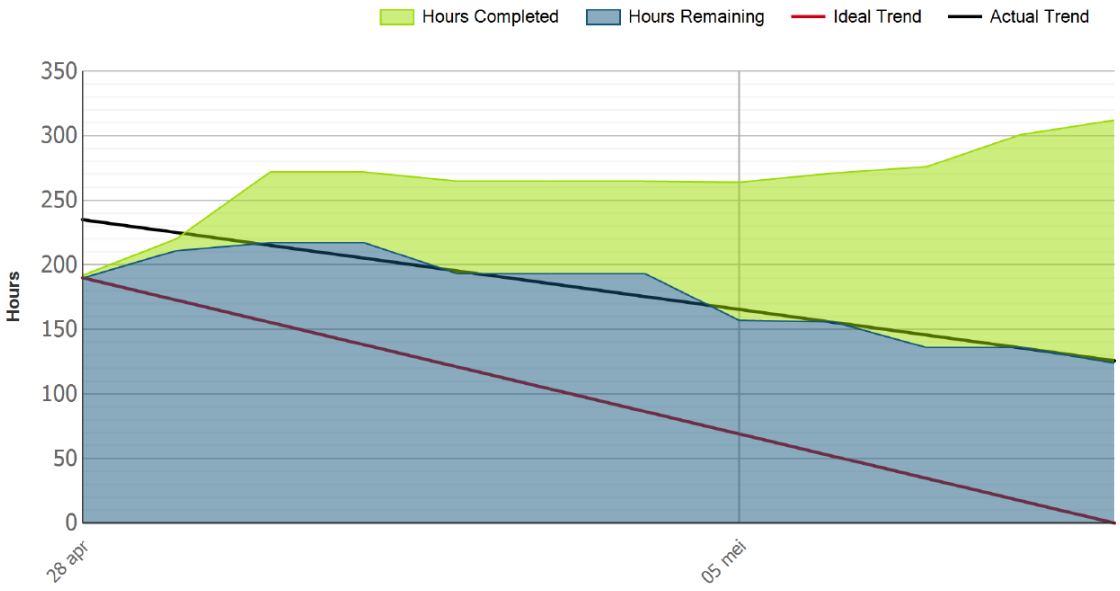
Burn Down Chart - Sprint 9 - April 1-11, 2014

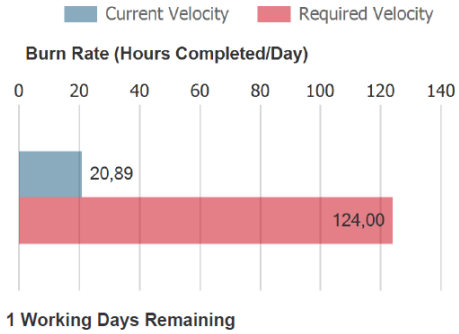


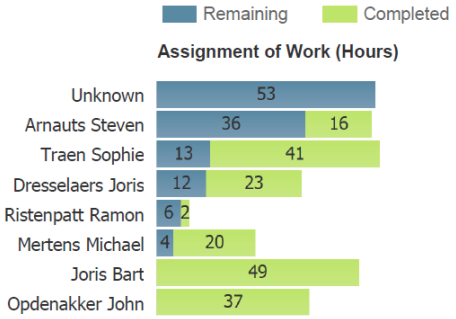




Burn Down Chart - Sprint 10 - April 15-25, 2014







Burn Down Chart - Sprint 11 - April 28, 2014 - May 9, 2015

Appendix E: Sprint Retrospectives OI v1 & v2

E.1 Sprint Retrospectives Online Invoicing version 1

	What went well?	Sprint
1	Beschikbaarheid Piet Gheysen	0
2	Overleg & medewerking binnen Core Team	0
3	Demo-omgeving beschikbaar	0
4	Resultaat	0
5	Interactie communicatie	0
6	User Story duidelijk gedefinieerd vanuit analyse	0
7	Opzet project (solution opzet door architecten)	0
8	Communicatie	0
9	Analyses voor 80% ok (20% -> navigatiediagram onduidelijk)	0
10	extra mensen in het team	1
11	nalezen & goedkeuren van docs reviewen beter & beter	1
12	Communicatie	1
13	snelle beslissingen genomen	1
14	open & eerlijke communicatie	1
15	communicatie dev team	1
16	communicatie (skype, e-mail)	1
17	aanwezigheid management & interesse	1
18	focus op kwaliteit	1
19	oplevering & demo's	1
20	TFS overzicht van use cases etc.	1
21	Communicatie	1
22	teamgeest zit goed	1
23	collega's (makkelijke onboarding)	1
24	team werkt goed	1
25	teamgeest zit goed	1
26	Betrokkenheid	1
27	UX meeting (met architecten & analisten)	1
28	spirit (iedereen wil vooruit)	1
29	Overgeschakeld naar telerik controls (wel nog leercurve) - goeie help telerik	5
30	controls -> telerik -> minder problemen daardoor voor architect	5
31	Logo is geïmplementeerd -> doel gesteld en halen ervan	5
32	One suite -> eindelijk (zeker naar DB)	5
33	Cross-domain groepswerk (analisten) -> moet nog beter	5
34	Demo wordt op amazon gedemonsteerd	5
35	Telerik Controls	5
36	Minder overhead gehad	5
37	Voorwaartse leercurve (CSP, entity framework) -> volgende sprints meer snelheid	5
38	Demo op het amazon platform	5
39	Vooruitgang gaat wel goed, maar perceptie daarvan moet beter	5

40	1 WKB suite	5
41	Telerik controls	5
42	Kort op de bal spelen tussen analist & developer (piet & hans & tom)	6
43	Meeste ontwikkelaars ontwikkelen momenteel stuk sneller (kennis entity framework) - sneller	6
44	Martine is erbij gekomen -> extra schouders onder het werk	6
45	Samenwerking met Piet (veel zaken kunnen afwerken)	6
46	Stijlgids op schermen op online invoicing	6
47	Demo	6
48	Snelheid van ontwikkelen lag stuk hoger (Kwaliteit is aan het verhogen)	6
49	Bart & Martine in het team	6
50	Excel way of working gaat vlot (change request & bugs erin)	6
51	Sprintdemo vandaag was top	6
52	Goeie sfeer binnen het team ondanks rough ride	6
53	Volledige focus op OI binnen ETNIA	6
54	Uitbreiding team	6
55	John als dedicated architect	6
56	Focus laatste weken op OI (vanuit alle hoeken)	6
57	John als dedicated architect	6
58	Flexibiliteit dev team - snel mee opgenomen & taken toegewezen	6
59	Perceptie door demo vandaag - signaalfunctie naar mngt	6
60	demo	6
61	Focus tijdens de sprint ifv. oplevering & demo + stabilisatie OI	8
62	Demo werd gegeven op QA omgeving (proces begint gestroomlijnd te geraken)	8
63	Korte beslissingslijnen en communicatie	8
64	demo goed verlopen & goed voorbereid	8
65	Meer realistische planning (60% capaciteit ingepland)	8
66	Invoering van definition of done	8
67	Aantal blocking zaken konden gedeblokkeerd worden	8
68	Styling begint er goed uit te zien (goeie toepassing)	8
69	Vooruitgang bij iedereen (kennis (framework)) en resultaten -> wordt stabieler	8
70	Throughput (snellere reactie, laatste halve week zijn er nog verschillende QA deployments geweest & verbeteringen)	8
71	F2Fes met Mark/Paul & Lars -> voor eerste keer echt geluisterd naar wat de problemen zijn -> verwachtingen!	8
72	Testen gebeurt goed door Piet, bugs komen vlugger naar boven	8
73	Extra sprints komen erbij	8
74	Beschikbaarheid van de architecten (altijd iemand beschikbaar)	8
75	Problemen die gesignaleerd worden zijn niet atypisch	8
76	focus & committment	8
77	willingness, gedrevenheid (ondanks kennisachterstand) -> continue wil tot bijleren	8
78	Eerste keer doel van sprint volledig gehaald worden	8
79	QA omgeving is beschikbaar	8

80	Demo is SUPER, maar nog niet alles is klaar	8
81	Goeie demo	8
82	Goed gevoel na demo, maar niet alles is getoond geweest (geen errors)	8
83	Communicatie	8
84	Pragmatiek boven procedures	11
85	Size doesn't matter -> elkaar niet voor de voeten gelopen ondanks groter team	11
86	Heel korte afstemming qua functionaliteiten & dan ontwikkeling	11
87	Team is gegroeid (meer mensen & meer kennis)	11
88	Eerste demo wanneer alles getoond werd wat nodig is voor OI v1.0	11
89	Eind augustus zal het er moeten zijn & kan het er zijn	11
90	Geen analyse statusmeetings meer -> geen tijdverlies meer	11
91	Grid & toetsen onderaan responsive	11
92	Enkel data ophalen van 1 gridscherm -> goeie verbetering qua performantie	11
93	Finish is in zicht, marathon einde komt in zicht	11
94	Vakantieperiode -> veel klaar gekregen + voldoende tijd (minder opgejaagd)	11
95	Goeie sprint -> 7 waard	11
96	Interactie in team heel hoog & veel initiatief	11
97	Minder bugs	11
98	ondanks vakantie toch stevig doorgewerkt	11
99	Heel goeie focus ondanks vakantieperiode	11
100	demo, goed stabiel snel, goed gegeven	11
101	heel goeie velocity	11
102	maturity van het team stijgt	11
103	Stabiliteit online invoicing	11

Table 11 Sprint Retrospectives Online Invoicing v1 - What went well

	What could be improved?	Sprint
1	Snelheid review analyses	0
2	Duidelijkheid rond UI guidelines	0
3	Visie over functionele/technische architectuur Kluwer Suite - globaal plan van aanpak?	0
4	Feedback rond requirements (duidelijkheid user story)	0
5	Stand van zaken autocomplete (concrete datums?)	0
6	Review use cases (zelfde use cases teveel moeten reviewen)	0
7	Efficientie development (meer snelheid zouden we moeten halen) - scherm volledig afwerken	0
8	Verkeerd gekozen user story voor sprint 0 (vanuit technische achtergrond)	0
9	Functionele & Technische Guidelines (CSP + UI)	0
11	Tijdsinschatting sprint 0	0
11	Testomgeving	0
12	Prioriteiten stellen tijdens de sprint	0

13	Snelheid review analyses	0
14	Duidelijkheid rond UI guidelines	0
15	DocGen tijd	1
16	Docgen document waar staat het & hoe lezen (JGE, RLI, HS)	1
17	opleiding tfs (source safe)	1
18	leercurve (tfs nieuw, csp nieuw) - nog te weinig csp specialisme (leercurve te traag)	1
19	overhead (probs met laptop) -> beschikbare capaciteit verminderen	1
20	meer één team (met iedereen tezamen)	1
21	tijdens ontwikkeling teveel oog aan detail (geen schrik van rework)	1
22	opstart developer (heel veel info op korte tijd) -tfs, sourcesafe	1
23	csp help (help van csp help nodig - weg in kwijt geraken)	1
24	betrokkenheid domein owner	1
25	communicatie met buitenwereld buiten het team	1
26	Autocomplete (specifiek daarover)	1
27	UX (user experience)	1
28	performantie	1
29	wie doet wat & bij wie moet ik terecht	1
30	TFS - unit tests, checken broken build	1
31	chaos waarbinnen gewerkt moet worden (meer structuur nodig)	1
32	inschatting qua uren (meer taken nodig & aantal taken)	1
33	features ontstaan nog niet spontaan uit combinatie requirements & technische	1
34	Queries goedzetten	1
35	code generatie bij bug -> heel weinig controle hierover -> tijdverlies	5
36	merge probleem -> door niet exclusief uitchecken van files (zaken kwijtgespeeld)	5
37	unittesting geraakt achterop -> strategie is nog niet helemaal duidelijk	5
38	problemen die opgedoken zijn bij mergen e.d.	5
39	Layout van de KluwerSolution moet veel beter	5
40	Manier op fouten te loggen in systeem -> bestaande fouten snel kunnen checken	5
41	Vertalingen -> frans/nederlands door elkaar, niet consistent vertaalde schermen	5
42	HIG & presentatie -> stijlgids zit er nog niet in -> look & feel moet beter	5
43	Autocomplete nog niet geïmplementeerd	5
44	Meer tijd stoppen in testen	5
45	Veel tijd gestopt in geïmplementeerd krijgen van de autocomplete	5
46	Backlog architecten -> dringend knopen door te hakken, zaken die van belang zijn voor OI	5
47	Tempo van ontwikkeling	5
48	Stakeholder management moet beter	5

49	Laatste versie moet gedemood worden op amazon & beter testen -> nu fouten	5
50	Zaken die gedemoed worden die niet af zijn en nog niet volledig klaar	5
51	Unit testing hoort bij definition of done van een taak	5
52	Er zit nog veel bij de architecten -> afstemming tussen guido & architecten noodzakelijk	5
53	Hoe ontwikkelingen in sprintplanning combineren in development met bugs & scope wijzigingen	6
54	Sprintplanning vorige keer - de user story moeten kleiner qua scope gedefinieerd worden om ze als afgerond geheel	6
55	**Cursus javascript is noodzakelijk -> maak er review van**	6
56	Verspreiding informatie (documentatie -> staat te fel verspreid)	6
57	Demo duurt te lang - vragen moment	6
58	**Styleguide implementeren kost veel tijd (nog maar paar controls gedaan & nog veel te doen)**	6
59	Leercurve -> blijft een berg beklimmen	6
60	Vooruitgang architecturale problemen (stijlgids (UI Shell); Filtering Grids, Select all over meerdere pages)	6
61	Snelheid development, blijft berg beklimmen - we halen niet genoeg snelheid	6
62	Kleinere user stories (om afgewerkte user stories te kunnen tonen)	6
63	Buglogging	6
64	Niet alles qua scope sprint 6 is afgewerkt	6
65	Onrust in het volledige team nav. demo	6
66	Verandering van requirements (recht evenredig met DO switch - Thierry vs. Mark/Paul)	6
67	Geen enkele performantietest kunnen doen	6
68	Bugs in TFS (vs. progress van burndown)	6
69	Te algemene user stories voor demo -> verfijndere user stories	6
70	Beslissingen - hoe descriptions ophalen & wegschrijven (CSP of eigen tabellen); EDMX (naamgeving infotabel) - hoe langer het duurt, hoe meer refactoring	6
71	Vragen die op demo gesteld worden (te gedetailleerd voor demo)	6
72	Demodata te verbeteren	6
73	**Integratie van de 2 projecten (SA & OI) -> waar komen de gemeenschappelijke delen terecht**	6
74	Leercurve - webontwikkeling/desktopontwikkeling	6
75	Documentatie analyse staat verspreid -> niet evident om info terug te vinden	6
76	webontwikkeling & javascript (zoeken op internet momenteel)	6
77	Juiste glossary & vertalingen voorzien	6

78	Dev beslissingen in OI documenteren in wiki of AB backlog?	6
79	Autocomplete functionaliteit is al paar keer achteruitgeschoven	6
80	code kwaliteit & definition of done is te respecteren	8
81	Toepassing van defintion of done moet nog groeien	8
82	Soms te lange statusmeetings, op letten! -> timekeeping	8
83	Testing van ingecheckte code kan beter (inchecken feature beter testen of alles ok is)	8
84	Verwachte throughput (80-20 regel) -> mee rekening te houden voor bug solving & planning	8
85	Performantie is belangrijk punt -> vrees is dat er heel ver terug in de code ingegrepen zal moeten worden	8
86	Beperkte verbetering van CSP zelf (tijd geleden input gegeven door DEV team, maar nog weinig verbetering merkbaar)	8
87	Tijdsdruk -> niet evident om de bugs op te lossen	8
88	Beschikbaarheid van product developer (momenteel +/- 0,5 FTE)	8
89	QA process, procedures nog duidelijker te krijgen & te volgen	8
90	UX/UI track krijgt te weinig aandacht (heel snelle review & validatie door management)	8
91	Gebrek aan duidelijke acceptatiecriteria	8
92	Probleem van gelijkaardige implementatie doorheen de suite -> met dev lead scenario doorspreken (code design)	8
93	Waarom zitten technische taken in architectuur team (roles & responsibilities)	8
94	Veel bugs gelogd, nefast om nieuwe zaken te kunnen ontwikkelen	8
95	In principe sprint 9 enkel oplossen van bugs	8
96	Performantie is serieus probleem	8
97	Eerste keer dat schouders & koppen naar beneden gingen tijdens de sprint	8
98	Performantie	8
99	Business model moet duidelijk worden (blijft aanslepen)	8
100	Veel overhead (statusmeeting + 30 min), 2d sprintplanning	8
101	Teveel volk op project (bugs in elkanders code)	8
102	Code review - definition of done -> momenteel niet de moment om daar nu mee starten (haalt dit de kwaliteit nu omhoog)	8
103	Motivatie is te verbeteren	8
104	Geen consistentie in demo's -> geen vragen gesteld over wat er naast wat gedemoot geweest is of ook het ook werkt	8
105	Performantie is heel heel traag (zelfs met wifi)	8

106	Probleem -> niet begonnen met begin, waardoor problemen om kwalitatief & consistent te programmeren	8
107	Problemen met CSP	8
108	Deploy (teveel black box vanuit development)	11
109	Procedure -> persoon vindt iets uit -> kennisoverdracht naar ganse team gaat moeilijk (knowledge sharing)	11
110	Logging - error tracking -> logging tools op dev omgeving (enkel error log) -> moeilijk op vreemde omgeving	11
111	Verbeteren van test scenario's	11
112	Aantal bugs blijft nog steeds groot (76-tal); 18 prio 1's; bugs die terugkomen	11
113	QA wordt laat bijgewerkt -> sneller naar QA (Johan test enkel op QA)	11
114	Opstarten en op F5 -> blijft vaak hangen bij Rudy Lievens	11
115	bug -> taak: vaak moeilijk te vinden waar het probleem juist zit (bv. op welke pagina)	11
116	Druk blijft hoog voor het ganse team, hopelijk waardering achteraf heel hoog	11
117	Demo vandaag niet op dev omgeving	11
118	Releasecycle -> wanneer wat gedeployt etc.	11
119	Demodata, reference data, scripts	11
120	Pintjes met het team	11
121	Demo vanop DEV, testen vanop QA	11
122	Ownership Usies & taken	11

Table 12 Sprint Retrospectives Online Invoicing v1 - What could be improved

E.2 Sprint Retrospectives Online Invoicing version 2

	What went well?	Sprint
1	Good flow within the OI team. Clear roles & responsibilities	1
2	Impressive quality of delivery	1
3	Good code quality. Good quality of code	1
4	Good teamwork	1
5	Feedback on Domain Model -> cooperation between Bart Joris & Tom Princen (=> improve use of Domain Model)	1
6	Sprintscope has been delivered	1
7	Good demo, no negative feedback	1
8	Teamdrive -> open discussions, enthusiasm within team	1
9	Quality: eye for design & testing	1
10	Good looking delivery after 2 weeks	1
11	From the beginning -> start with good testdata (for customers)	1
12	Already focus on (fast) automated deployment -> easier to test & bugfix	1
13	Kathleen is impressed by the result	1
14	Teamspirit values 5 stars -> everybody has grown in it's role	1
15	Impressive delivery after 1 sprint	1
16	Never seen an empty sprint scrumboard before	1
17	Nize team	1
18	Speed & structure	1
19	Collaboration	1
20	2 weeks sprint -> 2 weeks effective delivery time, only informal meetings when needed	1
21	Bart J & Tom P have been very explicit in terminology -> same language!	1
22	Sprint goal & scope have been reached -> ideal for 2 weeks (when more than 2 weeks -> would be difficult)	1
23	Very good demo	1
24	Team atmosphere was superb	1
25	John has learned a lot via the good structure of the solution	1
26	Bigger scope has been delivered than initially (strictly) planned	1
27	Pair programming was very usefull (keep on making use of this when possible)	1
28	Efficient teamwork (scope & goal)	1
29	Team: good communication, small irritations were solved quickly	1

30	Everybody takes his/her responsibility	1
31	Scope: more scope has been delivered than initially planned (webfoundation scope)	1
32	Good quality of delivery, no rework is needed	1
33	Planning of work via scrumboard -> focus	1
34	Standups gingen beter, betere focus	4
35	Interactie binnen het team	4
36	Architectuur zit op poten (voorlopig)	4
37	Veel gevloekt tijdens de sprint, maar uiteindelijk zag het er goed uit.	4
38	Indruk dat de applicatie kwalitatief heel goed zit	4
39	Testing methodologie voor stuk kunnen uitrollen (test cases & testen)	4
40	Elke demo blijft het waaauw gevoel	4
41	Onder de indruk -> kwaliteit -> er wordt nagedacht door het team	4
42	Demo's heel goed, zit goed in elkaar	4
43	3 keer op rij doelstelling gehaald + wel veel focus	4
44	Code review was heel heel nuttig	4
45	Fijne samenwerking & communicatie, zelfs in moeilijkere sprint	4
46	Code review meeting (elke sprint doen?)	4
47	alles werkt, heel kwalitatief	4
48	Code review was interessant, verhelderend voor FA	4
49	Veel functionaliteit opgeleverd - cirkel is rond	4
50	Veel changes vanuit Boondoggle -> opportuniteit om op kwalitatief op te leveren rekening houdende met hun input	4
51	Focus op features, maar ten koste van kwaliteit, moet opgelost worden	4
52	Daily standups werkten goed, maar briefjes beter organiseren, geen tijd verlies	4
53	Gisteren goeie dag - deploy naar DEV was goed dag ervoor	5
54	Alles loopt lekker	5
55	Heel veel minder broken builds	5
56	Bug fixing verdelen dag voor de demo -> pluim	5
57	Relatie met Piet loopt beter en beter	5
58	Sessie dag voor de demo heel goed, op laatste veel fixen -> goed gevoel voor de demo	5
59	We waren dag voordien klaar -> kwalitatieve demo	5
60	Totals & analysis was very good -> snel te implementeren	5
61	Veel business value opgeleverd	5
62	Continue work on quality	5
63	Good atmosphere	5
64	Meer gepland dat wat we aankonden, goed team, test data approach	5

65	Piet betrekken -> loopt beter	5
66	TEAM functions well - betrokkenheid	5
67	Quality has been improved (queuing, etc.)	5
68	Unit testing functions, builds functions	5
69	System Testing process	6
70	Update Scrumboard	6
71	Heel veel werk verzet	6
72	Samenwerking met Boondoggle	6
73	Bug tijdens demo -> rustig bekeken, alternatief gezocht, snelle communicatie wat het probleem wsa	6
74	Integratie van de styling ging supervlot, weinig problemen	6
75	Heel constructieve discussies binnen het team -> goeie beslissingen, toffe ideëen	6
76	Veel op unplanned work gewerkt, maar toch meer opgeleverd dan ingeschat	6
77	Koude pintjes -> veel te laat geleden, moeten we opnieuw doen	6
78	Heel veel opgeleverd	6
79	Heel veel verschillende componenten (ubl, scheduler, design UI)	6
80	Integratie van de styling, goed gepland	6
81	Query model (client side -> was getypescript -> zit goed in elkaar)	6
82	Aan heel veel gewerkt en kwalitatief	6
83	Demo was goed	6
84	Meer dan gepland, client side werk gedaan	6
85	Code review	6
86	Vlot oppikken van werk van scheduler	6
87	Analyse van PDF -> ging moeizaam vooruit, er waren geen pixels in het begin beschikbaar etc.	7
88	Heel goeie demo	7
89	Heel snel foutloos & goed geprogrammeerd	7
90	Meer OI functionaliteit uitgewerkt dan voorzien	7
91	Heel goeie demo	7
92	Flow van het team, ondanks het missen van aantal kleppers (meer focus) -> user story af, lijstjes gemaakt, dry run	7
93	Interactie met Kathleen & Piet (autosave, lijncreatie, tekstlijn) -> iedereen werkt & denkt mee	7
94	Autoprovisioning -> meest opmerkingen van Fabio zijn meegenomen -> gebruik dit communicatiekanaal (wanneer Steven & Joris terug zijn)	7
95	UX/UI integratie gaat heel vlot -> wijzigingen zijn vlot door te voeren (Wouter's styling & impact op functionaliteit)	7

96	User Stories goed afgewerkt, lijstjes, veel werk verzet -> pluim voor Sophie	7
97	Voorbereiding demo	7
98	Volume dat opgeleverd is (copiëren, bulkacties)	8
99	Problemen die opduiken, snel oppikken binnen het team -> geen opmerkingen die erop komen (heel goeie samenwerking)	8
100	Geen terugkerende bugs	8
101	Interactie dev, ana & UX	8
102	Functionaliteit die opgeleverd is, ook complexe zaken	8
103	Goeie demo want goed voorbereid	8
104	Goeie demo & voorbereiding, user testing	8
105	Samenwerking met piet & kathleen mbt. pdf (maten op pdf) - eindelijk af	8
106	Ramon die erbij is gekomen, er wordt snelheid gemaakt	8
107	Ambitieuze scope + aantal extra's	8
108	User Tests	8
109	Zwaarden van damocles (autosave, decimalen, provisioning) -> vlot opgelost	8
110	Dank aan Bart de poortwachter	8
111	Sophie & Piet -> supercombo	8
112	User Testing -> schitterend om uitdrukkingen op gezicht van de mensen te zien	8
113	User Testen met positieve feedback, van ruimere groep	8
114	Heel snel zaken oplossen (applicatie is gekend en zit goed ineen)	8
115	Bulk - joepie op 3d in orde!	8
116	Omgevingen worden ad hoc gebruikt	8
117	VIES = vlot geïmplementeerd	9
118	Change requests + flexibiliteit = mooie value	9
119	Veel fancy stuff opgeleverd	9
120	Demo was heel goed -> veel functionaliteit & value	9
121	Goeie samenwerking met Piet & Kathleen (VIES) - Goeie Demo	9
122	Background work (unplanned work)	9
123	Goeie demo - worker notifications, website licensing	9
124	Goeie demo	11
125	Veel bugs opgelost	11
126	Deployment	11
127	Samenwerking met tolk sybille ging heel vlot	11

Table 13 Online Invoicing v2 Sprint Retrospectives - What went well

	What can be improved?	Sprint
1	Good demo, but there was no feedback from business (Mark C or Johan George)	1
2	Organization within TFS should be improved	1

3	Scrummeeting should be shorter. However the discussions/communication is very usefull	1
4	Acceptance criteria between analists & developers -> more open & clearer communication is needed	1
5	Functionality to switch between languages is implemented differently than foreseen on mockup	1
6	Analists need to perform joint sprint planning for analysis/testing tasks to work towards joint goal	1
7	Anxiety about requirements concerning e.g. document templates and the room for open discussions about these requirements	1
8	Starting from product backlog with user stories (=OK) -> not yet a clear link with the user stories within sprint backlog(s)	1
9	Difficult to get all tasks fluently within TFS: tasks pop-up on scrum board -> how to get them easily in TFS	1
10	Faster system testing should be possible (sprint 1 -> no system testing was planned for sprint 1, PGH was able to test on Friday morning)	1
11	KDE will adapt Skype settings	1
12	Unit tests have been defined (client & server side) -> value for regression testing?	1
13	Duration of daily statusmeeting may be optimised	1
14	Make the analysis tasks within the sprint more relevant for the DEV team	1
15	Improve hour administration (TFS, Briljant, Excel, (AE Timesheet))	1
16	Ingeboet op kwaliteit van code - te weinig unit testen	3
17	Focus kan beter, minder gedaan dan vorige sprint, er komt teveel tussen (afleiding, elkaar onderbreken, meetings)	3
18	Technical debt zal opgebouwd worden - sommige dingen zullen aangepakt moeten worden id toekomst	3
19	demo data & testen - demo data in DB (=> probs) -> aanpak demo data	3
20	Deze sprint ahv. User stories gewerkt - alle taken waren nu op voorhand gemaakt (taken matchten niet op voorhand) - taken vaak placeholders - relevantie op voorhand bepalen is moeilijk	3
21	Deze sprint gevoel onder de analisten gegrommel dat er veel aangepast moest worden, veel change requesten vanuit boondoggle (change systeem voorzien)	3
22	Test Data voor Customers -> via API test data genereren?	3
23	Beter inplannen rond look-ahead modelling -> door Boondoggle?	3
24	Voor sprint demo issues doornemen vooraleer ze worden aangemaakt als bug	3
25	Duidelijker definiëren naar de kippen wat wel & niet in de sprint zit (hoe kunnen we dat duidelijker communiceren?)	3

26	Deze sprint enkel bezig geweest met boondoggle & kathleen -> moeilijk naar Tom toe. Moeilijk om te vertellen naar het team toe wat we zullen doen	3
27	Standups zijn heel nuttig maar duren te lang (halen het team uit de focus) (JOP)	3
28	Aantal To Do's in code & in hoofd -> veel focus op features, opletten voor technical debt	3
29	E-mail preview scherm (2 dagen in gestopt die verloren zijn gegaan) => template scherm gemaakt zonder specs	3
30	Boondoggle input is heel dringend	3
31	Wat lastige momenten onder de analisten	3
32	Meer kwaliteit tijd inplannen - quality time	3
33	Niet elke sprint evenveel functionaliteit op te leveren - moeilijk	3
34	Kwaliteit is een issue op dit moment -> meer aandacht (code coverage van 70 nr 65%)	3
35	Daily standup -> focussen op wat gezegd wordt	3
36	Demo data aanpak kan beter	4
37	Acceptance criteria is nog te verbeteren (misverstanden tussen sprint voorbereidingen en uiteindelijke sprintplanning)	4
38	Duidelijke toelichting wat de echte scope was bij begin demo	4
39	Product Ownership - inbreng team	4
40	E2E quality issue: test data, regressie, door onafgewerkte features	4
41	Scope	4
42	Sfeer in het team heel goed, goeie scrums	4
43	Kwaliteit van de unit testen moet beter -> refactoren	4
44	System Test rapport is niet ok -> dit geeft slecht rapport	4
45	Demo data -> zorgde voor veel irritatie -> scenario was niet gekend voor goeie demo data	4
46	Te laat begonnen met voorbereiding van demo, vroeger voor de algemene gemoedsrust	4
47	Doel, code review, focus van team	4
48	Acceptatiecriteria waren altijd voor volledige user story -> moet anders aangepakt worden. Proces bijsturen	4
49	Acceptatiecriteria komen niet op de juiste manier bij de juiste persoon terecht	4
50	Tijd om sprint 5 voor te bereiden (onder analisten) was ok	4
51	Kwaliteit is super	4
52	Planning look ahead modelling onder analisten zat goed	4
53	acceptatiecriteria te verbeteren	4
54	bugs & test rapportering te verbeteren -> beter rapport krijgen	4

55	Voor demo overlopen wat uit de system testing is gekomen tussen Steven & Piet	4
56	Scope is duidelijker geformuleerd -> duidelijk effect naar product developer & business solution owner	4
57	Heel goed opgevangen in het team	4
58	Communicatie & sfeer in het team zit heel goed	4
59	Bundelen wat er voor nieuwe starter nuttig is -> ligt klaar voor Michael M. (installation guide etc.)	4
60	Front-end code -> mooi! Paar stukken waar commentaar wel nuttig zou zijn (filters etc.)	4
61	Eerste sprint dat het gevoel goed zit dat we typescript gebruiken ipv. Javascript	4
62	Data moet beter, ook testen, unit testen naar het functionele toe -> op basis van de regressieproblemen die opduiken	4
63	Front-end -> architectuur met angular & bootstrap is flexibel en snel wijzigbaar	4
64	Overzicht, veel taken -> pas bij voorbereiding van demo was het duidelijk waar iedereen mee bezig was	4
65	Front-end/Back-end -> afstemming ertussen zat nog niet helemaal goed	4
66	Sinds vrijdag, tandje bijgestoken, iedereen was erbij betrokken	4
67	Sinds vrijdag, E2E issues -> commitment zit goed, iedereen was er vroeg om erop te vliegen	4
68	E2E testen	4
69	Heel straf dat de demo nog goed gekomen is -> zegt veel over robuustheid & kwaliteit van de solution	4
70	Joris, 1,5d capaciteit deze sprint, zou terug naar 80% moeten gaan	4
71	Joris op front-end gewerkt -> zit goed	4
72	Zaken die niet afgewerkt geraken, die blijven liggen (bv. totalen, etc.)	4
73	Acceptance criteria	4
74	Totals & back-end - contracts	5
75	Integratie van boondoggle -> aparte branch -> gebroken bindings etc -> zelfde branch als wouter gebruiken	5
76	Tegen het einde van de sprint liepen we elkaar voor de voeten -> onduidelijk hoe dit op te lossen	5
77	Als front-end developer -> heel veel inconsistente input -> teveel verschillende input	5
78	Feedback die van de demo komt, gaat meestal niet over de essentie -> gaat vaak over de design	5
79	Nog niks van webfoundation scope opgepikt -> integratie met andere pakketten	5
80	Heel veel wachten op elkaar -> contracten & totalen	5
81	How to integrate boondoggle designs	5
82	User Story ownership can be improved	5

83	Database changes	5
84	Acceptance criteria -> documenteren van user story mbt. sorteren zit niet logisch in elkaar	5
85	Impact on final decisions is not ok -> should be improved	5
86	Tijdsregistratieproces	6
87	Veel unplanned work -> waar komt dit uit? Functionaliteit die uit styling & designs kwam?	6
88	UI & Async calls (trager in geval van veel gebruikers) -> tool ervoor gebruiken?	6
89	Schattingen (due date toestand, registratie process)	6
90	Demo & bugs -> te laat opgepikt	6
91	GSS infrastructuur -> wordt niet gemanaged	6
92	User registration flow heel lange duur -> is niet kwalitatief -> moeten we oppikken	6
93	Misschien teveel willen opleveren -> scenario demo niet voorbereid	6
94	UBL lastig te testen -> QUA & Test omgeving voor andere pakketten	6
95	Demo preparatie -> op code review alle user stories dan beslissen wat gaan we tonen en wie?	6
96	Zien wat misgelopen tijdens de demo	6
97	In de logs ontbraken wat errors (using block voor exceptions)	6
98	Demo ging niet vlot -> alles was last minute, alles werd vandaag afgewerkt tot op het laatste moment	6
99	Terugkoppeling van kennis naar de groep die buiten het team ligt	6
100	Niet fris, kennis van client side reikt niet echt ver (duurt wat langer)	6
101	User registration analyse was niet klaar (heeft veel te lang geduurd) -> nog exceptions die niet ok zitten	6
102	Niet concreet genoeg getest ter voorbereiding van de demo (set aan data die we kunnen checken) -> paar setjes van juiste data	6
103	Unit testen altijd runnen	6
104	Jammer -> final designs van boondoggle nu pas klaar -> er kunnen nog wijzigingen opduiken	6
105	Analyse van PDF -> ging moeizaam vooruit, er waren geen pixels in het begin beschikbaar etc.	7
106	Andere scope dan PDF kunnen doen (John)	7
107	Terugkerende bugs -> Piet overloopt de test cases van in het begin van de sprints -> how come?	7
108	Buy in van business -> Johan & Mark, vooral Johan	7
109	Boondoggle beschikbaarheid van Wouter -> 1,5 dag beschikbaar voor Kluwer van de 3d dat Wouter hier was	7

110	Demo was heel goed en feature-wise valt alles op zijn plaats -> alleen naar happy path gekeken (connectietraag, veel mensen tegelijk)	7
111	Hoe zit het met fail-over, invloed op QA (load balancing, sql mirroring, etc)	7
112	Fabio input mbt. marginality - wat kost de beslissing (autosave, bulk acties)	7
113	Regressie! Tijd vrijmaken op te onderhouden	7
114	Timemanagement (Piet) -> puntjes op de i uitzoeken; op laatste spurtje moeten trekken om sprintplanning rond te krijgen	8
115	Meer bugs dan anders, maar wel snel opgelost	8
116	Tijdsregistratieprocess	8
117	Administratie neemt toe, .net verlof moet ingevoerd worden	8
118	Deploy op pc van Kathleen was frustrerend -> moet gedocumenteerd process zijn	8
119	Focus, chaotischer, frustraties	8
120	Te druk met user testing, weinig beschikbaarheid voor het team	8
121	Geen fijne sprint, overhead (meetings, docs schrijven, weinig kunnen doen!)	8
122	Bezorgd over to do's onder de motorkap (logging, cashing, DB tuning etc.)	8
123	Concurrency bugs	8
124	Productiefeeling, veel mensen tegelijk, kritische mensen -> uitrollen bij interne mensen	8
125	Goeie afspraken maken	8
126	Kathleen overbelast	9
127	Bij planning -> voldoende in vraag stellen of er voldoende werk is	9
128	Volgorde business - functionele analyse naar development	9
129	Niet voldoende verscheiden werk, waardoor wachten op elkaar	9
130	Frustratie omtrent licenses (tijdens de sprintplanning tijd voorzien om dit uit te denken	9
131	Eind eerste week -> zonder werk -> veel was ingezet op licensing (geen analyse voor handen)	9
132	Gevoel dat er weinig opgeleverd is door JD (aparte branch, veel merge conflicten etc.)	9
133	Samenwerking met GSS -> lange termijn visie & samen denken, teveel vanuit DEV (Joris & Steven)	9
134	Performance testing, security testing	9
135	Licensing: analyse + communicatie	9
136	Testing!	9
137	Werk was niet goed verdeeld	9
138	Infrastructuur -> weinig schot in de zaak	9
139	scrum is chaotischer	9
140	Lead DEV weinig betrokken bij deze sprint	9
141	Scrum duurt heel lang (wel nuttige elementen)	10

142	Demo was niet goed voorbereid, er is pas achteraf gedebugged	10
143	Samenwerking met GSS gaat traag, moeizaam, kan verbeteren	10
144	Voorbereiding Demo was slecht, eigenlijk geen voorbereiding en geen goeie demo - perceptie was wel goed	11
145	deployment loopt niet vlot -> wat kan er verbeterd worden? Te bekijken	11
146	Licensing verhaal, pdf's, is maand of 3 blijven liggen, mss niet goed opgevolgd? Veel werk om dit te vervangen door andere component	11
147	Unit testing	11
148	Code minder goed? Gevolg van unit testing -> minder business logica, maar wel te bespreken hoe dit te verbeteren	11
149	Service BUS (msnq & workers id achtergrond) -> is fundamenteel, wordt wrschlk bijgestuurd -> impact op infrastructuur	11

Table 14 Sprint Retrospectives Online Invoicing v2 - What could be improved

Auteursrechtelijke overeenkomst

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Case Study: Agility at Scale Wolters Kluwer Belgium

Richting: **Master of Management-Management Information Systems**
Jaar: **2014**

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Voor akkoord,

Devos, Tom

Datum: **21/08/2014**