Agile software development process improvement in large organizations

Bart Leusink

August 20, 2012

Supervisor:
Marko van Eekelen

Second assessor:
Theo Schouten

research number: 168IK

Abstract

This thesis deals with software development process improvement in large organizations. CMMI is combined with agile practices and concepts from literature to obtain an agile CMMI framework. This framework is consequently tested in a case study at a large energy supplier in the Netherlands. The framework proved helpful in determining problems and opportunities for improvement in an existing software development process. This shows that an agile CMMI framework can help a large organization improve their software development process.

Contents

Chapter 1

Problem Statement

In software development, waterfall and agile are both trade-offs between speed and risk [27]. Where waterfall provides the more risk averse option at the cost of speed, agile is less disciplined and consequently a faster and more dynamic way to develop software.

Agile software development in large organisations is an area of continuing research. In "Lots Done, More to Do: The Current State of Agile Systems Development Research" [1], Abrahamsson et al. state that more research is needed about the adaptability and extension of agile methods in large organizations.

In an effort to produce more, faster and at lower cost, an increasing number of large organizations is switching to agile software development methods [52] because they seek an alternative to more traditional methodologies such as waterfall, which they find too cumbersome, bureaucratic and inflexible [24]. There is common consensus that agile methodology implementations need to be tailored to a specific organization while at the same time organizations need to adapt to an agile way of working. Many large organizations opt for an agile–waterfall hybrid approach, obtaining a balance between risk and speed that fits their situation [6].

It is, however, easy to create an agile–waterfall hybrid which is far from efficient or even problematic [9].

The use of process improvement frameworks such as CMMI [51] can help organizations continually improve their software development processes. There have been many articles about combining CMMI with Agile development [18, 40, 19, 15, 3, 47] but none of them go into detail about specific agile-waterfall hybrid implementations of the various CMMI process areas.

In this paper, I will combine CMMI with an agility improvement model to measure and improve agile processes. This provides large organizations implementing a hybrid methodology a framework to improve their agility while at the same time maintaining or improving their process maturity.

My research question is: *Can an agile CMMI framework help a large organization improve their software development processes?*

1.1 Research methodology

1.1.1 Literature Review

I will start by reading up on existing literature regarding the adoption, successes and failures of agile software development methodologies in large organisations, as well as currently widely used and well-cited models for assessing the maturity of agile software development processes and improving them.

Based on this analysis I will then combine the agile principles behind these improvement models with CMMI to create a version of CMMI which can be used by large organizations to measure their current efficiency of agile-waterfall hybrid software processes as well as provide them with suggestions for process improvement. This requires I investigate the various dimensions in which a hybrid software development methodology can be tailored, what the specific choices between waterfall and agile approach entail and the corresponding consequences for the organization.

For example, the choice to not provide elaborate requirements up front has implications for the amount of customer involvement to arrive at the intended product. The same goes for documentation; not sharing knowledge explicitly through documentation puts more emphasis on tacit knowledge sharing, which requires the organization to provide opportunities to do so.

Which choice is best for the organization is dependent on multiple factors as each choice has its advantages and disadvantages.

1.1.2 Case Study

To test the aforementioned agile CMMI framework I will use it to perform a case study at a large electricity corporation in the Netherlands. I've chosen to focus my study on the B2C department because it's agile implementation is the furthest along and as such can provide a more interesting proving ground for my proposed model.

To assess the current agility of agile software development at the company I will use the indicators of the Agile Measurement Index [38] (see section 5.4 on page 20) to interview several stakeholders at different stages of the agile software development process. I've chosen to use this framework because it's well-cited and has a very comprehensive list of indicators [50] I have reduced the 300 indicators from [50] to the 45 interview questions in appendix 10 on page 58. These questions are meant to detect the presence of the various agile practices and concepts discussed in section 5.4 on page 20.

Before and after I ask the stakeholders about the state of the various agile principles in the software development process in the organization, I will ask them whether they can name problems or things that can be improved in their current software development process. This way, I hope to get a fairly extensive list of problems that exist with the software development process.

I will annotate the interview answers with whether they provide evidence for a full implementation of the agile practice or concept (+), a partly implemented agile concept or practice (+/-) or lack of an implementation (-). I will email the interviewees a copy of their annotated interview answers to assess whether they agree with my interpretation of their answers.

I will then use this information and the proposed agile CMMI framework to assess the agility of the software development process using the scoring method described in the indicators document of the agile adoption framework [50] and list opportunities for improvement in the software development processes of the B2C department. Every (+) annotation is worth 1 point for that specific agile concept or practice, every (+/-) is worth half a point and every (-) is worth no points. I will divide the amount of points for a specific agile practice or concept by the amount of annotations for that specific concept or practice to get a final score. If that score is higher or equal to 0.75 I will count the agile practice or concept as implemented.

Since the ultimate goal of both CMMI and Agile is maximizing value for the organization, the improvements proposed by an agile-CMMI hybrid model should contribute to the organizational goals. To determine whether my framework actually provides these improvements in practice, I will ask an agile expert to assess the applicability and value of the suggested improvements.

Chapter 2

Theoretical Framework

In this chapter I will discuss some of the important concepts which should be known when discussing agile software development process improvement. In section 2.1.1, the waterfall model is described. This is a traditional plan-based approach to software development process management. Agile software development is introduced in section 2.1.2. Agile software development differs from waterfall in the fact that it uses short iterations to incrementally create software with a minimum of planning upfront.

2.1 Software Engineering

A commonly used definition of Software Engineering is coined by Friedrich L. Bauer in 1968: [31]

> “The establishment and use of sound engineering principles to obtain economically software that is reliable and works on real machines efficiently.”

While this definition clearly establishes the desired end result, there are several software engineering process models to achieve this result.

There are a few common tasks in the software engineering process: [49]

- **Requirements gathering**: the developers sit down with the customer to talk about his demands for the product and to find out the purpose of the software.
- **Design**: in this phase the customer's requirements are analysed and based on this a design for the software product is made.
- **Coding**: during this step the software is actually programmed and subsequently integrated.
- **Testing**: the software is examined for errors and it is verified whether the delivered functionality matches the design and customer requirements.
- **Operations**: the product is put to use by the customer and is supported by the developers.

While these steps are present in most software engineering projects, the way of implementing them can vastly differ between various software engineering process models.

2.1.1 Waterfall

In many organizations the waterfall model or a variant thereof is used. This model was first described by Winston Royce [37]. Royce suggested implementing multiple levels of feedback to improve robustness of the software development process and increase the chance of a successful result. However, nowadays the waterfall model is often seen as a model in which all steps in the project are executed sequentially. In practice, a pure sequential order of the steps of the waterfall model is often not possible. It is often difficult for the customer to know and articulate all of the requirements in the beginning of the project and since he is only shown the end result after a long development process, there is considerable risk of there being a mismatch between the end result and the actual wishes of the customer [49].

2.1.2 Agile

Agile software development is based on the much older practice of iterative and incremental software development [22]. Iterative and incremental software development is based on the Plan-Do-Check-Act cycle, an easily understood and widely used model from quality management, popularized by W. Edwards Deming.

In the context of iterative and incremental software development, the steps in the software development process can be represented by the following parts of the Plan-Do-Check-Act cycle:

- **Plan**: the requirements are gathered and a single iteration is planned
- **Do**: the plan is implemented
- **Check**: the project team reflects on the plan and it's results
- **Act**: based on the "Check" step, improvements in the process are implemented

Compared to the waterfall process model, iterative and incremental software development has shorter cycles in which the previously mentioned software engineering process steps are carried out to deliver a working piece of software at the end of the cycle. This principle is essential for agile projects since relatively small iterations allow for frequent evaluation of customer value and continuous improvement [46].

The term *agile software development* was coined in 2001 by a group of 17 software developers discussing lightweight development methods. The end result of that meeting was *The Agile Manifesto* [46], a document outlining the purpose and principles of agile software development.

The writers of the agile manifesto value [46]:

- **Individuals and interactions over processes and tools.**
  The writers of *The Agile Manifesto* propose to build projects around motivated individuals and to trust them to get the job done when they get the environment and support they need. They believe that the best architectures, requirements and designs emerge from self-organizing teams.
- **Working software over comprehensive documentation.**
  In agile software development, working software is the primary measure of progress. Customers usually don't care about design documents and diagrams, they care whether they are delivered software that works as expected. That's why in an agile software development process, the highest priority of the project team is satisfying the customer through early and continuous delivery of valuable software. Due to the high volatility of today's project demands, this customer value needs to be reevaluated frequently. In large organizations, work on the specifications can take a lot of time and the end results are often obsolete by the time they are finished [24].
- **Customer collaboration over contract negotiation.**
  While the writers of *The Agile Manifesto* accept that contracts are often necessary and can provide some useful boundaries to the project, they point out that continuous customer collaboration is needed to make sure the team delivers what the client wants. Consequently, agile software developers should not expect a detailed set of requirements to be signed off at the beginning of the project, rather, they should gather a number of high-level requirements that is subject to frequent change. Since this is obviously not enough to design and code, this gap can be closed by continuous collaboration between business people and developers. The most efficient and effective method of conveying information with and within a development team is face-to-face conversation since a lot of the required information is tacit knowledge, which is not easily codified. Many large organizations face problems related to requirements, which can be a strong driver for a move to a more agile software development process [24]. For example, when deadlines require that software development is started when only partial or high-level requirements are available, organizations need to manage projects with incomplete requirements and find a way to better understand the end user's needs.
- **Responding to change over following a plan.**
  In a highly dynamic, competitive environment, faithfully following a plan instead of allowing for changing requirements can easily lead to an end product that no longer corresponds with the customer's wishes. Allowing changing requirements –even late in development– can be a competitive advantage. Rapid changes in the requirements and other environmental factors drives large organizations to seek a flexible software development process which is capable of adapting to volatile requirements [24].

While the writers of *The Agile Manifesto* do think there is value in the latter part of the statements, they believe that the first segment of the bullet point is more important. This distinction lies at the heart of agile software development [46].

In practice, there are many different ways in which these principles can be implemented. Currently, the most popular agile software engineering methodology is Scrum [52].

Chapter 3

Challenges to Agile improvement

Various authors [9, 24, 2] have described specific challenges that can prevent large organizations from achieving higher agility or maturity. In *Agile Software Development at Scale* [2], Scott Ambler mentions a number of complexity factors that exist when applying Agile strategies in large organizations. While each project will face a different combination of these challenges, they can be divided in the following 9 categories:

3.1 Team size

Large teams will work differently from small teams and need to be organized differently as well [2]. Many Agile practices are based on the assumption that teams are relatively small [7, 24]. With much larger teams, paper- and whiteboard-based processes as well as face-to-face strategies start to fall apart.

Some organizations have experienced successes with dividing labor within projects into smaller subteams and organizing teams-of-teams (e.g. scrum-of-scrums) [24, 21, 41, 52], where each subteam selects one member to take place in an overarching team, coordinating the efforts of the various subteams. However, critics [9] argue that multiple 15-minute daily meetings can quickly become untenable. Some organizations [24] believe that the solution to this problem lies in minimizing the need for cross-team communication.

3.2 Geographical distribution

In the ideal agile physical setup, the team is in the same room to facilitate easy communication [38]. However, in large organizations it is not uncommon for team members to be on different floors or even in different buildings. Frequent face-to-face communication between the developers and the customer is crucial to achieve a high level of agility [38]. Just like with large team sizes, effective collaboration and knowledge sharing becomes more challenging when the team is distributed since agile processes that require face-to-face interaction can no longer be used as effectively [2, 24, 7]. This creates the need for more formal communication, such as meetings and documentation [24].

One solution to facilitate cross-team communication is the creation of team-of-teams [24, 21]. These have described in section 3.1 and the same challenges apply here as well.

When the organization uses outsourcing for a part of the project, additional challenges such as timezone differences and cultural differences can appear [2].

Forrester [52] reports that some organizations have successfully been able to mitigate these concerns by adopting technologies and techniques that support distributed collaboration, such as wikis and web conferencing platforms.

3.3 Entrenched culture

Large organizations have often strived for years to achieve stability by using optimized and repeatable processes and producing large amounts of documentation [33]. This has created an entrenched culture in which people, processes and policies can not be easily changed to accommodate a more agile way of working [2].

Due to the agile concept of team empowerment, team members in an agile development process often have additional tasks next to their standard development position descriptions. These tasks may require significantly more skills and experience from the developers [9].

Agile project management requires the project manager to relinquish a lot of the authority he previously enjoyed in favor of a more facilitative role [33]. If an agile implementation does not have significant buy-in from middle management, it is much more likely to fail, since middle management is critical in driving change in large organizations [39].

The evolutionary or iterative process of agile software development creates a level of uncertainty and ambiguity in long-term estimates [9]. With a completely agile way of planning software development, which specific functionality will be finished can only be known shortly in advance but in large organizations business processes and infrastructure require near-perfect predictions of difficult to estimate tasks [9]. For example, departments in an organization need to know when new features or changes are finished to be able to do their job.

Since an agile way of software development requires a fair amount of customer collaboration, other departments need to allocate time to facilitate this collaboration to ensure that the resulting software meets their needs.

When traditional processes overlap or conflict with agile practices this creates double work, which greatly reduces agility [24].

3.4 System complexity

Large organizations often require large, complex systems which create the need for a viable architectural strategy [2]. The standard Agile practice of starting development before the requirements are well defined can lead to forgotten or misunderstood features in large, complex projects. Likewise, resources and time can not be easily estimated without a detailed plan.

While these risks might be acceptable for small or medium-sized projects, Agile development of large, complex systems requires a different trade-off between risk and development speed.

Boehm and Turner [7, 9] proposed creating an Agile-Waterfall hybrid methodology based on the amounts of potential- and acceptable risk. See section 4.2 on page 13 more a more detailed description of their risk-based method of creating a hybrid Agile-Waterfall software development methodology.

3.5 Legacy systems

Large organizations usually have a number of legacy systems that can't be easily changed by using an agile methodology because these systems often don't allow their replacements to be built in an incremental way [9].

Another issue is the quality of the legacy system, since some legacy systems may have code that is poorly written and documented or which has degraded over the years as multiple people worked on it [2].

3.6 Regulatory compliance

Large organizations often have a multitude of regulations they have to comply with.

A number of these regulations are external, for example imposed by the customer or the government. A few examples of these external regulations are financial regulations such as Sarbanes-Oxley and Basel III for financial risks and technical regulations such as ISO 27002 for security risks.

There are usually also internal regulations, such as CMMI and ISO 900x for process improvement. These regulations can increase the documentation and process burden on a project and complying with them while remaining agile can be a challenge [2, 9].

To address this issue, an Agile-Waterfall hybrid methodology can be tailored to the organization, ensuring that the project complies with all regulations while still benefiting from the use of Agile practices where appropriate. In chapter 4 on page 13 I will describe various methods for creating an organization specific blend of Agile and Waterfall practices.

3.7 Organizational distribution

Large organizations often hire consultants or contractors to help out with software development projects. Managing Agile software development processes and achieving high levels of agility becomes more difficult when the team is made up of people working for different departments or different companies [2].

3.8 Degree of governance

IT governance is closely related to regulatory compliance since IT governance usually leads to internal regulations. IT governance frameworks such as ITIL, CMMI and COBIT ensure alignment of IT with business goals and provide structure to the IT development and management processes. The structure established by these frameworks may hinder agility improvement [2, 9]. Traditional forms of governance, such as contracts, milestones and progress measurement techniques may be inadequate to support the rapid pace of agile processes [9].

A solution to this is a more agile approach to governance. In chapter 6 on page 26 I describe an implementation of CMMI with Agile practices.

3.9 Enterprise focus

Large organizations often want to leverage common infrastructure platforms to lower costs, reduce development time and improve consistency. This requires effective architecture, portfolio management, business modelling and reuse disciplines.

Combining these disciplines with agile processing may prove challenging [2].

Chapter 4

Hybrid Agile

Hybrid methodologies are software development methodologies which combine waterfall processes with aspects of agile methodologies. Many organizations choosing to implement agile methodologies opt for a hybrid approach [52], where instead of strictly sticking to a specific methodology, they cherry-pick practices from several agile and plan-based methodologies. There are various methods for composing an Agile-Waterfall hybrid software development methodology from individual Agile- or plan-based practices.

In this chapter I will discuss some of the ways specific practices can be selected which best match an organization. In section 4.1, I will discuss the adapted base methodology of creating a hybrid methodology. A risk-based approach to choosing the amount of agility in a software development process is described in section 4.2. A method based on cost/benefit analysis is discussed in section 4.3.

4.1 Adapted Base

When using the adapted base methodology [20], teams choose a specific methodology - such as Scrum- and then evaluate each practice of that methodology to determine whether its purpose is congruent with the goals of the organization and project. If not, the practice can be removed or replaced with a similar practice from e.g. waterfall or another agile methodology. Boehm and Turner [7] argue that it's better to start with a relatively minimal set of practices and only adding extra ones when they can be clearly justified by a cost-benefit or risk analysis.

4.2 Risk based

Boehm and Turner [7, 8, 9] proposed a method for creating a hybrid Agile-Waterfall methodology based on risk management. Organizations should rate a project's risks to determine which aspects of the software development process are better suited for a plan-based approach and which aspects can benefit from an agile way of working. Method selection is based on 5 dimensions:

- **Personnel**: How knowledgeable and collaborative are the developers?
- **Dynamism**: How stable or dynamic is the environment? Will there be a lot of requirements changes?

- **Culture**: Does the organization have a culture where people feel comfortable and empowered by having a lot of freedom or would they rather have their roles defined by clear policies and procedures? The challenges that the entrenched culture poses to agile implementation and improvement are detailed in section 3.3 on page 10.
- **Size**: How many developers are in a team and how big is the software product? The risks related to team size are described in section 3.1 on page 9. The challenges that large software products provide to the team are described in section 3.4 on page 10.
- **Criticality**: Wat is the impact of defects? Will bugs in the software cause loss of large amounts of money or even lives?

Depending on these factors, organizations can choose between plan-based- and Agile practices to create their own hybrid methodology, fitted to their organization.

The risk-based method does not explicitly say which software development practice corresponds to which score in a specific dimension and measures only a few of the many possible obstacles to agile improvement.

4.3 Cost/Benefit analysis

The cost/benefit analysis methodology of creating an agile-waterfall hybrid is similar to the risk-based method, using cost/benefit considerations instead of risk analysis. Austin and Devin [5] propose comparing the benefits and costs of specific agile practices with their equivalent practices in other agile methodologies and traditional methodologies such as waterfall to find the practice with the highest benefit relative to its costs.

Chapter 5

Agile improvement models

Large organizations need to be able to determine the agility of their hybrid methodology to find out what can be improved.

The goal of agile improvement- or maturity models is to describe evolutionary stages in the agile adoption process with which organizations cannot only assess their current maturity level, but also gain insight in *what's next* in the journey to more agility and discipline. In these models and frameworks, increasing maturity means the progression towards more agile practices.

Thus, for a maturity model to be useful for agile improvement, organizations have to be able to do two things:

1. determine the current state of the organization's software development processes in the model
2. identify actions that can be taken to *improve* the organization's software development processes relative to the model

It's important that the model not only measures the current state of the team, but also measures the environment in which it has to operate. The lack of customer- or management buy-in can severely hamper the agility of a software development team.

In this chapter, I will discuss a few proposed agile maturity/improvement models with respect to these two requirements. I will begin by discussing the 4-dimensional analytical tool in section 5.1. The agile adoption and improvement model -which uses the 4-dimensional analytical tool for its measurements- is analysed in section 5.2. In section 5.3, the agile maturity map will be described. Finally, in section 5.4 I will discuss the Agile Adoption Framework, which combines ideas from some of the previously mentioned agile maturity models.

I will also determine the usefulness of these models for large organizations with a hybrid software development methodology.

5.1 4-Dimensional Analytical Tool

The 4-Dimensional Analytical Tool (4-DAT) [35] is an analytical tool to measure the agility and adoptability of agile software development methods in practice. It is meant to be used by organizations to make decisions about the selection or adoption of agile methods.

4-DAT facilitates the analysis and comparison of agile methods from four perspectives:

1. **Method scope characterization**: this first dimension provides a set of key scope items on which methods can be compared at a high level. These items pertain to various environmental factors in which the methodology has to work, such as business culture, project- and team sizes, styles of development and coding, technology- and physical environments.
2. **Agility characterization**: the second dimension provides a set of agility features with an agility measurement approach. Combined, these can be used to check the agility of methods at both a process level and a method practices level. This is the only dimension in the 4-Dimensional Analytical Tool that is quantitative.
3. **Agile values characterization**: in the third dimension, methods are compared by their support for the Agile values described in the Agile Manifesto (see page 8). Maturity in this dimension is determined by looking for practices that support the values described in the Agile Manifesto, as well as practices to keep the process agile and cost effective. In practice, hybrid methodologies are usually not completely compatible with the values from the Agile Manifesto, since they incorporate several processes from a more plan-based approach.
4. **Software process characterization**: the last dimension examines the practices that support product engineering processes and process management processes. This dimension is basically a very simple version of CMMI (see page 26).

5.2 Agile Adoption and Improvement Model

The Agile Adoption and Improvement Model (AAIM) [36] is a model for the adoption, assessment and improvement of agile software development processes. It can be used as a gradual road map for the adoption of an agile approach so that the required agile level can be achieved and improved over a period of time. Contrary to the Agile Maturity Map (see page 19), in the Agile Adoption and Improvement Model each level contains specific agile practices that need to be implemented successfully to achieve that level.

The AAIM consists of 6 maturity levels, ordered in three agile blocks, from basic to advanced. At each block the agility measurement modelling approach of the 4-Dimensional Analytical Tool (see section 5.1 on page 16) is used to measure the degree of agility of the agile processes.

Agile Block: Agile-Prompt

Agile-Prompt is a starting point for basic Agile process practices in an organization.

1. **Agile Infancy**: At this level, only the basic Agile properties (speed, flexibility and responsiveness) in a software development process are established in practice. Improvement is focused on implementing iterative and incremental development and preparing the team and environment for further changes in the software development processes.

Agile Block: Agile-Crux

The Agile-Crux block consists of the core of the AAIM levels and is focused on the establishment of key agile practices and properties in a software process.

What's immediately noticeable is that the levels of the Agile-Crux block can be easily mapped to the points from the Agile Manifesto (see page 8).

2. **Agile Initial**: At this level the organization should focus on enabling communication and collaboration with developers, stakeholders and customers. This corresponds with the "Customer collaboration over contract negotiation" value from the Agile Manifesto.
3. **Agile Realization**: This level emphasizes the production of software over comprehensive documentation. This is almost literally the value "Working software over comprehensive documentation" from the Agile Manifesto.
4. **Agile Value**: At this point, the practices are established and focused on developers and customers without ignoring tools and processes. This level corresponds to the agile value "Individuals and interactions over processes and tools" of the Agile Manifesto.

Agile Block: Agile-Apex

In this block, the focus is on further reducing production costs while maintaining or improving the production quality.

5. **Agile Smart**: This level focuses on the establishment of a learning environment. Various agile methodologies provide guidance on how to integrate learning with the

software development process. Scrum does suggest implementing Communities of Practice and some organizations have had a lot of success with that [21].

6. **Agile Process**: At this level, the practices are focused on the establishment of quality production with minimal resources and within a minimum time frame.

5.3 The Agile Maturity Map

The Agile Maturity Map [34] is a way of thinking about Agile adoption in terms of goals for Agile software development teams to help them assess the current maturity of their Agile processes and to provide them with a clear roadmap for implementing and improving Agile practices.

The Agile Maturity Map consists of the following five maturity levels:

1. **Awareness**: the team understands the improvement goals and their value
2. **Transformation**: the team demonstrates commitment towards achieving the improvement goals
3. **Breakthrough**: the team now consistently uses Agile practices that satisfy the goals
4. **Optimizing**: the team makes continuous improvements in the goal area
5. **Mentoring**: the team is mentoring other teams in the goal area

While the Agile Maturity Map does not explicitly name the Agile practices belonging to each maturity level, it does provide five categories in which the use of Agile practices can be improved:

- **Acceptance Criteria**: increasing the quantity and quality of information communicated to the developers just in time for them to use that information in development
- **Green-Bar Tests and Builds**: automated development builds and tests
- **Iterative Planning**: continuous planning
- **Learning and Adapting**: focus on improving skills and learning from doing
- **Engineering Excellence**: implementing or improving practices that improve the quality of the software

Each Agile software development team in the organization is tasked with assessing their maturity level for each of these categories and creating user stories based on these categories which each present an improvement in the use of Agile practices within the team.

Letting the software development teams decide on the improvement goal user stories helps the teams overcome their resistance to change and lets them implement the Check- and Act parts of the Deming cycle for continuous improvement on which Agile software development is based. However, many improvements in Agile practices are dependent on environmental factors, management or customers. The implementation of these agile practices or concepts requires organizational changes which in large organizations are hard to implement from the bottom up. However, since agile is a collaboration-oriented approach to software development, grassroots level buy-in is very important [39]. The Agile Maturity Map could therefore be used as a way to introduce new agile practices and concepts to existing agile teams.

5.4 Agile Adoption Framework

The Agile Adoption Framework [38] combines various aspects of the previously discussed maturity models and is intended as a structured process to guide organizations in adopting agile practices. As a part of this framework, the Agile Measurement Index contains clear indicators [50] for determining the agility of a specific software development process. The other part of the Agile Adoption Framework is a four stage plan to adopt additional agile practices. This plan gives organizations a clearly defined method of introducing new agile practices to teams to improve agility.

The five agility levels in the Agile Measurement Index are inspired from the core agile values and beliefs as

1. **Collaborative**: This level focuses on making the software development process more collaborative through enhancing communication and cooperation between stakeholders. Communication and collaboration lies at the foundation of agile software development and as such is represented by the "Individuals and interactions over processes and tools" and the "Customer collaboration over contract negotiation" values of the Agile manifesto (see page 8) .
2. **Evolutionary**: Evolutionary software development deals with the early and continuous delivery of software. Delivering software early and often is an important aspect of all agile methodologies as it corresponds with the "Working software over comprehensive documentation" value of the agile manifesto (see page 8).
3. **Effective**: This level improves the development of high-quality working software in an efficient and effective manner.
4. **Adaptive**: The objective of this level is to make the software development process more responsive to change through multiple levels of feedback. This is related to the agile value "Responding to change over following a plan" from the agile manifesto (see page 8).
5. **Ambient**: This agility level focuses on establishing the ideal agile environment, a vibrant environment needed to sustain and foster agility in an organization.

The following five dimensions are a distillation of the twelve principles from the agile manifesto (see page 8): Each dimension contains a number of agile concepts and practices, essential characteristics that must be included in the software development process before that process can be considered agile.

- **Embrace change to deliver customer value**: Agile methodologies measure the success of a software development effort on the extent to which it helps deliver customer value. Since some requirements that realize additional customer value can not be known beforehand, an attitude of embracing and welcoming change should be maintained throughout the software development process. This is reflected in the value of "Responding to change over following a plan" the agile manifesto (see page 8).
  - *Reflect and tune process*: Reflecting about the process and tuning it are the check- and act steps of the Deming cycle and as such an important part of any agile methodology. This agile practice concerns whether management and de-

velopers are willing and able to reflect and tune the process after every iteration and release [50].

- *Evolutionary requirements*: Instead of gathering all requirements upfront and refusing changes when implementing them, agile teams try to decide on requirements and features as late as possible. This agile concept deals with whether managers and developers can deal with the uncertainty associated with deciding on features and requirements as late as possible and whether managers and developers are willing to accept changes from the customer [50].
- *Client driven iterations*: Whether or not the customer has the power to dictate the scope of the iterations [50].
- *Continuous customer satisfaction feedback*: Whether the customer is encouraged continually give feedback/criticism and rethink their requirements during the development process [50].
- *Low process ceremony*: Process ceremony is the level of paperwork involved with a process. It is important to keep the amount of process ceremony low to maintain agility. Large organizations may need to maintain a high process ceremony due to audits or regulations (see section 3.6 on page 11). This agile concept covers whether or not developers are trusted to make decisions on their own without explicit management approval [50].

- **Plan and deliver software frequently**: Delivering working software early and frequently is reflected in the agile value "Working software over comprehensive documentation" from the agile manifesto (see page 8). This provides the customer with a functional piece of the product to review and provide feedback on, which in turn is essential for planning the scope and direction of upcoming iterations.
  - *Collaborative planning*: Whether or not customers, developers and management plan together [11, 50].
  - *Continuous delivery*: Incremental and iterative development [22] is an important part of any agile methodology. It means that the team has to deliver a working product after very iteration. This agile practice deals with whether or not management and developers can handle and are willing to use an incremental and iterative development approach [50].
  - *Planning at different levels*: Whether or not management is willing to commit to the process of continuously planning instead of developing a one-time plan upfront [50].
  - *Risk driven iterations*: Whether the developers and managers agree to have risks drive the scope of each iteration [50].
  - *Plan features not tasks*: In agile methodologies, planning implemented through the product backlog, a list of features that need to be implemented. This forces the agile team to think about the product at the feature level before separating the features in to tasks for the sprint backlog. This agile concept deals with whether the high-level agile planning is based on features instead of tasks.
  - *Maintain a list of features and their status (backlog)*: Whether or not management is willing to maintain an up-to-date list of all the remaining features for the

project and their status [50].

  - *Smaller and more frequent releases*: Smaller and more frequent releases are an important part of any agile software development methodology since they give the customer quicker feedback, allowing him to intervene early in case requirements are implemented incorrectly. This agile concept deals with whether the team delivers a fully functional release after every iteration [50].
  - *Adaptive planning*: Adaptive planning is part of the act step from the Deming cycle and allows planning for the next iteration to be based on the customer's feedback from the current iteration. This agile practice deals with whether or not management is willing to plan immediately before the iteration instead of earlier to allow the customer's feedback to be incorporated into the planning [50].
  - *Agile project estimation*: Whether or not developers are estimating the effort and duration of various user stories themselves [50].

- **Human centric**: The reliance on people and the interactions between them is a fundamental part of all agile methodologies [11].
  - *Collaborative teams*: Whether or not people are willing to work in teams and help others [50].
  - *Empowered and motivated teams*: This agile concept covers whether or not management has empowered teams with decision making authority and whether or not team members are motivated [50, 23].
  - *Self organizing teams*: This agile concept is about whether management agrees to have self-organizing teams and employees feel comfortable working in self-organizing teams [50].
  - *Frequent face-to-face communication*: Frequent face-to-face communication is one of the cornerstones of any agile methodology. This agile concept involves whether frequent face-to-face communication between team members is achievable [11, 50, 23].
  - *Ideal agile physical setup*: Whether all the developers are in a common room, furnished to facilitate the agile process (i.e. whiteboards) [50, 23].

- **Technical excellence**: High-quality code is essential in an agile, fast-paced development environment.
  - *Coding standards*: Whether or not coding standards exist and developers see the benefit and are willing to apply them [50].
  - *Knowledge sharing tools*: Whether or not knowledge sharing tools, such as wikis are available and are used by the whole team [50].
  - *Task volunteering*: Whether employees volunteer for tasks instead of being assigned to them [50].
  - *Software configuration management*: Whether or not the organization has tools for software configuration management [50].

- *Tracking iteration progress*: Whether or not a mechanism exists to monitor the iteration progress [50].
- *No big design up front*: Whether design is a continuous process or done once at the beginning of the development process [50].
- *Continuous integration*: Continuous integration [45] means that the product is automatically integrated and tested every time new code is added, which allows developers to find defects early. This agile practice is about whether or not the developers are willing and able to use continuous integration [50].
- *Continuous improvement (refactoring)*: Refactoring is the discipline of improving the quality of code by modifying the internal structure of existing code without changing the external behavior. This agile practice deals with whether or not developers agree to adopt an approach of continuous software improvement and whether or not they are competent enough to refactor code without jeopardizing the quality of functionality of the code [50].
- *Unit tests*: Unit tests are tests of small individual parts of code. In agile software development, unit tests are very important, since they ensure that functionality doesn't break when refactoring code. This agile practice is about whether developers are willing to write unit tests during development and have the competence and previous experience to do so [50].
- *30% of level 2 and level 3 people*: Level 2 people are at the point of learning to break the rules, i.e. they have a level of software development method understanding and use which enables them to tailor a method to fit a precedented new situation by using heuristics [7]. In addition, level 3 people are able to revise a method (i.e. break its rules) when they encounter an unprecedented new situation [7]. While level 2 people can function well when managing a small, precedented agile or disciplined project, they need the guidance from level 3 people on large, unprecedented projects [7].
- *Daily progress tracking meetings*: Whether or not management and developers are willing to meet daily to discuss the progress of the project [23, 50].
- *Agile documentation*: Since agile methodologies value "working software over comprehensive documentation" (see page 8), an agile approach to documentation entails writing minimal or "just enough" documentation. In large organizations, this might not be possible due to regulatory requirements (see section 3.6 on page 11). This agile concept involves whether developers are willing and able to take an agile approach to documentation [50].
- *User stories*: User stories are an elicitation method/form for high level requirements which in a few sentences capture the *who*, *what* and *why* of a requirement [9]. Regulatory requirements for the elicitation of requirements may prevent teams from using user stories. This agile practice is about whether management and developers are willing and able to use user stories [50].
- *Test-driven development*: In test-driven development [13], a developer writes a failing automated test case that defines the requirement before implementing code to pass the test. This agile practice deals with whether or not the developers are motivated and willing to apply test driven development and whether

or not management will encourage test-driven development and tolerate the learning curve [50].

- *Pair programming*: Pair programming [43, 14, 44] is a technique in which two programmers work on one workstation. This allows one programmer to review the code and design of the other programmer while the code is being typed in. Pair programming is an important element of extreme programming but can also be used within other software development methodologies. This agile practice deals with whether management can see the benefit of pair programming and whether or not developers are willing to try pair programming.
- *No/minimal number of level -1 or 1b people on team*: Level -1 people are unable or unwilling to collaborate or follow shared methods [7]. Since collaboration and following shared methods are very important for the effectiveness of an agile team, level -1 people need to be identified to be able to get rid of this counterproductive mindset. Level 1 people are are still learning and need procedures, heuristics or a set of rules to follow. Level 1b people can perform well in straightforward software development in a stable situation but are likely to slow down an agile team when dealing with rapid change [7]. This makes level 1b people ideal for work in a traditional software development environment but counterproductive in an agile team.

- **Customer collaboration** The agile manifesto value "Customer collaboration over contract negotiation" emphasizes the importance of significant and frequent interaction between customers, developers and all other stakeholders of the project to ensure that the agile project satisfies the requirements of the customer.
  - *Customer commitment to work with developing team*: Whether the customer is willing to dedicate time to take an active role in the project [50, 23].
  - *Customer contract reflective of evolutionary development*: Whether or not the contract with the customer reflects that the system is developed in an iterative and incremental fashion [50].
  - *Customer immediately accessible*: Whether a knowledgeable customer representative that is authorized to make decisions on the spot regarding the product is immediately accessible to the development team if needed [50].
  - *Customer contract revolves around commitment of collaboration*: Whether instead of features, the customer contract revolves around commitment of interaction and collaboration [50].
  - *Frequent face-to-face interaction between developers & users (collocated)*: Whether the frequent face-to-face interaction between developers and customer is achievable [50, 23].

The Agile Measurement Index contains indicators for every maturity level of a dimension [50]. This sets this maturity model apart from the previously mentioned models and gives organizations an easy way to objectively measure the agility of software development processes.

	Agile Principles				
	Embrace Change to Deliver Customer Value	*Plan and Deliver Software Frequently*	*Human Centric*	*Technical Excellence*	*Customer Collaboration*
Level 1: **Collaborative** *Enhancing communication and collaboration*	Reflect and tune process	Collaborative planning	Collaborative teams Empowered and motivated teams	Coding standards Knowledge sharing tools Task volunteering	Customer commitment to work with developing team
Level 2: **Evolutionary** *Delivering software early and continuously*	Evolutionary requirements	Continuous delivery Planning at different levels		Software configuration management Tracking iteration progress No big design up front	Customer contract reflective of evolutionary development
Level 3: **Effective** *Developing high quality, working software in an efficient and effective manner*		Risk driven iterations Plan features not tasks Maintain a list of all features and their status (backlog)	Self organizing teams Frequent face-to-face communication	Continuous integration Continuous improvement (refactoring) Unit tests 30% of level 2 and level 3 people	
Level 4: **Adaptive** *Responding to change through multiple levels of feedback*	Client driven iterations Continuous customer satisfaction feedback	Smaller and more frequent releases Adaptive planning		Daily progress tracking meetings Agile documentation User stories	Customer immediately accessible Customer contract revolves around commitment of collaboration
Level 5: **Ambient** *Establishing a vibrant environment to sustain agility*	Low process ceremony	Agile project estimation	Ideal agile physical setup	Test-driven development Pair programming No/minimal number of level -1 or 1b people on team	Frequent face-to-face interaction between developers & users (collocated)

Table 5.1: The 5 Levels of agility populated with agile practices and concepts (from [38])

Chapter 6

Agile and CMMI

The Capability Maturity Model Integration (CMMI) for Development [51] is a model for software development process management. CMMI is comprised of a set of process areas, organized into maturity levels. Each process area consists of a number of goals An organization has to implement practices to satisfy the goals of all process areas of a specific maturity level in order to claim its software development processes are at that maturity level. The CMMI document contains example practices for each goal but since these practices are based on traditional plan-based software development they are often in conflict with the values of agile software development.

Just like agile software development, the CMMI is based on the Plan-Do-Check-Act cycle popularized by Deming. Using this cycle, CMMI can be used as a model for continuous improvement of software development processes. However, CMMI is not intended to be used as a standard, and not all organizations will benefit from reaching the highest CMMI maturity level [47]. Instead, organizations should consider whether there is business value in satisfying the goals of a specific process area.

Since both Agile and CMMI are based on the Plan-Do-Check-Act cycle and CMMI and Agile share many of the same values [47], we can use this information to map agile practices to CMMI process areas [3].

Where CMMI describes *what* should be done, the various agile methodologies describe *how* this could be achieved[47]. For each CMMI goal there are multiple possible implementations from agile methodologies such as Extreme Programming and Scrum, but also from plan-based approaches such as waterfall. Since all these practices have their own advantages and trade-offs, organizations need to find a combination that fits their situation.

To determine which approach is most suitable, organizations first need to determine their situation with respect to the obstacles to agile improvement [38, 7]. With that information they can choose a practice that matches each factor to fill in each CMMI goal. A few common approaches to choosing these practices can be found in chapter 4 on page 13. By mixing and matching CMMI goals with a combination of agile and waterfall practices, organizations can choose their own trade-off between agility and discipline.

To improve their methodology, they can try to eliminate roadblocks, which allows the organization to implement more agile practices and work more efficiently. To help guide this improvement process, one of the agile improvement models (described in chapter 5 on

page 15) can be used.

The check- and act steps of the Deming cycle can be used to continuously improve the agile software development process by allowing experimentation with various possible implementations of the CMMI goals. This means we can use CMMI roadmaps to guide process improvement while using an Agile improvement roadmap to continually improve the agility of the software development practices implementing the CMMI roadmap.

On the next few pages, I will describe an agile implementation of CMMI process areas, showing how agile principles relate to IT- and business processes. For each CMMI process area, relevant agile practices and concepts are *emphasized*. In order to meet the requirements of a CMMI process area, an organization must have practices implementing the goals of that process area. However, these practices do not need to be the ones described in the CMMI specification as the organization is free to propose alternative practices and appropriate evidence to meet a CMMI appraisal [3, 47].

6.1 Causal analysis and resolution

This process area deals with the identification of problematic or exceptional process performance and their consequences for process improvement. This corresponds strongly with the check and act steps from the Deming cycle.

The analysis of past performance to implement continual improvement is a crucial part of any Agile methodology [43]. CMMI mentions that everyone should be empowered to propose improvements, which is in line with the Agile way of thinking about optimizations. This is reflected in the *reflect and tune process* practice from the agile adoption framework (see page 20). In Scrum for example, this is implemented as the Sprint retrospective. After each sprint, all team members reflect on the questions of what went well during the past sprint and what could be improved.

6.2 Configuration management

Software configuration management (see page 22) involves establishing and maintaining the integrity of work products. In agile projects, such configuration items are code, design, tests, requirements, drawing and specifications [14]. An example of a software configuration management tool is version control software such as Subversion or Git.

While configuration management is not specifically called out in any agile methodology, it is even more important in Agile than in more traditional software development methodologies due to the frequently changing environment.

To reduce the amount of work this process area takes for Agile teams, large parts of configuration management can be automated, especially when combining it with *continuous integration* (see page 23) [14].

6.3 Decision analysis and resolution

This process area involves the analysis of possible decisions using a formal evaluation process that evaluates identified alternatives against established criteria.

The focus on establishing specific processes for team functions in the practices of this process area is in conflict with the spirit of “Individuals and interactions over processes and tools” from the Agile Manifesto (see page 8). Decision making in agile projects is more complex than in traditional development environments since in agile projects decision making is a collaborative process.

Furthermore, agile projects should be able to adapt quickly to the situation rather than be bound to preconceived criteria, a strict evaluation of alternatives or decision analysis process [43].

In large organizations, having a formal process for taking difficult decisions can be very beneficial, especially if the issue being decided on can pose significant risks for the project or organization. The risk-based approach to establishing an Agile methodology (see section 4.2 on page 13) can help an organization decide whether a more traditional implementation of this process area’s goals is needed.

6.4 Integrated project management

The purpose of integrated project management is to establish and manage the project and the involvement of relevant stakeholders.

CMMI explicitly mentions the coordination and collaboration with relevant stakeholders. In an agile software development process, the continuing involvement of stakeholders and *customer commitment to work with the developing team* (see page 24) is very important. This is why the *customer contract should be reflective of evolutionary development* (see page 24) and *revolve around commitment of collaboration* (see page 24). The integrated project management process area deals with processes that make sure that the concerns of relevant stakeholders are addressed. Since Agile methodologies are focused on responding to change, it is easier to respond to stakeholders' concerns than in more traditional development methods. However this requires stakeholders to actively participate in the development process [33]. *Frequent face-to-face interaction between developers & users* (see page 24) -especially when collocated- ensures that the *customer is immediately accessible* (see page 24) to validate the product, clarify requirements and answer questions.

Agile software development methodologies embrace the empowerment of the software development team, which changes the role of the project manager from planning and controlling to facilitating the collaboration of team members [33].

6.5 Measurement and analysis

The purpose of this process area is to make sure that sufficient measurement capability is available to make informed management decisions. Measurement and analysis are the check part of the Deming cycle and as such well represented in both CMMI and Agile. Agile software methodologies continually measure both the product and the process used to create it to allow improvement.

Hartmann and Dymond [17] point out that traditional measuring methods such as used in the waterfall method can hamper performance in agile development. They propose that the key metric with which organizations measure agile performance is the amount of business value created for the organization.

Another useful and often used diagnostic is velocity, the amount of story points an agile team can process in a single iteration.

Another thing to measure is the agility and maturity of the software development process itself. This is described in more detail on page 15.

6.6 Organizational process definition

The purpose of the organizational process definition process area is to establish and maintain organizational process assets, work environment standards, and rules and guidelines for teams.

Some people [43] consider an agile methodology as being essentially an informal process asset repository. For establishing an agile process asset repository, large organizations can use one of the approaches described in chapter 4 on page 13.

The work environment of agile teams needs to facilitate various agile processes. In the *ideal agile physical setup* (see page 22) all the developers are in a common room, furnished with e.g. whiteboards to facilitate the agile process.

The rules and guidelines for agile teams differ significantly from those applicable to teams in more traditional software development environments. *Self organizing teams* (see page 22) can organize again and again in various configurations to meet challenges as they arise [11].

In a highly dynamic environment, team members in an agile team need to be able to respond to change quickly by being empowered to make some decisions without seeking explicit managerial approval. Therefore, *empowered and motivated team members* (see page 22 are important underlying concepts in the agile manifesto (see page 8.

The agile concept *collaborative teams* (see page 22) states that even more so than in traditional software development, agile practices require developers to work in teams and help each other.

6.7 Organizational process focus

This process area deals with planning, implementing and deploying organizational process improvements based on an analysis of the strengths, weaknesses, opportunities and threats of the organization's processes and process assets. These process improvements improvements occur in the context of the organization's needs and should be used to address organizational objectives.

The traditional infrastructure implied by the goals of this process area conflict with a more agile way of working [43]. Within a team or a project, Agile processes evolve under their own experience as part of the check- and act steps of the Deming cycle. An Agile example of soliciting input for this analysis is the sprint retrospective. However, in large organizations it can become challenging to make sure these improvements are shared with all other projects. This problem can be solved by establishing organization-wide repositories containing best practices of previous projects [14] or creating communities of practice [21].

6.8 Organizational performance management

The organizational performance management process area is another manifestation of the check- and act steps from the Deming cycle, in this case specifically for the organization's performance to meet its business objectives. To that end, organizations need to iteratively analyze performance data and implement improvements to close the performance gaps.

Continually improving processes based on the performance to meet organizational goals is an important aspect of most Agile methodologies and the focus of the *reflect and tune process* agile concept (see page 20. For example, in Scrum the sprint retrospective provides an opportunity for team members to identify practices that could be improved in the next sprint.

CMMI Level 5 advocates that the organization looks at where they are overdoing processes like documentation and then trim those back [27]. Examples of such agile software development process improvements are *lowering the amount of process ceremony* (see page

21) and taking a more *agile approach to documentation* (see page 23), so that only "just enough" documentation is written.

6.9 Organizational process performance

Organizational process performance deals with establishing and maintaining process performance baselines and models needed for quantitative project management (see section 6.15 on page 33).

The idea of measuring a process and maintaining baselines and models conflicts with the spirit of "Individuals and interactions over processes and tools" from the Agile Manifesto (see page 8). However, Scrum has metrics which could be characterized as process metrics [43] and can be used by large organizations to implement the goal of this process area.

6.10 Organizational training

The purpose of this process area is to develop the knowledge and skills of team members so they can perform their roles effectively and more efficiently.

Many authors agree that sufficiently training people is critical to agile project performance [27, 33, 6, 7]. The agile concepts of *30% level 2 and level 3 people* (see page 23) and *no/minimal level 1 or 1b people on team* (see page 24) ensure that the team members are sufficiently qualified and capable to work in an agile way.

Since Agile methodologies value individuals and interactions over processes and comprehensive documentation, organizational knowledge sharing should reflect this. Therefore, an Agile implementation of organizational training should focus on training and mentoring [43]. *Frequent face-to-face communication* (see page 22) is essential for sharing knowledge between developers. This requires an *ideal agile physical setup* (see page 22) in which developers are located closely together in a common room.

Agile practices for sharing tacit knowledge are for example communities of practice [21] for organization-wide knowledge sharing and *pair programming* (see page 24) for sharing code-related knowledge.

Knowledge sharing tools (see page 22) such as wikis can help teams codify and share knowledge. By adopting these technologies organizations can overcome knowledge sharing challenges created by geographical distribution (see section 3.2 on page 9).

6.11 Product integration

Product integration is the assembly of the product from the product components and consequent delivery.

In software development projects following traditional software development methodologies such as the waterfall model, the product is integrated and shipped at the end of development. This is in stark contrast with agile methodologies, which features *smaller and more frequent releases* (see page 22). In agile projects, product integration is a frequent -at least daily- activity and the agile concept of *continuous delivery* (see page 21)

means that working software is shipped early and often. In extreme programming for example, *continuous integration* (see page 23) is an important practice to find defects early. Research [52] shows that while the agile discipline of always checking in working code to make sure the automated build doesn't break is not easy to adopt, it will quickly pay off since it catches defects early and often. Since integration steps are performed very often, a thorough preparation of the continuous integration environment is critical [14].

To make sure interfaces between different components match, *test-driven development* (see page 23) can be used. In this practice, tests are written before implementation to describe how the interfaces should behave but without focussing on implementation details. When combining *test-driven development* with *continuous integration*, the continuously decreasing amount of failing tests can provide a powerful metric on the amount of implementation work that remains to be done.

6.12 Project monitoring and control

This process area involves providing an understanding of project progress so that corrective actions can be taken when the project gets behind schedule.

While monitoring and taking corrective actions are crucial to the progress of any project, the practices CMMI suggests for doing so seem very oriented to traditional project management [43]. However, agile methodologies provide excellent support for implementing this process area.

For example, Scrum uses burn-down or burn-up charts to show how much functionality is left to complete, while the project task board is used to track the progress of individual user stories. During the iteration, there are *daily progress tracking meetings* (see page 23) such as Scrum stand-ups to allow developers and management assess the progress of the individual team members and overall progress of the project. Due to the strict system of short iterations and the regular commitments to the plan, monitoring an agile project against the baseline is relatively easy [14].

Another relevant agile practice is *tracking iteration progress* through working software (see page 23). Most agile methodologies state that the ultimate measure of project progress should be whether working software is released with improvements for the end-user.

Stakeholder involvement can be crucial to the success of agile projects when the collaboration with the customer and potential end users replaces detailed requirements. In this case, stakeholder involvement in project activities should be monitored.

6.13 Project planning

The purpose of project planning is establishing and maintaining plans defining project tasks.

In agile methodologies project planning is an activity that is performed more frequently than in traditional software development environments due to the short incremental iterations. In addition, most agile methodologies require a level of start-up planning and risk assessment [43].

It is easy to interpret the specific CMMI practices in this process area as requirements for big planning up front, something that is in direct opposition to the agile way of working. However, in large organizations some degree of planning up front is usually required to

coordinate separate teams and departments. In order to allow agile teams some degree of adaptability to environmental factors, up front planning should be kept to a bare minimum.

Planning at different levels (see page 21) means that instead of developing a detailed plan up front, management commits to a process of continuous planning. Anderson [3] proposes that organizations provide a loose project plan to approximate the scope of a project and lay out a plan for a series of iterations that provides an outline of what will be developed in each one. The agile concept *plan features not tasks* (see page 21) states that the low-level planning should be done by *maintaining a list of features and their status* (see page 21) and letting *developers volunteer for tasks* (see page 22) instead of planning the tasks explicitly.

In addition, the agile practice of *collaborative planning* (see page 21) means that managers, developers and the customer plan together.

Since scrum is focused on project management, it specifically has a number of practices which can easily be used to replace the waterfall-based practices from CMMI for project planning [28]. The concept of *agile project estimation* (see page 22) means that team members themselves estimate the effort and duration of the user stories. An example of this is planning poker [30], a technique based on consensus to estimate the amount of time and effort a user story should take. In Scrum, story points are the preferred way to estimate how much time the implementation of a requirement should take. Estimates for stories and tasks are established and can be corrected during the project. The precision of the estimates is increased through a short planning horizon due to short iterations [14].

As part of the act step from the Deming cycle, the agile concept of *adaptive planning* (see page 22) states that planning for the next iteration should be based on the customer's feedback from the current iteration.

6.14 Process and product quality assurance

The purpose of this process area is to objectively evaluate processes and associated work products to gain insight in their quality.

Since the emphasis of this process area lies on processes and work products instead of the end product, the process and product quality assurance process area is not completely applicable to agile projects [43]. However, objectively evaluating the maturity and agility of the software development process can reveal new insights on how to improve it.

In agile methodologies there is no explicit evaluation of processes, work products and services against the applicable process descriptions [14]. In Scrum, some basic quality assurance activities are performed by the Scrum Master when he checks that the Scrum process is being followed. In XP, the main method of controlling that the method is applied in the right way is the XP coach [14]. Using this agile CMMI framework to measure agility and maturity is another way to evaluate the software development process.

6.15 Quantitative project management

The purpose of this process area is to achieve the project's established quality and process performance objectives by monitoring performance and quality and addressing deficiencies in achieving the quality and performance objectives of the project.

While measuring and analysing performance and quality is an important part of agile software development (see section 6.5 on page 29), Agile methodologies do not contain specific practices for implementing quantitative project management since statistical methods focus on defined processes instead of individuals [14] which is in conflict with the spirit of the agile manifesto (see page 8). However, the agile manifesto does not preclude implementing this process area, as long as the resulting processes are kept light-weight [43].

Due to the short iterations in agile software development, project performance issues quickly become apparent when team members fail to implement the requested functionality in the course of one iteration. This allows project managers to address project performance issues after each iteration.

6.16 Requirements development

Requirements development is the process of eliciting, analyzing and establishing requirements.

This CMMI process area supports the Agile concepts of close customer relationships, customer-based requirements elicitation and stakeholder involvement [43].

In plan-based software development methodologies such as waterfall, requirements are elicited at the start of the project and are relatively stable. This is in stark contrast with most agile methodologies, where the agile concept of *evolutionary requirements* (see page 21) means requirements are defined incrementally, rather than trying to get them all in advance.

In agile projects requirements are written down as a few sentences in a *user story* (see page 23) which concisely captures the who, what and why of a requirement [9]. In extreme programming, test-driven development (see page 23) means that requirements are communicated as automated acceptance tests. In both cases however, the requirements specification remains quite vague [14].

The requirements are finally analysed and specified when a developer starts working on a specific task [14].

Which way of requirements development is more appropriate for a specific project can be determined using Boehm and Turner's risk-based approach (see page 13).

6.17 Requirements management

The purpose of requirements management is to make sure that the project's plans and products are aligned with the customer's requirements.

While the goal of this process area is completely applicable in an agile project environment, the plan-based practices CMMI suggests are not [43]. Fortunately, several agile methodologies contain practices to implement requirements management. In agile software development *maintaining a list of features and their status* (see page 21) or product backlog plays a huge part in the management of requirements. The Product owner and team jointly review the product backlog to help the team develop an understanding of the requirements. The traceability of requirements is not an explicit goal of most agile methodologies, but is nonetheless supported by artifacts such as stories, tasks and tests [14].

Another important aspect of any agile methodology is the ordering of requirements by priority. This ensures that the requirements with the highest business value are implemented first.

6.18 Risk management

The purpose of risk management is identifying potential issues so they can be prevented or their effect mitigated.

Most agile methodologies are designed to mitigate certain types of risks – particularly those from changing requirements and schedules [43]. For example, the agile mantra of releasing early and often, combined with customer involvement means that the effect of the risk of delivering the wrong product to the customer is relatively small compared to waterfall-based projects.

When using *risk driven iterations* (see page 21), risks for the organization determine the scope of each iteration. If -for example due to regulations- there is a risk for the organization when something is not delivered on time or is of poor quality, it should be prioritized to make sure the organization is not fined or missing out on large opportunities.

Organizations can manage the amount of risk in a project upfront by combining an agile methodology with plan-based practices as described by Boehm and Turner in their risk-based methodology for creating a hybrid software development process (see section 4.2 on page 13).

6.19 Supplier agreement management

The purpose of this process area is to manage the acquisition of products and services from suppliers.

It can be difficult to combine communication with a supplier with the agile way of working in an organization. For example, suppliers often require planning upfront to get artifacts in time. These kind of issues are left largely ignored in agile methodologies [43, 14] so the practices CMMI proposes can be a welcome addition to agile projects in large organizations.

6.20 Technical solution

The purpose of this process area is to select, design and implement solutions to requirements.

In agile development, the concept of *no big design up front* (see page 23) means that an initial design is kept as simple as possible and instead, code is used as a design document [14]. After the initial design, design is carried out iteratively.

Having *coding standards* (see page 22) helps ensure that the created code is of sufficient quality. To improve code quality, *continuous improvements (refactorings)* (see page 23) to the internal structure of the code can be made.

In *test-driven development* (see page 23) implementing the product means developing the minimum viable product to stop the tests from failing.

The CMMI practices in this process area can help agile teams make higher quality implementation decisions by making the rationale for selection and trade-offs more explicit. Whether this is necessary depends on the amount of risk that is involved with these decisions. The risk-based approach to creating an agile methodology can help organizations determine whether implementing the recommended practices from this process area is beneficial for a specific project (see section 4.2 on page 13).

6.21 Validation

The purpose of validation is to demonstrate to the customer that the product will fulfill its intended use. This is consistent with the spirit of customer collaboration in the agile manifesto [43] (see page 8).

In agile projects, the main criterion for validation is acceptance by the customer. Smaller and more frequent releases empower the customer to participate and constantly perform validation [14] through *continuous customer satisfaction feedback* (see page 21). In Scrum, the sprint review meeting after each iteration features a demonstration of the product to the stakeholders, allowing the stakeholders to check whether the developers "built the right thing". This is especially important in agile software development when only high-level requirements are communicated to the developers. The participation of the customer improves the chance that the product will fulfill its intended use.

6.22 Verification

The purpose of verification is to check that the product meets its requirements.

Writing *unit tests* (see page 23) during development can help developers verify that the software works as planned and ensure that refactorings do not break any functionality.

Evidence [13] suggests that the practice of writing tests before implementing can lead to productivity gains and quality improvements. When using *test-driven development* (see page 23), the lack of failing acceptance tests can be used as an indicator that all requirements are sufficiently implemented. Behavior-driven development [48] is an extension of test-driven development with a strong focus on business value and collaboration with various stakeholders. To that end, test cases are written in a natural language. Whatever method of testing is used, is especially important to verify that the implemented business logic is correct since since these mistakes can be costly and it's quite easy to miscommunicate business rules or e.g. forget to implement a specific edge case.

The CMMI verification goal of peer reviews is closely aligned with *pair programming* (see page 24). Various studies [26, 12, 10, 44] have demonstrated that pair programming can result in better design and fewer defects, at the expense of total developer effort. Research [4, 25] shows that pair programming can be especially beneficial for novice programmers. Organizations need to carefully examine the costs and benefits of pair programming for a specific project and team, since pair programming is not always beneficial or effective [16]. Experience [23] shows that the knowledge sharing benefits are greatest when developers regularly switch programming pairs.

	Maturity level 2: **Managed**	Maturity level 3: **Defined**	Maturity level 5: **Optimizing**
Agility level 1: **Collaborative**	Task volunteering Collaborative planning	Collaborative teams Empowered and motivated teams Coding standards Knowledge sharing tools Customer commitment to work with developing team	Reflect and tune process
Agility level 2: **Evolutionary**	Tracking iteration progress Planning at different levels Software configuration management	Continuous delivery Evolutionary requirements No big design up front Customer contract reflective of evolutionary development	
Agility level 3: **Effective**	Maintain a list of all features and their status (backlog) Plan features not tasks	Risk driven iterations Self organizing teams Frequent face-to-face communication Continuous integration Continuous improvement (refactoring) Unit tests 30% of level 2 and level 3 people	
Agility level 4: **Adaptive**	Daily progress tracking meetings	Client driven iterations Continuous customer satisfaction feedback Smaller and more frequent releases Adaptive planning User stories Customer immediately accessible Customer contract revolves around commitment of collaboration	Agile documentation
Agility level 5: **Ambient**	Agile project estimation	Ideal agile physical setup Test-driven development Pair programming No/minimal number of level -1 or 1b people on team Frequent face-to-face interaction between developers & users (collocated)	Low process ceremony

Table 6.1: The 5 Levels of agility and relevant CMMI levels, populated with agile practices and concepts (from [38])

Chapter 7

Applying the Agile CMMI Framework at a large energy company

The company where I held my case study is one the biggest suppliers of energy in the Netherlands.

The energy sector is a very dynamic market in which new chances, challenges or regulations appear very often. For example, green energy, increasing globalization of the energy market, increased competition, more demanding customers and new technological developments pose new challenges to the organization every day. This requires that the organization can rapidly implement changes in its organization and create or change associated software. An Agile software development methodology should be beneficial for software changes in this dynamic environment and as such the organization has started implementing Agile methodologies in their software development process. At this organization, the customer of the agile software development process is usually another department in the same organization.

To measure the presence of the various agile principles and concepts (see page 20) I interviewed employees at various positions within the agile software development process. The entire interviews can be found in appendix B on page 62.

I focused my interviews on the B2C (Business to Consumer) department since they have been developing software using scrum for about 2 years, making it the department with the most mature agile implementation. Within B2C there are a number of projects and corresponding teams. While these teams should be at about the same level of agility, small differences still exist.

In the rest of this chapter, I will describe the results of this series of interviews. In section 7.1 on page 39, I will discuss the results of the interviews in the context of the agile principles from the agile adoption framework (see section 5.4 on page 20). In section 7.2 on page 47 I will examine the agility and maturity of the software development process at the B2C department in the context of the agile CMMI framework.

7.1 Results of the interviews on agile principle level

7.1.1 Results

In this section the agile concepts and practices belonging to the various agile principles are listed along with the level of support found in the interviews with various employees of a large energy company. The interviews can be found in appendix B on page 62 and are annotated with the various agile practices and concepts that are mentioned in the answers to the interview questions, along with whether they provide evidence for a full implementation of the agile practice or concept (+), a partly implemented agile concept or practice (+/-) or lack of an implementation (-), which are worth respectively 1, 0.5 and 0 points in the tables below. Any agile practice or concept with an implementation percentage of 75 percent or higher is counted as present.

Embrace Change to Deliver Customer Value

Level 1

Agile practice or concept	Score	Max	Percentage	Present?
Reflect and tune process	3	3	100%	Yes

Level 2

Agile practice or concept	Score	Max	Percentage	Present?
Evolutionary requirements	4	5	80%	Yes

Level 4

Agile practice or concept	Score	Max	Percentage	Present?
Client driven iterations	4.5	6	75%	Yes
Continuous customer satisfaction feedback	3.5	6	58%	No

Level 5

Agile practice or concept	Score	Max	Percentage	Present?
Low process ceremony	1	3	33%	No

Plan and Deliver Software Frequently

Level 1

Agile practice or concept	Score	Max	Percentage	Present?
Collaborative planning	4.5	6	75%	Yes

Level 2

Agile practice or concept	Score	Max	Percentage	Present?
Continuous delivery	3	3	100%	Yes
Planning at different levels	2	2	100%	Yes

Level 3

Agile practice or concept	Score	Max	Percentage	Present?
Risk driven iterations	1.5	3	50%	No
Plan features not tasks	2	2	100%	Yes
Maintain a list of all features and their status (backlog)	6	6	100%	Yes

Level 4

Agile practice or concept	Score	Max	Percentage	Present?
Smaller and more frequent releases	3	3	100%	Yes
Adaptive planning	6.5	8	81%	Yes

Level 5

Agile practice or concept	Score	Max	Percentage	Present?
Agile project estimation	2.5	3	83%	Yes

Human centric

Level 1

Agile practice or concept	Score	Max	Percentage	Present?
Collaborative teams	3	3	100%	Yes
Empowered and motivated teams	4	6	67%	No

Level 3

Agile practice or concept	Score	Max	Percentage	Present?
Self organizing teams	4	5	80%	Yes
Frequent face-to-face communication	3	3	100%	Yes

Level 5

Agile practice or concept	Score	Max	Percentage	Present?
Ideal agile physical setup	1.5	3	50%	No

Technical Excellence

Level 1

Agile practice or concept	Score	Max	Percentage	Present?
Coding standards	2	3	67%	No
Knowledge sharing tools	2.5	3	83%	Yes
Task volunteering	2.5	3	83%	Yes

Level 2

Agile practice or concept	Score	Max	Percentage	Present?
Software configuration management	2.5	3	83%	Yes
Tracking iteration progress	3	3	100%	Yes
No big design up front	3	3	100%	Yes

Level 3

Agile practice or concept	Score	Max	Percentage	Present?
Continuous integration	1	3	33%	No
Continuous improvement (refactoring)	2	3	67%	No
Unit tests	3	3	100%	Yes
30% of level 2 and level 3 people	1	1	100%	Yes

Level 4

Agile practice or concept	Score	Max	Percentage	Present?
Daily progress tracking meetings	2.5	3	83%	Yes
Agile documentation	3.5	4	88%	Yes
User stories	6.5	7	93%	Yes

Level 5

Agile practice or concept	Score	Max	Percentage	Present?
Test-driven development	0	4	0%	No
Pair programming	0.5	3	17%	No
No/minimal number of level -1 or 1b people on team	3.5	6	58%	No

Customer Collaboration

Level 1

Agile practice or concept	Score	Max	Percentage	Present?
Customer commitment to work with developing team	2.5	8	31%	No

Level 2

Agile practice or concept	Score	Max	Percentage	Present?
Customer contract reflective of evolutionary development	2	2	100%	Yes

Level 4

Agile practice or concept	Score	Max	Percentage	Present?
Customer immediately accessible	4.5	7	67%	No
Customer contract revolves around commitment of collaboration	0.5	6	8%	No

Level 5

Agile practice or concept	Score	Max	Percentage	Present?
Frequent face-to-face interaction between developers & users (collocated)	1	6	17%	No

7.1.2 Level of agility

As can be seen in table 7.1 on page 46, there is a huge discrepancy between the agility levels according to the agile adoption framework (see section 5.4 on page 20) and the percentage of implemented agile concepts or practices. This means that implementing only a few more agile practices or concepts would greatly increase the agility scores. However, some agile concepts -such as empowered teams- can be difficult to implement in a large organization. I have decided to use percentages in figure 7.1 on page 46 since this gives a more faithful representation of the level of implementation of the various agile principles.

7.1.3 Suggestions for improvements

As can be seen in figure 7.1 on page 46, the agile principle which implementation is the most lacking is *Customer Collaboration*. According to the agile adoption framework, the *Customer commitment to work with developing team* concept is a constraining practice [38, 50], meaning it will impede progress on the other agile principles if not implemented. Since this agile principle has the lowest implementation percentage and can impede the agility of the other agile principles, finding a way to increase customer commitment should be the number one priority in the improvement process.

Another customer related practice is *continuous customer satisfaction feedback* which -if implemented- could increase the agility level of the *Embrace Change to Deliver Customer Value* principle from 2 to 4.

On the *Technical Exellence* front, implementing *Coding standards* could raise the score of this agile principle from 0 to 2.

Risk driven iterations is the only agile concept not implemented on the *Plan and Deliver Software Frequently* agile principle. Implementing it would mean an agility level increase from 3 to 5.

These suggestions are described in more detail in section 7.2.

Agile Principle	Agility level	Number of implemented agile practices or concepts	Total number of agile practices or concepts	Percentage implemented
Embrace Change to Deliver Customer Value	2	3	5	60%
Plan and Deliver Software Frequently	2	8	9	89%
Human centric	0	3	5	60%
Technical Excellence	0	10	16	63%
Customer Collaboration	0	1	5	20%

Table 7.1: The level of implementation of the various agile concepts and practices at the B2C department

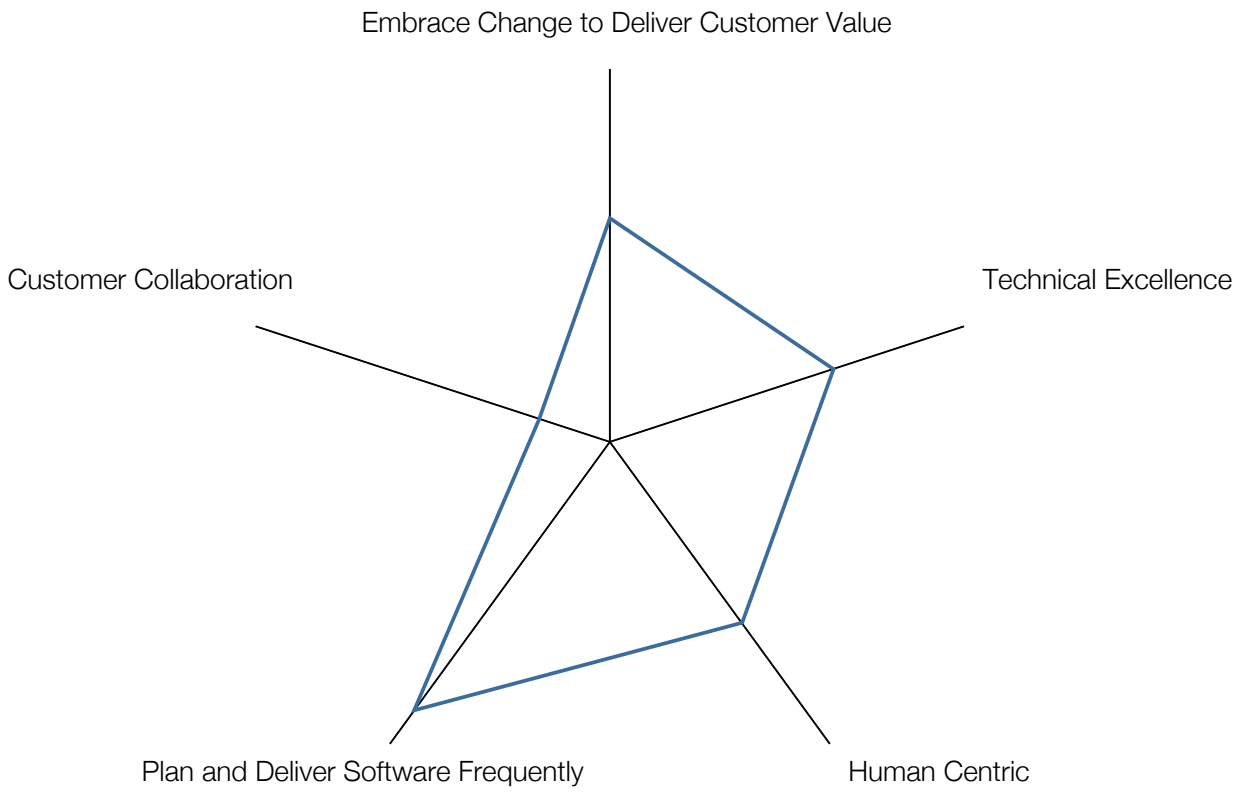

Figure 7.1: The level of implementation of the various agile concepts and practices at the B2C department

7.2 Results of the interviews on CMMI process area level

7.2.1 Results

Causal analysis and resolution

The organization is constantly *reflecting about and tuning the software development process* at B2C. It's important to involve stakeholders in this process to make sure that suggested improvements are feasible and make sure everyone is aware of the new way of working. Currently, this is not always the case (see the answer to question 46 on page 105). While involving lots of stakeholders creates a lot of conflicting interests (see question 3 on page 63), this is the only way to create buy-in for the proposed changes.

Configuration management

Software configuration management is not yet thoroughly performed in all fields (see question 27 on page 70) but overall this process area has been implemented in most fields.

Integrated project management

As can be seen in figure 7.1 on page 46, customer collaboration is the most important aspect to be improved as it's currently hindering agile improvement in other dimensions. The interview answers show that while the customer may be willing to work with the agile teams, in practice it's very difficult for the business to allocate time for collaboration with developers (see the answer to question 41 on page 104).

The specific situation at this company makes it hard to make all business representatives available to the agile teams full-time, since projects often have many different business units as request owners (see the answer to question 45 on page 114 and question 15 on page 90). However, keeping developers waiting on answers of requirement elaborations can waste a lot of time. The *customer accessibility* differs, depending on which department is the request owner (see question 43 on page 93). In general, the time it takes for the customer to answer a question should decrease to improve productivity of the agile teams.

While there are key performance indicators [KPIs] for the agile teams to deliver software every 3 weeks (see question 42 on page 73), there is no obligation to collaborate with the agile teams (see the answer to question 44 on page 73 and question 44 on page 93).

Organizational process definition

While the work environment is sufficiently furnished for day-to-day agile software development, there are some things that can be improved to make it an *ideal agile physical setup*. Working from home erects a small barrier to face-to-face communication, decreasing the amount of tacit knowledge sharing (see the answer to question 23 on page 76 and question 23 on page 98). When multiple people have to work at the same desk, the lack of workspace can limit productivity (see question 23 on page 76 and question 23 on page 69).

While the teams are *self organizing* and *collaborative* they are not fully *empowered*. This is due to the challenges system complexity (see section 3.4 on page 10) and enterprise focus (see section 3.9 on page 12) which are to be expected in a large organization with

a complex IT landscape (see the answer to question 18 on page 68). When the workload increases, the developers become less motivated and collaborative (see the answers to question 17 on page 67 and question 19 on page 68).

Organizational performance management

Two ways in which agile methodologies can improve software development process performance is by *lowering the amount of process ceremony* and taking an *agile approach to documentation*.

Since this organization is pretty large, a certain amount of bureaucracy is unavoidable (see the answers to question 7 on page 64 and question 7 on page 95).

The concept of *agile documentation* -writing only the bare minimum of documentation, only when needed- should not be abused as an argument to write no documentation at all (see question 35 on page 71). However, most of the time the developers write a sufficient amount of documentation (see the answers to question 34 on page 78 and question 35 on page 100).

Organizational training

Frequent face-to-face communication helps spread tacit knowledge within agile teams. Team members at the B2C department have a lot of face-to-face communication (see question 22 on page 69 and question 22 on page 97). However, when people are working from their home, the geographical distribution (see section 3.2 on page 9) makes the barrier to face-to-face communication higher (see the answer to question 22 on page 76).

Knowledge sharing tools can be used to store and share explicit knowledge. The agile teams at B2C store a lot of information in wikis (see the answer to question 25 on page 70). However, since each team has its own wiki, the knowledge is spread over a number of knowledge sharing tools (see question 25 on page 77).

Pair programming can help tacit knowledge sharing and helps create knowledge of an unfamiliar codebase. While *pair programming* is currently not commonly used for this purpose at the B2C department, some developers are already occasionally using pair programming (see question 38 on page 101) and the interviewees were generally favorable about implementing it (see the answer to question 36 on page 79).

Product integration

The answers to the interview questions show that the agile teams of the B2C department use *smaller and more frequent releases* to *continuously deliver* working software after every iteration (see the answer to question 9 on page 65).

However, they currently do not use *continuous integration* in most fields (see the answers to question 30 on page 70 and question 30 on page 85). Continuously building and testing software automatically can quickly give developers feedback when introducing bugs into the codebase and can therefore be a valuable tool for agile teams.

Project monitoring and control

The answers to the interview questions show that project monitoring is sufficiently implemented by the *daily progress tracking meetings* (standups) and *tracking iteration progress* (burnup charts) agile practices (see questions 28 on page 91 and 34 on page 92).

Project planning

All agile principles and concepts related to the CMMI project planning process area are sufficiently implemented. Since the organization is pretty large, it is to be expected that the amount of planning that is actually left to the agile teams is limited.

A high-level planning is actually made by the business board by means of a prioritization of projects (see the answer to question 8 on page 65 and question 15 on page 66). However, teams will have to slightly deviate from that plan to make sure that team members from all fields have some work to do (see question 8 on page 88).

The team members themselves estimate how much time and effort a specific user story is going to take (see the answer to question 16 on page 67.

Requirements development

Since customers at can't create *user stories* on their own due to lack of technical knowledge (see question 2 on page 102), creating the requirements should be a collaborative process. The principle "Customer collaboration over contract negotiation" of the agile manifesto (see page 8) is based on the fact that it's nigh impossible to know all requirements of a project in advance and the concept of *evolutionary requirements* means that requirements get clarified during development. While this should not be used as an excuse for scope creep (see question 4 on page 87 and question 46 on page 114), one of the big advantages of agile software development is that requirements can change and new requirements can surface during development to ensure that the ultimately delivered software fits the needs of the customer. However, this requires close collaboration with said customer to elaborate requirements, since things that can be obvious to the customer might be interpreted differently by the developers (see question 2 on page 102).

Requirements management

The answers in the interviews show that *maintaining a list of features and their status* or backlog works pretty well as a way of managing requirements at the B2C departement. However, some issues with the backlog being either too small (see question 4 on page 87 and question 12 on page 83) or too big (see the answer to question 4 on page 63) do exist.

Risk management

The answers to the interview questions show that sprints at the B2C department are usually not based on the amount of risk for the organization (see the answer to question 11 on page 66). The organization has to deal with a lot of regulatory compliance (see section 3.6 on page 11). *Risk driven iterations* can help ensure that changes that can cost the organization a lot of money if implemented incorrectly of not delivered on time are prioritized.

Supplier agreement management

The organization works with a lot of external parties which often don't work with an agile software development methodology (see question 2 on page 62). This requires a lot of planning up front to make sure the right software is delivered at the end, especially if agile teams at the organization are dependent on functionality that is implemented by external parties. The supplier agreement management process area of CMMI describes a number of practices that can be implemented to ensure that dealing with external parties is a trouble-free experience, such as creating and executing agreements, acceptance testing the released products and providing a smooth transition.

Technical solution

While *no big design is made up front*, due to enterprise focus (see section 3.9 on page 12) and system complexity (see section 3.4 on page 10) some degree of design up front is required (see the answers to question 2 on page 62, question 18).

Since at this organization, external developers come and go on a regular basis, having *coding standards* becomes extra important to ensure a consistent quality of the code base. Not all agile teams at the B2C department have *coding standards* or are actively using them. Since this is an agility level 1 concept, introducing *coding standards* and making sure they are used should be a high priority.

While developers generally react favorably to the idea of *continuous improvement (refactoring)*, it is rarely done in practice. The reason for this is the lack of time and the fact that it is not a priority for the business. However, improving code quality does not contribute directly to customer value but does make future changes easier and can prevent bugs from being introduced, which both indirectly contribute to customer value in the long term.

Validation

Continuous customer satisfaction feedback can help customers ensure that what developers are building actually corresponds with their needs. This allows customers to notice discrepancies between the software being built and their actual wishes early in the development process and prevent costly 180 degree turns late in the project (see the answer to question 4 on page 87). At this organization, this requires the customer to allocate more time to be present at demos. Currently, the customer is not always present at demos (see question 42 at page 104). Agile teams should agree with the customers upon a specific day at the end of every sprint for the demo, allowing all parties to allocate time for it far in advance. This prevents issues where customers can not be present at the demo because of lack of time or because they were only notified a few days in advance (see the answer to question 41 on page 108).

Verification

Agile teams at the B2C department are all writing a sufficient amount of unit tests (see question 32 on page 71 and 36 on page 79). However, they are currently not working with *test-driven develoment* and are only occasionally *pair programming* (see question 38 on page 101. The developers are not big fans of test-driven development, as can for example be seen in the answer to question 37 on page 100.

7.2.2 Suggestions for improvements

To implement a CMMI level 3 software development process using agile practices and concepts, there are still a few practices and concepts that need to be implemented.

As could be read in section 7.1, the area that is currently most lacking and should be improved first deals with the collaboration between business and IT. As part of the integrated project management CMMI process area, stakeholder involvement should be managed. In the case of the B2C department, this means that the *customer commitment to work with the developing team* should be improved. Since customer input is very important in evolutionary development, the *customer should be immediately accessible* to for example clarify requirements. Since the business board has given a high priority to a specific project instead of other projects, delivering high-quality software on time should be in the interest of the customer which should motivate them to collaborate with the agile teams as much as possible.

The *integrated project management* CMMI process area provides a number of practices that can be implemented to improve stakeholder involvement. As mentioned in the *causal analysis and resolution* process area, the improvement process for this should involve both parties to create buy-in and ensure that the resulting process is practical.

Continuous customer satisfaction feedback helps the team ensure that what they are building corresponds with what the business actually wants. Therefore it is very important that the customer frequently visits demos to see the current state of the software and give his feedback. One way to ensure customer feedback is by putting KPIs on customer presence at demos.

Since customer collaboration is crucial to deliver a high-quality product on time, the *project monitoring and control* CMMI process area advises to monitor customer involvement as well and take action if the customer is not allocating sufficient time to collaboration with the agile team.

Since the organization deals with a lot of risk, implementing *risk-driven iterations*, can help ensure the prioritization of changes that can cost the organization the largest amount of money if not implemented correctly on time.

Another area that can benefit from the implementation of additional agile practices is *Technical Excellence*. Improvements in the technical maturity of the software development process start with improving the quality of the code base. To improve code quality *coding standards* need to be drafted and all teams should start using them. In addition, time should be allocated for *continuous improvement (refactoring)* of the existing code to improve the quality of the code base and make the code more maintainable. *Pair programming* improves knowledge sharing and code quality and can also be a valuable improvement to the *organizational training* and *verification* CMMI process areas.

Another practice that can improve the implementation of the *verification* CMMI process area is *test-driven development*. *Test-driven development* requires a *continuous integration* infrastructure to continuously test the current state of the software so this should be implemented before attempting to use *test-driven development*.

Since both *test-driven development* and *pair programming* are level 5 practices, implementing lower level concepts and practices should be prioritized.

7.3 Validation of the suggestions at a large energy company

To check whether the advice from the agile CMMI framework (see section 7.2.2 on page 51 is applicable in practice and can be useful for large organizations, I've asked an agile coach at the company to give his opinion about the proposed improvements. The entire interview can be read in appendix B.9 on page 115.

Not all suggested improvements proved to be applicable in practice. Some of the challenges mentioned in chapter 3 on page 9 are responsible for the fact that it's very hard to achieve a high agility level in large organizations. Nevertheless, introducing new agile practices and concepts into an existing software development process can help increase productivity and software quality.

Some improvements can be implemented by the agile teams themselves without customer involvement. Increasing the use of coding standards and taking into account the risks of a specific user story during the planning are relatively low level agility concepts which can help reduce risks for the organization. Pair programming and test-driven development are both level 5 agility practices which means they should only be implemented when lower level practices and concepts are already present. The answer to question 8 on page 117 shows that the agile teams at the B2C department currently don't have sufficient technical know-how to implement test-driven development but that implementing test-driven development is a long-term plan. Since pair programming does not require any additional technical skill, it can already be implemented and is in fact already occasionally done (see question 9 on page 117).

System complexity (see section 3.4 on page 10) prevents the organization from implementing continuous integration (see question 4 on page 115).

The most important point on which the software development process at the B2C department can be improved is the collaboration with the customer. The amount of contact between business and IT should be increased to for example elaborate requirements and check that what is being built corresponds with the business needs. While the maturity and agility of the agile teams are pretty good, an analysis of the interview answers (see section 7.1 on page 39) makes it clear that most of the agile practices and concepts that interface with the customer show problems. The improper functioning of agile practices and concepts that deal with the customers has a negative effect on the productivity of the agile teams. This shows that it's important that the improvement process is not limited to the IT departement, but is actually done in consultation with other stakeholders. To improve this situation, the entrenched culture (see section 3.3 on page 10) of the business needs to be changed. The answers to question 1 on page 115 and question 6 on page 116 show that the agile consultant at the organization agrees with this analysis.

Overall, the views of the agile coach agree fairly well with the suggestions from the agile CMMI framework.

Chapter 8

Reflection

From the various agile improvement models (see chapter 5 on page 15), I chose the agile adoption framework (see section 5.4 on page 20) to combine with CMMI and use in my case study. This has proven to be a good choice since the various agile practices and concepts from that framework map clearly to the different CMMI practices and the indicators make measuring the agility of a software development process easy. The concrete agile practices and concepts result in a clear advice and roadmap for improvement of software development agility.

The results in section 7.1 on page 39 show that the lower levels are often implemented while the highest levels lack an implementation. While this one case study is not enough to state that this is the case for all organizations, it does suggest that the levels of the agile adoption framework reflect the order in which organizations adopt agile practices and concepts in practice and it therefore is a good roadmap for the improvement of the software development process agility.

While the views of the agile coach agree fairly well with the suggestions from the agile CMMI framework, challenges to agility improvement (see chapter 3 on page 9) prevent some suggestions from being applied in practice, in which case the default CMMI practices of a process area need to be implemented to achieve a certain maturity level. Further research could combine the current framework with the list of challenges described in chapter 3 on page 9 and a hybrid methodology model (see chapter 4 on page 13) to get a more comprehensive list of agile constraining practices and their implications for the software development process.

My case study was mainly focused on the agility aspect of the agile CMMI framework. While I did not do a full CMMI appraisal, the information I got from the interview questions based on the agile indicators was enough to provide some useful suggestions for improvements. Further research should combine the CMMI appraisal process with the agility measurement process to get a more comprehensive image of the software development process maturity.

Chapter 9

Conclusion

In this master thesis I've looked at software development process improvement in large organizations.

My research question was: *Can an agile CMMI framework help a large organization improve their software development processes?*

To that end, I've mapped specific agile practices and concepts to CMMI process areas. Hereafter I've applied this agile CMMI framework in a case study at a large energy company.

While there are some CMMI process areas where CMMI and agile software development methodologies conflict, it's possible to implement a CMMI level 3 software development process using agile practices and concepts. The resulting agile CMMI framework can provide useful advice about next steps to take in the journey to more agility and maturity in the software development process of large organizations.

Chapter 10

References

10.1 Academic peer-reviewed papers

[1] P. Abrahamsson, K. Conboy, and X. Wang. Lots done, more to do: The current state of agile systems development research. *European Journal of Information Systems*, 18(4):281–284, 2009.

[2] S. Ambler. Agile software development at scale. *Balancing Agility and Formalism in Software Engineering*, pages 1–12, 2008.

[3] D.J. Anderson. Stretching agile to fit cmmi level 3 – the story of creating msf for cmmi process improvement at microsoft corporation. In *Proceedings of the Agile Conference, 2005*, pages 193–201. IEEE, 2005.

[4] E. Arisholm, H. Gallis, T. Dybå, and D.I.K. Sjøberg. Evaluating pair programming with respect to system complexity and programmer expertise. *IEEE Transactions on Software Engineering*, 33(2):65–86, 2007.

[5] R.D. Austin and L. Devin. Weighting the benefits and costs of flexibility in making software: Toward a contingency theory of the determinants of development process design. *Information Systems Research*, 20(3):462–477, 2009.

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

[7] B. Boehm and R. Turner. Observations on balancing discipline and agility. In *Proceedings of the Agile Development Conference, 2003. ADC 2003*. IEEE, 2003.

[8] B. Boehm and R. Turner. Using risk to balance agile and plan-driven methods. *Computer*, 36:57–66, 2003.

[9] B. Boehm and R. Turner. Management challenges to implementing agile processes in traditional development organizations. *Computer*, 22(5):30–39, 2005.

[10] G. Canfora, A. Cimitile, F. Garcia, M. Piattini, and C. Visaggio. Evaluating performances of pair designing in industry. *The Journal of Systems and Software*, 80(8):1317–1327, 2007.

[11] A. Cockburn and J. Highsmith. Agile software development: The people factor. *Computer*, 34:131–133, 2001.

[12] A. Cockburn and L. Williams. The costs and benefits of pair programming. In *eXtreme Programming and Flexible Processes in Software Engineering – XP2000*, pages 33–42, 2000.

[13] H. Erdogmus, M. Morisio, and M. Torchiano. On the effectiveness of the test-first approach to programming. *IEEE Transactions on Software Engineering*, 31(3):226–237, 2005.

[14] M. Fritzsche and P. Keil. Agile methods and cmmi: Compatibility or conflict? *e-Informatica Software Engineering Journal*, 1(1):9–26, 2007.

[15] H. Glazer. Love and marriage: Cmmi and agile need each other. *CrossTalk*, pages 29–34, 2010.

[16] J. Hannay, T. Dyba, E. Arisholm, and D. Sjøberg. The effectiveness of pair programming: A meta-analysis. *Information and Software Technology*, 51(7):1110–1122, 2009.

[17] D. Hartmann and R. Dymond. Appropriate agile measurement: Using metrics and diagnostics to deliver business value. In *Proceedings of AGILE 2006 Conference*, pages 126–134. IEEE, 2006.

[18] C.R. Jakobsen and K.A. Johnson. Mature agile with a twist of cmmi. In *Agile, 2008. AGILE'08. Conference*, pages 212–217. IEEE, 2008.

[19] C.R. Jakobsen and J. Sutherland. Scrum and cmmi – going from good to great. In *Agile Conference, 2009. AGILE'09*, pages 333–337. IEEE, 2009.

[20] F. Karlsson and P. Ågerfalk. Exploring agile values in method configuration. *European Journal of Information Systems*, 18(4):300–316, 2009.

[21] T. Kähkönen. Agile methods for large organizations – building communities of practice. In *Agile Development Conference, 2004*, pages 2–10. IEEE, 2004.

[22] C. Larman and V.R. Basili. Iterative and incremental development: A brief history. *Computer*, 36(6):47–56, 2003.

[23] A. Law and R. Charron. Effects of agile practices on social factors. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–5, 2005.

[24] M. Lindvall, D. Muthig, A. Dagnino, C. wallin, M. Stupperich, D. Kiefer, J. May, and T. Kähkönen. Agile software development in large organizations. *Computer*, 37(12):26–34, 2004.

[25] K.M. Lui and K.C.C. Chan. Pair programming productivity: Novice-novice vs. expert-expert. *International Journal of Human-Computer Studies*, 64(9):915–925, 2006.

[26] K.M. Lui, K.C.C. Chan, and J.T. Nosek. The effect of pairs in program design tasks. *IEEE Transactions on Software Engineering*, 34(2):197–211, 2008.

[27] M. Lutz, T. DeMarco, and B. Boehm. The agile methods fray. *Computer*, 35(6):90–92, 2002.

[28] A. Marçal, B. Freitas, F. Soares, and A. Belchior. Mapping cmmi project management process areas to scrum practices. In *Software Engineering Workshop, 2007. SEW 2007. 31st IEEE*, pages 13–22. IEEE, 2007.

[29] J. McAvoy and T. Butler. The role of project management in ineffective decision making within agile software development projects. *European Journal of Information Systems*, 18(4):372–383, 2009.

[30] K. Molokken-Ostvold and N. C. Haugen. Combining estimates with planning poker–an empirical study. In *Software Engineering Conference, 2007. ASWEC 2007. 18th Australian*, pages 349–358. IEEE, 2007.

[31] P. Naur and B. Randell. Software engineering: Report on a conference sponsored by the nato science committee. In *NATO Software Engineering Conference*. NATO Scientific Affairs Division, 1968.

[32] M.K. Nayak and M.R. Patra. Agile project management – redefining the role of managers. In *Proceedings of the 2nd National Conference; INDIACom-2008*, 2008.

[33] S. Nerur, R.K. Mahapatra, and G. Mangalaraj. Challenges of migrating to agile methodologies. *Communications of the ACM*, 48(5):72–78, 2005.

[34] J. Packlick. The agile maturity map – a goal oriented approach to agile improvement. In *AGILE 2007*, pages 266–271. IEEE, 2007.

[35] A. Qumer and B. Henderson-Sellers. Measuring agility and adoptability of agile methods: A 4-dimensional analytical tool. In *Proceedings IADIS International Conference Applied Computing*, pages 503–507, 2006.

[36] A. Qumer, B. Henderson-Sellers, and T. McBride. Agile adoption and improvement model. In *Proceedings European and Mediterranean Conference on Information Systems 2007*, pages 1–9, 2007.

[37] W. Royce. Managing the development of large software systems: Concepts and techniques. In *Proceedings of IEEE WESCON*, pages 328–339. IEEE CS Press, 1970.

[38] A. Sidky and J. Arthur. A disciplined approach to adopting agile practices: The agile adoption framework. *Innovations in systems and software engineering*, 3:203–216, 2007.

[39] M. K. Spayd. Evolving agile in the enterprise: Implementing xp on a grand scale. In *Proceedings of the Agile Development Conference, 2003*, pages 60–70. IEEE, 2003.

[40] J. Sutherland, C. Jacobsen, and K. Johnson. Scrum and cmmi level 5: The magic potion for code warriors. In *Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, pages 466–466. IEEE, 2008.

[41] J. Sutherland, A. Viktorov, J. Blount, and N. Puntikov. Distributed scrum: Agile project management with outsourced development teams. In *Proceedings of the 40th Annual Hawaii International Conference on System Sciences*. IEEE, 2007.

[42] R. Turner. Agile development: Good process or bad attitude? *Product Focused Software Process Improvement*, pages 134–144, 2002.

[43] R. Turner and A. Jain. Agile meets cmmi: Culture clash or common cause? In *Extreme Programming and Agile Methods-XP/Agile Universe 2002*, pages 153–165. Springer, 2002.

[44] L. Williams, R. R. Kessler, W. Cunnningham, and R. Jeffries. Strengthening the case for pair programming. *Software*, 17:19–25, 2000.

10.2 Other

[45] M. Fowler and M. Foemmel. Continuous integration, 2006.

[46] M. Fowler and J. Highsmith. The agile manifesto. *Software Development*, 9(8):28–35, 2001.

[47] H. Glazer, J. Dalton, D. Anderson, M.D. Konrad, and S. Shrum. Cmmi or agile: Why not embrace both! Technical report, Software Engineering Institute, 2008.

[48] D. North. Introducing bdd. *Better Software*, 2006.

[49] R. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Science/Engineering/Math, 6th edition, 2005.

[50] A. Sidky and J. Arthur. Agile adoption process framework – indicators document, 2006.

[51] CMMI Product Team. Cmmi for development, version 1.3. Technical report, Software Engineering Institute, 2010.

[52] D. West and T. Grant. Agile development: Mainstream adoption has changed agility. Technical report, Forrester Research, 2010.

Appendix A

Interview questions

1. What is your job at [company] and how does it relate to the agile software development process of the B2C department?
2. What are in your opinion the most important problems with the agile software development process of the B2C department?

Embrace change to deliver customer value

3. Are management and developers willing and able to reflect and tune the process after every iteration and release?
4. Are managers and developers willing and able to deal with evolutionary requirements?
5. Has the customer the power to dictate the scope of the iteration?
6. Is the customer encouraged to continually give feedback/criticism and rethink their requirements during the development process?
7. Can developers make decisions on their own without explicit management approval?

Plan and deliver software frequently

8. Do customers, developers and managers plan together?
9. Are management and developers willing to use an incremental and iterative development approach?
10. Is management willing to commit to the process of continuously planning instead of developing a one-time plan upfront?
11. Do the developers and managers agree to have risks drive the scope of each iteration?
12. Is the high-level agile planning based on features instead of tasks?
13. Is management willing to maintain an up-to-date list of all the remaining features for the project and their status?

14. Does the team deliver a fully functional release after every iteration?
15. Is management willing to plan immediately before the iteration instead of earlier to allow the customer's feedback to be incorporated into the planning?
16. Are developers estimating the effort and duration of various user stories themselves?

Human centric

17. Are people willing to work in teams and help others?
18. Has management empowered teams with decision making authority?
19. Are team members motivated?
20. Does management agree to have self organizing teams?
21. Do employees feel comfortable working in self organizing teams?
22. Is there frequent face-to-face communication between team members?
23. Are all the developers are in a common room, furnished to facilitate the agile process?

Technical excellence

24. Are developers using coding standards?
25. Are the team members using knowledge sharing tools?
26. Do employees volunteer for tasks instead of being assigned to them?
27. Does the organization have tools for software configuration management?
28. Does a mechanism exist to monitor the iteration progress?
29. Is design a continuous process or done once at the beginning of the development process?
30. Are the developers willing and able to use continuous integration?
31. Are the developers successfully refactoring code?
32. Are the developers successfully writing unit tests during development?
33. How many team members have a level of software development method understanding and use which enables them to tailor a method to fit a new situation?
34. Are management and developers willing to meet daily to discuss the progress of the project?
35. Are developers willing and able to take an agile approach to documentation?
36. Are management and developers willing and able to use user stories?
37. Are the developers successfully using test-driven development?
38. Are the developers successfully using pair programming?
39. How many employees are unable or unwilling to collaborate or follow shared methods?

40. How many employees are unable to deal with rapid change?

Customer collaboration

41. Is the customer willing to dedicate time to take an active role in the project?
42. Is the customer contract reflective of evolutionary development?
43. Is the customer immediately accessible?
44. Does the customer contract revolve around commitment of collaboration?
45. Do the developers and customer frequently interact face-to-face?

46. Did you think of any other problems with the agile software development process of the B2C department during the course of this interview?

Appendix B

Interviews

B.1 Interview 1

1. **What is your job at [company] and how does it relate to the agile software development process of the B2C department?**
   I work at [company] as solution architect specializing in integration and SOA [service oriented architecture]. I am also the team leader of the complete integration team; around 25 people who do the development work here.

2. **What are in your opinion the most important problems with the agile software development process of the B2C department?**
   The largest challenge we have here at [company] is that we are doing agile in a 5-layer architecture and development regularly impacts multiple different applications which are not in the scope of a specific epic or maybe even not in the same business unit.
   Another challenge I see with the way we are doing agile is that we involve external parties. Those often don't work with an agile software development methodology so they only release at the end of development. This creates problems when agile teams at [company] are dependent on functionality being implemented by external parties. It's possible to encounter issues during the sprint and if you haven't integrated them in your agile team - which is often impossible because you're working with an external party- then the issue might not be picked up adequately which forces you to add the issue to the backlog. This can result in you being unable to deliver what the customer asked and you may not even be able to run the software in production because it doesn't work correctly.

Embrace change to deliver customer value

3. **Are management and developers willing and able to reflect and tune the process after every iteration and release?**

   Reflect and tune process (+)

   The teams do have retrospectives in which they discuss what went right and what went wrong. I notice that this does not result in a lot of process adjustments but instead it mostly deals with how should pick up certain things next time however the process is not or barely adjusted because of this.

   What does happen however is that once in a while people from delivery and team leaders meet and reflect about the process to see whether there are shortcomings or opportunities for optimization but this is not after every sprint. For example, we started out with backend-, frontend- and online-teams, which we changed to mixed teams and now we are working with chameleon teams so we are constantly changing.

   The problem with people from different fields helping to drive the process is that you have mixed interests. For example, my focus is to get people who have nothing to do within their team to support other teams because a team does not have to be fixed and if there is no work for a certain employee on his field he can easily help another team. Other people have the another point of view on this and think that we should implement more T-shaping which means that a developer who doesn't have anything to do should start testing. These are different views on how to implement such a process. This shows that due to this conflict of interests process adaptations sometimes go left and sometimes go right.

4. **Are managers and developers willing and able to deal with evolutionary requirements?**

   Evolutionary requirements (+)

   Features and requirements for the sprint are decided on as late as possible. In most cases, the customer can request changes during development and those will be picked up provided they fit within the borders of the epic. If the change request is too big -which will be decided on by the team immediately upon request of the change- then most often will be decided to implement the features and requirements as they were and implement the requested changes in the following sprint, provided the change gets enough priority since the backlog is currently quite big.

5. **Has the customer the power to dictate the scope of the iteration?**
   No, the scope of the sprint is not dictated by the customer. Instead, the scope is determined by the product owner, in cooperation with the team and potentially the attached project leader but the customer does not determine the entire scope. This would not be possible because we deal with a lot of customers and consequently a lot of conflicting interests which requires an umbrella body to determine the scope of the iteration.

Client driven iterations (+/-)

6. **Is the customer encouraged to continually give feedback/criticism and rethink their requirements during the development process?**
   We're seeing that the customer -the requester of specific functionality- is not sufficiently involved. In practice the product owner by proxy is trying to represent the interests of the customer and the customer only gets back in the picture at the very end.

Continuous customer satisfaction feedback (+/-)

7. **Can developers make decisions on their own without explicit management approval?**
   In the past the team decided themselves how to implement the changes in the IT landscape based on the epic. We are currently rolling that back because at [company] we are working with a lot of external developers. The average external developer is only at [company] for about half a year to a year after which they leave and are replaced by another external developer. You can't expect from everybody to know the entire architecture when they start at [company]. As some articles about agile software development mention there's the concept of a "pre game" in which a part of the design is implemented. This phase -which we call "IT readiness"- we reintroduced at [company] In this phase we receive the customer requests, look at which architectural principles go with them and start implementing a small part of the design based on those architectural principles. After that phase, we hand over the design to the team and tell them to stay within the bounds of that design. Within the bounds of the previously created design the team has complete freedom.

Low process ceremony (+/-)

Plan and deliver software frequently

8. **Do customers, developers and managers plan together?**
   No, we have the business board for that so that doesn't involve the customer. Indirectly the customer is involved because he is in the business board as well but that is not the real request owner, instead the managers and representatives of the various departments decide on the planning. The developers gat a list of epics that need to be picked up in the sprint and they can state which epics they want to pick from the backlog and want to commit themselves to. However, nowadays there is a planning for some sprint items as well because for example legal requirements and market changes can dictate deadlines. Therefore sometimes teams are scheduled to pick up specific items in a specific sprint, but this is based on certain priorities.

Collaborative planning (+/-)

9. **Are management and developers willing to use an incremental and iterative development approach?**
   Yes, after every sprint we release working software. In the two years we are working with an agile software methodology, it only only once occurred that we could not release and had to extend the sprint with two weeks. Other than that one time we have always delivered working software after three weeks.

Continuous delivery (+)

Smaller and more frequent releases (+)

10. **Is management willing to commit to the process of continuously planning instead of developing a one-time plan upfront?**
    The big one-time plan is basically for what the project leaders want picked up. In practice this is sometimes too much to fit in one single sprint. This is one of the big challenges we see with big projects versus agile, how do you handle this when you are running multiple big projects at the same time? This requires you need to assign a priority to a certain project of certain change that needs to be picked up quickly to find the right balance per team and keep the right priority for the team. We find this very hard.

Planning at different levels (+)

11. **Do the developers and managers agree to have risks drive the scope of each iteration?**
    A project leader looks at the date it needs to be released to fulfill legal requirements or a business case and this is calculated back to how many epics they need. These epics are the features, the separate pieces of functionality that need to be developed. A project leader then states how many epics he needs within a certain amount of sprints to achieve the project goals. The team itself decides on how the work within the epic is handled.

Risk driven iterations (+/-)

13. **Is management willing to maintain an up-to-date list of all the remaining features for the project and their status?**
    Working with a backlog works well for us but at the moment it's quite large. There are a lot of items on the backlog because a lot of new work is submitted and we only have 7 teams, which currently can't handle the amount of work coming our way. This is caused by the number of very large projects that are currently supplying us with their requests.

Maintain a list of all features and their status (+)

15. **Is management willing to plan immediately before the iteration instead of earlier to allow the customer's feedback to be incorporated into the planning?**
    The customer's feedback is incorporated into the planning because the customer is in the business board and thus the customer can participate in discussions about what the priority of the project is. This shows that there is a good cooperation between customer and IT. However, it should be noted that we are talking about the customer representative here, not the request owner himself, so we are usually not talking with the person who requested the work but the person responsible that the customer goals are met.

Adaptive planning (+)

16. **Are developers estimating the effort and duration of various user stories themselves?**
The teams themselves determine how much time and effort a specific user story will take.
The business board decides on a higher level that a certain change is worth more to them according to a certain value they gave to it or that a certain project is worth more to them. Based on that value they will determine whether a team should give priority to a certain change instead of another epic.
The team ultimately decides about the amount of time and effort an epic should take when they are playing planning poker. So when a project manager indicates that in this sprint 10 epics need to be finished and the team gets those 10 epics on their plate, the team themselves poker with those epics to see what they can commit to in that sprint and all other epics will have to be postponed to the next sprint.

Agile project estimation (+)

User stories (+)

Human centric

17. **Are people willing to work in teams and help others?**
I'm currently working on the concept of swapping out people between teams but also between agile projects and projects using a traditional software development methodology. A close team works well but they shouldn't become too close because it becomes counterproductive when they start working like an autonomous island within the organization. That's why I'd rather have people supporting each other from time to time. At the moment there are very little problems with the willingness of people to work in teams and help others. It is noticeable that if all teams are very busy, the willingness to help each other out drops quite a bit because everyone is too busy with his own work.

Collaborative teams (+)

18. **Has management empowered teams with decision making authority?**

Empowered and motivated teams (+/-)

It's not only management that determines the framework within which the project should stay but the architecture should be taken into account as well. We simply do have a SOA architecture and you need to consider that agile is designed for monolithic applications where actual development work occurs with 2 to 3 developers per team, with the testers, customer and documenters collocated. However, 2 to 3 developers is nice if you would simply need java programmers. We have many different types of programmers and that often brings some challenges.

You have to make sure that developers don't go for the "quick win" in a 5 layer architecture but that they instead look for a good solution. You do not want business logic in your integration, you want your data stored in your backend applications, you want your front end to remain free of business logic so that only a small amount of content management is performed there. The "readiness team solution architecture" guards against these kind of mistakes, we create a design based on the 3-5 year architectural plan and tell the team they have to stay within that framework to keep the architecture landscape stable for the next few years. If you don't do that, you will have a problem later because in that case you'll have a hodgepodge of business logic in all layers, services which unnecessarily are duplicated, that kind of things. You can only prevent that by keeping a focus on the architecture.

Maybe the most important obstacle I'm seeing with agile in that context is that textbook agile gives teams almost complete freedom. In my opinion this may work in a software development house with internal developers, all facing the same direction and knowing the same architectural principles. However if you have multiple applications in your IT landscape, are working with external developers that are replaced regularly, I think it becomes much harder to do proper agile and the organization needs to keep an eye on the architecture.

19. **Are team members motivated?**

Empowered and motivated teams (+)

Motivation in itself is pretty good, but you do notice that when a disproportionate amount of work is expected from people, so the amount of poker points that needs to be done in a sprint is more than they are able to do, that motivation is impacted.

Motivation is also impacted when specialists are asked to for example test instead of applying their development specialism for 2-3 sprints, so 6 to 9 weeks. You can notice that this results in a severe drop in motivation.

However, on the whole everyone is pretty satisfied with the concept of agile.

20. **Does management agree to have self organizing teams?**
Yes.

Self organizing teams (+)

21. **Do employees feel comfortable working in self organizing teams?**
Yes, they are pretty happy to be working in self organizing teams. Once in a while a tester supports a developer or vice versa.

Self organizing teams (+)

Task volunteering (+)

22. **Is there frequent face-to-face communication between team members?**
Yes, I'm seeing that team members are having a lot of face-to-face meetings. In addition, every morning there's a standup and each afternoon the team members are having a meeting about the question whether they are going to play planning poker -which they do twice a week- and once in a while to talk about complications that are occurring, something they obviously do in the standups as well. So there is a lot of face-to-face communication between team members.

Frequent face-to-face communication (+)

23. **Are all the developers are in a common room, furnished to facilitate the agile process?**
Yes, all facilities are available. However, I have to admit I personally think that the office space here is pretty bad. For example, I've seen 2 people work at the same desk. The concept of "anders werken" [telecommuting, hot desking] provides some challenges.

Ideal agile physical setup (+/-)

Technical excellence

24. **Are developers using coding standards?**
In integration we have standard guidelines and those are used. I know that in other fields they aren't there yet of are still in development. We also have a quality assurance review at the end to check whether we stayed within those guidelines. We check that because integration is the backbone of our organization and if we make a small mistake the complete interface between A and B is jammed so we keep a close eye on it.

Coding standards (+/-)

25. **Are the team members using knowledge sharing tools?**
Yes, we are using our wiki fulltime, for example you can find our coding standards there. Our service catalog is on there as well and that is always updated when changes occur. Basically we have a lot of information on the wiki currently.

Knowledge sharing tools (+)

27. **Does the organization have tools for software configuration management?**
Yes, in integration we have subversion for that, which in itself works pretty well. We have yet to have problems checking in code or checking out code.
The only disadvantage is that we don't have software configuration management when for example you have a dependency between a backend system and an integration.
Software configuration management is currently mainly used in the web field.

Software configuration management (+/-)

28. **Does a mechanism exist to monitor the iteration progress?**
Yes, we are using burnup charts and Jira for tracking epics. Furthermore we have the delivery board and the like to monitor that the epics that are delivered are what was requested. So in that way we try to keep an eye on things.

Tracking iteration progress (+)

29. **Is design a continuous process or done once at the beginning of the development process?**
This also unfortunately depends on which field you are talking about. When I'm talking about my own department, we create the design beforehand and check afterwards to see whether we stayed within that design. Basically we ensure that design is a continuous process. Designs might get adapted at then end as well when needed. When it's obvious that a solution doesn't fit within the current design this is communicated with the designer who will change the design.

No big design up front (+)

30. **Are the developers willing and able to use continuous integration?**

We are using the term continuous integration but when I'm looking at theory versus practice than we are not fully using it. We use only a part of the concept behind continuous integration; we don't do automated testing and automated building but we are currently implementing a part of continuous improvement, so we're constantly challenging our landscape.

Continuous integration (-)

31. **Are the developers successfully refactoring code?**
Yes, we're constantly refactoring code. We always make sure we adapt existing code to fit the new standards.

Continuous improvement (+)

32. **Are the developers successfully writing unit tests during development?**
Yes, we create unit tests ourselves. We have a tool for unit tests we are currently not using that much but we are soon going to roll it out further within the organization.

Unit tests (+)

33. **How many team members have a level of software development method understanding and use which enables them to tailor a method to fit a new situation?**
I think that 30 to 40 percent of the team members is capable of doing that and others basically try to follow that 30-40 percent.

30% of level 2 and level 3 people (+)

34. **Are management and developers willing to meet daily to discuss the progress of the project?**
Managers actually play no role in the agile process, they are pretty much on the sidelines but are for example as delivery lead responsible for what is delivered. Occasionally they are present at standups but there is no structured meeting between managers and developers.

Daily progress tracking meetings (+/-)

35. **Are developers willing and able to take an agile approach to documentation?**
I think this is a big downside of agile, because this point is often abused for not having to write documentation. I deliberately call this abuse because I know agile says you must provide minimal documentation, only when needed. However, this is often interpreted as that documentation is not necessary.
I think people still need to deliver towards administration what one has modified, how it was changed and what needs to be administered. This way system administration always knows what's gonna be their delta.
The documentation we are still producing ourselves are design documents up front, the solution architectures and at the end for example our service catalogus is updated.
Occasionally I have the feeling that too little is documented or documentation is not updated because there is too much coding work in a sprint and too little time is freed up for decent documentation of what has been delivered. I think documentation belongs in the "definition of done" tasks but this is currently often not the case.

Agile documentation (+/-)

37. **Are the developers successfully using test-driven development?**
Not yet, but they are working on it. Just like automated testing, which is also something we want to do.

Test-driven development (-)

38. **Are the developers successfully using pair programming?**
No, we're not using pair programming but we are using another concept in which people from different fields sit next to each other to for example quickly determine what will be the input and what will be the output. For example, a SAP guy indicates how the IDocs are going to look, the integration guy that sits next to him immediately starts making the mapping and when he encounters issues he can easily ask his neighbour to re-route something and he can immediately change it to fix the mapping. That concept of pair programming we do have.

Pair programming (-)

39. **How many employees are unable or unwilling to collaborate or follow shared methods?**
There are a few developers that are pretty dominantly present in the teams and when they are of the opinion that something is not supposed to be done in a specific way they will simply not pick that up.
That is not agile because another way to deal with that would be picking it up as is this sprint and correcting it in the following sprint. I think that the opinion of one person should never influence the entire landscape.

No/minimal number of level -1 or 1b people on team (-)

40. **How many employees are unable to deal with rapid change?**
Here as well the answer is some can and some can't. Some people have a lot of difficulty with changes -especially testers- while others have no problems at all with change and can adapt quickly.

No/minimal number of level -1 or 1b people on team (-)

Customer collaboration

41. **Is the customer willing to dedicate time to take an active role in the project?**
The product owner does, but the request owner 9 out of 10 times doesn't want to dedicate time to take an active role in the project. The request owner should play an important role since he's the one requesting the change. In practice the request owner actually throws it over the fence and basically says: I'll find out what you'll deliver at the end, keep me posted. That's not the commitment you would expect from a request owner.

Customer commitment to work with developing team (-)

42. **Is the customer contract reflective of evolutionary development?**
Since the customer is actually a department in the same organization there is no customer contract. There are KPIs [Key Performance Indicators] about that every 3 weeks something new needs to be released and if we don't make that delivery deadline then we will be judged by that.
Previously we had big releases and believe me, the business wasn't happy we only delivered something after half a year of development.

Customer contract reflective of evolutionary development (+)

43. **Is the customer immediately accessible?**
This depends on the customer because we have a lot of different customers and his accessibility depends on the person. Often the request owner is also someone who is extremely busy and if he's very busy then it would often happen dat he says he has no time that week for the agile team.

Customer immediately accessible (+/-)

44. **Does the customer contract revolve around commitment of collaboration?**
No, as far as I know there are no KPIs or contract for the customer.

Customer contract revolves around commitment of collaboration (-)

45. **Do the developers and customer frequently interact face-to-face?**
There's the product owner by proxy, who is actually the representative of the business. When I'm talking about the concept of the customer I'm actually talking about the request owner, the person who requested to apply a certain change in the landscape. In my experience that person is very little present because he feels that he has delegated his responsibilities to the product owner by proxy and has transferred the work to the agile team.
The current debate is about whether the product owner by proxy can represent the customer well enough because how can a product owner determine the right priorities for all processes in the landscape and make the right decisions?

Frequent face-to-face interaction between developers & users (-)

46. **Did you think of any other problems with the agile software development process of the B2C department during the course of this interview?**
No.

B.2 Interview 2

1. **What is your job at [company] and how does it relate to the agile software development process of the B2C department?**
   I am application consultant and functional consultant in the area of SAP IS-U.

2. **What are in your opinion the most important problems with the agile software development process of the B2C department?**
   I think the most important problem is the costs model, the way we pass on costs to the customer, the IT as cost center, which basically means that we get money and don't make a profit. All the money we get we put in IT. However, what we're currently seeing is that the business side is still working project based which means that a budget is allocated for a specific project and subsequently it's intended that somewhere along the way the budget is transferred to the agile teams. However, the agile principle is: we have a certain capacity during this period and everyone -so all customers- pays the same amount. That's not the way it's currently going, now people are saying: I have a budget for this project so I want people to work on it. According to the agile theory you should not do that and in practice we're seeing that that doesn't work very well because you're quickly regressing to project teams, which is what we're doing now. I think it's unfortunate that we are creating project teams instead of agile teams.

Embrace change to deliver customer value

7. **Can developers make decisions on their own without explicit management approval?**
   What I'm experiencing is that the team can make many decisions on their own but in practice the priority definition is often overruled by a manager because an arrangement has already been made on another level which the team didn't know about. I've already seen this happen quite a few times. I think this is just a process we have to go through. In practice it occasionally happens that a manager from the business side has already made a deal, didn't communicate this and then tells the team what needs to be picked up in a specific sprint. For example, in the NTA project, suddenly the deadline was July 1st and that deadline had to be made at all cost because the agreement with the business was already there so 5 teams were needed, working overtime, evenings and weekends simply to ensure that the promised functionality could be delivered by July 1st. The teams couldn't say that the didn't want to do that because there was a fair amount of pressure from the business to make it happen.

Low process ceremony (+/-)

Plan and deliver software frequently

8. **Do customers, developers and managers plan together?**
   No, actually this is done by the customer, what you see is that in the background there are already agreements on management level, but basically the planning is made by the customer, that's what the business board is for. The contribution of the developers in this is that when they see technical dependencies or a certain order in the things they need to develop then they will point it out, it's the responsibility of the developer to do that.

Collaborative planning (+/-)

Human centric

22. **Is there frequent face-to-face communication between team members?**
    Yes, I'm seeing this happen more and more, it takes some getting used to because a lot of people were used to sending a quick email but I'm trying to coach people to make sure they just do it face-to-face or make a quick phone call so that at least they have an answer quickly. This is done more and more so I expect that this will be fine.
    There's also the teleworking program we have -"anders werken" it's called- which means you're not here for 2 days a week and these are different days for all team members so there's always a few people on the team who are not available on location and the threshold for communication is just a little higher because you have to consciously call someone on Communicator.

Frequent face-to-face communication (+)

23. **Are all the developers are in a common room, furnished to facilitate the agile process?**
    If you're working working from home you're not close to the other developers but apart from that, 2 days a week all developers are here so then everyone is sitting closely together. In fact, there a 3 people who sit at the same desk, there are 3 desks with 6 people in total sitting at them, which is not good but is necessary due to lack of space. At least we're all sitting closely together and if you're working at home you're just one button click in Communicator away so that all works fine.

Ideal agile physical setup (+/-)

Technical excellence

24. **Are developers using coding standards?**
Yes, there have been coding standards for quite a while and every developer here knows them. A while ago maintenance checked the conformance to coding standards of existing code but the costs for that checkup were very high and they found very few problems. Currently each team has a colleague -I happen to be that in our team- that knows all coding standards and overlooks the work of the developer, you can't have a better quality check than that.

Coding standards (+)

25. **Are the team members using knowledge sharing tools?**
Knowledge sharing tools are available and are used for transferring knowledge but every team has their own wiki site or their own Sharepoint site so the information is spread over a number of places. After development the changes are documented and that documentation is put in GForge -that's another knowledge sharing tool- and all other teams can access and view that documentation so when they are going to continue development they will firs read about what's currently there and how it looks, that information is available in the documentation. So knowledge sharing tools are used quite a bit. GI documents on wikis but other than that the same story applies.

Knowledge sharing tools (+)

26. **Do employees volunteer for tasks instead of being assigned to them?**
Well, what you often see in practice -and that is different from the agile theory- is that tasks rely on rather specific knowledge and expertise. That means than when you hire a GI developer he has accumulated, i don't know, 10 years of GI knowledge and experience. That's not something about which you can say to the team: in 2-3 sprints we can all do the same he can, that just doesn't work. So his specific skills require him to work on all stories that are GI related. So in that sense he can't really choose what he wants to do, it just depends on someone's skills. You can still do a little T-shaping because everyone in the team can have global knowledge about an application but once you have to develop something it's almost impossible to do that without detailed knowledge.

Task volunteering (+)

User stories (+)

31. **Are the developers successfully refactoring code?**
No, we don't really do that. All changes we make are related to requests for changes in functionality. We don't really do IT driven development at all since it's not a priority for the business.

Continuous improvement (-)

32. **Are the developers successfully writing unit tests during development?**
Yes, of course it varies from person to person but if I look at our team then I'm seeing some very extensive unit testing. This means that testers really have to search for things they can still test because we already covered most of it. This means the testers can focus on the really rare exceptions.

Unit tests (+)

33. **How many team members have a level of software development method understanding and use which enables them to tailor a method to fit a new situation?**
The steps in the process are really solid, which is good in my opinion because without prioritization from the business you don't know if what you're working on is actually the right thing. So the process is just a few steps that should be followed. In terms of requirements, you'll see that many people still follow the "old" way: a business support engineering will talk with the customers, then a list of requirements is produced which are given to the agile teams. For all requests I'm involved in, I try to do things differently. I begin with a session with the business, the business support engineer, the people of the team that will be involved; the tester, a developer etc. Then I just ask the business again what they want to have exactly. Every time I've done that new requirements have surfaced, or ambiguities that the business needs to deliberate about. So you can see we still have to grow in this aspect.

34. **Are developers willing and able to take an agile approach to documentation?**
Yes, if I look at my area we do. We just create functional and technical designs. GI has wikis where they put their information and I think that works well. As far as I'm seeing it's all going well but you only miss something when you need it but can't find it of course but this hasn't happened yet.

Agile documentation (+)

35. **Are the developers successfully using test-driven development?**
We don't do it yet but [company] wants to implement it because they want to rise in agile maturity.
I'm seeing some disadvantages of test-driven development and I don't know whether we should really want to do that because you would be focussing on very specific situations. However, at [company] currently the biggest problems are exactly in the odd exceptional situations you didn't think of. Those are precisely the things yo can not think of beforehand and the business can't think of these situations either so if you're immediately going to say what things we should be testing that would probably result in around 50 percent of the cases we would have normally done so that's only half, the other halve usually comes gradually during development and preparation. We really need that time of preparation and such and you can't just say well we're going to see what test cases we think we should have and after that we know how the program should work. I think the situation is a little more complex and therefore test-driven development is not really going to work.

Test-driven development (-)

36. **Are the developers successfully using pair programming?**
We don't do it yet but [company] wants to implement it because they want to rise in agile maturity.
I think pair programming will happen but it currently isn't often done, partly because of pressure to finish things and pair programming is still considered ineffective, 2 people doing 1 thing, that can't be efficient. In the long run you will be more efficient but long-term vision is also something that's lacking at [company]. [company] is very good in putting out fires in the short term. When you would propose pair programming to a team then everyone would say they wouldn't want to do that because it's not efficient at all, that would mean I'm only sitting there, watching somebody else make something. Then I think, very good, that is exactly the point and then you can do it yourself in a couple of weeks, plus you prevent some mistakes, spark some discussions and help create better structure so yes, I do believe in pair programming but it currently isn't done, partly because of the time pressure and the belief that it's inefficient.

Pair programming (-)

37. **How many employees are unable or unwilling to collaborate or follow shared methods?**
There are some but in my current team there are none so that's nice. In my last team there were two of them.
You have varying degrees of course, you have people who are aware of this and for example say: agile is nice but I'm external and I'm hired here for SAP so I'll work on just SAP. The percentage of such people is not that large, I think 5 to 10 percent. In addition, you also have a big group that just leans back and thinks: you want me to do this that way so I'll do it that way but those people show very little initiative regarding agile. Those people are just along for the ride and are fine with that but they are not really the pioneers of agile. I think this is a fairly large group, I think about 70-80 percent.

No/minimal number of level -1 or 1b people on team (+/-)

38. **How many employees are unable to deal with rapid change?**
Most employees are able to deal with rapid change. You hear the occasional complaining of course when something has been changed again but most of the can handle it and are pretty flexible. I think that it's not that much of a problem at [company] because we work with a lot of external hires, these people already have quite a flexible nature because they are already flexible in terms of work and location. They don't really care who hires them and what they'll have to do so maybe that plays a part because rapid change does not really affect us.

No/minimal number of level -1 or 1b people on team (+)

Customer collaboration

41. **Is the customer willing to dedicate time to take an active role in the project?**
That depends on what you consider an active role. If you call them with a question then they will answer it, although this might take a day.
However, for example for the NTA project I told them I want business people here on the floor because we had a lot of time pressure, only 3 weeks, 1 sprint in which we had to do a tremendous amount of development work so we really didn't have time for delays or to wait for the answers to questions, etc. That's why I told them I wanted business people here on the floor. Finally this didn't happen because they could not or did not want to allocate time. So that shows that the business still needs to improve their agility

Customer commitment to work with developing team (+/-)

.

43. **Is the customer immediately accessible?**
    It depends but what you often see is that they don't immediately know the answer to a question. So if you pose a question then they will have to discuss it with 10 other people first before they can make a decision. So you can see that there isn't really an agile mindset there. The decision power is there because they are the ones who take the decisions, however they only take the decision after discussing it with 3, 5 or 10 people. It seems like they first need to approach a lot of people before they dare give an answer so there is definitely a loss of time. They are certainly accessible, if you call them they will pick up their phone.

Customer immediately accessible (+/-)

44. **Does the customer contract revolve around commitment of collaboration?**
    No, not currently. In fact, their own managers just say that they'd rather have that they sit there and answer the questions by phone. They always ensure that those people don't have to sit on our floor because that would hinder them in their daily work they have as well.

Customer contract revolves around commitment of collaboration (-)

45. **Do the developers and customer frequently interact face-to-face?**
    I would prefer the customer to be here, if you are working on something for him, let him sit here but I've noticed that that's still a bridge too far for [company].

Frequent face-to-face interaction between developers & users (-)

46. **Did you think of any other problems with the agile software development process of the B2C department during the course of this interview?**
    No.

B.3 Interview 3

1. **What is your job at [company] and how does it relate to the agile software development process of the B2C department?**
   I am tester in one of the agile teams and also the scrum master of that team.

Embrace change to deliver customer value

3. **Are management and developers willing and able to reflect and tune the process after every iteration and release?**
   Yes, we have a retrospective every sprint and each time this results in actions which we try to pick up.

Reflect and tune process (+)

4. **Are managers and developers willing and able to deal with evolutionary requirements?**
   Basically, very small changes we sometimes accept but if there are really big changes then we might stop development or tell the customer to create a new story which we will pick up next sprint.

Evolutionary requirements (+)

5. **Has the customer the power to dictate the scope of the iteration?**
   No, but they of course determine what gets on the backlog so in that sense they do dictate what we pick up, they can indicate what has the highest priority but they can't tell us what we have to do next sprint, that's for the team to decide.

Client driven iterations (+/-)

Maintain a list of all features and their status (+)

6. **Is the customer encouraged to continually give feedback/criticism and rethink their requirements during the development process?**
   Yes, that occasionally happens. Of course you often start when 80 percent of the requirements are know and 20 percent aren't. When those 20 percent become known that's of course okay but when that 80 percent starts to change then we usually estimate as a team how much work the change will be and whether we want to do that. Of course we want to think along with the business but if it's a really big change then -like I said earlier- we stop working on it if it makes no sense to continue of we continue building it as it was originally designed and tell the customer to create a new epic for the change request.

Continuous customer satisfaction feedback (+)

Plan and deliver software frequently

8. **Are management and developers willing to use an incremental and iterative development approach?**
   I can't really judge from a business standpoint but I can imagine that not everyone from the business is happy with it. Of course we are on the IT side and we all work this way so I think that we are all pretty happy with it but it's difficult to gauge what the business thinks of incremental and iterative development.

Continuous delivery (+)

9. **Is management willing to commit to the process of continuously planning instead of developing a one-time plan upfront?**
   On the business side they run projects and for that they undoubtedly have a certain planning but on our team we prepare what's coming and we play per sprint so for three weeks what we pick up so we don't plan far ahead.

Planning at different levels (+)

10. **Do the developers and managers agree to have risks drive the scope of each iteration?**
    No, usually in that case it has a high priority so we only pick it up earlier, but we don't pick up less. Something that's more complex gets more poker points so we don't pick up fewer poker points but we do pick up less different stories for the same amount of story points.

Risk driven iterations (-)

User stories (+)

11. **Is the high-level agile planning based on features instead of tasks?**
    Yes, that is mainly done by the business, they are running the projects and planning the features.

Plan features not tasks (+)

12. **Is management willing to maintain an up-to-date list of all the remaining features for the project and their status?**
    We have people from multiple fields in our team and sometimes the backlog is too small or not enough differentiated, so there's a lot of work for people in 1 field but too little work for people in other fields, so there's not an even work balance for the team members so that's pretty unfortunate but usually working with a backlog works well.

Maintain a list of all features and their status (+)

13. **Does the team deliver a fully functional release after every iteration?**
We will always release something at the end of a sprint but we of course commit to building multiple things and it sometimes happens that we don't release something that we committed to but we always release something.

Smaller and more frequent releases (+)

14. **Is management willing to plan immediately before the iteration instead of earlier to allow the customer's feedback to be incorporated into the planning?**
No, we get feedback from the customer but it doesn't influence our planning.

Adaptive planning (-)

15. **Are developers estimating the effort and duration of various user stories themselves?**
Yes, not only do the developers estimate but the entire team participates so that includes testers and that works well.

Agile project estimation (+)

Human centric

17. **Are people willing to work in teams and help others?**
Yes, I think that's one of our strengths. I think that works very well.

Collaborative teams (+)

18. **Has management empowered teams with decision making authority?**
Yes and no, as a team you'll make decisions but there are always delivery managers that have ultimate decision power so in the end the decisions our team makes are never final.

Empowered and motivated teams (+/-)

19. **Are team members motivated?**
Yes, certainly.

Empowered and motivated teams (+)

20. **Does management agree to have self organizing teams?**
No, we don't have self organizing teams.

Self organizing teams (-)

Technical excellence

28. **Does a mechanism exist to monitor the iteration progress?**
Yes, mainly what's on the scrum wall, the epics and stories that have been picked up and the burn-up chart we make based on that.

Tracking iteration progress (+)

User stories (+)

29. **Is design a continuous process or done once at the beginning of the development process?**
Yes, it's mainly a continuous process.

No big design up front (+)

30. **Are the developers willing and able to use continuous integration?**

No, we don't currently do that, we do our builds and tests manually. I know they are currently trying to implement it for regression tests.

Continuous integration (-)

35. **Are management and developers willing to meet daily to discuss the progress of the project?**
Project managers have a daily meeting with product owners but they don't have a daily meeting with developers.

Daily progress tracking meetings (+)

46. **Did you think of any other problems with the agile software development process of the B2C department during the course of this interview?**
No, not really.

B.4 Interview 4

1. **What is your job at [company] and how does it relate to the agile software development process of the B2C department?**
   I am a delivery manager, I map the demand on the supply. I make sure that we work conform the priorities set by the business board and I connect the teams, the capacity on the back end to that. I also do the overall escalations so when decisions about the availability of the system have to be made I do so because I'm also owner of the OT environment. I also make sure that I keep out the hassle that goes on outside of the team as much as possible.

2. **What are in your opinion the most important problems with the agile software development process of the B2C department?**
   The first one is that planning is pretty difficult because we are dealing with a quite often changing prioritization. Last time two weeks before the sprint it became known that the deadline was Juli 1st so that changes everything. The constant rescheduling brings a fair amount of unrest on the back end. Another point is of course what we don't have a high level portfolio that describes what is about to arrive and how full our capacity is.

   Adaptive planning (+)

   Embrace change to deliver customer value

3. **Are management and developers willing and able to reflect and tune the process after every iteration and release?**
   Yes, we do have a retrospective. The result doesn't have to be improvements to the software development process, it can be general improvements, it doesn't by definition be related to the software development process. In my opinion the retrospective is meant to get improvements in productivity, quality, the three pillars basically and whether those improvements are for the software development process or for example the team, that doesn't matter to me.
   The retrospective is a precondition, the last sprint there wasn't a retrospective because of a lack of time, well I think that's a mortal sin, I think we should begin each Thursday at the end of the sprint and the start of the new one with a retrospective.

   Reflect and tune process (+)

4. **Are managers and developers willing and able to deal with evolutionary requirements?**
   You know what, I think that's the fourth one from the agile manifesto: "Responding to change over following a plan". Yes and no, let's say it this way: we would actually want our features that need to be built known two sprints ahead of time which allows the teams to compose their sprints from a variegated backlog. If you only have a backlog for one sprint for example then it's possible that there's a lot of GI work and very little SAP work due to which the team can't be optimally used so I certainly don't want it a month ahead or two months because you have to build what the business currently wants but if you want to get efficiency then you have to make sure that you have at least 1.5 or 2 sprints of backlog.
   Do you want a change within a sprint? Actually by definition no since at the moment you commit to the contents of a sprint that's basically the piece you are going to deliver but in my experience in practice the business does change things during the sprint and if you ignore that and tell them we don't change anything during the sprint then you deliver software but the software doesn't do what the business wants so the principle is that we don't change anything during the sprint but in practice it does happen. However in that case it only happens provided I and others know about it because we don't want people to change everything under the radar and the business must know that the need to think carefully about what they want and not want to change everything at the last moment. If the customer wants to change something it usually isn't only a quarter turn, they really want a 180 degree turn the other way which completely changes the IT solution.

Evolutionary requirements (+/-)

Maintain a list of all features and their status (+)

5. **Has the customer the power to dictate the scope of the iteration?**
   They do. Through the backlog they have they determine the scope of the sprint.

Client driven iterations (+)

Maintain a list of all features and their status (+)

7. **Is the customer encouraged to continually give feedback/criticism and rethink their requirements during the development process?**
That's exactly the problem and it's in line with what I said previously, I think the business commitment varies so we get more commitment from one customer than from another. We do have a mandatory demo, so on the O or T environment -that will be predominantly T because we can show an integral solution there- we want to show a demo directly to the business and within that demo there is time -because we do that as early as possible- for the demo to put the finishing touches on it. If they see a letter there and there is a word they want to change then we can quickly change it while sitting next to them and ask them what way they want it. We want the demo to be as early as possible so that the customer feedback is as early as possible on what we are going to deliver.

Continuous customer satisfaction feedback (+/-)

Plan and deliver software frequently

8. **Do customers, developers and managers plan together?**
You should actually prioritize from agile on the lowest possible level and that's the epic. However, our business prioritizes on project or on theme within a project, well that are usually big things so if you subsequently put that into a planning, which you then have to use which doesn't work on a project level. NMM is an example which has a lot of GI work but if you don't prioritize and plan anything for other fields then there wouldn't be any work for the SAP people so -and there's the crux- you have to translate from prioritization from the business on project level to a piece from delivery in which you will see things that not always correspond with the prioritization made by the business board which has everything to do with using our capacity.

Collaborative planning (+/-)

10. **Is management willing to commit to the process of continuously planning instead of developing a one-time plan upfront?**
Planning is a continuous process because today may turn out on the last day of the sprint that something could not be realized which leads to overflow to the next sprint. You could have a planning moment but that wouldn't work because during the sprint you start to see more clearly what will be delivered and what won't which influences what we are going to do next sprint and then there's also the last Thursday when the definitive planning for the next sprint is made but that takes a lot of interaction in the three weeks prior.

Continuous planning (+)

11. **Do the developers and managers agree to have risks drive the scope of each iteration?**
    Our principle is: one epic, so the smallest thing, should be such that it can be built, test, demoed, all in 1 sprint, well we are occasionally seeing some problems with when they get bigger so they require 2 sprints. Another point is the particular risks in the area of CI [Configuration Item] locking, so we know from what is requested which CIs, which configuration items we use so which software components are going to be changed and the issue is that the longer you are keeping something in a sprint, if something takes multiple sprints then that CI is locked for a longer period of time for a certain part and then it's not usable for another part so that's a risk that has to be looked into and then it's always the one with the highest priority which initially creates the lock.

Risk driven iterations (+)

Software configuration management (+)

12. **Is the high-level agile planning based on features instead of tasks?**
    In the end the planning is created per team on story level. A story is basically a part of an epic so you could see that as a task because what we're currently doing is that an epic is always associated with a chameleon team. A chameleon team can handle all aspects so what you'll see is that to realize something end-to-end you need something from streamserve, something from SAP, something from GI, a little bit of test work. Well this is all worked out in stories, tasks, under an epic and then the entire team starts working on it so the planning is made by the team on story level.

Plan features not tasks (+)

User stories (+)

13. **Is management willing to maintain an up-to-date list of all the remaining features for the project and their status?**
    We are dealing with an often changing backlog and the problem is -and I think that is one of the issues that using agile brings with it- we have a complex organization with multiple business units that are on the same backlog. What you would want ideally is that you have one product owner on one team that simply prioritized on epic level however here they prioritize on a much higher level, on project level so there's the crux, you would actually want that prioritization is on epic level but I don't think you could ask that from the deputy directors, who are absolutely not going to be planning on that level so that's a bit related to agile. I think agile is basically if you have one team, one product owner, one bag of money, I think that is the most ideal situation, we currently have one bag of money but that are actually 3 bags of money divided by 3 business directors that need to compete over priorities in a set amount of teams, which is difficult.

Maintain a list of all features and their status (+)

14. **Does the team deliver a fully functional release after every iteration?**
Well, like I said, ideally we would have one epic within one sprint and yes, we do release working software after every sprint but it certainly happens that we bring things technically live because it simply needs multiple sprints to get something fully functional and so it is deployed but it is for example never invoked so it's technically live and not functionally. So yes it does happen that we release non-working software but there will be no sprint in which we don't deliver anything functional at all because we are so big that we can deliver multiple things every sprint.

Smaller and more frequent releases (+)

15. **Is management willing to plan immediately before the iteration instead of earlier to allow the customer's feedback to be incorporated into the planning?**
Feedback is requested from the customer, we do have a post implementation review but that doesn't involve feedback on the planning because there's continuously feedback on the planning. The product owner by proxy -the business representative- is certainly at the sprint retrospective but it happens that a team delivers work in a sprint for 4-5 business units -who are the request owners here- and those request owners are not present at a retrospectives but they do give their feedback on cooperation, quality and applicability of the functionality we deliver through a post implementation review on Jira, this could be feedback on a team, a product or whatever and that feedback is used but not really for the planning. The product owner by proxy is a business representative who basically guards the prioritization for the business. They ultimately decide what the team will do but it isn't the real user, they are actually process consultants of the business who do this here, this is related to the diversity of business units we have here. Like I said, ideally we would have one business representative who is responsible for the budget but that's the rift between IT on the one hand and business on the other, the business needs to want and be able to allocate people for that, well the actual manager isn't going to do it, I'll tell you that.

Adaptive planning (+/-)

Technical excellence

28. **Does a mechanism exist to monitor the iteration progress?**
Burndown gives the complication that you have to extend the Y-axis at the underside of the chart, which is very unpractical so we have burnup charts which show how far along in a sprint you are and I must say that this is strongly dependent on the team's ability to cut up tasks in small pieces so they can burn up small amounts.
What you will often see is that if a team is not capable of doing that, the burnup is very even in the beginning and at the end you'll see it quickly rising so this results in the fact that you know your liability only at a very late point in the process which can create quite a few problems but basically the mechanism is simply the standups and I make rounds. I simply ask random team members what their progress is, so I ask a tester or a developer how it's going and then you get a feeling what the progress of the sprint is. You would want to measure it by that burnup but like I said, the burnup isn't very reliable until the very end. We have a reliability which must be higher than 95 percent so from everything we commit to at the start we have to deliver 95 percent of at then end which goes pretty well so based on that we have a little certainty that the teams will actually deliver what the promise but the problem with the burnup charts is that you won't see a lot in the beginning and then suddenly it goes up. That's because things are often tested later, writing a technical design and a functional design is part of our definition of done, which can be done in the regression so we say we deliver the software and then we create the documentation since documentation is less important than software but because it's a part of your burn process all those burn points are only counted at the end so that's why I pay little attention to the burn rate.
For us it's also much more important that there's a long-term trend upwards in quality, productivity -they call that your velocity-, quality, reliability. That's pretty hard because we have to deal with KPIs which are pretty black and white, if you tell a team to increase productivity then you'll get poker inflation because they have to get as many points as possible. If you tell them to increase quality then you'll see that productivity lessens. So you have this axis of reliability, quality and productivity and those three indicators you have to keep an eye on to make sure they rise evenly. If you push and focus excessively on one of them then that will have a negative influence on the other aspects.

Tracking iteration progress (+)

agile documentation (+)

34. **Are management and developers willing to meet daily to discuss the progress of the project?**
Every day the teams do a standup in which the progress and deliverables are discussed so if we are working on something than we will discuss who will deliver what and then we also know right away what the impediments are. Every day after all teams have had their own standup we have a scrum of scrums every day in which the overarching impediments are discussed, so if a team is unable to resolve an impediment on their own then hopefully they won't always wait but then can request assistance so if the team for example could deliver something extra if they had one additional testers then they can ask for help during the scrum of scrums, whether the other teams have excess capacity or knowledge about how to deal with this situation.

Daily progress tracking meetings (+)

Customer collaboration

41. **Is the customer willing to dedicate time to take an active role in the project?**
Partly yes and partly no. For example, last sprint I heard: yes, we also have other projects and operational work. Yes, that's right but if we have to deliver something on July 1st and we need you to do that then you will have to raise this to a very high priority, but that rarely happens. There is a huge language barrier between business and IT and I think that the business might not realize that IT is a core competence for them and I don't know whether we can and may expect that.
Business testers so user acceptation are now simply team members, they used to be in completely separate teams but now they are right next to us. We also have the demo that's given directly to the business people, they come to us for that but it's not like -what I would preferred last sprint as well- there are business people permanently in the teams, it usually becomes a compromise, for example two days a week, I don't know whether you can expect more from the business.

Customer commitment to work with developing team (+/-)

42. **Is the customer contract reflective of evolutionary development?**
The business has KPIs, for example about prioritization, e.g. how many times the prioritization can change.
I just had a conversation with someone from the business about the last sprint and he told me that they are not yet ready for agile so I told him: we've been doing agile for two years, how can you tell me you aren't ready for it yet? There is still a lot of progress to be made, especially with marketing, who are much less committed. We've seen that marketing does their own software development for certain things, so without involving IT. There's clearly something wrong with the perception and expectations on both sides.

Customer contract reflective of evolutionary development (+)

43. **Is the customer immediately accessible?**
Yes, by means of escalation, so if we really need answer to this question because otherwise we can't deliver this at the end of this sprint then we will escalate the question. It's very diverse, if you for example look at payment processing and back office things, they are usually more willing then for example marketing to make someone available continuously. Marketing expects to give us the specs, have us build is and they'll come look at what we've made at the end while others are more accommodating and I think see the advantages as well of working together.

Customer immediately accessible (+/-)

44. **Does the customer contract revolve around commitment of collaboration?**
No, we do have KPIs but conversely, the business doesn't have KPIs. They have KPIs about communication, provision of information -which means for example that the business boards gets the minutes- but they don't have KPIs about the mount of time they allocate to working with us or that they must have had a demo of all the software they get from us. It's difficult to measure the collaboration with a team, it differs per team and per person from the business.

Customer contract revolves around commitment of collaboration (-)

45. **Do the developers and customer frequently interact face-to-face?**
Yes, but it's hard to answer whether it's enough. It depends on the team, whether or not they are going to make and on the business. I think there's never enough face-to-face interaction. I think we could act a lot better in this and I think it's not only the supply but also the demand. We should insist on frequent interaction, we should tell the customer we will not deliver if they don't come and look tomorrow. I think where we could really improve is that currently our testers are only involved when the requirements are almost finished. Actually I would like that the testers start thinking along at the start about which scenario's they should test and when the software is successful so that you are basically using your test scripts as requirements for your software and then you are doing test-driven development, I would like to do that. However, currently you see developers and requesters communicating while I think that a lot of process and business knowledge is available from the testers and if you involve the testers, you can do a risk analysis in an early stadium, define your scripting, match your scripting with the requirements so that you'll know you are testing the right things. It's currently quite often a problem that what we are testing and what is requested do not match and test-driven development is one of the ways to prevent that.

Frequent face-to-face interaction between developers & users (+/-)

Test-driven development (-)

46. **Did you think of any other problems with the agile software development process of the B2C department during the course of this interview?**
I think we're pretty complete already.

B.5 Interview 5

1. **What is your job at [company] and how does it relate to the agile software development process of the B2C department?**
   I am SAP developer and also scrum master for chameleon 5 so I have a double role but mainly I'm a developer within an agile team.

2. **What are in your opinion the most important problems with the agile software development process of the B2C department?**
   As an external consultant I'm very used to working on a project basis and after the last sprint and all the hectics we had I noticed that working on a project basis is a somewhat smarter way of working, a little bit more focused on one thing, making sure all your tests run correctly, testing smartly, that sort of thing. In my experience in an agile way of working, due to the fact you release small components which you test individually you miss the complete picture of what you are actually making. For small changes I think agile is ideal, but for the large projects which we have here I think agile is a bit of a mismatch.

Embrace change to deliver customer value

7. **Can developers make decisions on their own without explicit management approval?**
   If it's about implementation specific details then nobody is stopping me. However, before the project is brought to the teams the entire process has to go through so many different steps that I think this is all way too bureaucratic. The same applies to communication with the business. I think business is still not used to agile. Of course you would want the business to be available semi full-time for questions and such. However, here they would like you to send an email or contact them through another person so that's all pretty bureaucratic. That [company] used to be a public company might have something to do with it in my opinion.

Low process ceremony (-)

Customer immediately accessible (-)

Plan and deliver software frequently

9. **Are management and developers willing to use an incremental and iterative development approach?**
   Yes, it has its advantages and its disadvantages but I certainly see the advantages of agile, especially because it forces you to think smarter about how you are going to do things in advance and that can make a lot of difference which allows you to create business value much earlier so it definitely has its advantages. It also has a few disadvantages but development wise I would say that agile certainly has its advantages.
   For managers there's a division between types of managers. Project managers are still thinking in projects so aren't big agile fans. However, department managers are pretty involved with the agile process so they do like the fact that we develop incremental and iterative. I think that de business is still not used to agile and likes the advantages, that they get things earlier but they dislike the disadvantages, so that it possibly is a bit less efficient. They don't really get agile and are themselves not thinking that way.
   I think that the agile teams basically know what they are doing but business and management and especially project management are not always thinking about agile in the right way.

Continuous delivery (+)

16. **Are developers estimating the effort and duration of various user stories themselves?**
   Yes, there's an estimate made in advance by -I think- IT readiness which describes how much time a request will approximately take. The agile team prepares a story and based on that we indicate how many poker points it will take and those poker points can be converted to the amount of work we will pick up in a sprint so then you'll know how much time it will approximately take.

Agile project estimation (+/-)

Human centric

17. **Are people willing to work in teams and help others?**
   On average people are definitely ready to help each other, especially within teams. Like I said, the communication with the business could be improved in my opinion but within the teams people certainly help each other out a lot.

Collaborative teams (+)

18. **Has management empowered teams with decision making authority?**
    In certain areas they have, such as the order of delivery but as soon as it's about whether or not we are going to deliver something or a deadline we're not going to make then usually management gets involved to tell us that we have deliver everything in time and to make sure everyone gives their best to make sure that we manage to get it done. But usually we can decide many things ourselves about the daily affairs.

Empowered and motivated teams (+/-)

19. **Are team members motivated?**
    It differs, a number of people are a little less comfortable working with agile and need a little bit more motivation but on average I would say that more than half of team members are pretty motivated.

Empowered and motivated teams (+/-)

20. **Does management agree to have self organizing teams?**
    Yes, basically the teams are self organizing. Management -as in the immediate manage layer above the teams- is not really involved in that. Organization is done in the team itself, mostly by scrum masters in cooperation with the rest of the team. This involves how they organize things, when people will work remotely, where they are going to sit, etc. This basically all works well. We don't actually have a lot of difficulty with it but I think you shouldn't be too strict in this aspect.

Self organizing teams (+)

21. **Do employees feel comfortable working in self organizing teams?**
    As far as I know they do like it.

Self organizing teams (+)

22. **Is there frequent face-to-face communication between team members?**
    We try to do as much as possible face-to-face and that's usually works best and it works definitely much better than through emails and such, that's an advantage I see of agile, people are now talking with each other more instead of just sending a quick email. The development and test departments are now setting together, that used to be two separate departments but this way is much easier because now everyone knows exactly what everyone is working on and there is a lot more cooperation.

Frequent face-to-face communication (+)

23. **Are all the developers are in a common room, furnished to facilitate the agile process?**
People are actually asked here to work two days a week at home due to the occupancy of the building. I personally don't do that, I'm actually here 90 percent of the time. A number of people in my team does do that and except from the fact that there are sometimes some small technical issues with teleworking we do have technical resources to our disposal such as Communicator with which you can use a webcam to call each other. Okay, it has its occasional hiccup but it really helps for communicating with people who work remotely. Still, I think that normal face-to-face communication works much better.

Ideal agile physical setup (+/-)

Technical excellence

24. **Are developers using coding standards?**
There are coding standards but they are from a pretty distant past, they are not used very actively but I know that currently a number of people are busy to start using coding standards again and create new coding standards. In principle I as a developer am basically in favor of coding standards but a little bit more pragmatic, certain solution guidelines are fine but I don't think standardizing all naming and such is going to work.

Coding standards (+/-)

25. **Are the team members using knowledge sharing tools?**
Within TIPCO as far as I know they do, they use a wiki, everything they have built is on there and a lot of other stuff as well. Within SAP development we currently have 4 or 5 people sitting here and I have a reasonable amount of contact with 4 of them, we talk a lot with each other but we use few to no tools to share knowledge.

Knowledge sharing tools (+/-)

26. **Do employees volunteer for tasks instead of being assigned to them?**
No, since we only have one developer within a team it means that automatically all development work ends up with that developer. However, I've noticed lately that if I'm too busy then I can ask a colleague of mine who sits here as well to help me and he will pick up some things for a while. We exchange work from time to time, which is on a purely individual basis: Hey do you have some free time, then I'll help you for a bit. So basically we can decide that for ourselves if there is time.

Task volunteering (+/-)

27. **Does the organization have tools for software configuration management?**
    It's a very large area, SAP doesn't have many problems with it because it does a lot of software configuration management internally so a lot of things are already covered by the transport mechanism which also ensures that two people don't work on the same item and such so basically SAP does need little to no external software configuration management tools.

Software configuration management (+)

29. **Is design a continuous process or done once at the beginning of the development process?**
    The high level design is created before the sprint, that's on a high level how the process is roughly supposed to go but the design -including documentation- is actually done during development in the three week sprint.

No big design up front (+)

30. **Are the developers willing and able to use continuous integration?**
    That's actually a standard mechanism within SAP so you don't actually have to do anything. When we make a change, that will go into what's called a transport file, which will be automatically transferred to the test system and compiled after which you have your actual new build.
    At TIPCO they usually communicate to all testers and as far as I know they must manually install a new build on the target system.

Continuous integration (+)

31. **Are the developers successfully refactoring code?**
    I would love to do it and when I'm working on a big change then I will do it but I won't do it all the time because you can't guarantee afterwards that the code works the same so you would have to retest everything, which could of course be solved by automatic testing.

Continuous improvement (+)

32. **Are the developers successfully writing unit tests during development?**
    I think we have a pretty limited development system in which we can test only a little but the amount of testing we do differs per change but we create a fair amount of unit tests with wich we try to cover a reasonable basis. For the more difficult cases and integration you'll go to the next system but we'll do a few unit tests for each change.

Unit tests (+)

33. **How many team members have a level of software development method understanding and use which enables them to tailor a method to fit a new situation?**
If we look at the previous sprint in which we actually did a small project, that's really exceptional, you wouldn't usually be able to do that. Usually we do normal changes during a sprint and require large chunks to be cut up in to smaller pieces. Possibly we'll go what they call "technically live" which means you build something and pretend to bring it to production but it doesn't do anything and you will later continue working on it within agile.

35. **Are developers willing and able to take an agile approach to documentation?**
I think we keep ourselves to the relevant documentation. I'm talking about the functional design and the technical design of how it all fits together and I think that's the minimal basis and apart from that little to no useless documentation is created.

Agile documentation (+)

36. **Are management and developers willing and able to use user stories?**
I think that the user stories should come from the users but it's often the case that we have to create the user stories ourselves, they will tell us what they approximately want which is a pretty big request so we have to cut it up into smaller pieces and that finally arrives at our desks and that shouldn't be the way it works because actually the business should create small stories themselves.
We do not do something for one department, we do something for for example twenty departments so it's very difficult to appoint one user who should design it and that's the problem but I still think that one of the layers between the business and the developers should be a little bit more clear and should create the stories based on functionality they want because now the stories are usually technical in nature.

User stories (+/-)

37. **Are the developers successfully using test-driven development?**
No, we don't actually do that here and I must say that I'm not a huge fan of it myself. I think that in that case you would program with the goal of making these few cases work well and there's too little thought about the functionality, it's reasoning too much from the developer's standpoint and not from the point of we want certain functionality so I think that's a disadvantage of test-driven development.

Test-driven development (-)

38. **Are the developers successfully using pair programming?**
We do it sometimes but mostly for the short, complex things. It's not like we sit next to each other all day but we do do short code reviews but never real pair programming all day long, that's done very little.

Pair programming (+/-)

39. **How many employees are unable or unwilling to collaborate or follow shared methods?**
I can't say that's a problem.

No/minimal number of level -1 or 1b people on team (+)

40. **How many employees are unable to deal with rapid change?**
Most developers are able to deal with rapid change but I occasionally think that the business is abusing that fact by requesting a lot more things during the sprint or changing requirements during the sprint. Up until 30-40 percent of the functionality should be able to change a little bit while working on a sprint but there's a certain base that needs to be solid. Sometimes it's like let's do A, oh no we're going to do B instead and yews that means we have to start over again if you are in a sprint so then you have a problem so I think that most developers can handle rapid change, sometimes there's a little resistance, especially in the area of letters, where they mess a lot with the requirements while you are almost finished so they first want to see something and they the tell you they want it a different way and then you have to change it, which causes resistance but in most cases you try to deal with it as best you can.

No/minimal number of level -1 or 1b people on team (+)

46. **Did you think of any other problems with the agile software development process of the B2C department during the course of this interview?**
No, like I said at the start I think that it's mostly the integration between business and IT which could be improved a little and I think that should come naturally if the business was a little bit more involved with what happens in IT and vice versa, I think that would make the agile software development process a lot more successful.

Customer commitment to work with developing team (-)

B.6 Interview 6

1. **What is your job at [company] and how does it relate to the agile software development process of the B2C department?**
   I am a business process consultant, we are the ones creating the epics. I'm also gatekeeper for billing.

2. **What are in your opinion the most important problems with the agile software development process of the B2C department?**
   An often occurring problem is that you have a requirement but completely specifying it is difficult for the business because that requires a level of IT understanding. It sometimes happens that you are specifying requirements but IT interprets them differently and because of that the end result is something unexpected or you don't mention a requirement because it seems obvious to you from a business perspective. A good example of this is the usage feedback, customers who own a smart meter are given a bimonthly usage feedback which includes charts. Only the customers with return supply would get a chart for consumption feedback (CMB) because there is no data supplied for the rest. However, the developers created a chart for all customers. For me that requirement is obvious, because a customer that doesn't get a restitution does not get a chart while IT says that we supply NULL data and NULL is also a value so they make a chart. So then you get the discussion: was this built as designed or was the requirement not clear. That's the kind of problems we are facing.

Embrace change to deliver customer value

3. **Are managers and developers willing and able to deal with evolutionary requirements?**
   We always try to submit changes during development as well but we often experience that the agile teams are of the opinion that the requirements are final. Once a specific solution is chosen then it's difficult to deviate from the chosen path, while I think that agile should be about stopping when we're on the wrong path and starting again with another solution. I miss that in the agile software development methodology of B2C.

5. **Has the customer the power to dictate the scope of the iteration?**
   No, the agile team determines the scope of the iteration based on the amount of work they think they'll be able to do so the amount of poker points they think they'll be able to process. However, as the business we can indicate the priority but what will be delivered at the end of a sprint is determined by the agile teams.

Client driven iterations (+)

6. **Is the customer encouraged to continually give feedback/criticism and rethink their requirements during the development process?**
   Yes, we try to give feedback but it is hard to stay connected from the business because for us it is only a small part of our job while agile teams are working on it for the entire day. We are not always available on the right moments and we don't always have the time to answer questions directly. However, we did voice our intention to answer an issue within 4 ours but this is not always possible.
   I've been a product owner by proxy for half a year at which team I was a real part of the agile team and because I know the business processes it was pretty easy for me to arrange things but that job is not sustainable because you basically have 2 functions at the same time. That's currently a bit of a handicap, because we don't know exactly who has which role within the agile process. There's the gatekeeper -that's me currently- who decides whether a release goes live, there's the product owners by proxy who represent the business but in my opinion they side too much with IT so they actually represent the interests of IT and then you have the business solution engineer that creates the stories for us.
   They want to change some of this but we think that's not workable because we business consultants don't have the time or the knowledge to create all user stories.

Continuous customer satisfaction feedback (+/-)

User stories (+)

Plan and deliver software frequently

8. **Do customers, developers and managers plan together?**
   Yes, on a high level we do.

Collaborative planning (+)

15. **Is management willing to plan immediately before the iteration instead of earlier to allow the customer's feedback to be incorporated into the planning?**
   Yes, our feedback is certainly used. We have discussed some things but often there's a lack of communication. I don't know how to improve this because we can't fulfill the needs of the agile team.

Adaptive planning (+)

Customer collaboration

41. **Is the customer willing to dedicate time to take an active role in the project?**
We are certainly willing but because of everything that's happing it's just not feasible to take an active role in the project. You have to understand that we have a fulltime job and next to that we try to answer all issues from the agile team and clarify requirements.

Customer commitment to work with developing team (+/-)

42. **Is the customer contract reflective of evolutionary development?**
We're quite happy with the concept of evolutionary development. We often get demos of what is built and what is being delivered. This is different for each agile team and also depends a bit on what is being built.
The business involvement could certainly be better but considering the amount of time it takes I sometimes rather use the time to work on some other things instead of going to a demo. I would expect the product owner by proxy to play a more prominent role, he should really start representing the business side more because I think that I'm currently not having enough contact with the product owner by proxy. He knows exactly what is being built by the agile team and which issues did arise. I would like to have a weekly meeting of an hour with the product owner to discuss the progress of the agile team.

43. **Is the customer immediately accessible?**
I've sometimes had the request to come sit close to the agile team. I did that a few times in the last few months but I only got 2-3 questions a day and the downside is that I'm sitting in a hectic environment and doing my other work is more difficult.
I can be reached by phone, email and Communicator, that should really be sufficient for the agile teams.

Customer immediately accessible (+)

44. **Does the customer contract revolve around commitment of collaboration?**
No, we only voiced our intention that we would try to answer issues within 4 hours. We strive to do that but it isn't always possible because sometimes you're not at the office and as a result don't look at your email that much, making responding in time a challenge.

Customer contract revolves around commitment of collaboration (+/-)

45. **Do the developers and customer frequently interact face-to-face?**
Yes, often when there are issues we sit together to discuss things but actually more time should be freed up for that. However, we don't have more free time so that's really the bottleneck here.
The agenda of the agile teams does not fit with our calendars. For example, on Thursday I'm never in Den Bosch and that's the day that usually a demo is given so that's providing a challenge. I regularly follow that demo by using Office Communicator which works pretty well except when the demo is given to a larger group, so for 2-3 people it works well but with 10 people usually start talking over each other and then it becomes much more difficult to follow. This is sometimes prevented by someone who manages the demo very strict in which case demos with large groups as easy to follow as well.
Another problem is that we are working with a lot of Infosys people [Indians] and they can be somewhat difficult to understand through Communicator. It's very strange, when you talk with them face-to-face then they can easily be understood but sometimes when using Communicator I have no idea what the other party is talking about, so the language barrier can play a role as well.

Frequent face-to-face interaction between developers & users (+/-)

46. **Did you think of any other problems with the agile software development process of the B2C department during the course of this interview?**
I think we need more clarity about who has what role and what is expected of everyone. I feel that perhaps it is defined somewhere but it was never really clearly communicated. We currently don't really know who has what role.
They also devised a new method of working, I don't know who approved it but it certainly wasn't us. It could have been senior management but we have expressed that we currently can't work that way, it just doesn't fit within our daily activities. The will is there but it just can't work this way. There should be more alignment about how we work together instead of imposing a way of working on us.

B.7 Interview 7

1. **What is your job at [company] and how does it relate to the agile software development process of the B2C department?**
   Since april I'm the product owner by proxy for two chameleon teams.

2. **What are in your opinion the most important problems with the agile software development process of the B2C department?**
   In my opinion currently the biggest problem is that we are saying we are working agile and we're saying we're using an agile methodology but actually we are only using the pieces that we like. If you look at how the team is really working and how for example the cooperation between projects and agile teams is going then it looks a lot like waterfall to me. I'm wondering to what extent we are working according to an agile methodology or whether we call something agile but have created our own "[company]" methodology.

Embrace change to deliver customer value

3. **Are managers and developers willing and able to deal with evolutionary requirements?**
   You can submit changes until the moment we start playing planning poker and when the planning poker game is finished then the scope of the sprint is fixed. If it so happens that in a sprint something very small like a dot or a comma needs to be changed it depends on capacity whether or not it is possible to change it in the same sprint. When we have time to make the change then we do, otherwise we tell the requester that we can't. In practice you see that the scope is pretty fixed at the moment we start the planning poker so that at that time the estimates can be made about the amount of work.

Evolutionary requirements (+/-)

5. **Has the customer the power to dictate the scope of the iteration?**
   Not directly but is does happen indirectly. A team has a certain backlog and a team decides what they pick up from that backlog so a team does not meet with a customer directly who tells them what he wants to change, except that the product owner by proxy may indicate a certain preference in the order that things are picked up. However, you can see that projects try to exert influence from the outside on [company] development management about when they can expect their change requests to be fulfilled.
   Besides whether or not a team commits to or plans something, you see that a variety of forecasts are made about the contents of the next few sprints.
   So directly the customer doesn't have a lot of influence over the scope of the sprints but indirectly they have some influence.

Client driven iterations (+/-)

6. **Is the customer encouraged to continually give feedback/criticism and rethink their requirements during the development process?**
   They do not constantly give feedback but this is due to the fact that they are not constantly involved in the development process. They often know that we are starting a certain epic and it depends on the person how much interaction the team has with the customer.
   The customers do not give feedback on their own initiative but after every story we ask someone to fill out a survey of 3-4 questions about how satisfied they are with the solution, how it fits, how they experienced the collaboration and so on.

Continuous customer satisfaction feedback (+/-)

Plan and deliver software frequently

8. **Do customers, developers and managers plan together?**
   Yes.

Collaborative planning (+)

15. **Is management willing to plan immediately before the iteration instead of earlier to allow the customer's feedback to be incorporated into the planning?**
   I can recall no situation in which the business said they were unhappy with how things were going and that that feedback was incorporated into the planning for the next sprint. However, the content of sprints is discussed, for example when the customer needs something changed with a much higher priority than other things so he asks whether we can make that a priority. Things like that occur a lot more than discussions about the way we work.

Adaptive planning (+)

Customer collaboration

41. **Is the customer willing to dedicate time to take an active role in the project?** No. The business is in my experience involved with the agile team in a number of ways. We have 6 chameleon teams here which all have half a dedicated product owner by proxy so we have 3 proxies on 6 teams. This is the fulltime involvement of the business. The request owner -the applicant of a particular epic- is sitting somewhere else entirely and for him the answer is no, the customers are not sufficiently involved and they do not have enough time to take an active role in the project.
    Basically the product owners by proxy look at what is required in an epic and then try to involve those people from the business that submitted the initial request by for example asking them if they want to come by to get a demonstration but that does not happen a lot. Sometimes that's simply because we are too busy and therefore announce 3 days in advance that someone can come by and see the result but at such short notice people may not be able to allocate time for that.
    It's not as black and white as that they are unwilling to dedicate time to take an active role in the project, sometimes it's simply because we are too late in making the appointment.

    Customer commitment to work with developing team (-)

42. **Is the customer contract reflective of evolutionary development?** As far as I know the customer is pretty happy with this way of working.

43. **Is the customer immediately accessible?**
    Yes, usually the customer is immediately accessible for answering questions but often these questions are simply about whether he wants left or right so then you can call him to ask that and most of the time after a short deliberation he can give you the answer. So yes, the customer is certainly accessible but I doubt he's involved enough.

    Customer immediately accessible (+)

44. **Does the customer contract revolve around commitment of collaboration?**
    Not as far as I know.

    Customer contract revolves around commitment of collaboration (-)

45. **Do the developers and customer frequently interact face-to-face?**
No, not enough.

Frequent face-to-face interaction between developers & users (-)

46. **Did you think of any other problems with the agile software development process of the B2C department during the course of this interview?**
Basically it all comes down to the point I made at the beginning of this interview, we say we're doing agile but what we're doing is not completely agile. In the meantime there are -just like in any other large organization- a lot of other things that are being changed and due to all these changes you can see that we as an organization do not fit with the theory of agile. Due to this you can see that agile is not always as efficient and effective as a method at [company].
The business is expected to show involvement with the agile team next to their normal job. If I had a number of KPIs for delivering processes than I would make sure I was working on that, at the expense of people developing software and wanting to ask questions.
It would actually be in the interest of the business to be more involved. Ultimately, they need the best possible product. Currently the business is depending a lot on other people to determine that the product is good.
I see improvement opportunities on both sides, sometimes everything is very hectic here, for example we are still working on the preparations for the next sprint, which starts next Friday, so that's all pretty last minute.

Customer commitment to work with developing team (-)

Adaptive planning (+)

B.8 Interview 8

1. **What is your job at [company] and how does it relate to the agile software development process of the B2C department?**
   I am the manager of the business process support department, we are responsible for managing IT insofar that we have budget responsibility. We actually determine where the money goes that is spent on IT on what we spend it and we monitor that as well. We also prioritize the work that enters the agile process, on processes but also changes, small changes but also large projects. In addition, I also manage the UAT department. We do User Acceptance Tests as part of B2C, we test what is actually delivered by the agile teams. We do that from a business perspective because we believe that we are the ones that have to work with the software and we know our processes best. We test the new functionality to make sure it does what it should do and whether the existing functionality still works after completion of the sprint. So basically I am responsible for the front of the agile process, so everything that goes in. Then I make sure that it is delivered on time and within budget and I keep monitoring it until finally we acceptance test what comes out.

2. **What are in your opinion the most important problems with the agile software development process of the B2C department?**
   A number of things that stand out at the moment in our experience. The most important is that large changes or large projects do not currently match with the agile way of working. When you're starting a large project involving multiple sprints then to deliver the final product on time it's very important to define precisely what pieces of work there are, what needs to be done and in what order. The risk is that if you don't correctly estimate the number of sprints needed and specify how many teams and poker points are needed, the deadline will slip because your estimate and consequently the planning was wrong, which results in not being able to finish the project in time.
   Another problem we see is that because we currently have many projects that need to be finished, we're increasing capacity but this leads to a decrease in quality of the deliveries. Those who build the functionality actually have little affinity with the system and processes so they are mostly just creating code without understanding the rationale. We try to improve that situation with the product owners by proxy which are in the teams at B2C but we're experiencing that that takes a lot of time to pass down knowledge like this, especially when you create completely new teams consisting of new external people, many of them Indian and having no knowledge of the energy market at all.

Embrace change to deliver customer value

Evolutionary requirements (+)

3. **Are managers and developers willing and able to deal with evolutionary requirements?**
   A team should only commit to the work they are going to deliver in a sprint when the requirements are formulated clearly. If that isn't the case then the developers don't really know what to build so a team should say that the requirements aren't specific enough and they won't pick them up because of that, even if the change or project has the highest priority from the business. This is a field of tension between business and IT because at the moment we're seeing that we need stuff really quickly because it has a lot of business value for us and/or can help us save a lot of time or money or can help another project progress more quickly then it's possible that we say that we want it at all costs while the team isn't completely clear on what needs to be done, how much time it's gonna take etc. At such times you'll see that during a sprint the pressure is increased and people have to work overtime because the task turns out to be much bigger, more coordination is required, etc.
   On the other hand, you sometimes see that the requirements are not completely clear and during the sprint some customer contact is required about whether to turn left or right or color something green or blue. I don't find this much of a problem because this is the power of agile or at least should be the power of agile because you have a team which includes a business representative it should be possible to do such things as long as it's small things and no big changes so it should be specified, no large chunks but whether there should be two spaces in an invoice line or whether a green or a red text needs to be added, such things could in my opinion still be changed during the sprint, as long as it's a small change or a small choice that needs to be made and it doesn't influence the rest of the delivery. However, sometimes something looks small from a business perspective but actually has a huge impact and for example results in double the amount of work to make it happen. At that point you should question whether you did specify your request sufficiently beforehand and at this point the team should say that since it's not in the requirements, they can't build it. They can build it in a separate epic if it still needs to be delivered.

5. **Has the customer the power to dictate the scope of the iteration?**
   Yes, because very simply: we are the ones paying for it. I always say: the customer is paying, if we weren't here, IT didn't have a job. It's pretty black and white, but ultimately it comes down to that. That's why we decide on the scope per sprint based on priorities we give to all changes and projects that are in the pipeline It may just be that we think that we want to do a certain project next sprint but that next week something else becomes more important so we give that a higher priority so it has to be picked up earlier. In the end the business board determines the definitive priority.

Client driven iterations (+)

6. **Is the customer encouraged to continually give feedback/criticism and rethink their requirements during the development process?**
   Yes and no, it very much depends on the person requesting the change. I see that some people have a higher involvement than others and it is exactly the role of the product owner by proxy to involve the person who submitted the change request when necessary. On the other hand, if the requirements are good and clearly specified and are simply implemented without problems then basically the applicant does not need to be involved during the software development process. At the moment that fundamental choices need to be made -for example whether something needs to be green or blue- he needs to be present.
   We try to make sure the applicant is present during demos to show him how it looks and ask him whether this is what he actually wanted but that is not always necessary.

Continuous customer satisfaction feedback (+/-)

Plan and deliver software frequently

8. **Do customers, developers and managers plan together?**
   No, the developers basically don't make a planning, to the extent that we indicate priorities and when something should be finished and then it's up to the preparing party -so in this case business alignment- to create a specification and ultimately the development teams that declare what they will deliver at the end of a sprint.

Collaborative planning (+)

15. **Is management willing to plan immediately before the iteration instead of earlier to allow the customer's feedback to be incorporated into the planning?**
Yes, that's mandatory. We also have a KPI on the business side for the people in my team that control IT about the prioritization effectiveness, so how effective they are in the prioritization. In other words: if we have a top 10 that needs to be picked up, how many items from that top 10 are actually included in the next sprint?

Adaptive planning (+)

Customer collaboration

41. **Is the customer willing to dedicate time to take an active role in the project?**
Yes.

Customer commitment to work with developing team (+)

42. **Is the customer contract reflective of evolutionary development?**
We are happy with evolutionary development when requirements are sufficiently specified and business alignment indicates if this isn't the case as well. The risk is that for example we say we want a car and when business alignment then says they will build it and finally they deliver a green Fiat while we actually wanted a red Ferrari then something is wrong.
What I would like is that during the intake, IT is much more assertive when it's not exactly clear what is requested. We are currently losing too much money because we build something that's ultimately not what was actually requested or something is just slightly different or an essential part was not included. This means we spend money on something we don't really want and I think it's the responsibility of IT to make sure they understand the requirements when they accept a request, that they implement it correctly and tell us when something is not completely clear. That's not weakness, that's just being professional. They should be saying: listen, you requested this, it isn't clear, I need this specification now or I won't pick up the request and that can be done in a very normal businesslike manner.
This is something which is currently still a problem and then subsequently you get a discussion about whether it was not clearly specified. Yes, but listen for a moment, we said we wanted a car but you should have asked what kind of car it should be. You should not want to conduct this discussion with each other.

43. **Is the customer immediately accessible?**
Yes, in my experience that is the case and we try to leave the customer communication to the product owners by proxy.

Customer immediately accessible (+)

44. **Does the customer contract revolve around commitment of collaboration?**
No.

Customer contract revolves around commitment of collaboration (-)

45. **Do the developers and customer frequently interact face-to-face?**
No, actually only at demos.
The product owner by proxy is in this case also an extension of the customer because otherwise you would have 8 customers around the table when 1 team is working on 8 separate functionalities, such as for example payment, billing, collections, customer acquisition, marketing and smart energy. Since we don't want that, we have a product owner by proxy, the business representative in the teams.

Frequent face-to-face interaction between developers & users (-)

46. **Did you think of any other problems with the agile software development process of the B2C department during the course of this interview?**
We often encounter large projects you would actually do in a waterfall structure. Doing these projects in an agile way means that you need to be very focused from the very beginning and you should be very clear about the requirements. In this case the process starts to look a bit more like waterfall but it's impossible to work on one project for 10-20 sprints without having very clear requirements otherwise the project will never finish.

B.9 Interview 9

1. **Do you agree that the amount of continuous customer satisfaction feedback should be improved?**

   Indeed one of the important things we want to achieve with our move to an agile way of working is the close cooperation and visibility between business and IT. If you have a sprint of 3 weeks then you should know that there is a demo after ever sprint so you could put the demo in your agenda for the rest of the year. The business could already know for the rest of the year on what days they should be present at the IT department to check that what they requested corresponds with what is delivered.

2. **Do you agree that risks should be taken into account when planning sprints?**

   Yes, it's one way to look at the question of how to arrive at priorities. What I think is really nice and what you actually suggest is that you take risks as the starting point to determine the priority order of things. So a risk can be either indeed something is very business critical and technically difficult, so let's try to finish this one as early as possible or indeed a high risk as in if we leave this one we are going to miss out on a lot of money. So it's prioritizing but it also feels a bit like it's from a negative undertone.

3. **Do you agree that coding standards should be used more?**

   Yes, I think this is true for the [company] case. It depends on how broad you want to stretch the subject or theme. If you are talking about wanting to work in an agile way, a high level of software development maturity and experience on that level is indeed an important precondition to have in place, so that you have experienced people who can look with a certain standard -also in quality- to code and technology.

   So what you notice is that agile and Scrum at a certain point have the power to show that this is a problem so that actually your software development maturity is too low and you need to improve the technical skills. I think that at [company] this is one of the things we need to improve in.

4. **Do you agree that continuous integration should be implemented?**

   We decided not to implement this. This is related to the technical landscape we have. We have a 5-layer architecture -or decoupled architecture as it's actually now called- with SAP components, intermediate layers, TIBCO, iProcess on the frontend and what you notice is that various teams are working on different parts of the code and creating one new integrated build is a lot of work, which is also related to the technology behind it whether this can be continuously integrated or not. I don't think we currently have the technology in place to implement that milestone and there is no energy at [company] to grow towards that goal, it's simply accepted as that's it's actually one of the suggestions or preconditions that the books prescribe but we are not going to implement it now.

5. **Do you agree that continuous improvement (refactoring) should be done more often?**

   Yes, that is a good one because there are proponents and opponents and what you'll see is that interests within the company are divided on this and I find it difficult in my role as agile coach to deal with this in this organization. If you ask the business what they need they don't care whether the code is maintainable, they want as much software for as little money to help them carry out their business. So the pressure from them to finish a project and start something new is enormous. What you'll notice is that the effect of this is that the teams don't feel the freedom or are willing to take the time to work on this kind of things. I actually think that they shouldn't explain it and just organize it themselves. So that's a pretty difficult discussion so if you ask whether [company] will go there in the short term, if you ask me we have to create the air and space in the work so that teams are going to do it. I don't think it's necessary for the business to know that this takes a certain amount of time and money because it's part of the IT job. However, on the short term we probably won't be able to implement it.

6. **Do you agree that the customer should be more committed to collaboration with the agile team?**

   What I actually see is that -and I'm appealing to my lean background here- there is still one large form of waste left and that's the amount of work we have in stock. This means that the business is currently working on other things that the request they asked of IT roughly half a year to a year ago. That effect is reinforced by that we are now in 2011-2012 mostly dealing with compliance projects so eighty percent of our IT projects involve complying with laws and regulations from outside of [company], such as the new market model, SEPA. Those are all not thought up by ourselves but we have to do them because otherwise we are fined or not licenced anymore and have to stop as a company. This means that the business asked IT to take care of it for them and in the meanwhile they are busy with thinking up new campaigns, new rates, a new form of customer centricity. This means they are talking with each other about these kind of things and if we are asking a question about how they see something in the new market model or how to we implement specific SEPA rules, they have to make a context switch. What I think is important is the question: how do we ensure that that cooperation is optimal, so how do we ensure that IT is working on the same things that are topical for the business. If the business is currently talking about what they want to do with the new campaign then IT has to be sparring partner and immediately think with them -build- about the implementation of campaigns. So if they say: this is how it's going to look that the software is finished as well and they can put the campaign live and that's currently not going well because there is too much stock between the projects IT is working on and the business is already working on the next thing.

7. **Do you think that there should be KPIs on the amount of customer collaboration?**

   No, I don't believe in that kind of KPIs. What I think is important is -and that's an important we are currently actively implementing- measuring and demonstrating how much work there is in the pipeline, where the stock is and how big the gap between the agenda of business and IT is, which should be as close to each other as possible because in that case you can talk with the same people about the same topics, which means that as an organization you have to consciously choose about where time is allocated. We currently aren't that far, there are now a few people working on important things and a lot of other people around them working on less important things and creating noise. It could be a lot leaner.

8. **Do you agree that test-driven development should be implemented?**

   Yes, I believe in the power of test-driven development but I'm also afraid -and that is related to the software development maturity I was talking about earlier- that we have an insufficient amount of people who have knowledge and experience about that to do this well in the short term. I think this is currently a bridge too far.

9. **Do you agree that pair programming should be implemented?**

   We are currently do it already in certain places, in particular you'll see that the people from Infosys that are sitting here are doing it a lot and that works. I also have very good personal experience from early 2000 with pair programming at ABN Amro bank. I love working that way and really believe in it.