TAMPERE UNIVERSITY OF TECHNOLOGY

TIMO TENHUNEN

CHALLENGES IN SCALING AGILE SOFTWARE DEVELOPMENT

Master's thesis

Tarkastaja: professori Kai Koskimies
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan
tiedekuntaneuvoston
kokouksessa 3. helmikuuta 2010

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
TENHUNEN, TIMO: Haasteet ketterien ohjelmistokehitysmenetelmien skaalaamisessa
Diplomityö, 61 sivua, 4 liitesivua
Toukokuu 2010
Pääaine: Ohjelmistotuotanto
Tarkastaja: professori Kai Koskimies
Avainsanat: Ketterät ohjelmistokehitysmenetelmät, laihat ohjelmistokehitysmenetelmät, kansainvälinen ohjelmistotoimitus, ketterien menetelmien skaalautuvuus

Ketterien ohjelmistokehitysmenetelmien skaalautuvuuteen liittyy monia haasteita. Tämä diplomityö on jaettu kahteen osaan. Ensimmäiseksi perinteisiä ohjelmistokehitysmenetelmiä käydään läpi ja verrataan niitä iteratiivisiin ja ketteriin ohjelmistokehitysmenetelmiin kuten Scrum. Ketteriä menetelmiä käydään läpi teoriatasolla, joka mahdollistaa myöhemmän tarkastelun skaalautuvuuden yhteydessä. Esimerkiksi jonoteoria on tärkeä osa laihoja ohjelmistomenetelmiä kun toimitaan suurten työerien kanssa.

Tärkeimpiä praktiikoita käydään läpi kuten testivetoinen kehitys, jatkuva integraatio ja Extreme Programming. Työssä tuodaan esille eri näkökulmia ja ratkaisuja skaalautuvuushaasteisiin toimittaessa isoilla tai hajautetuilla kehitystiimeillä. Näihin kuuluvat mm. ”Scrum of Scrums” –malli, ketterä julkaisuketju ja erilaiset vaatimukset globaalissa julkaisussa.

Työn toinen osa koostuu tutkimuksesta, joka teetettiin muutamalle ohjelmistoteollisuuden ammattilaiselle. Heidän vastauksiaan analysoidaan ja esitetään kahden samantapaisen tutkimuksen yhteydessä.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
TENHUNEN, TIMO: Challenges in Scaling Agile Software Development
Master of Science Thesis, 61 pages, 4 Appendix pages
May 2010
Major: Software engineering
Examiner: Professor Kai Koskimies
Keywords: Agile software development, lean, agile, global delivery, Scrum, agile at scale

Many challenges arise when agile software development methods are being used on larger scale. This thesis consists of two parts. First the thesis will go through the traditional software development processes and compare them to iterative and agile software development practices such as Scrum. Agile methods are represented so that the theory can be used on a basis of scaling analysis. For example queuing theory is relevant when using lean principles and working with larger batches.

The most common practices are explained such as Test Driven Development, Continuous Integration and Extreme Programming. Different aspects of scaling issues and solutions, when working with large or distributed teams, are represented. These include the Scrum of Scrums model, agile release train and different requirements in the global delivery.

Second part of the thesis is the survey which was conducted to a few software industry professionals. Their answers are being analyzed and represented with two related surveys.

PREFACE

The thesis was written in the winter 2009-2010 while working as a part of Accelerated Solution Delivery team in IBM Finland. During the time, I had a chance to observe an agile project of a well established major company and also have discussions with very experienced people in the software development area. Insights were also got from the Agile Finland association's agile dinners which I participated in the cities of Tampere and Helsinki.

I'd like to thank my family and following persons for giving me professional advice or helping with the work related to the thesis:
Juha-Markus Aalto, Scott W. Ambler, Lauri Fjällström, Satu-Maria Jauhiainen, Tomas Kaulakys, Kai Koskimies, Thomas von Kulessa, Ville Lehto, Lasse Lilja, Christoph Nocula, Ville Paalanen, Pirkka Palomäki, Jason Papadopoulos, Tapio Peltomäki, Pekka Savolainen, Sjoerd Sommen, Ville Säkö, Marko Taipale and Tuomas Vanhanen.

TABLE OF CONTENTS

ABBREVIATIONS

ABS	Absolute value
ASM	Agile Scaling Model
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
CT	Cycle Time
IP	Internet Protocol
JIT	Just In Time
MIN	Minimum value
QA	Quality Assurance
ROI	Return On Investment
RUP	Rational Unified Process
ST	Service Time
TDD	Test Driven Development
WIP	Work In Progress
XP	Extreme Programming

1. INTRODUCTION

Since the introduction of Scrum in the 90's, agile software development methods have been widely adopted on the team level. Many benefits of the agile practices have been welcomed by the professionals in the relatively new area of the software development. Customers are also interested as late changes can be made with relatively less cost compared to the traditional development methods. Not to mention that the quality of the delivered products tends to be higher. As agile has become mainstream many enterprises are initializing their own agile adoption programs and are often facing the common agile scaling challenges. Agile teams are ideally sized under 10 members, but in the organizations teams with hundreds of developers might exist. In addition, the enterprises might have existing regulatory compliances such as Capability Maturity Model Integration (CMMI) which should be aligned with the new agile processes. Therefore agile adoption might involve changes in all of the organization levels.

Global Delivery is a part of software engineering as organizations are off-sourcing their resources and taking benefits from follow the sun development. When developers are geographically dispersed, effective communication methods are needed. In addition, for cost reduction and to avoid high carbon footprint no face-to-face meetings are even arranged with the near shored team members. Therefore teams need to find the best alternatives for daily communication across the globe.

Agile development practices have been invented by researching traditional manufacturing companies such as Toyota. Principles of lean development have been found useful and adopted as a basis of many agile practices. It might seem that there's even some fanaticism within some agile groups. Sometimes the theory of some specific methodology is followed so blindly that the actual outcome doesn't really fit into the industry needs.

This thesis has been made to understand the challenges the enterprises will most likely face when adopting agile on large scale. The findings could be used to ease up the agile adoption work in the whole industry and to concentrate on the right issues. The context is in companies which already have their existing processes in place and are now adopting agile practices. Companies have projects which may involve multiple dispersed teams with different skill sets and cultures. In addition, these enterprises are mostly working in strict financial chains with annual and quarterly budgets so that up-front financial planning is needed even for agile projects.

For the thesis I gathered theories from today's most known agile literature and latest agile scaling trends by authors such as Craig Larman, Bas Vodde and Dean Leffingwell. Often we hear comments that the practices described in the books hardly satisfy the needs of the real industry. These needs and the most important large scale practices were charted by conducting a survey to a few top agile performers on the

software development area. Survey was mostly concentrating on the challenges in the team distribution and size.

Thesis is divided into two main parts, the theory and the survey. Chapter 2 goes through the theory of traditional and agile software development processes and scaling issues. Chapters 3 and 4 contain the survey to find out the state of current agile scaling methods in few known big companies. Chapter 5 has an analysis of the survey results and it also contains comments gathered when interviewing the survey participants. In Chapter 6 two globally known surveys are reflected against the results and finally conclusions are wrapped together in Chapter 7.

2. PROCESSES

Software engineering has still relatively short roots compared to other areas of engineering. Development methods are constantly evolving when software is produced more efficiently. Iterative agile and lean methodologies are taking over the traditional waterfall and sequential software development. Even though the agile manifesto was introduced in the software engineering area in the year 2001 [1], the Scrum principles were already presented by Takeuchi and Nonaka at 1986 [2]. Scrum is currently widely used in both small and the largest of IT companies and certified Scrum masters are being trained with intensive 2-day courses.

In sequential development the software products are being manufactured in different phases. The next phase cannot be started before the previous state has been ended and the work is being handed over. This kind of development has been causing handover overhead and is stiff compared to the more overlapping and flexible agile development methods, which are illustrated in Figure 2.1.

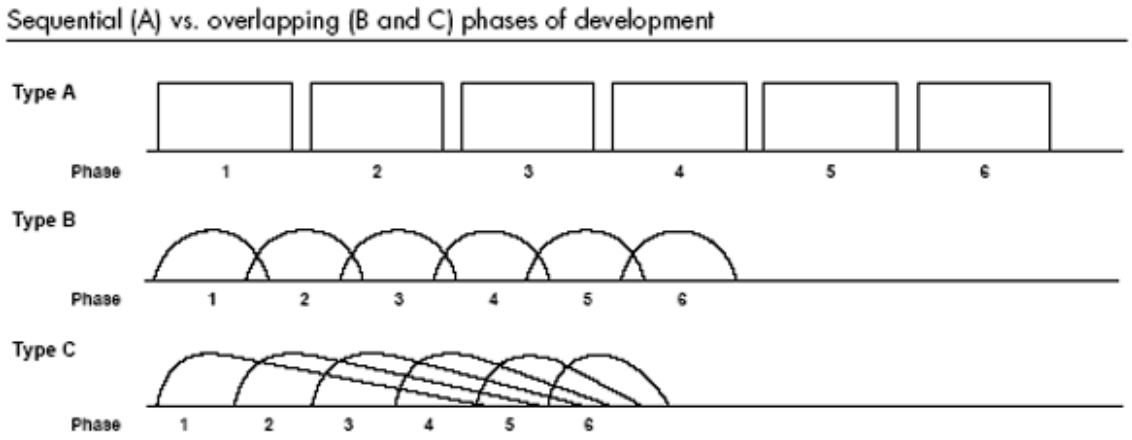

Figure 2.1 *Sequential and overlapping types of development [2]*

On the other hand, lean principles see work in process and overlapping tasks as a waste which should be eliminated. Takeuchi & Nonaka [2, p.2] use relay race as a bad example whereas Craig Larman [3, p.39] sees that the aspect of lean thinking is in the sport of relay racing. Even though there is a contradiction, the basic idea of watching the baton not the runners is similar in both writings. Queuing theory proves that high utilization of workers will lead to queues which cause waste. Larman and Takeuchi and Nonaka watch the relay race from different perspectives. Takeuchi and Nonaka are worried about the overhead caused by the baton being handed over from person to person. They suggest that more flexible “rugby” approach, where the team passes the

baton back and forth as a unit, would serve better today's development needs. This approach is used as a basis of Scrum. On the other hand Larman is worried about the traditional management advices where utilization of the workers should be kept high and describes relay race as a pipeline in which the baton passes through as fast as possible. It's essential to understand the different context between these writings.

As we can see in Figure 2.2, some software development methods give more rules and are therefore more prescriptive than the more adaptive ones.

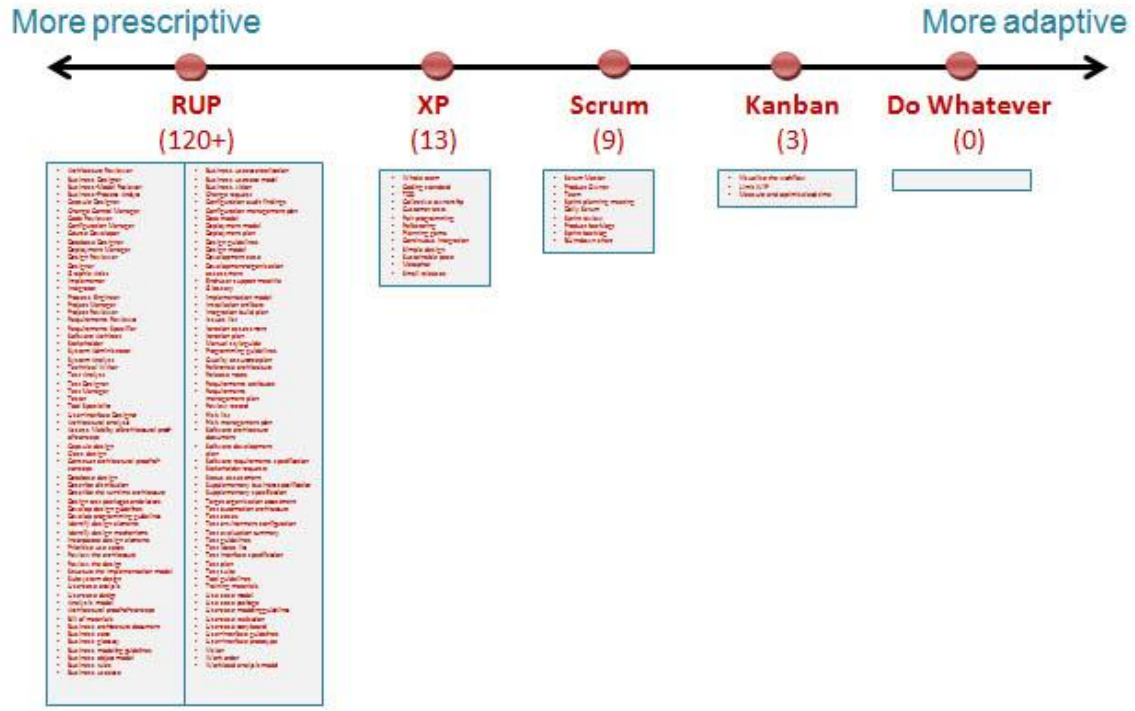

Figure 2.2 *Amount of rules in different software development methods [4]*

For example in the lean Kanban process only few basic rules are given. In this kind of development the development team has more freedom to choose their own rules in addition to the basic rules given. On the other hand the Rational Unified Process (RUP) gives many rules or guidelines from which the development should only choose the most suitable subset for their project [4].

2.1. Traditional

Traditional software development is usually done in the way of a waterfall shown in Figure 2.3. Project starts with extensive planning where the whole product is planned and documented up-front. Estimates are being made of the important milestones far to the future. Usually Gantt charts are used to plan steps and activities. When the plans are ready they are handed over to the development team. After implementation the development team passes the product for the Quality Assurance (QA) before finally reaching the customer. This kind of sequential life cycle doesn't really embrace changes late in the project. With the waterfall, a great idea late in the release cycle is not a gift, but a threat [5].

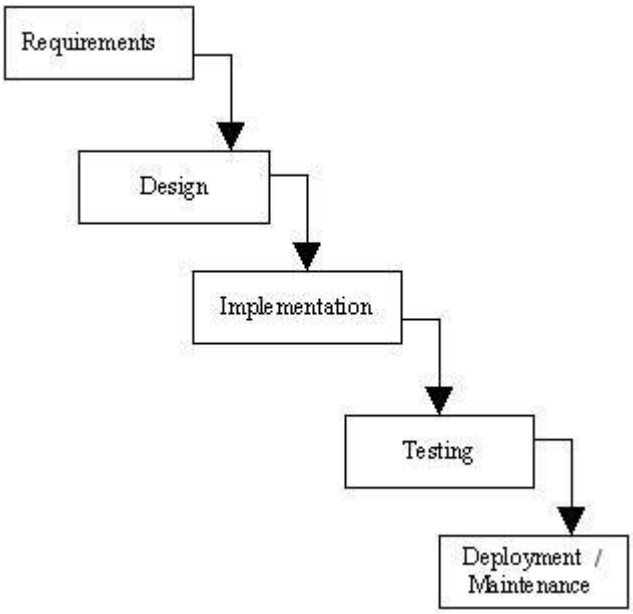

Figure 2.3 *Waterfall model*

In the waterfall there's also a pressure to produce huge amount of documentation. However most of the times the documentation is not ever read or it's misunderstood. As the documentation is the main channel of communication, misunderstandings can be easily made. [5]

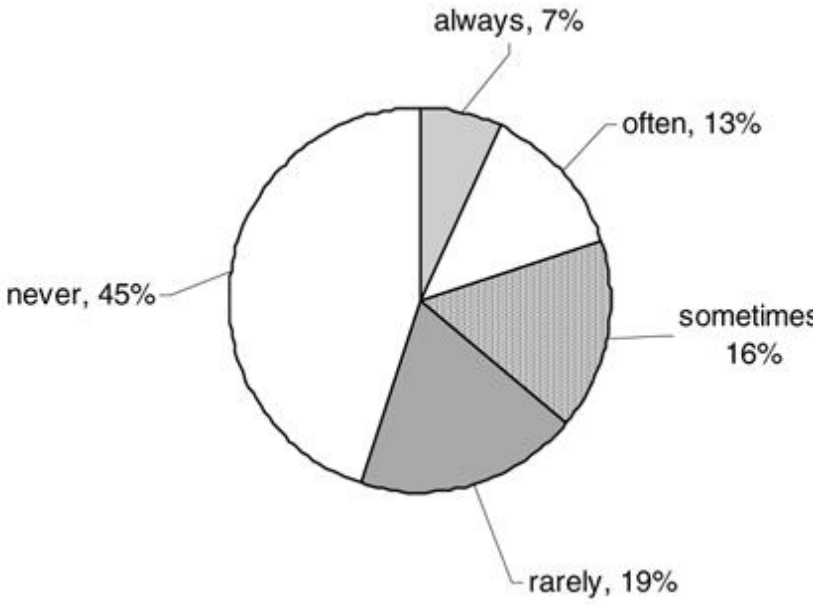

Figure 2.4 *Actual use of waterfall-specified features [6, p.56]*

Another shortcoming in the waterfall is that the customer rarely knows beforehand what she really wants of the product. If the plans are being made up-front the product deployed year after may not really be what the customer, or the other stakeholders, want at the current situation. Figure 2.4 shows that on average 45% of the features in the waterfall requirements are never used.

2.1.1. Capability Maturity Model Integration

Capability Maturity Model Integration is a process maturity model which was originated to support complex and high risk processes. It's the successor of the capability maturity model (CMM) which was developed from 1987 until 1997. [7] CMMI doesn't really give you practices or tools and therefore CMMI is said to be focusing on 'what' and not 'how'. CMMI describes an improvement path from immature process to a mature disciplined process. It's designed to combine the management and engineering disciplines in software development and systems engineering. Since over the last decade, various versions of the CMM were mixed to different disciplines. CMMI reduces these duplications supporting process and product improvement. [8] CMMI is widely adopted in larger organizations and it contains five maturity levels which can be seen in Figure 2.5.

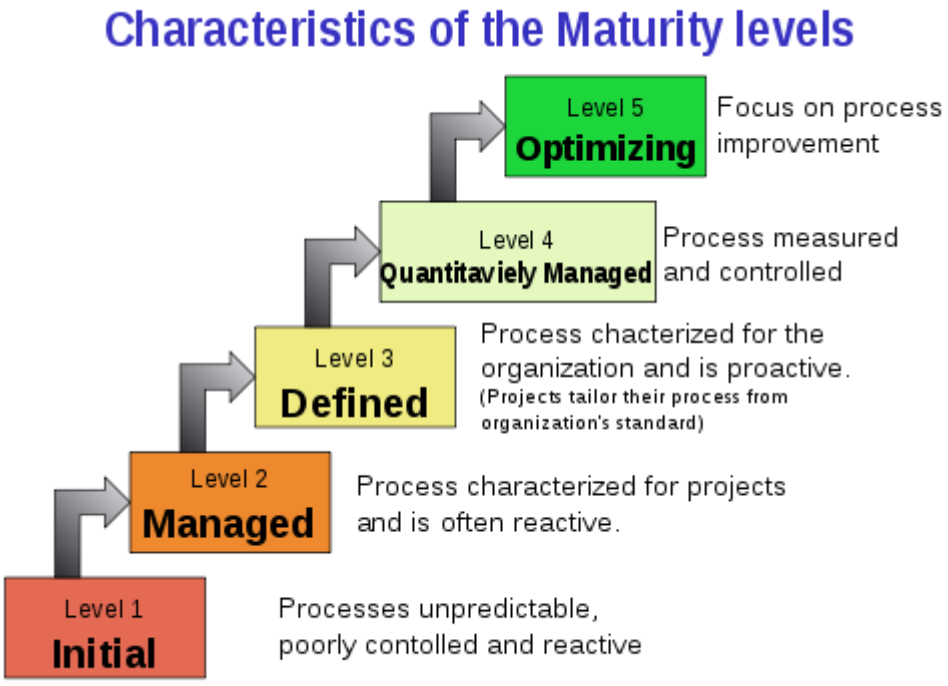

Figure 2.5 *Characteristics of the Maturity levels [9]*

CMMI and agile are seldom used together because of the common misconception that these have contradictions. However as enterprises are adopting agile methods, ways of combining these two have been found.

2.1.2. ISO 9000

ISO 9000 is a family of standards for quality management systems. It provides a set of requirements that, if effectively implemented, will provide that for example the supplier can deliver goods and services that meet the expectations and follow the applicable regulations.

Requirements include for example supplier's top management commitment to quality, customer focus, adequacy of resources, employee competence, process management, quality planning, product design, review of incoming orders, purchasing and monitoring and measurement of the processes and products [8]. Organizations might face unexpected challenges when trying to fit agile development processes into ISO 9000 standards.

2.2. Iterative and Agile

In iterative development the software is being developed in small pieces. Usually this means development with repeating and short time iterations. Each iteration contains its own requirements, analysis, design and implementation. The outcome of each iteration is a tested and integrated executable subsystem [6, p.19]. Therefore new business value is added to the system in the end of the each iteration. Along with small increments, the whole system may be ready for production deployment after, for example, 10-15 iterations. In addition, iterative and evolutionary development involves early programming and testing [6, p.18]. The system isn't necessarily fully detailed up-front but the current iteration is only needed for thorough design instead. This makes it easier to make changes in the middle of the project compared to the waterfall process model. Early feedback is gathered from the end-users and used for evolving specifications. Therefore the team doesn't have to speculate on the complete, correct requirements or design. At this early stage the end-users have also a chance to try out if some feature was really what they wanted [6, p.21]. Research has shown that iterative development methods have resulted in higher project success and productivity rates and better quality than sequential or waterfall projects [6, p.18]. Key statistics show that on average, 25% of the requirements change in the software projects [6, p.55].

Agile software development was officially found in 2001 when the Agile Manifesto was published [1]. Agile is not a practice but is based on a set of values which support more flexible and adaptable software development. The Agile Manifesto is following:

> *"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*
>
> *Individuals and interactions over processes and tools*
> *Working software over comprehensive documentation*
> *Customer collaboration over contract negotiation*
> *Responding to change over following a plan*
>
> *That is, while there is value in the items on*
> *the right, we value the items on the left more."* [10]

As the manifesto states it was released to concentrate more towards communication and involving different stakeholders to achieve more responsive software development.

Excessive documentation is discouraged but against common misconception the needed documentation is justified. Keep it simple enough but not too simple is a principle for many agile practices. In addition to the agile values, the 12 following agile principles exist to support being agile [11]:

- *Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.*
- *Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.*
- *Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.*
- *Business people and developers must work together daily throughout the project.*
- *Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.*
- *The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.*
- *Working software is the primary measure of progress.*
- *Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.*
- *Continuous attention to technical excellence and good design enhances agility.*
- *Simplicity--the art of maximizing the amount of work not done--is essential.*
- *The best architectures, requirements, and designs emerge from self-organizing teams.*
- *At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.*

There are various software development methods based on agile principles. Most of these methods also encourage working iteratively. Many practices and tools are evolved around these methods. Methods include for example, Scrum and Extreme programming. Practices are for example Test Driven Development, Planning poker, Pair programming and Continuous Integration. Most of the practices are helping the agile teams to work along the agile principles. Mike Cohn lists main principles in which agile teams work [12, p.23]:

- *Work as one team*
- *Work in short iterations*
- *Deliver something each iteration*
- *Focus on business priorities*
- *Inspect and adapt*

Agile teams work in short time boxed iterations and deliver a working product in the end of the each iteration. Features implemented are selected along the business priority, which ensures that the most important features are developed first. Agile teams accept

that the plans can change in the late of the project and therefore are ready to adapt to new plans when needed. [12, p.32]

As agile gives only few disciplines, any iterative method, including the Unified Process, can be applied to agile spirit. Another example is the Toyota Production System from which the agile development community has adapted for example the following techniques [13, p.140]:

1. Pull system
2. Just In Time (JIT)
3. Visual management
4. Multi-skill development

These are also some of the main principles in the lean thinking.

2.2.1. Unified Process

Unified process is an iterative process and is heavily based on use cases. Use cases are needed to describe the behavioral requirements for the software. Table 2.1 shows an example of use case with extended detail.

Table 2.1 *Example of a use case*

Use case #	*1.0 Login to Zapmeet*
Description	*As a user I want to login to Zapmeet*
Actors	*User*
Pre-Conditions	*User navigates to page with login GUI, user account exists*
Post-Conditions	*User logged in and navigated to personal home page*
Steps	*1. User types the username* *2. User types the password* *3. User confirms the action by pressing login button*
Alternative execution paths	*1. User can cancel the action and navigate to other pages.* *2. User types the wrong username or password and is asked to retype.*
Non-Functional	*1. Typed password not shown* *2. User should be informed if typed information is wrong*
Issues	*-*
Input	*Username: String (Required)* *Password: String (Required)*
Output	*Result: String (error message or confirmation message)*

Project can be divided into four different phases Inception, Elaboration, Construction and Transition which are illustrated in Figure 2.6. Each phase consists of time-boxed iterations. However the length of the Inception is usually only single iteration and its purpose is to check if the project is really feasible business-wise, approximate vision, scope and give vague estimates. In elaboration a vision is refined,

core architecture is being developed, highest risks are tackled and estimates and requirements are mostly made. In Construction-phase the iterative implementation is made to the remaining lower risk elements and preparations are made for deployment. The transition-phase consists of beta tests and actual deployment of the final product.

The Unified Process encourages risk driven and client driven development to tackle the highest risks early as possible and build visible the most important features for the customer [6, p.27].

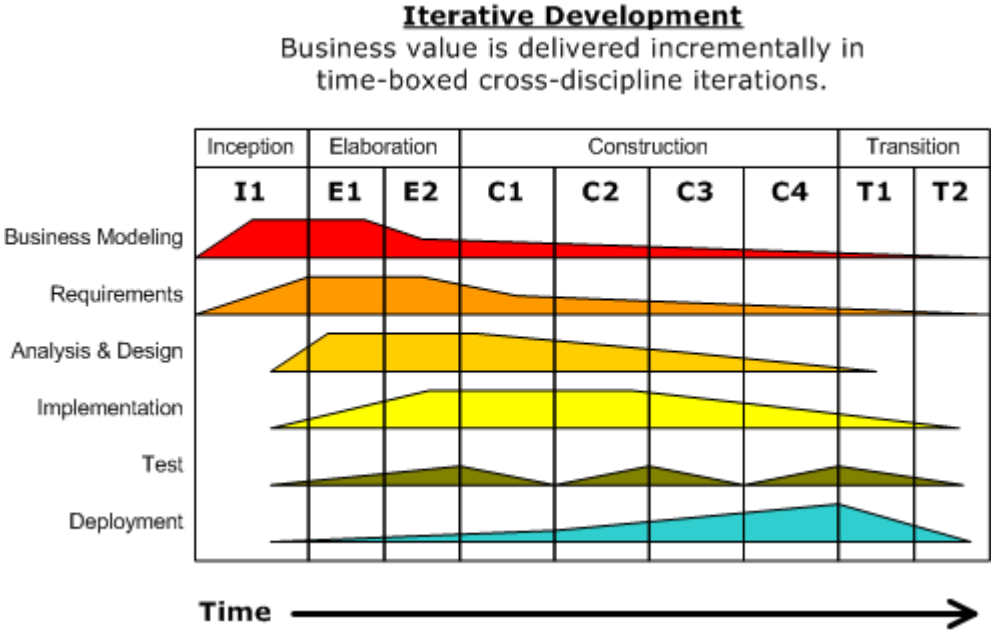

Figure 2.6 *iterative development in unified process*

It's also possible to use practices from other agile methods such as Scrum and Extreme Programming. These would be for example daily Scrum meeting and Test Driven Development [6, p.18]. Rational Unified Process provides additional disciplines to the Unified Process. However, almost all activities and artifacts are optional so a suitable subset can be chosen for each project.

2.2.2. Test Driven Development

Test Driven Development (TDD) is a software development technique where tests are written before the actual code. TDD makes developers to work in smaller steps and ensures that test cases are written on time. The main benefit is achieved by reduced defect density and by making the subject of work crystal clear to all involved. When the defect density can be reduced enough, the QA can shift from reactive to proactive work [14, p. x].

The general TDD cycle goes as follows [14, p.11]:

1. Developer starts with writing a simple test which doesn't pass. This involves thinking how the developer would like the operation appear in the code and inventing the interface. All the elements which will be necessary to calculate the right answers should be included in the test.

2. Developer implements the code to make the test to pass. If there's a clean solution it should be used, but the main idea is to make the test to pass as fast as possible.
3. Developer refactors the code. Now when the test is passing the developer needs to clean up the code and write the possible cleaner solution to make the test pass.

Even though initializing TDD in a project might involve relatively big amount of work, extensive and constantly passing tests usually cause a remarkable decrease to all stakeholders' stress levels.

2.2.3. Continuous Integration

Continuous Integration is a set of practices to support software integration and quality. Continuous Integration involves that the developers integrate their code frequently to some centralized repository. Integrations are relatively small and therefore possible conflicts can be tracked and resolved at early phase. There are practices as one-button-build or even automated build environment which minimize the integration effort of the developer. In this environment all code from distributed teams, hundreds of developers is constantly compiled, linked and run through test suites before transferred and shared through code repository. This happens many times each day. [3, p. 181]

Continuous Integration leads to higher quality code that progress more rapidly over time. Less time is needed in hunting bugs which are caused when integrating multiple developers' code. Possible defects are discovered while they're still fresh in everyone's minds and all the team members are still available to make the corrections efficiently [15, p.171].

2.2.4. Extreme Programming

Extreme Programming (XP) is agile programming method which was first time used in 1996 project at DaimlerChrysler managed by Kent Beck. He and a team of a dozen or so programmers were able to implement a financial system in 2 years. Formerly a team of 30 had failed to do the same in over many years [15, p.29]. According to Beck the XP is distinguished from other methodologies by [16, p. xvii]:

- *Its early, concrete, and continuing feedback from short cycles.*
- *Its incremental planning approach, which quickly comes up with an overall plan that is expected to evolve through the life of the project.*
- *Its ability to flexibly schedule the implementation of functionality, responding to changing business needs.*
- *Its reliance on automated tests written by programmers and customers to monitor the progress of development, to allow the system to evolve, and to catch defects early.*
- *Its reliance on oral communication, tests, and source code to communicate system structure and intent.*

- *Its reliance on an evolutionary design process that lasts as long as the system lasts.*
- *Its reliance on the close collaboration of programmers with ordinary skills.*
- *Its reliance on practices that work with both the short-term instincts of programmers and the long-term interests of the project.*

In practice, the most distinct feature of XP is the pair programming where two programmers simultaneously implement functionality. This is most often done with a single workstation even though distributed pair programming practices exist. Extreme Programming welcomes changes during the project and also assumes that the cost of a late change will be much less than, for example, in the waterfall development. Practices such as TDD and Continuous Integration are required for successfully applying XP.

2.3. Lean thinking

Lean thinking has its proven roots in the Toyota. MIT researchers were visiting Toyota and gave the English term ‘lean‘ to the Toyota system in the 1990 published article *The Machine That Changed the World*. [3, p.44] Lean thinking was introduced to the software developers by Mary and Tom Poppendieck in their 2003 published book *Lean Software Development: An Agile Toolkit* [17].

Lean processes are formed around creating value for the customer and waste reduction. Lean Enterprise Institute lists the following five-step lean implementation process [18]:

“The five-step thought process for guiding the implementation of lean techniques are easy to remember, but not always easy to achieve:

1. *Specify value from the standpoint of the end customer by product family.*
2. *Identify all the steps in the value stream for each product family, eliminating whenever possible those steps that do not create value.*
3. *Make the value-creating steps occur in tight sequence so the product will flow smoothly toward the customer.*
4. *As flow is introduced, let customers pull value from the next upstream activity.*
5. *As value is specified, value streams are identified, wasted steps are removed, and flow and pull are introduced, begin the process again and continue it until a state of perfection is reached in which perfect value is created with no waste.”*

Figure 2.7 *Lean implementation process [18]*

All the tools and thinking around lean development is based on the process shown in Figure 2.7. The term 'lean' is used now within the Toyota for example in their Toyota Way –internal booklet. Craig Larman has made a summary of the modern Toyota Way as a Lean Thinking house shown in Figure 2.8 [3]. The house consists of foundation, is held up by two pillars and has the roof as a goal. Inside the house you can find the 14 principles of Toyota Way and the basics of lean product development.

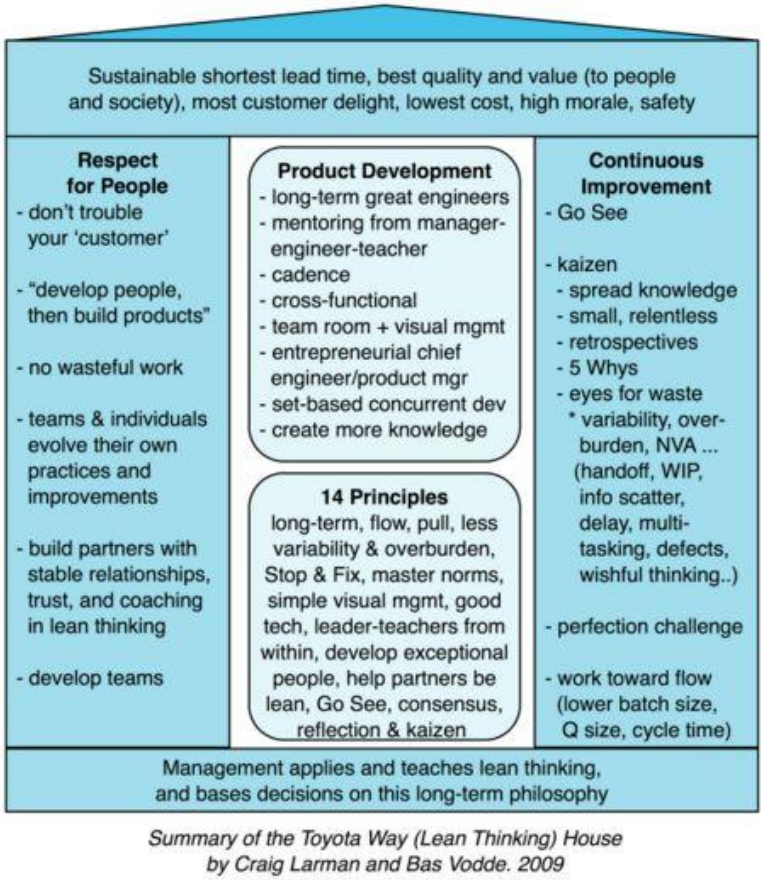

Figure 2.8 *Lean Thinking house [3]*

The pillars of the house are Respect for People and Continuous Improvement, which are also found in the agile principles. Major part of Toyota Way is the waste reduction, which can be seen in the items of the pillars. Respect for People pillar states that no wasteful work should be made and Continuous Improvement advices having eyes for the waste.

Poppendieck lists the seven wastes of software development [17, p.4] as:

- Partially Done Work
- Extra Processes
- Extra Features
- Switching the tasks
- Waiting
- Motion
- Defects

Larman adds other Non-Value-Adding actions to this list. One good example for large scale development purposes is [3, p.61]: Knowledge and information scatter or loss which may be caused by:

- Information in many separate documents rather than centralized for example to a wiki
- Communication barriers such as walls between people or people distributed to multiple locations

On the contrary to the common misconception, the lean thinking is not just tools and removing waste. The whole idea is based on the management's commitment to continuously keep investing and respect its people and promote a culture of continuous improvement [3, p.41]. This is also the foundation of the Lean Thinking house.

2.3.1. Kanban

Kanban comes from the Japanese for "visual card" and it can be used to signal a pull event in a pull driven lean environment. It's used as the operating method in the Toyota Production system [19, p.27]. The classic example of Kanban is the pie store. First a withdraw Kanban card labeled "one pie" is put into shelf behind all pies. When the last pie is eventually sold and taken off the shelf the card is revealed. The "one pie" card is then taken to the bakery to get refill pie for the shelf. This is possible since bakery already had one pie ready in inventory for this purpose. At this time a creation Kanban is also sent to the baker so that he knows to replenish his inventory with one more pie. [3, p.72]

Kanban consists of three rules. Under the first two the Kanban serves as a withdrawal order, an order for conveyance or deliver, and as a work order. The third rule of Kanban prohibits picking up or producing goods without a Kanban. [19, p.40] Kanban is essential in achieving JIT system without excess or urgent need inventory. Lean principle of continuous improvement also exists in Kanban. Therefore users of

Kanban should have a duty to keep improving it with creativity and resourcefulness without allowing it to become fixed at any stage [19, p.42].

Since product and software development differs from the traditional way of manufacturing, tailored ways of using Kanban have been introduced for example in the lean software development. Visual management has proved to be effective in supporting self-directed work and teams. Work is split into smaller pieces or tasks and one card is written to represent each task. The cards are shown to the whole team on a Kanban task board from which someone may volunteer or fulfill the card. [3, p.73] Example of the Kanban task board is shown in Figure 2.9.

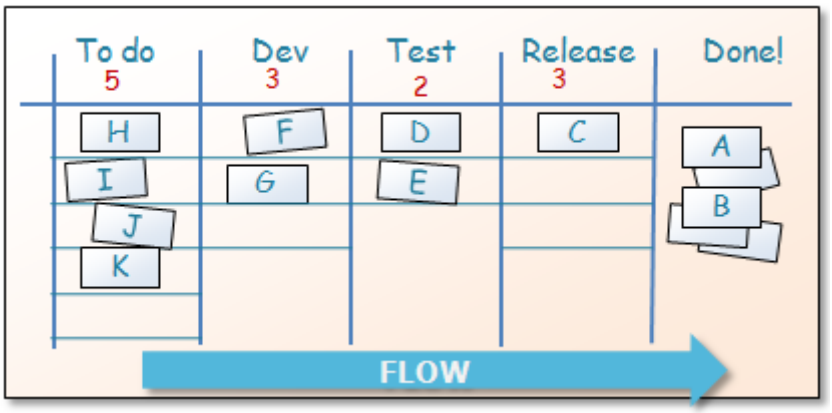

Figure 2.9 *Kanban board [4]*

One of the principles of lean was to reduce waste which can many times be seen as simultaneous work tasks. Therefore it's essential to limit the work in progress (WIP) in different workflow states. This is also shown in the Kanban board where WIP-constraint number is marked to each state. In Figure 2.9 for example the development workflow state can have three ongoing tasks at once. Additionally, each workflow state can have two more phases "Doing" and "Done". The cards are pulled from previous workflow state to the "Doing" phase and then moved to the "Done" after being done. It's then straightforward for the people in the next workflow state to pick the items from the "Done" sub-column.

The cycle time (CT) is the amount of time something takes to go through the process. It's important to measure the CT for each card in the pipeline and make the time as small and predictable as possible. Kanban board gives a clear visual indication if there's a bottleneck for example in the Test state, which can be seen in Figure 2.10. As the flow is disrupted in this state, the tasks will pile up in the previous states and the following state will eventually run out of tasks. The key influence on CT is the variability in time which it takes to develop a new feature. One of the best ways to reduce CT and the variability is to work with small and similar size units of work [12, p.252].

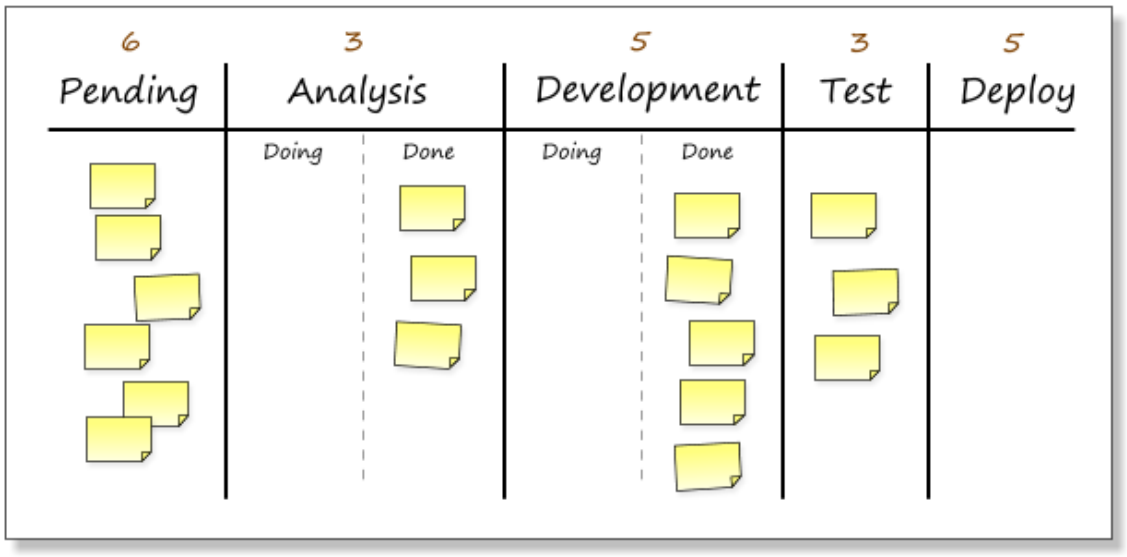

Figure 2.10 *Bottleneck in the test phase [20]*

Overall Kanban is really adaptive as a software process tool. The only constraints are Visualize Your Workflow and Limit Your WIP. [4]

2.3.2. Queuing theory

Queuing theory is applicable in areas which have large products and big features. Usually large batches and long queues exist in these domains. Queuing theory was originally developed to understand the high variability and randomness in telecommunication systems. [3, p.93] Queue management can be used to reduce cycle time in product development where different types of queues exist. In sequential development, like waterfall development, there are WIP-queues. For example specification documents waiting to be programmed and code ready to be tested. Queues may also arise from constrained or shared resources. Many types of queues might exist in development and portfolio management [3, p.97]:

- Projects in portfolio
- New features for a single product
- Detailed requirements in need of concepting and design
- Design documents waiting to be coded
- Code ready to be tested
- Code or modules ready for integration with other developers
- Large components waiting to be integrated
- Large components and systems waiting to be tested
- Features ready for customer demonstration

WIP-queues are identified as a waste and should be removed. They are inventory which are binding time and money where there has not yet been return on investment (ROI). They also increase cycle time and can hide surprising about of technical debt. For example unintegrated code can have lots of hidden problems.

There are two different approaches in the queue management. The optimal one is to eradicate the queue by changing the system. For example by changing from sequential software development to cross functional teams, the WIP-queues related to the item handoffs between groups will vanish. [3, p.99] The second approach is to reduce the batch and queue size, however so that we don't forget the average cycle time. There's a pitfall that if we want to reduce queue size for example by multitasking, we don't actually reach the finish line any sooner since the average cycle time will soar. If the queues must exist, the queuing theory helps to deal with them. [3, p.101]

Common misconception is that no delay or overload exists until the capacity utilization reaches 100 percent. However, slow down happens well before the maximum capacity is reached. Product development is a stochastic system with queues [3, p.109]. Queues can be observed with the ratio of Cycle Time (CT) divided by Service Time (ST).

CT = queue time + ST
Ratio = CT/ST

Queue size and the ratio are correlated and ratio of 1 means that there is no queue. In addition variability increases also the queue size. For example working with bigger batch size causes variability which increases the CT and ratio compared to working with just a single item. One big single requirement is in fact itself a batch of sub-requirements. [3, p.106]

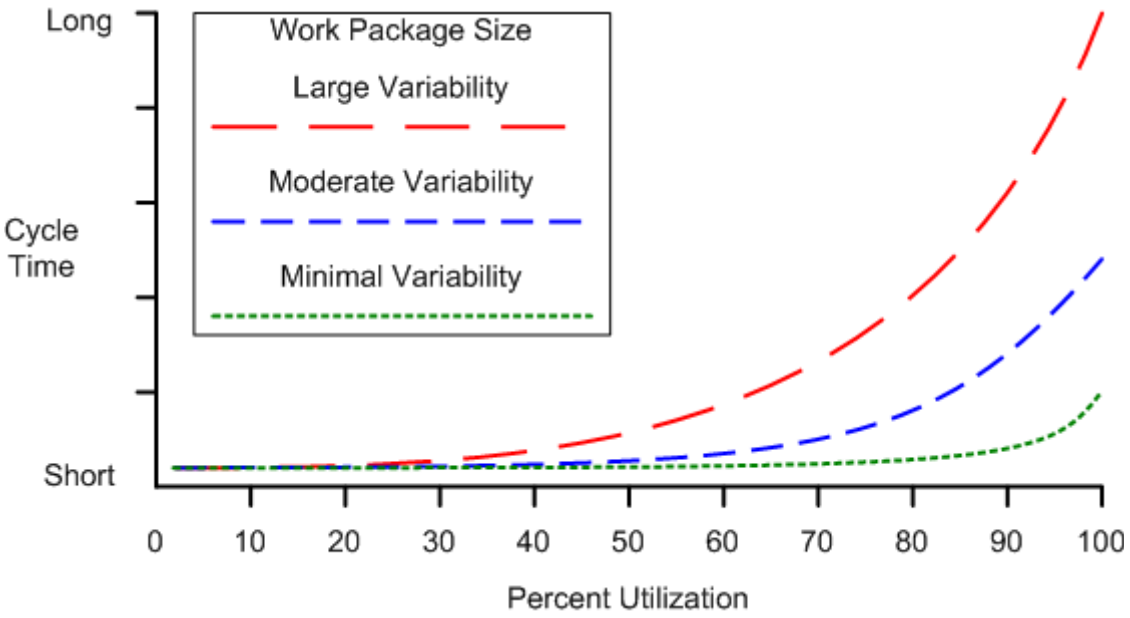

Figure 2.11 *Effect on variation in package size on cycle time [21, p.1]*

Arrival of single item will have minimal variability. This can be modeled mathematically as Markovian queuing model M/M/1/∞, where inter-arrival rate into the queue is Markovian, the service rate is Markovian and it has one server and an infinite

queue. Markovian is a random stochastic process in which the future state cannot be clearly known. Random variability in the process can be caused for example by [3, p.104]:

- Requests arriving at different times with different effort
- Programming or testing effort taking variable time
- People having variation in skills. People work faster or slower and might get sick.

When larger batch sizes arrives we have a M^x/M/1/∞ -system which is a better analogy for traditional product development [3, p.107]. As Figure 2.11 shows when larger batch sizes arrive, the variability and CT will increase.

2.4. Scrum

Scrum was brought to the area of software development in 1993 by Jeff Sutherland in the Easel Corporation [22, p.3]. However Scrum was already named 1986 in Takeuchi & Nonaka's article as a project management style in product manufacturing companies [2, p.4]. In the waterfall software development the whole requirements are documented up-front at a point when it may not be even clear what the customer really wants. On the other hand the Scrum project starts with a vision of the system to be developed. This vision might be blurry at first but gets clearer as the project moves forward. The product owner is responsible for listing the needed functional and non-functional requirements to the prioritized product backlog. Items in the product backlog will be then divided into proposed releases. This is a starting point for a team who will start transforming the Product Backlog into functionality. [23]

In traditional waterfall software development one functional or component team passes the product phases to the next one. The idea of Scrum is that the team chooses the required features from the Product Backlog. Each requirement should be small enough to fit into a single iteration. Requirements are then broken into small pieces or tasks to help estimation and the same team works with the story from start to finish. This is based on the Japanese sashimi-model developed by Fuji-Xerox [24, p.5], which was evolved from the experiences of the waterfall model. Sashimi means sliced fish where each part rests partially on the slice before it. Sashimi model is illustrated in Figure 2.12. Sashimi is taken a bit further in Scrum where all the overlapping phases are reduced to one.

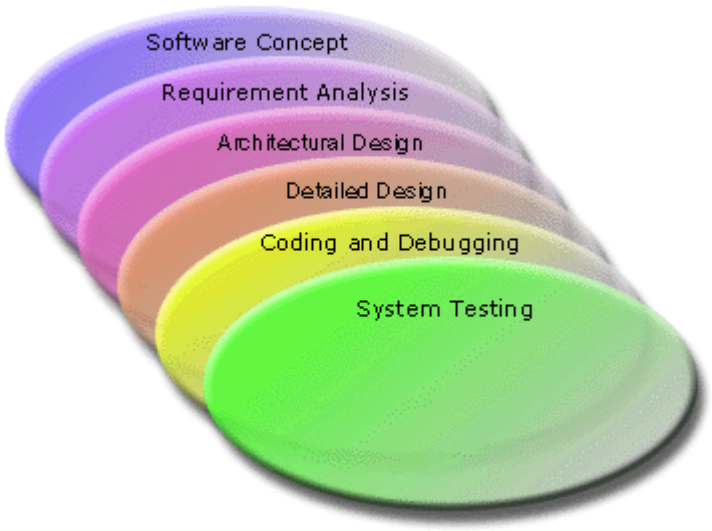

Figure 2.12 *Sashimi model [25]*

After every iteration, or a sprint, a small increment has been made to the potentially shippable product. Iteration should be from two to four weeks and even though the product may don't have all the features for the release, it will be executable after every sprint. The advantage is that if the project runs into problems, with funding for example, some value can still be released. Tasks are kept in the separate prioritized sprint backlog and monitoring can be done for example with a burn down chart showing remaining story points in the current sprint. Requirements or features are often called stories since they are usually documented in the form of user stories. As the sprints are time boxed and regular length, it's possible for team to track their velocity and do more accurate estimates. All this is done in the sprint planning meeting which is held in the beginning of the sprint. The product owner informs the team which features he wants completed. The team estimates the feature size on a scale of relative story points and chooses a suitable amount of work to fit into the next sprint. The team will now commit to the product owner to do its best for the features in the upcoming iteration. The product owner will know how the team is doing after every sprint when the team gives demonstration about the product. The overview of the Scrum is illustrated in Figure 2.13.

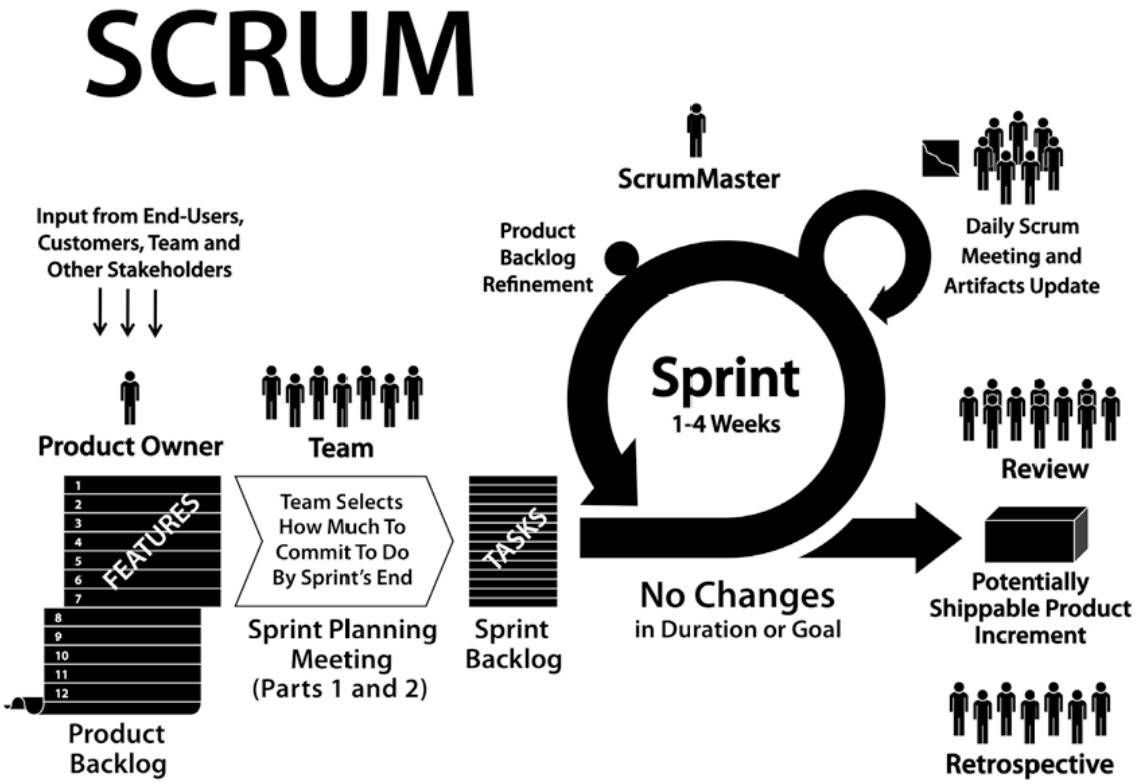

Figure 2.13 *Overview of Scrum [5]*

It's important that both the team and the product owner agree on the definition of done. There might be issues that the team understood the feature be done when the code is checked in but the product owner wanted everything to be deployed to the test server and be verified by an integration test team [26, p.32].

2.4.1. Scrum roles

Scrum introduces three roles, the product owner, the Scrum master and the team [27]. The optimal team size in Scrum is 5-9 members. Team should be cross functional and manufacturing type of teams doing only a part of single component should be avoided. Team individuals should be more like a generalist types than specialists since team member may need to take part of tasks outside her speciality. Members need broad variety of skills since one sprint contains all of the traditional phases in software development; requirement, analysis, development, testing and deployment. The team can make its own ways to reach the sprint goals and therefore the best teams are self organized without too much management involvement. The successful team members have to be responsible and do what it takes to meet their goals. The team has to be prepared to accept the failures and learn from them, so that every team member could comfortable share difficulties and ask for help. [28, p.17] Team will present the sprint results by holding a demo to the product owner after every sprint.

Team has a Scrum master who is not the team leader, but only removes impediments, external interferences and ensures that the team is productive by providing needed resources. He also ensures that the process is being followed and enables close cooperation across all roles and functions.

The product owner defines the features of the product and also prioritizes the items in the backlog according to the market value. Product owner might want to know estimate for specific feature from a team so that ROI value can be calculated by dividing the market value with cost. Product owner sets the release dates and accepts or rejects the results after every sprint.

2.4.2. Daily Scrum

Scrum is relatively adaptive and requires only few items to be followed; sprint planning meeting, daily Scrum, sprint review, product backlog, sprint backlog and the burn down chart. In addition to these an extra sprint retrospective and in large scale development Scrum of Scrums could be included.

Scrum has gained popularity because it's relatively easy to understand and many metaphors exist to reflect different daily situations. The most famous metaphor describes commitment in the daily Scrums:

"*A pig and a chicken are walking down a road. The chicken looks at the pig and says, "Hey, why don't we open a restaurant?" The pig looks back at the chicken and says, "Good idea, what do you want to call it?" The chicken thinks about it and says, "Why don't we call it 'Ham and Eggs'?" "I don't think so," says the pig, "I'd be committed, but you'd only be involved."*

In daily Scrum or daily stand-up meeting everyone involved, the team, the product owner and the Scrum master, take the role of pigs and all other stakeholders such as managers are chickens. Only pigs are allowed to talk, hence meeting should be max 15 min meeting with three questions to each of the team members:

- What did you do yesterday?
- What will you do today?
- Is anything blocking your progress?

Sometimes team has to deal with seagulls:

"Seagulls, like chickens, are birds. But unlike a chicken seagulls are noisier and tend to crap all over the place. Seagulls like lots of other large birds don't live in the farmyard. They just drop in occasionally and make a mess." [29]

Seagull could be a manager or a specialist who pops uninvited to give his advices to the stand-up meeting. The role of Scrum master is to remove such impediments or at least clarify that the person should be either a pig or a chicken.

2.4.3. Estimating in Scrum

Story points are a way to give relative estimate to a task. Size is proportional to the number of story points associated to a task. Story points are given by team after judging

the amount of effort involved in developing the feature, the complexity of developing it, the risk inherent in it and so on. [12, p.36]

Velocity is an amount of story points a team can complete during the iteration. For example completing four stories each estimated two story points, will result to a velocity of eight. Velocity is a great tool for estimating. If features for a release are estimated to take 60 story points, team velocity is 20 and iteration length is two weeks. It will most likely take three iterations or six weeks to have all the features ready for the release. If the estimation of the feature's story points changes it's relatively easy to derive duration by using velocity.

Alternative of story points is to use ideal days. Ideal days will vary from the actual elapsed time, since it's common knowledge that programmer working full time will not have the whole day for programming. Time is spent also answering email, talking to the manager and getting interrupted every fifteen minutes. Mike Cohn has made an observation that most individuals who are assigned to a project full time spend between four and six hours per day on that project [12, p.182]. Examples of why ideal time does not equal elapsed time [12, p.45]:

- *Supporting the current release*
- *Sick time*
- *Meetings*
- *Demonstrations*
- *Personnel issues*
- *Phone calls*
- *Special projects*
- *Training*
- *Email*
- *Reviews and walk-throughs*
- *Interviewing candidates*
- *Task switching*
- *Bug fixing in current releases*
- *Management reviews*

Ideal day size might vary between developers whereas story points are a pure measure of size which won't change even when the team gains experience with a technology [12, p.71] The whole Scrum team participates in the estimation even though the one doing the work would probably give it the best estimate. However we can't be sure who does the work in the agile team so it's important that everyone can give their input.

Cohn states that one of the reasons why agile planning works is that the work in process is always eliminated at the end of each iteration. When all the work is practically eliminated at the start of each iteration, teams don't have the burden of multiple ongoing tasks and can work more easily in short iterations. [12, p.253]

2.5. Scaling issues

Scrum is best used in co-located team with five to nine team members. However if the team size starts to exceed, say, 15 persons the daily stand-up meetings are already taking relatively long. In addition new challenges are introduced if the team members are geographically distributed or many teams are working for a single product owner. There are some means of scaling agile methods. Rather than creating a single 100-person team, the agile encourages to split developer into multiple smaller teams. [12, p.203] Planning a large, multiple team project may require [12, p.203]:

1. Establishing a basis so that all teams understand how to estimate. Teams should have a common unit for estimating. For example all teams could use story points to estimate user stories. There should be some reference user stories which would assure that five story points mean approximately same amount of work within all teams. [12, p.204]
2. Adding details to user stories sooner to understand the big picture and help the teams to coordinate their work better.
3. Performing look ahead planning so that teams can successfully work together. This is especially useful if teams have interdependencies which require them to coordinate their work for next couple of iterations. [12, p.206]
4. Incorporating feeding buffers so that unexpected events in one team won't risk the whole project. If teams have complex interdependencies and just performing look ahead planning isn't sufficient, teams can include a feeding buffer in iterations that deliver capabilities needed by other teams. This lessens the chance of being in the situation where one team is still implementing a feature needed by the second team to start its work. [12, p.208]

However you don't need to do plans in this depth if you only have a single team. This level of planning is not even needed if project contains three or four approximately seven-person teams as long as those teams communicate often [12, p.210].

There isn't any special large scale Scrum framework and there isn't going to be, but tips and tools exist of how to make Scrum work in larger scale. Large-scale Scrum suggestions often reflect the principles from the lean thinking pillar of Continuous Improvement [3, p.290]. They are for inspecting and adapting the product in environment with multiple teams and also for groups of 500 or 1000 people.

Often involvement of the whole organization is needed to adopt agile methods in large scale. Management needs to provide enough resources to support for example effective communications. On the business-level large scale agile development affects also other than the development department, such as sales, QA and submissions. Usually systematic weaknesses are found in the organizational design – in structure, processes, rewards, people, and tasks. Therefore findings of large scale Scrum may be a driving force for an organizational change, which may be needed for successful agile development. Different agile scaling factors can be found to understand the challenges in different areas.

2.5.1. Scrum of Scrums

Large scale Scrum for multiple teams can be arranged with Scrum of Scrums, which is a quite straightforward scaling method for Scrum. It has proven to be an important technique in scaling Scrum to large project teams. It allows smaller Scrum teams to collaborate on large scale with each other especially on areas of overlap and integration. The idea behind Scrum of Scrums is that if the project has for example seven teams, each team will still hold their own daily Scrum meeting. Each team would also have at least one representative in the Scrum of Scrums meeting. The amount of representatives should be chosen so that each Scrum of Scrums meeting would have the optimal amount of five to nine attendees. The teams should decide themselves who will attend to the meeting and the attendees should change over the course of typical project. The team representatives should be chosen based on who will likely to have most to understand and comment on the issues which are currently present at the project. [30] Similar strategy can be used to manage multiple teams with just one product owner as seen in Figure 2.14.

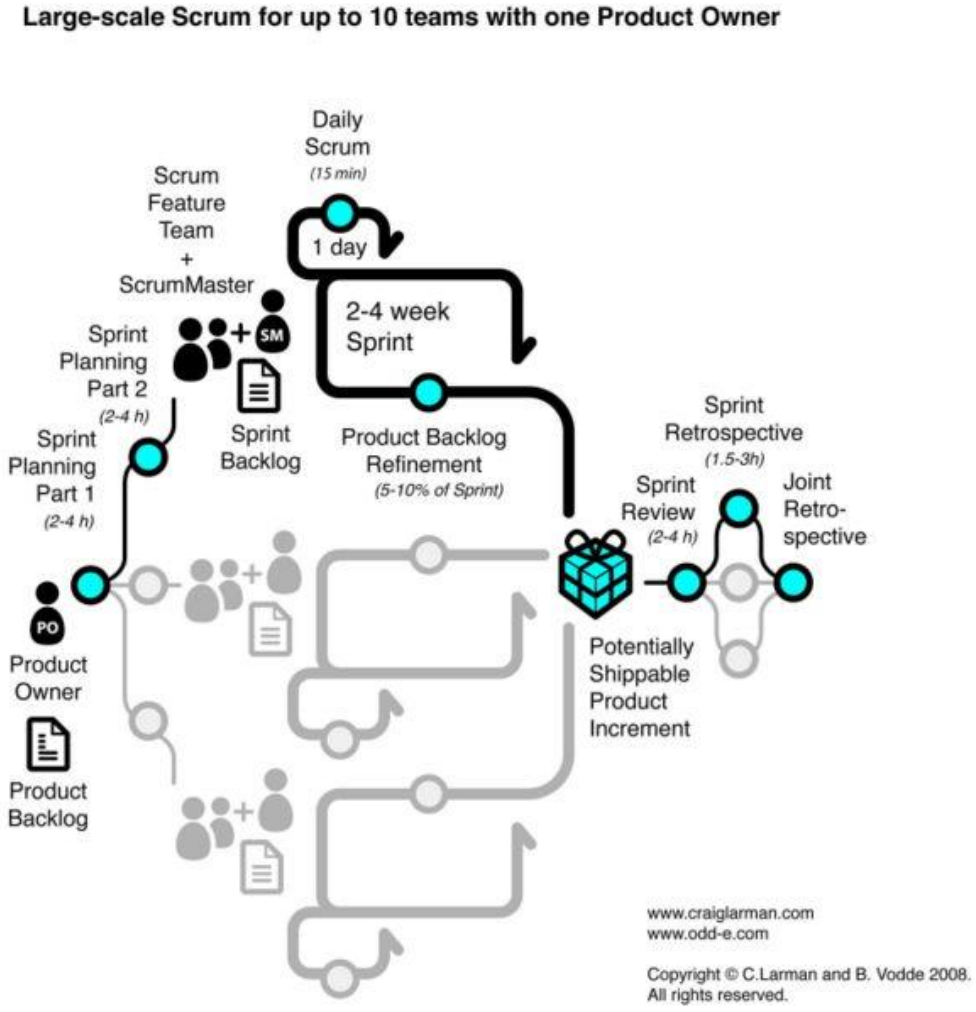

Figure 2.14 *Large scale Scrum with one product owner [3, p.292]*

The teams should be formed around features so that there should be little need for the teams to interact or coordinate, except at the level of integration of code [3, p.294]. The feature team is a long-lived, cross-functional team that completes customer-centric features from the backlog, one by one [3, p.149]. Larman summaries the ideal feature team [3, p.153]:

- *long-lived – the team stays together so they can 'jell' for higher performance; they take on new features over time*
- *cross-functional and cross-component*
- *co-located*
- *work on a complete customer-centric feature, across all components and disciplines (analysis, programming, testing, ...)*
- *composed of generalizing specialists*
- *in Scrum, typically 7 ± 2 people*

According to Ken Schwaber Scrum of Scrums should, like Scrum meetings, be daily 15-minute meetings. On the other hand Mike Cohn suggests that Scrum of Scrum meeting should usually be held only two or three times a week at the time which would be suitable for all the teams [30]. Duration of the meeting should be little longer than the normal daily Scrum. A block of 30 to 60 minutes should be reserved from the calendar for the Scrum of Scrums meeting. Reason for this is that the issues handled in the Scrum of Scrums usually can affect the work up to hundred people and therefore they should be addressed and, if possible, resolved during the meeting. [30] Since Cohn's Scrum of Scrums isn't conducted daily and persons are representing whole teams the following questions suit better than the traditional three questions in the daily Scrums:

1. What has your team done since we last met?
2. What will your team do before we meet again?
3. Is anything slowing your team down or getting in their way?
4. Are you about to put something in another team's way?

The fourth question is important for example when there are interdependencies between teams' tasks.

2.5.2. Large-Scale Scrum with requirement areas

It's also possible to scale the Scrum of Scrum meetings. For example if project has seven Scrum of Scrum meetings, a Scrum of Scrum of Scrums meeting could be held to collaborate on higher level. "*It isn't usually called this, though, because things start to sound a bit silly at some point. Scrum of Scrums often suffices even for these higher levels of meeting.*" [30]

When the project has more than five or ten feature teams it's difficult for teams to work on the whole product and the product owner to work with so many teams. Feature teams can therefore be scaled up by grouping related teams in a requirement

area. This area consists of features which are strongly related from the customer perspective. Features are divided into different area backlogs for the corresponding requirement area teams. [3, p.217] Examples of requirement areas are for example protocols, performance and management related areas [3, p.219]. Each area will have their own area product owner who will act as the product owner in iteration planning and reviews [3, p.221].

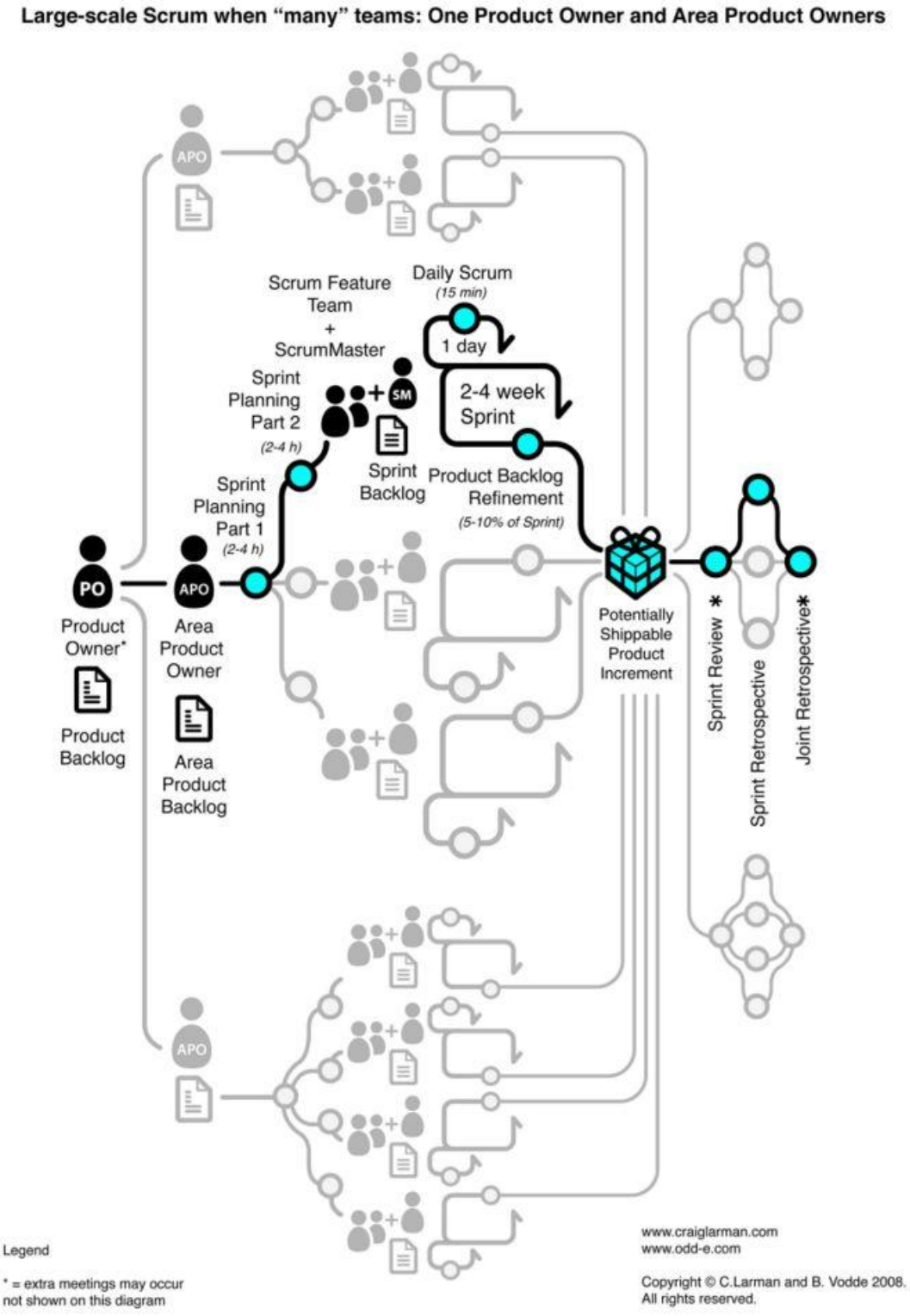

Figure 2.15 *Large scale Scrum with requirement areas [3, p.299]*

The advantages of this approach are that on the high level the large projects can potentially manage with just one product owner and the product backlog as seen in Figure 2.15. The product owner will make product-wide prioritizations but he doesn't need to know the low level details of all the requirement areas.

2.5.3. Distributed Scrum models

Scrum has proved to be effective within co-located teams of five to nine members. However there are different ways of using Scrum in distributed environment. Scrum is even used to tackle different failures in traditional off-sourcing: *"Distribution of individual Scrum teams across geographies eliminates communication failures, XP practices solve integration problems, and daily team meetings maintain high focus on customer priorities."* [31, p.1]

Adding business value in each iteration reduces the overall risk in the global delivery. If one of the teams has misunderstood the requirements, it can be steered to the correct course relatively easy. Success of a global development project shouldn't be measured by the effectiveness of the communication, or engineering techniques. It should be measured by the results delivered and if the teams met the expectations of the end-users [8]. Therefore delivering what customer wants should be the priority in all of the teams. Teams will have different needs when distributing their development. As shown in Figure 2.16, three distributed Scrum models can be described [31, p.2]:

- Isolated Scrums - teams are isolated and distributed across geographies.
- Distributed Scrum of Scrums – Scrum teams are isolated across geographies but they are integrated by a Scrum of Scrums
- Fully distributed Scrums – Scrum teams are having team members distributed across geographies.

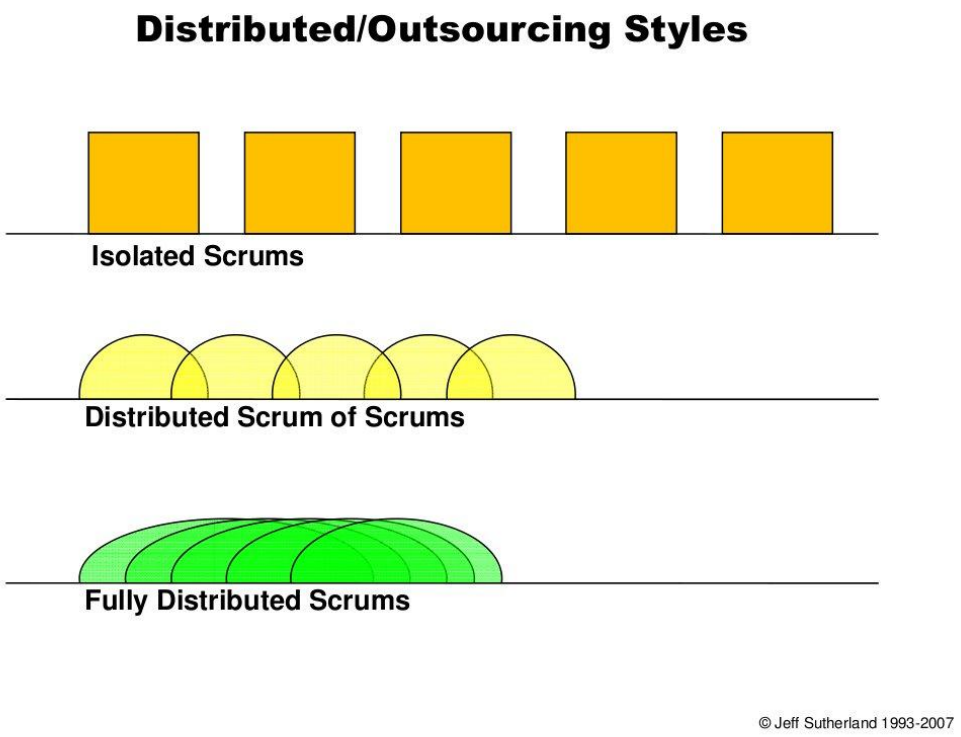

Figure 2.16 *Distributed Scrum models [32]*

Isolated Scrum teams may have problems in transparency and therefore might need to improve their communication practices. The best practice by Scrum Alliance to improve communication is the Distributed Scrum of Scrums model where Scrum masters meet regularly across working locations. In this model the teams have a link with each other but they are still working isolated without having too much dependency. [31, p.2]

In Fully distributed Scrum each team has members dispersed at multiple locations. This is an efficient way to overcome cultural barriers and disparities in the work styles. Team is communicating daily in the daily Scrums, which enhances customer focus and offshore understanding of customer needs [31, p.2].

In all three models so called ambassadors can be used. Ambassador is a team member who visits other sites for a certain period of time. Ambassador usually has a mission to gather as much knowledge and understanding of system and processes as possible, so that when he returns back home he can share the gathered information with the co-located team. Ambassadors are also especially useful for example in overcoming language barriers and improving personal relationships between the team members who otherwise wouldn’t have a chance to meet face-to-face.

2.5.4. Global Delivery Model

Organizations are more and more distributing their production and delivery globally to find global talent, reduce costs and enter the global markets. Technology and increased bandwidth have made the Earth relatively small place for example if compared to the beginning of the previous century. Even the most remote areas can now be accessed relatively fast and highly skilled people can be found from all around the globe. Internet has significantly altered the barrier of entry for many countries. Internet access and the availability of high-speed data lines mean every country with the proper focus and incentives can benefit from global delivery [8].

Software development professionals are hired in large numbers from India, China, Russia, Philippines, Eastern Europe and South-America. There are a few key ingredients for countries to be taken seriously on the global sourcing stage [8]:

- *Attractive tax and business incentives*
- *High amount of qualified and talented people with skills and experience*
- *Long-term commitment to educational systems necessary to produce required talent*
- *Mature processes*
- *Decent infrastructure*

Distribution of teams globally introduces new challenges to the software development. Ways of effective communication are essential to ensure that the benefits of global delivery aren't lost. Traditionally off-sourcing is made so that the remote team works independently and only requirement or design documents are passed from home to the remote team. Lots of highly detailed documentation is needed in this type of approach and even then the failure rates tend to be high. This model is far from ideal and fails to involve all the team members to the key design discussions [8]. Distributed agile practices are the solution to make the global delivery to work effectively, be able to adapt if plans change and produce items that satisfy the customer.

2.5.5. Follow the sun development

Follow the sun development can be used when productive hours of the day need to be maximized. For example if testing is being made outside the teams a well-executed global test program allows you to shorten the test cycles by taking advantage of the time zone differences [8]. Software teams can work with the shared backlog over the 24h cycle if daily integrations are being handled effectively at least between three different sites.

The idea of Conway's Law is that organizations are producing designs which are copies of the communication structures of these organizations [33]. Therefore there's a risk that there exists three different architectures to the same feature if three sites are working on the same backlog without proper communication. It's essential to find shared time between teams during every day. Example of time differences around the globe is shown in Table 2.2.

Table 2.2 *Example of global time differences*

Location	Time	Δ Time	Overall time difference
San Diego, USA	17:00	+00:00	+00:00
Sào Paulo, Brazil	23:00*	+06:00	+06:00
Riga, Latvia	03:00	+04:00	+10:00
Pune, India	06:30	+03:30	+13:30
Shanghai, China	09:00	+2:30	+16:00

*Daylight Saving Time

Teams can do integration in standard working hours (Standard working hours may vary in different countries) when:

MIN(abs(Overall time difference), abs((24 – Overall time difference))) <= 9 h

Also, generally people like to extend their day to the evening instead of arriving early morning.

Table 2.3 *Example of multi site development: San Diego, Riga, Shanghai*

San Diego	Riga	Shanghai
7:00	17:00	23:00
8:00	18:00	0:00
9:00	19:00	1:00
10:00	20:00	2:00
11:00	21:00	3:00
12:00	22:00	4:00
13:00	23:00	5:00
14:00	0:00	6:00
15:00	1:00	7:00
16:00	2:00	8:00
17:00	3:00	9:00
18:00	4:00	10:00
19:00	5:00	11:00
20:00	6:00	12:00
21:00	7:00	13:00
22:00	8:00	14:00
23:00	9:00	15:00
0:00	10:00	16:00

Integration in working hours (08:00 - 17:00)
Integration in extended hours

Some time planning is needed when multiple globally dispersed teams are working with the same tasks. Each team needs to integrate with the other team in the start and in the end of their shift. Effective use of this shared communication window is vital in succeeding in multi site development. For example in Table 2.3 the teams are dispersed between San Diego, Riga and Shanghai and the day could be following:

1. Team members in San Diego integrate in the morning with the team Riga which is currently ending their day. The either team has to integrate on extended hours because of the time differences.
2. When finishing their shift the San Diego passes task to the Shanghai team
3. Shanghai handles the baton to Riga in the afternoon

As three handovers are held each day it's critical that effective processes and good communication channels are established between teams. Cultural differences and bank holidays add extra challenges to the global delivery.

2.5.6. Unified Communications

Currently many different communication channels exist. Real time communication has been made with phone calls, Internet Protocol (IP) calls, audio/video/web conferencing or instant messaging. Non real-time communication can be seen as masses of e-mail, SMS, voicemail and faxes. In addition different companies may have different tools in use which collaborate poorly or not at all. As the teams grow larger the high amount of potential intercommunication channels creates new kind of challenge for the project management. Potential intercommunication channels are illustrated in Figure 2.17 and can be calculated [33, p.18]:

Amount of channels = N(N-1)/2 (N=team members).*

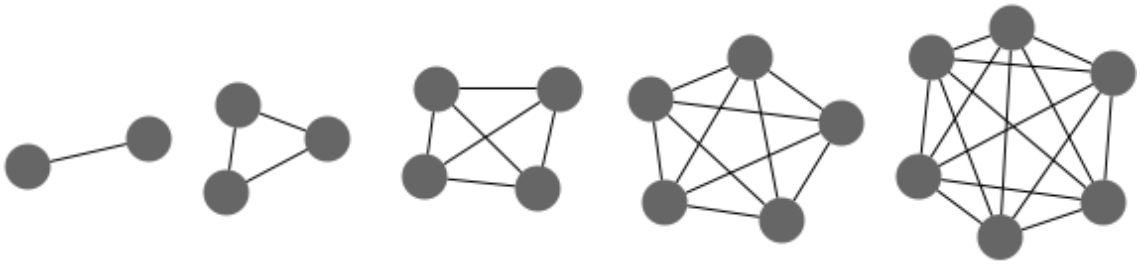

***Figure 2.17** Communication channels [34]*

Effective means of information sharing should be found in the environment with lots of variation. Communication inefficiency is often the root cause for a project failure. The need of communication is different for development teams, customers, end users and executives [35, p.7].

Unified Communications brings together both real and non-real time communication channels. Presence information is used to choose the most suitable communication channel at the given time. If person is for example in the meeting an unobtrusive way of communication could be chosen instead of voice call. A person can then act to the communication when he has a suitable time slot.

Programmed bots can be used to track information sharing and communication so that further optimization in communication channels can be made. Flow of information can be channelled to many different devices from mobile phones to information radiator board in the team room and information quality can be chosen according to the participants. As different geographical locations have different quality

in network infrastructure, voice call can be chosen instead of videoconferencing if not enough bandwidth is available.

An information radiator displays information in a place where passersby can see it [36, p.84]. Information radiators can radiate information into the hallway, team room, across the intranet and onto the people passing by. They are excellent ways of sharing information without disturbing the people whose status is being reported. [36, p.85]

Many times we are indirectly communicating just by radiating information to our surroundings and also by picking up traces of background information even if we are not consciously paying attention. This happens when we overhead conversations and simple just notice things happening around us. Cockburn describes this as an osmotic communication and suggests that the team members would be located in the same room to get the full benefit of it. [36, p.81] On the other hand osmotic communication can be sometimes harmful and disrupt the teams if the background noise doesn't contain any relevant information. This may be the case if for example multiple teams have been packed into the same working area.

2.5.7. Agile enterprise

When the amount of information and work at hand starts to increase, it's critical to control which information is needed by each individual. Dean Leffingwell has made the Agile Enterprise Big Picture shown in Figure 2.18, which reflects different levels or system of systems in the agile enterprise. On the each level people work with different items and are only having the information they need to work on the current scope and context. Agile component teams work on the Project level and may represent a classical Scrum team with maximum of 10 members. On the contrary to the traditional manufacturing component teams, Leffingwell defines ideal software component teams as Define/Build/Test teams which are accountable for their results [15, p.113]. In the big picture these kinds of component teams are working with stories that fit into iterations.

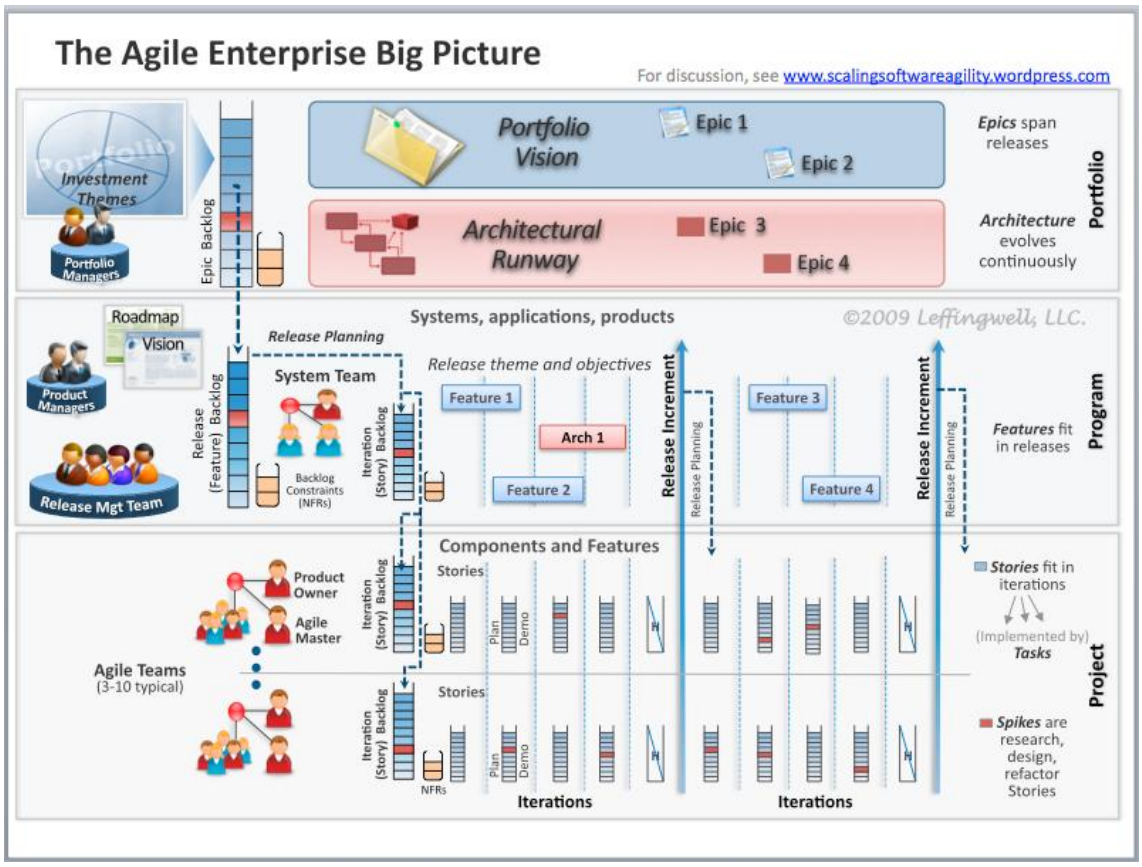

Figure 2.18 *The Agile Enterprise Big Picture [37, p.4]*

In the Leffingwell's picture the Program is above the Project level. There might be dozens of teams working in one Program so coordinating each team's stories on a Program level will become unbearable. Therefore Program introduces features which act as a driving force for stories and typically span multiple teams. In larger enterprises there might be many programs running and each is delivering dozens of features. Therefore features are too fine grained to be handled on the most top portfolio level. Business managers are defining investment portfolios which consist of epics. Unlike features and stories, the epics are not testable but act as a driving force for the programs with features. Epics contain key values which can be used to gain competitive advantage and may be implemented during very long time period, even years.

It's very critical to manage releases at a program level as many teams might be working with the same features in the same program. If the teams are not working on the same cadence, there's a risk that a slow progress of one team will actually delay the external release on the project level as seen in Figure 2.19. The worst thing is that this delay may be revealed at a very late stage.

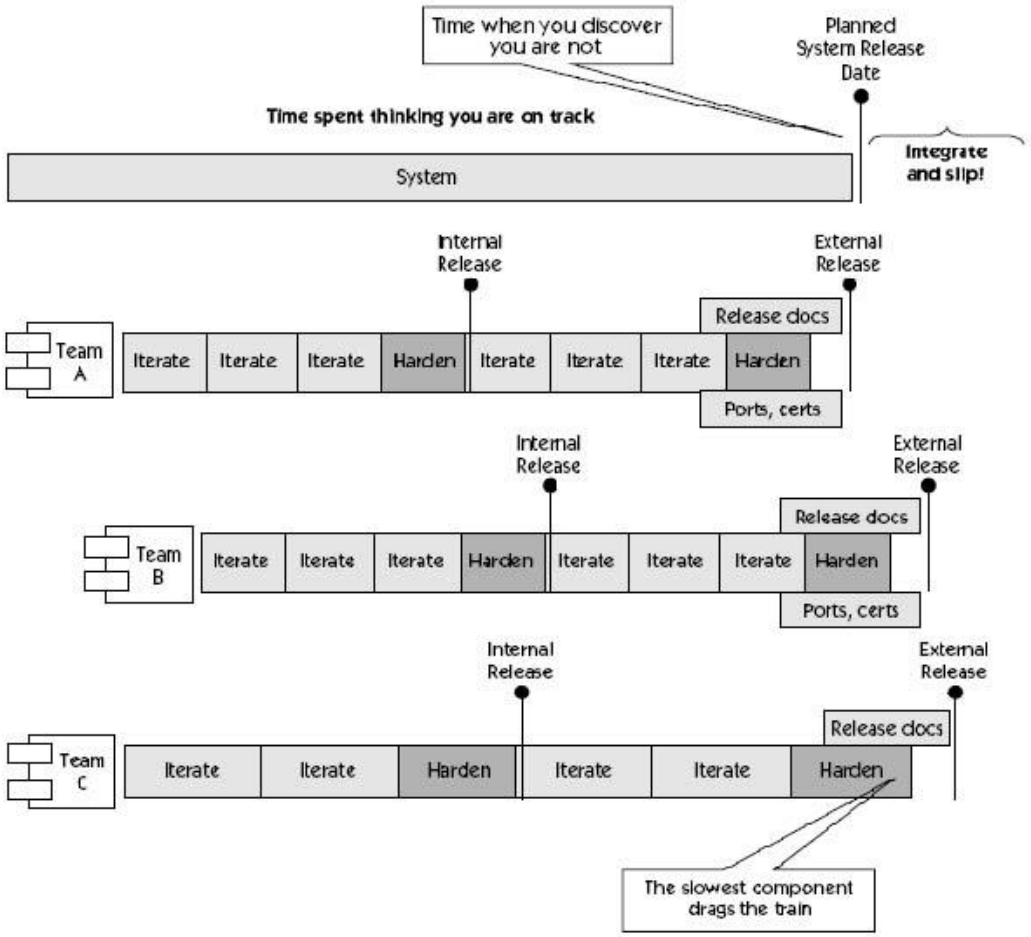

Figure 2.19 *Unsynchronized agile release pattern [15, p.240]*

Figure 2.20 illustrates the agile release train which is synchronized. In the synchronized agile release train model all the component teams are synchronized to the same iteration schedule. The benefit of this is that all the teams will have the internal releases simultaneously and if one of the team lacks behind its relatively slow progress can be detected early. At this point rescoping can be made on the system or component level and resources can be added to the needed teams. However, constraining all the teams to exact release dates means that the components need to be flexible. Therefore teams can still decide themselves what functionality to fit into every iteration. [15, p.242] When iterations are synchronized it's also possible to do simultaneous changes between iterations to multiple teams without actually interrupting teams' ongoing iteration.

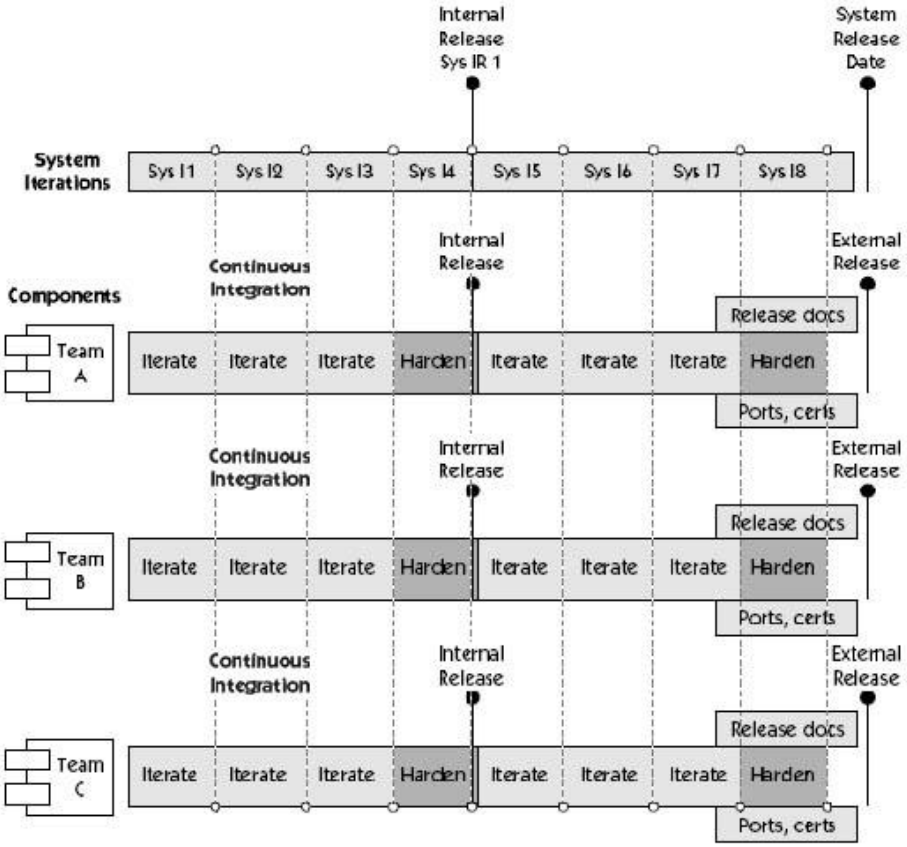

Figure 2.20 *Synchronized agile release train [15, p.243]*

Enterprises often have very complex working environments. Agile Scaling Model (ASM) by IBM concentrates on how the agile methods are adapted to suit the enterprise needs. In addition to the classic core agile development and disciplined agile delivery, the ASM introduces eight scaling factors which will potentially affect the agile requirements strategy [38, p.2]:

1. *Team size*
2. *Geographical distribution*
3. *Regulatory compliance (CMMI, ISO)*
4. *Domain complexity*
5. *Organizational distribution*
6. *Technical complexity*
7. *Organizational complexity*
8. *Enterprise discipline*

Example of the simple and the complex ends of the scaling factors can be seen in Figure 2.21.

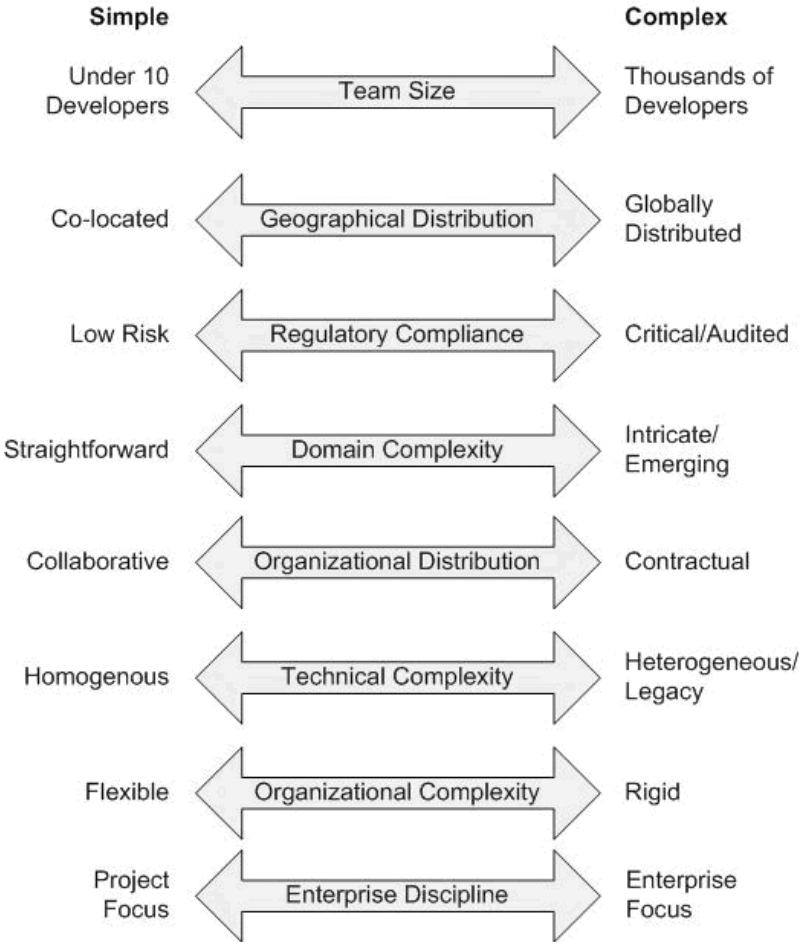

Figure 2.21 *Agile scaling factors [38, p.21]*

The most recognized scaling factor is the team size. It's relatively easy to notice when team is getting too large for example of fitting everyone into common daily stand up meeting. However agile methods will work with larger teams and tasks when teams can be divided into sub teams which will handle the different requirements.

Geographical distribution is getting more relevant nowadays since the ways of communication have broadly improved since the beginning of the millennium and development is dispersed globally. However, even when the technology improves, the face-to-face communication is still the most effective way of communicating. Therefore distribution of the teams, even just to near located cubicles, will increase the communication and coordination risk.

Enterprises might have different types of processes or standards in use. The high level of regulatory compliance with bureaucracy can be a burden when scaling agile methods. When possible issues are recognized it's possible to fit agile for example with CMMI. [39, p.26]

Domain complexity will have an effect on depth of requirements. User stories may be enough for requirements modeling for example in simple informational web site, but for more complex domains, such as air traffic control, more sophisticated modeling techniques are needed. [38, p.22]

Organizational distribution will rise when projects contain members from different departments, companies and involve many different subcontractors. Each organization will have their own work habits and processes which may cause conflicts and make tracking of progress difficult.

Different systems will have a different technical complexity. It's different to build a single platform system from a scratch than if you're building a product based on a legacy system which involves multiple platforms.

Organizational complexity can be a major obstacle in scaling agile methods. When the culture of the organization is flexible it's easy to introduce agile principles. On the other hand in a rigid organization, the organization may for example require heavy up-front planning to get funding for a project. It may be impossible to do any changes to the plans or requirements on the way. This is in contradiction with the agile core principles about evolving and iterative development. Phenomenon was already observed in the Toyota Production System: "*The spine of an older person, like myself, does not bend easily. And, once bent, it does not unbend quickly. This is definitely a phenomenon of aging. We observe the same phenomenom in a business.*" [18, p.46] In addition, different subgroups in the organization can have different visions how they should work. It may be difficult to align these variations of strategies on the larger scale.

Some enterprises introduce more discipline in a way of common enterprise architecture, enterprise business modeling, strategic reuse and portfolio management. Teams need to align the disciplined agile delivery processes to fit into the enterprise level. This may involve of having an enterprise professional as a stakeholder in a team. [38, p.23]

3. SURVEY SETUP

Survey about scaling agile development was conducted in the early 2010. The purpose of the survey was to rank the most known agile and scaling practices and chart in which context they work best. The survey consisted of two phases. First the attendees were asked to fill a web survey form and then more specific questions were asked with e-mail or face-to-face interviews. Answers received in the second phase were used to understand the context and help in the analysis of the survey results.

The attendees were chosen amongst people who had experience of the large projects and agile software methodologies. Total of six attendees participated in the survey from four different companies.

IBM:

- **Scott W. Ambler**, Chief Agile Methodologist
- **Lasse Lilja**, Senior IT Specialist, Accelerated Solution Delivery Team Leader with experience of 88 projects
- **Tuomas Vanhanen**, IT Architect, several years of experience leading agile projects as Scrum Master for multiple customers and has participated in development of agile working ways in IBM Finland.

Nokia:

- **Juha-Markus Aalto**, Agile Transition Program Manager with 20 years of professional life

F-Secure:

- **Pirkka Palomäki**, Chief Technology Officer

Huitale Ltd (Finnish Agile Coaching company):

- **Marko Taipale**, Chief Technology Officer

The web survey had total of 10 questions and was concentrating on the agile scaling challenges related to the team size and geographical distribution. The other agile scaling factors shown in Figure 2.21 had less or no weight.

Questions 2 and 3 were asked to find out if there's a limit in the manageable agile project, or team, size and to see if the answers were in line with the Scrum team size recommendation of 5-9 members. Question 4 was chosen to find out on which level teams can cooperate effectively. Options to this question were intentionally given from two different ends to raise discussion. In Questions 5 and 6 popular agile methods and

practices were asked to see which of them are really kept essential in the industry. Questions 7 and 8 concentrated on distribution and were chosen to conclude which are the greatest challenges and the most effective strategies for distributed agile development. The last two communications questions were chosen from the Ambysoft's Agile Principles and Practices survey [40]. The purpose of asking these same questions was to track if effective communication methods are evolving together with the technology and infrastructure. To further clarify this, a new answering option of high quality videoconferencing was added.

In Questions 5-10, the rating average was calculated for the answers. Answers were given points in the scale from -5 to +5. For example in Question 5 "very effective" option was equal to 5, "effective" 3, "neutral" 1, "ineffective" -3 and "very ineffective" -5 points.

4. RESULTS

Following results were gathered from the survey.

Question 2 *How big is the largest project you've seen to run successfully with agile methods?*

	Response Percent	Response Count
20-50 members	33.3%	2
100-500 members	16.7%	1
500+ members	33.3%	2
I haven't seen a successful scaled agile project	16,7%	1

Question 3 *In your opinion, what is the maximum single agile team size?*

- 8 (team members)
- 6
- 12
- 5
- 9
- 5000

Question 4 *Which working method would you prefer for multiple teams working for same project?*

	Response Percent	Response Count
Multiple teams working with shared requirements / backlog	66.7%	2
Isolated and self sufficient teams divided to different requirement areas	33.3%	1

Other (Please specify. For example the middle approach, where both methods are used)

1. Layered approach for requirements management. Feature teams working toward a unified theme, epic, feature set
2. I must agree shared requirements leads to more coherent outcome, but the feeling is different and some individuals at least have to suffer compared to isolated (teams)

3. Agile Release Train + Lean & Scalable Requirements Model (Epics, Features, Stories), ref. Leffingwell & Aalto
4. Depends on the product / project the teams are working on

Question 5 *Rate following agile methods, based how they work on larger scale*

	Very Effective	Effective	Neutral	Ineffective	Very Ineffective	No idea	Rating Average
Scrum		50.0% (3)	16.7% (1)		33.3% (2)		0,00
Lean / Kanban	16.7% (1)	50.0% (3)	16.7% (1)			16.7% (1)	3,00
(Rational) Unified Process		66.7% (4)		33.3% (2)			1,00
Extreme Programming		16.7% (1)	33.3% (2)		16.7% (1)	33.3% (2)	0,00

Question 6 *How essential are following practices on a project level in large scale agile development?*

	Essential	Good to have	Average need	Little need	Not needed	No idea	Rating Average
Test Driven Development		83.3% (5)	16.7% (1)				2,67
Continuous Integration	100.0% (6)						5,00
Self organizing teams	33.3% (2)	66.7% (4)					3,67
Active stakeholder participation	83.3% (5)	16.7% (1)					4,67
Collective code ownership	33.3% (2)	33.3% (2)	33.3% (2)				3,00
Coding standards	33.3% (2)	33.3% (2)	16.7% (1)	16.7% (1)			2,33
Documentation treated as a requirement		33.3% (2)	33.3% (2)	16.7% (1)	16.7% (1)		0,00
Producing potentially shippable software each iteration	66.7% (4)	33.3% (2)					4,33
Daily Stand-Up Meeting		100.0% (6)					3,00
Burndown tracking	16.7% (1)	50.0% (3)	33.3% (2)				2,67
Demonstrations after iteration	66.7% (4)	33.3% (2)					4,33
Pair programming		16.7% (1)	50.0% (3)	16.7% (1)	16.7% (1)		-0,33
Retrospectives	33.3% (2)	66.7% (4)					3,67

Question 7 *Rate following challenges in succeeding with distributed agile development.*

	Blocker	Major challenge	Average challenge	Minor challenge	No factor	No idea	Rating Average
Geographical distance		50.0% (3)	50.0% (3)				2,00
Time zones		33.3% (2)	50.0% (3)	16.7% (1)			1,00
Culture in different working locations	16.7% (1)	33.3% (2)	50.0% (3)				2,33
Language barrier	16.7% (1)	16.7% (1)	33.3% (2)	33.3% (2)			0,67
Political situation in different working locations		16.7% (1)	50.0% (3)	16.7% (1)	16.7% (1)		-0,33
Infrastructure(bandwidth etc.) in different working locations		16.7% (1)	66.7% (4)	16.7% (1)			0,67
Communication between working locations		83.3% (5)	16.7% (1)				2,67
Organization distribution(members from different companies)		66.7% (4)	16.7% (1)	16.7% (1)			1,67
Processes, standards (CMMI, ISO)		33.3% (2)	33.3% (2)	16.7% (1)		16.7% (1)	1,00

Question 8 *The following strategies for distributed agile development are:*

	Very Effective	Effective	Neutral	Ineffective	Very Ineffective	No idea	Rating Average
Isolated teams		33.3% (2)	33.3% (2)	16.7% (1)		16.7% (1)	1,00
Isolated teams with few members as co-located ambassadors to share knowledge		66.7% (4)	16.7% (1)	16.7% (1)			1,67
Isolated teams having shared stand up meetings with other teams (Distributed Scrum of Scrums)	16.7% (1)	33.3% (2)	50.0% (3)				2,33
Teams having both co-located and near-shore distributed members (Fully distributed Scrum)		33.3% (2)	50.0% (3)	16.7% (1)			1,00
Teams having both co-located and off-shore distributed members (Fully Distributed Scrum)		16.7% (1)	33.3% (2)	33.3% (2)	16.7% (1)		-1,00

Question 9 The following strategies for conveying information WITHIN THE TEAM are:

	Very Effective	Effective	Neutral	Ineffective	Very Ineffective	No idea	Rating Average
Face-to-face (F2F) communication	83.3% (5)	16.7% (1)					4,67
F2F at a Whiteboard	100.0% (6)						5,00
Detailed documentation			16.7% (1)	50.0% (3)	33.3% (2)		-3,00
Email			50.0% (3)	16.7% (1)	33.3% (2)		-1,67
Overview documentation		50.0% (3)	33.3% (2)	16.7% (1)			1,33
Overview diagrams	16.7% (1)	50.0% (3)	33.3% (2)				2,67
Online Chat		66.7% (4)	33.3% (2)				2,33
Teleconference Calls		50.0% (3)	50.0% (3)				2,00
Telepresence / Very high quality videoconferencing	16.7% (1)	50.0% (3)	33.3% (2)				2,67
Videoconferencing	16.7% (1)	33.3% (2)	50.0% (3)				2,33

Question 10 The following strategies for conveying information WITH STAKEHOLDERS are:

	Very Effective	Effective	Neutral	Ineffective	Very Ineffective	No idea	Rating Average
Face-to-face (F2F) communication	83.3% (5)	16.7% (1)					4,67
F2F at a Whiteboard	66.7% (4)	16.7% (1)	16.7% (1)				4,00
Detailed documentation		16.7% (1)	33.3% (2)	33.3% (2)	16.7% (1)		-1,00
Email			100.0% (6)				1,00
Overview documentation	33.3% (2)	33.3% (2)	16.7% (1)	16.7% (1)			2,33
Overview diagrams	16.7% (1)	66.7% (4)		16.7% (1)			2,33
Online Chat	16.7% (1)	16.7% (1)	66.7% (4)				2,00
Teleconference Calls		83.3% (5)	16.7% (1)				2,67
Telepresence / Very high quality videoconferencing	16.7% (1)	66.7% (4)	16.7% (1)				3,00
Videoconferencing	16.7% (1)	66.7% (4)	16.7% (1)				3,00

5. ANALYSIS OF THE RESULTS

In Question 2 the participants were asked “How big is the largest project you’ve seen to run successfully with agile methods?” Some variation was noticed in the answers due to different backgrounds of the respondents. In addition, discussion was raised of the fact that how could we measure the successful use of agile methods in a project. Therefore, all of the attendees were asked if there exists some key numbers which could be monitored or is the subjective opinion of different stakeholders the only real way to judge the successful agile transition. One of the attendees stated that he hasn’t seen a successful scaled agile project because no objective indicators could be shown. There seems also to be a challenge in keeping up the overall architecture when using agile methods on the long run. Similar challenge with architecture in the agile teams was discussed in the survey by Forrester Research. It seems that besides the challenge, the teams are successfully figuring out how to balance long-term architectural consideration with each iterations deliverables. [41, p.13] Most of the attendees accepted that it’s hard to find genuinely objective indicators for successful agile adoption. However, the following indicators were suggested:

Team:

- Delivered business value by monitoring burn down charts
- Efficiency by monitoring velocity
- Agility which can be measured with agile self assessment tools for teams

Organization:

- Delivery reliability / predictability
- Quality increase, Quality debt
- Transparency
- Get rid of the item overhead and low ROI items
- Increase in customer satisfactory due to regular feedback
- Reputation as a convincing vendor
- According to agilists like Jeff Sutherland, Jim Coplien: 100% increase in productivity and 50% less defects in the final product
- Remarkably shortens the time from the initial idea to the delivery of the final product.
- Ability to change direction, business agility

Financial:

- Positive effect on the company’s profit and loss statement.

- ROI, noticeable increase in savings and profit for investments

Question 3 asked about the maximum agile team size. The responses were in line with the Scrum recommendation of 5-9 team members. However answer by Scott Ambler was 5000 and he clarifies it as follows: "*I've seen teams up to 1000 people, so 5000 could be a reasonable upper limit for a program. I don't see why you couldn't apply agile on very large programs.*" Ambler's answer was in the context of program level but on a team level he admits that you'll never have for example a team of 200 people, but instead a collection of sub teams that add up to 200 people [42]. According to the Forrester research many agile teams are however larger than the Scrum recommendation of 5-9 members [41, p.15].

In Question 4 two options from different ends were given. Two of the answerers recommended the first option with shared backlog for multiple teams. One admitted that shared backlog may result in the best outcome but some of the individuals may suffer from this approach. Basically the fastest team with most skill will slow down and the slowest will have to work on its limit when using shared backlogs. The decision depends of the project and most of the teams will choose something between these two ends. Two of the respondents suggested Leffingwell's big picture approach.

Question 5 got critic because it was hard to know in which context the different methodologies should be used. For example, Scrum and Kanban contain good team level practices but work poorly on the enterprise level. On the other hand there are many good lean practices which scale well. Most of the answerers indicated that there's no silver bullet and usually practices from different methodologies have to be tailored to fit the enterprise needs.

Question 6 was about the need of different practices in large scale agile development. Answers show that the most essential practices are Continuous Integration with average rating of 5, Active stakeholder participation (4.67), Producing potentially shippable software each iteration (4.33), Demonstrations after iteration (4.33), Retrospectives (3.67) and Self organizing teams (3.67). The most of the practices with the highest rating were the ones which involve getting feedback and using it for continuous improvement. This never ending cycle of learning, also called kaizen, is one of the main principles in lean.

In Question 7 the respondents were asked to rate the challenges in succeeding with distributed agile development. Highest rating was given to the challenge in the communication between working locations (2.67). Therefore teams should be co-located whenever possible to minimize this challenge. The second highest challenge was the culture in different working locations (2.33). One of the respondents rated this as a blocker since very hierarchical and bureaucratic working culture in one site can block using agile development in multiple sites. Therefore it's important that the agile principles and culture is driven in across all the sites on enterprise level. Geographical distance got average rating of 2.00 whereas time zones got rating of 1.00. One of the

respondents rated geographical distance as a major challenge and time zones as a minor. He clarifies that even though these two might correlate, the first principle is to keep the teams co-located because most of the communication is happening inside the teams. Communication between two distributed teams can be arranged to some level. For example the daily Scrums can be arranged between Finland and Japan (+7h) so that suitable time can be found for both teams. Finding suitable time starts to become too big of a challenge if the time difference grows larger than this.

Question 8 concerned of different strategies for distributed agile development. The options which involved having isolated teams got higher average rating than options in which teams have members in different locations. These results are similar with the results gathered from Question 7. However it seems that some communication is still needed between the teams, since the best average rating was given to the distributed Scrum of Scrums strategy (2.33). One of the answerers explained that in Scrum of Scrums model the different teams have to work towards a common goal, communication overhead will be smaller and teams can learn from each other. One of the respondents was concerned that the isolated teams shouldn't be too isolated but still need to have connection to their stakeholders. Fully distributed Scrum with off-shore members (-1.00) was seen as the most ineffective of the given strategies.

Questions 9 and 10 were chosen from the Ambysoft's Agile Principles and Practices survey [40]. The responses show that no technology can match face-to-face communication. Videoconferencing is seen as the best communication alternative to face-to-face communication within the team and with stakeholders. One of the respondents rated very high quality videoconferencing more effective than normal videoconferencing. He had experience in using sophisticated telepresence rooms as a communication channel within a team. These rooms are like normal meeting rooms but with big screens and very high quality video connection. According to the respondent, using the room had a positive effect on the team spirit and he considered team meetings being fun.

6. RELATED WORK

There are quite many books and researches available now, when the agile development methods have become more popular in the software development community. For example Ambysoft and Forrester have conducted surveys to the IT professionals working with agile methods.

I've reflected my results to two agile surveys. Ambysoft's Agile Practices and Principles survey contains relevant information about the agile practices and also communication strategies. Forrester's survey has important knowledge about agile adoption in the current state of the IT industry.

6.1. Agile Practices and Principles survey

Ambysoft's Agile Practices and Principles survey [40] was conducted in the July 2008 to 337 respondents.

- 36.9% were developers, 35.9% management
- 42% had 10-20 years IT experience, 17.3% had 21+ years
- 31.3% worked in organizations of 1000+ people
- 57.3% respondents were working in North America, 22.7% in Europe, 7.2% in Asia

Rating from -5 to +5 was used in the questions. Following average rating results were gathered:

Project Management Practices

- Iteration planning (3.54)
- Daily Scrum Meeting (3.29)
- Prioritized worklist (3.08)
- High-level release planning (2.19)
- Retrospectives (1.84)
- One Product Owner (1.55)
- Burndown chart (1.51)
- Potentially Shippable Software (1.51)
- Status Reports (1.15)
- Story Board with Task Breakdowns (0.83)

Development Practices

- Coding Standards (2.30)
- Collective Code Ownership (1.97)
- Continuous integration (1.94)
- Database standards (1.86)
- UI standards (1.65)

- Pair programming (-1.34)

Quality Practices

- Code Refactoring (1.79)
- UI Testing (1.54)
- Automated Developer Testing (1.08)
- TDD (-0.08)
- UI Refactoring (-0.22)
- Database refactoring (-0.31)
- Automated Acceptance Testing (-0.87)
- Database regression testing (-1.03)
- Executable Specs (-1.43)

Modeling Practices

- Active Stakeholder Participation (1.95)
- Requirements Envisioning (1.50)
- Architecture Envisioning (1.31)
- Documentation as Requirement (0.49)
- Model Storming (-0.84)

Table 6.1 *Effectiveness of Communication Strategies [40]*

Communication Strategy	Within Team	With Stakeholders
Face to face(F2F)	4.25	4.06
F2F at Whiteboard	4.24	3.46
Overview diagrams	2.54	1.89
Online chat	2.10	0.15
Overview documentation	1.84	1.86
Teleconference calls	1.42	1.51
Videoconferencing	1.34	1.62
Email	1.08	1.32
Detailed Documentation	-0.34	0.16

When we reflect the results from the Ambysoft's survey to Question 6 in the thesis, we find out that the only practice in both surveys with average rating 3.00 or above is the Daily Stand-Up Meeting, which got average 3.29 in Ambysoft's survey and 3.00 in the thesis. Practices with most difference in the average ratings were Continuous Integration (1.94, 5.00), Potentially Shippable Software (1.51, 4.33), TDD (-0.08, 2.67) and Active Stakeholder Participation (1.95, 4.67). One should also note that the average of the average ratings was lower in the Ambysoft's survey (1.39, 2.81), when practices found only from both surveys are included. If we look at the mere rank of the practices found from the both surveys, the most difference can be found from the project management practices. In Ambysoft's survey Potentially Shippable Software is ranked the last and in the thesis it's seen as the most important project management practice. In the

development practices the Ambysoft's survey ranks Coding Standards as the most essential practice whereas in the thesis it's behind Continuous Integration and Collective Code Ownership.

Table 6.1 lists average ratings gathered by Ambysoft about effectiveness of communication strategies within a team and with stakeholders. These can be compared to thesis Questions 9 and 10. Both surveys got similar results with face-to-face communication being clearly the most effective and detailed documentation the least effective communication strategy. However there's one interesting difference when it comes to using technology as a communication strategy. In Ambysoft's survey the teleconference calls and videoconferencing got relatively low average rating and ranked behind the overview documentation. However, the results gathered one and half year later for the thesis show that tele- and videoconferencing have lessened the gap to the face-to-face communication and clearly passed the overview documentation as a communication strategy.

6.2. Agile Development: Mainstream Adoption Has Changed Agility

Forrester conducted three agile related surveys in 2009. The summary of the surveys is gathered to a document by Dave West and Tom Grant, Ph.D [41].

Global Agile Adoption Online Survey was fielded to 60 technology professionals. One of the questions was "Which of the following aspects of agile do you use?" Continuous Integration and build was chosen by 13 out of 25 IT departments and 22 out of 27 technology departments. Test Driven Development was picked by 11 and 11 respectively. High adoption rate of Continuous Integration indicates that investment in it pays off quickly. However on the contrary to the Continuous Integration, the TDD still remains an objective for many teams.

Agile is entering to the mainstream since teams pragmatically mix methodologies. Instead of sticking strictly with one agile methodology, teams pick different agile methods and often include non-agile techniques as well.

Adoption of Agile: "How would you characterize your adoption of Agile?" was asked from 52 development professionals who have adopted agile

- 27% We stick to a particular Agile methodology as closely as possible
- 35% We deliberately mix Agile with other methodologies
- 39% We deliberately mix different Agile methodologies

Team size: Only about 50% of the teams surveyed for the Global Agile Adoption Online Survey reported having 50 members or fewer. Third of the teams consisted of more than hundred members.

Distribution: Only 17% of the surveyed teams are co-located. For most of the teams (78%) up to 50% of their members are co-located. Remaining 22% reported that more than 50% of the team members are distributed in the different locations.

Few observations can be made when we compare the findings of the Forrester survey and the survey in this thesis. In Question 5 none of the agile methods got negative average rating which is in line with the Forrester's results of Agile Adoption. Professionals are mixing different agile methodologies and using the most applicable for current context.

When looking at the agile aspects, Continuous Integration was given the highest average rating (5.00) in Question 6 and also the Forrester's survey shows that it's widely adopted in the technology departments. In addition, the lower average rating of TDD (2.67) is correlating with the lower adoption rate in the industry. Team sizes observed in the Global Agile Adoption Survey are much bigger than which were suggested as a successful agile team size in Question 3. On the other hand, most of the teams were reported to have up to 50% of their members co-located which is reflecting the average ratings gathered in Question 8 about the agile distribution strategies.

7. CONCLUSION

Using agile methodologies in larger scale has many challenges. Team level practices like Scrum and Kanban work poorly on the organizational level, but there are different strategies to get benefits of agile to the whole enterprise. Dean Leffingwell and Craig Larman both point out that "divide and conquer" is the correct way to handle teams and tasks on large scale. Scott Ambler has introduced different agile scaling factors, which can be used in the enterprise to identify the agile scaling challenges and choose the right tools for the current context. Working on the organizational level can introduce many unpredicted events and constraints. For example Mike Cohn gives an advice that iterations shouldn't be in cadence with the fiscal quarters: "*There are too many unknowns, unknowables, and uncertainties in software development for me to want to risk recognizing revenue by targeting an end-of-the-quarter release."* [12]

Communication within distributed teams is still seen as a major challenge even though advanced technology is available. Face-to-face communication method is the most effective way of communicating and therefore teams should be co-located whenever possible. Being co-located, the team members will also benefit from the indirect osmotic communication, as most of the communication is being made inside the teams. When isolated, the distributed teams can work together by using for example Scrum of Scrums. However, global distribution can introduce cultural challenge which can be a blocker for successful agile scaling.

As agile adoption has become mainstream, the organizations use practices from many different agile methods and tailor them to fit their needs. Successful use of agile methods is not seen as a black and white event, in which the organization is either agile or not. It's difficult to point out genuinely objective indicators to measure the success of agile and most of the times judgment is based on the opinion of different stakeholders. Like generally in the world of software development, there's no silver bullet for scaling agile methods either. There are no best practices, but only adequate for current context.

REFERENCES

[1] Highsmith, J. History : The Agile Manifesto [WWW]. 2001. [Cited 28/11/2009]. Available at: http://agilemanifesto.org/history.html

[2] Takeuchi, H. and Nonaka, I. The New New Product Development Game. Harvard Business Review January-February 1986.

[3] Larman, C. and Vodde, B. Scaling Lean & Agile development: Thinking and Organizational Tools for Large-Scale Scrum. 2008, Addison-Wesley. 368 p.

[4] Kniberg, H. Kanban vs Scrum [PDF]. Version 1.1, 2009. [Cited 02/12/2009]. Available at: http://www.crisp.se/henrik.kniberg/Kanban-vs-Scrum.pdf

[5] Deemer, P., Benefield, G., Larman, C. and Vodde, B. The Scrum Primer [PDF]. Version 1.2 2009. [Cited 23/11/2009]. Available at: http://scrumtraininginstitute.com/home/stream_download/scrumprimer

[6] Larman, C. Applying UML and Patterns: Introduction to Object-Oriented Analysis and Design and Iterative Development. Third Edition. 2004, Prentice Hall PTR. 736 p.

[7] Wikipedia. Capability Maturity Model Integration [WWW]. [Cited 28/11/2009]. Available at: http://en.wikipedia.org/wiki/Capability_Maturity_Model_Integration

[8] Hussey, J. M. and Hall, S. E. Managing Global Development Risk. 2007, Auerbach Publications. 256 p.

[9] Godfrey, S. What is CMMI? [PPT]. [Cited 11.1.2010]. Available at: http://software.gsfc.nasa.gov/docs/What%20is%20CMMI.ppt

[10] Manifesto for Agile Software Development [WWW]. 2001. [Cited 28/11/2009]. Available at: http://agilemanifesto.org/

[11] Principles behind the Agile Manifesto. [Cited 28/11/2009]. Available at: http://agilemanifesto.org/principles.html

[12] Cohn, M. Agile Estimating and Planning. 2005, Prentice Hall PTR. 368 p.

[13] Furugaki, K., Takagi, T., Sakata, A. and Okayama, D. Innovation in Software Development Process by Introducing Toyota Production System. Fujitsu Sci. Tech. J., 43,1, p.139-150(January 2007)

[14] Beck, K. Test Driven Development: By Example. 2002, Addison-Wesley. 240 p.

[15] Leffingwell, D. Scaling Software Agility: Best Practices for Large Enterprises. 2007, Addison-Wesley. 384 p.

[16] Beck, K. Extreme Programming explained: Embrace change. Second Edition. 2004, Addison-Wesley. 224 p.

[17] Poppendieck, M and Poppendieck, T. Lean Software Development: An Agile Toolkit. 2003, Addison-Wesley. 240 p.

[18] Lean Enterprise Institute. Principles of Lean [WWW]. [Cited 03/02/2010]. Available at: http://www.lean.org/WhatsLean/Principles.cfm

[19] Ohno, T. Toyota Production System: Beyond Large-Scale Production. 1988, Productivity Press. 152 p.

[20] Kanban Distilled for Managers [WWW]. [Cited 02/12/2009]. Available at: http://www.kanbandistilled.com/

[21] Berteig, M. Agile Work Uses Lean Thinking [PDF]. [Cited 03/12/2009]. Available at: http://www.berteigconsulting.com/Whitepaper%20-%20Agile%20Work%20Uses%20Lean%20Thinking.pdf

[22] Sutherland, J. Inventing and Reinventing SCRUM in Five Companies [PDF]. 2001. [Cited 19/12/2009]. Available at: http://www.agilealliance.org/show/888

[23] Schwaber, K. Agile Project Management with Scrum. 2004, Microsoft Press. 192 p.

[24] Umemoto, K., Endo, A. and Machado, M. From Sashimi to Zen-in: The Evolution of Concurrent Engineering at Fuji Xerox [PDF]. [Cited 12/12/2009]. Available at: http://www.jaist.ac.jp/ks/labs/umemoto/Fuji-Xerox.pdf

[25] Acidaes Solutions Pvt. Ltd. Sashimi model [GIF]. [Cited 21/12/2009]. Available at: *http://www.acidaes.com/Graphics/sashimi.gif*

[26] Kniberg, H. Scrum and XP from the Trenches: How we do Scrum [PDF]. 2007. [Cited 17/11/2009]. Available at: http://www.infoq.com/minibooks/scrum-xp-from-the-trenches

[27] Scrum Alliance. Scrum Roles [WWW]. [Cited 02/12/2009]. Available at: http://www.scrumalliance.org/pages/scrum_roles

[28] Elssamadisy, A. Agile Adoption Patterns: A Roadmap to Organizational Success. 2008, Addison-Wesley. 408 p.

[29] Miller, A. Scrum Bestiary – The Seagull [WWW]. 2007. [Cited 02/12/2009]. Available at: http://www.ademiller.com/blogs/tech/2007/10/scrum-bestuary-the-seagull/

[30] Cohn, M. Advice on Conducting the Scrum of Scrums Meeting [WWW]. 2007. [Cited 21/12/2009]. Available at: http://www.scrumalliance.org/articles/46-advice-on-conducting-the-scrum-of-scrums-meeting

[31] Sutherland, J., Schoonheim, G., Rustenburg, E. and Rijk, M. Fully Distributed Scrum: The Secret Sauce for Hyperproductive Offshored Development Teams [PDF]. [Cited 30/11/2009]. Available at: https://blog.itu.dk/KF12-F2010/files/2010/02/xebia-distributed-scrum-final.pdf

[32] Sutherland, J., Schoonheim, G. and Rijk, M. Fully Distributed Scrum: Replicating Local Productivity and Quality with Offshore Teams [PDF]. 2009, Proceedings of the 42nd Hawaii International Conference on System Sciences. [Cited 30/11/2009]. Available at: http://www.computer.org/portal/web/csdl/doi/10.1109/HICSS.2009.740

[33] Brooks, F. P. The Mythical Man-Month: Essays on Software Engineering. Second Edition. 1995, Addison-Wesley. 336 p.

[34] Francl, L. Communication Channels [PNG]. [Cited 04/12/2009]. Available at: http://railspikes.com/assets/2008/1/7/Communication_Channels.png

[35] Rational Software, IBM. Getting requirements right: avoiding the top 10 traps [PDF]. 2009. [Cited 01/02/2010]. Available at: http://library.dzone.com/whitepapers/getting-requirements-right

[36] Cockburn, A. Agile Software Development. 2001, Addison-Wesley. p. 304.

[37] Leffingwell, D. The Big Picture of Enterprise Agility [PDF]. Rev. 2. 2009. [Cited 01/03/2010]. Available at: http://scalingsoftwareagility.files.wordpress.com/2009/08/the-agile-enterprise-big-picture-2.pdf

[38] Ambler, S. W. The Agile Scaling Model (ASM): Adapting Agile Methods for Complex Environments [PDF]. 2009. [Cited 17/01/2010]. Available at: ftp://ftp.software.ibm.com/common/ssi/sa/wh/n/raw14204usen/RAW14204USEN.PDF

[39] Glazer, H., Dalton, J., Anderson, D., Konrad, M. and Shrum, S. CMMI® or Agile: Why Not Embrace Both! [PDF]. 2008. [Cited 20/11/2009]. Available at: http://www.sei.cmu.edu/library/abstracts/reports/08tn003.cfm

[40] Ambler S. W. Agile Principles and Practices survey [WWW]. 2008. [Cited 01/03/2010]. Available at: http://www.ambysoft.com/surveys/practicesPrinciples2008.html

[41] West, D. and Grant, T. Agile Development: Mainstream Adoption Has Changed Agility [PDF]. 2010. [Cited 01/03/2010]. Available at: http://www.forrester.com/rb/Research/agile_development_mainstream_adoption_has_changed_agility/q/id/56100/t/2?action=5

[42] Ambler, S. W. Large Agile Teams [WWW]. 2008. [Cited 01/03/2010]. Available at: https://www.ibm.com/developerworks/mydeveloperworks/blogs/ambler/entry/large_agile_teams?lang=en_us

8. APPENDIX 1: WEB SURVEY

Page 1: General, Scaling Agile
In the questions consider situations from ~20 to hundreds of people. Survey has 10 questions and ideally results would be individually discussed afterwards.

1. Please give your name (Will be kept private and used to analyse the context of answers):

_____

2. How big is the largest project you've seen to run succesfully with agile methods?

- 10-20 members
- 20-50 members
- 50-100 members
- 100-500 members
- 500+ members
- I haven't seen a successful scaled agile project

3. In your opinion, what is the maximum single agile team size?
Number of team members: _____

4. Which working method would you prefer for multiple teams working for same project?
Multiple teams working with shared requirements / backlog
Isolated and self sufficient teams divided to different requirement areas
Other (Please specify. For example the middle approach, where both methods are used) _____

5. Rate following agile methods, based how they work on larger scale

	Very effective	Effective	Neutral	Ineffective	Very Ineffective	No idea
Scrum						
Lean / Kanban						
(Rational) Unified Process						
Extreme Programming						

6. How essential are following practices on a project level in large scale agile development?

	Essential	Good to have	Average need	Little need	Not needed	No idea

Active stakeholder participation						
Burndown tracking						
Coding standards						
Collective code ownership						
Continuous Integration						
Daily Stand-Up Meeting						
Demonstrations after iteration						
Documentation treated as a requirement						
Pair programming						
Producing potentially shippable software each iteration						
Retrospectives						
Self organizing teams						
Test Driven Development						

Page 2. Distributing, communication

Distributed agile teams may work in the different rooms, neighbouring countries (nearshoring), or across the globe (offshoring)

7. Rate following challenges in succeeding with distributed agile development.

	Blocker	Major challenge	Average challenge	Minor challenge	No factor	No idea
Communication between working locations						
Culture in different working locations						
Geographical distance						
Infrastructure(bandwidth etc.) in different working locations						
Language barrier						
Organization distribution(members from						

different companies)						
Political situation in different working locations						
Processes, standards (CMMI, ISO)						
Time zones						

8. The following strategies for distributed agile development are:

	Very effective	Effective	Neutral	Ineffective	Very Ineffective	No idea
Isolated teams						
Isolated teams with few members as co-located ambassadors to share knowledge						
Isolated teams having shared stand up meetings with other teams (Distributed Scrum of Scrums)						
Teams having both co-located and near-shore distributed members (Fully distributed Scrum)						
Teams having both co-located and off-shore distributed members (Fully Distributed Scrum)						

9. The following strategies for conveying information WITHIN THE TEAM are:

	Very effective	Effective	Neutral	Ineffective	Very Ineffective	No idea
Detailed						

documentation						
Email						
F2F at a Whiteboard						
Face-to-face (F2F) communication						
Online Chat						
Overview diagrams						
Overview documentation						
Teleconference Calls						
Telepresence / Very high quality videoconferencing						
Videoconferencing						

10. The following strategies for conveying information WITH STAKEHOLDERS are:

	Very effective	Effective	Neutral	Ineffective	Very Ineffective	No idea
Detailed documentation						
Email						
F2F at a Whiteboard						
Face-to-face (F2F) communication						
Online Chat						
Overview diagrams						
Overview documentation						
Teleconference Calls						
Telepresence / Very high quality videoconferencing						
Videoconferencing						