EÖTVÖS LORÁND UNIVERSITY

FACULTY OF INFORMATICS

DEPARTMENT OF COMPUTER ALGEBRA

Scalablity on IT Projects

Advisor:

Dr. Attila Kovács

Associate Professor

ELTE IK, Department of Computer Algebra

Author:

Norbert Bartha

MSc student in Software Engineering,

Software Technology specialization

Budapest, 2016

Acknowledgements

I would like to thank to my principal supervisor, Attila Kovács, for his help, patience, understanding and constant guidance.

I am also grateful to Kristóf Szabados, co-supervisor and my colleague, for his time, help and support and for providing me valuable and reliable methods and tools for measurements.

Contents

1 Introduction

Nowadays software is everywhere. IT projects become an important competitive element in many industries. Most of today's goods and services are realized with the help of software systems we are depending on more and more. In areas that were traditionally reserved for hardware, software has become the major driving force of overall progress. The size and complexity of software systems in various domains has increased rapidly and these growth trends seem to be continuous. In addition, in domains, where software is playing the key role, there is an increasing pressure upon software to progress. Hence, software and software development are getting more and more complex entailing a fundamental shift in their cost, size, functionality, time-to-market and quality requirements.

The growing can be seen in every aspects of the project. It can be seen in the number of participants, stakeholders, code size of design and test, the number of integrated functionality, involved sites or tools are used and so on. Projects' complexity, interdependencies, and communication challenges grow dramatically as projects grow in size. In addition the wish list is typically quite long but in general the software must come to the market before the competitors, with less cost and show good quality at the same time. These requirements cause justified worries for business leaders when taking decision to embark on a large-scale IT project.

As projects are getting larger, they touch more parts of the organization and pose a risk to the company if something goes wrong. Unfortunately, things often do go wrong. The number of large projects failed is higher than in any other discipline, which has already been proven by the global surveys and researches [1]. These show that around 50 percent of all large IT projects massively blow their budgets. On the average, these large IT projects run 45 percent over budget and 7 percent over time, while delivering 56 percent less value than predicted. It proves that large IT projects are vastly more difficult to deliver successfully than smaller ones. Which works well for the execution of small projects, rarely suffices for the execution of larger ones. Although every major IT project and its circumstances are unique, projects that are flagging often have common characteristics that we can analyze, draw conclusions and prepare suggestion for a solution. The reasons

behind are usually similar so we have a chance to build a process for detection and even try to give a common recommendation for the solution.

1.1 Base project

I have been working in a big project for 15 years in telecom area, where even the tests of this system grow to 3 million lines of code. I will show some examples from the function test area about how to measure a big system and how to use the results in large scale. I have played different roles in the test area of this project: test engineer, Scrum master, team leader, system test leader, and project coordinator.

For business reasons, I cannot share much detail on the product we are testing, but as it usually is, the growth of the function test system is in line with the newly introduced features into the system under test. The following diagram shows the lines of test code in our function test environment:

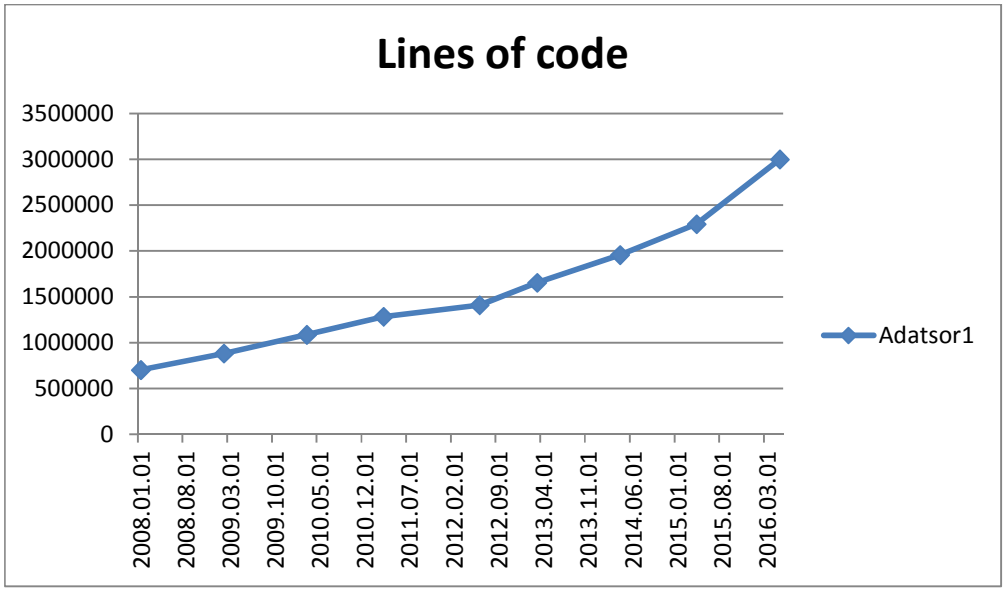

Figure 1-1: Lines of code in the function test system

In the above figure the growth of the system from 2008 until 2016 can be seen on yearly basis. The measurement is discrete for each year. They are connected with lines together to get a better view on the trends. The lines of code increase from more than half a million (in 2008) till 3 million code lines that we have there today in 2016. From 2012 it can be seen

that the increase became faster thanks to the new contracts and new customers who ordered the product. Certainly they have put requests and requirements for new features and updates that should have been tested. It resulted a faster grow in the number of function test cases covering new areas. Also, it is expected a growth in human resources including function test area too.

The test system includes about 1500 modules currently and their relationship is very complex in terms of which one includes the other. Looking at the imports between the modules, the following diagram shows the module connection graph of the function test system in a layered layout:

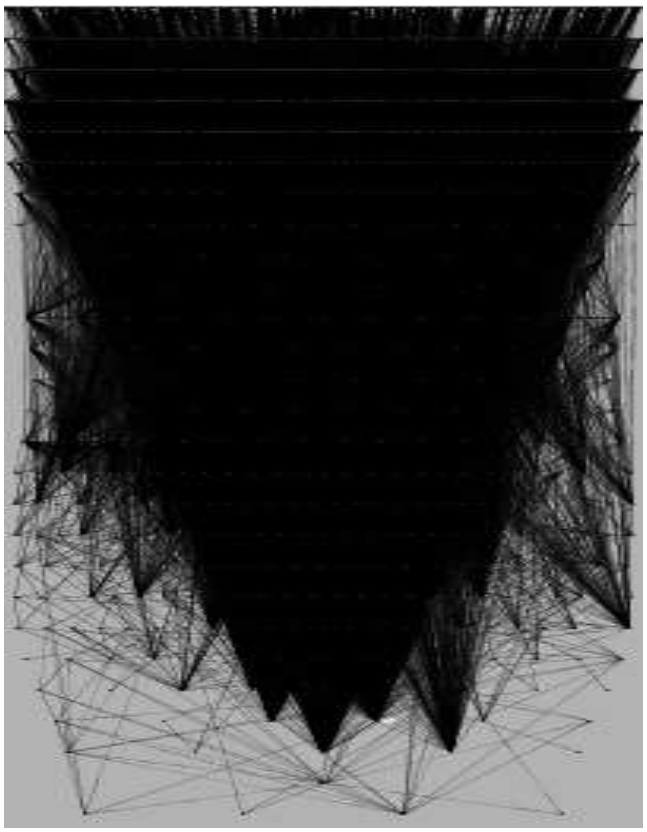

Figure 1-2: Module satellite of the function test system

The nodes of the graph represent the modules and the edges represent the import between two modules. The number of edges in the system is 25564 currently, showing the complexity. I will analyze further in the coming chapters in relation to their topic and in case there is any relevance to the current subject.

2 Problem description

Starting from the title, Scalability of IT projects is an extremely complex and heavy question. What can be done to save the projects from their size and maximize the chances that they deliver the expected value on time and within budget? How to identify if we reached a certain point when we need to change? How to scale out in different areas of the project and how to solve specific problems? What experiences and techniques are available? How to apply them? How to develop and test at large scale? How to improve ways of working?

We have to face a huge number of questions during a large project. We will see that for IT projects, almost all problems are scaling problems. Some of them have already been considered in discussions, presentation or articles, but most of them have not even been mentioned or realized. In addition, to put them together and try to summarize or even draw conclusions are still awaited. I will make an attempt to answer these questions from different project point of views and try to find significant points, theories that can be success factors on projects and recommendations for the solutions.

In the context of complexity we should emphasize that managing at scale is a learned skill rather than a natural ability. No one is born with the attributes of being able to know how to manage thousands of people. Everybody learns at some point for the certain situation in the actual company. Also do not try to predict the distant future. In environments of uncertainty, the best course of action is to make decisions that optimize/scale for the near and known future. Employ people, tactics, and processes that solve the problems in front of you immediately. Avoid the trap of solving for problems that aren’t staring at you yet. Work on surviving long enough to confront those problems later on [15].

One way to go through the several aspects of the project is to follow the widely used and popular standard 12207:2008 from ISO [2]. It establishes a common framework for software life cycle processes that is shown on the following figure:

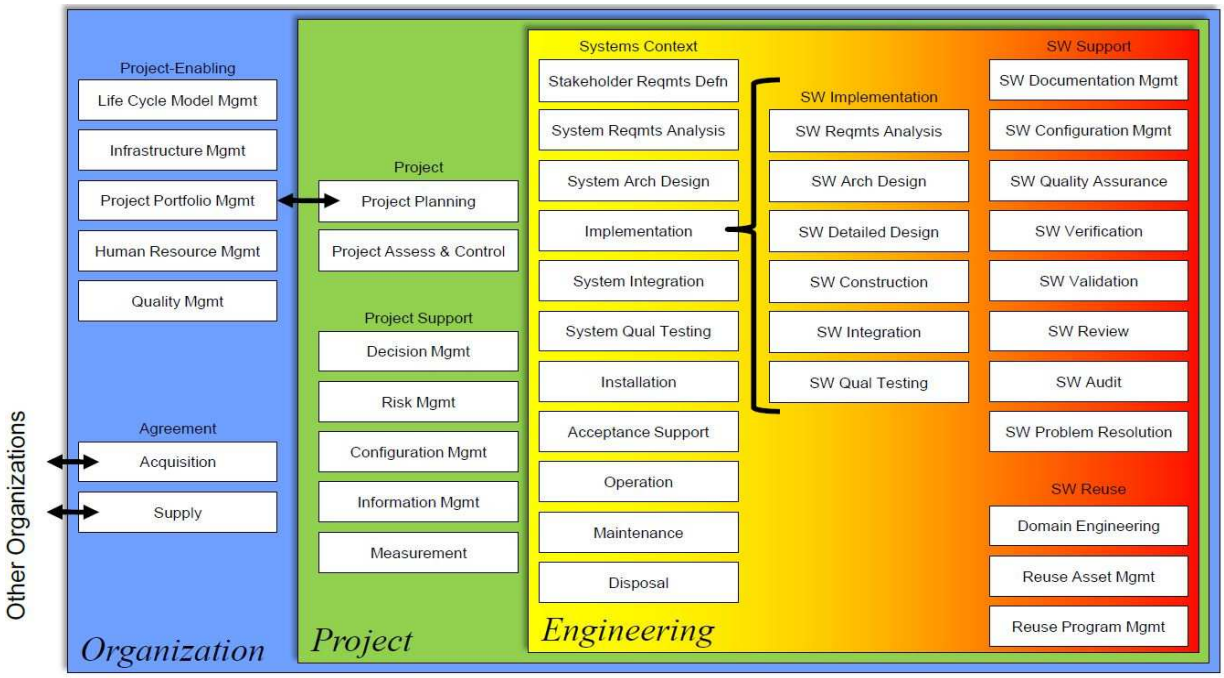

Figure 2-1: Software Life Cycle Processes of ISO/IEC 12207

This International Standard establishes a common framework for software life cycle processes, with well-defined terminology and structure to provide a comprehensive set of life cycle processes, activities and tasks for software that is part of a larger system, and for standalone software products and services. This International Standard does not detail the life cycle processes in terms of methods or procedures required to meet the requirements and outcomes of a process, showing what we should measure. We will see examples for processes in use, difference in between, pros and cons, and how to scale on them.

Naturally there are several ways and view point where this issue is approachable, the different project aspects and grouping won't change the issues to be analyzed and solved. However, they could help to show them in a more understandable way.

I will try to briefly touch all main project areas in order to give a more complete picture on scaling as *Organization*, *Project* and *Engineering*. I will go into details for subareas I

have developed experience in. These areas are mainly in the group of *Technical Processes* and are the following:

- Acquisition and supply
- Ways of working on team and on project level
- Software support and verification
- Software Implementation and design
- Project Support
- Measurement in large scale
- Architecture and detailed design
- Maintenance

Certainly to go through all of the project perspectives and check every procedure and possibilities in large scale would exceed a scope of a thesis and worth to have several books to have or give a complete picture. Therefore I will try to focus as many areas as I can and put focus on those ones where I have experience or I can give good examples and advice.

3 Literature review

The literature of this subject is still limited and it consists of mainly articles and treatises on different project aspects as we will see. However, popular each of these topics is and lots of discussions are opened on them, a global summary that describes them well together is still missing. It can be understandable if we take the continuous growth into account or the complexity and size of this topic.

3.1 Scalability

What is scalability? One definition could be: “Scalability is the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged in order to accommodate that growth.” [3]. To say it simpler the scalability is an ability to grow. I would extend the word *enlarge* and add *reduce* to the picture as well or even the word *balance*, too. As in many cases there could be a need to reduce or balance on the existing resources rather than to grow.

The dimensions of scalability also need to be defined. Some of them are already available, but needs some explanation in terms of an IT project [3]:

- *Administrative scalability*: The ability for an increasing number of organizations or users to easily share a single distributed system. In terms of project perspective administrative scalability includes team management, documentation handling, release management and last but not least the administration of different tasks during the project.
- *Functional scalability*: The ability to enhance the system by adding new functionality at minimal effort or even to decrease or balance on the requirements to be planned in to the next release.
- *Geographic scalability*: The ability to maintain performance, usefulness, or usability regardless of expansion from concentration in a local area to a more distributed geographic pattern. From project perspective expanding the geographic scope of projects is an opportunity to distribute the project tasks on more sites and/or involve different companies into the project.

- *Load scalability*: The ability for a distributed system to easily expand and contract its resource pool to accommodate heavier or lighter loads or number of inputs. Alternatively, the ease with which a system or component can be modified, added, or removed, to accommodate changing load.
- *Generation scalability*: refers to the ability of a system to scale up by using new generations of components. Thereby, heterogeneous scalability is the ability to use the components from different vendors.

We can introduce some new dimensions to the above to make it more complete when we are going to scale on IT projects:

- *Verification scalability*: Ability to extend or reduce the number of test levels, test strategy and mode.
- *Human scalability*: Scaling on the involved people, to grow or reduce their assignments.
- *Scalability of project structure*: Ability to add or remove project management layers, involve or outsource project areas and also to scale on number of teams and their sizes.
- *Scalability of finance*: Financial resources need to be mobilized to support the scaled up intervention. The size, timing, sourcing, and cash flow requirements of the project.
- *Scalability of environment*: Impact on natural resources and environment must be considered at scaling. Harmful effects are mitigated, beneficial are promoted.
- *Scalability of complexity*: In terms of for example technical or organizational complexity scaling could help to balance and give an optimal solution for a scaled up solution.
- *Scalability of skills*: To have the necessary amount of experts and to measure on the experience level of employees is essential either internally or externally in large scale.
- *Scalability of time and duration*: Scaling on the resources in relation of time and timeframe for resource commitment.
- *Scalability of assess*: Scaling on the known or required information of the actual status of the project from all project's aspects.

- *Scalability of importance and priority*: Scaling on the priority of the project itself, the backlog items, goals, objectives to give the right priority order in align with the organization's size, resource and knowledge.
- *Scalability of business cases*: Scaling between benefits verses cost, risk, and investment.
- *Scalability of risks*: Scaling on threats, opportunities and likelihood verses the projects' objectives when growing towards the next level.
- *Scalability of size*: Scaling on growth that can be handled in comparison to the existing delivery organization.
- *Scalability of Experience*: Scaling on available experience and capabilities required to handle actual and planned projects.
- *Scalability of communication*: Scaling on organization which is no more than communication channels and control directions.
- *Scalability on Maturity*: Scaling the organization and project maturity to be able to keep them optimal and healthy.

We will meet all of these scalability aspects in the coming chapters where I will try to give the details and describe the meaning of them. It is important to mention as well that these scalability points of the view, I have highlighted, are the most important ones in my view. Although the list is not short it is far from completeness. There could be several other aspects, would worth mentioning here as well, that I let the reader to think them further.

3.2 Large-scale IT projects

Managing large IT projects became focus of the interests from the early 60th. Many articles and studies conducted on how to run projects effective.

As an example *Delivering large-scale IT projects on time, on budget, and on value* [1] emphasizes the triangle of the project management, time, cost and value. There is a fourth one worth to mention here: the quality.

Architecting for large scale agile software development is touching the view of project planning and tries to modernize the program and review of architecture-centric risks [10].

Another view on the problem to see how the new methods like “agile” can be adopted and scaled in large projects and with what limitations described by Hina, Hannan, Mukhtar, Sameer, Fahim in *Systematic Literature Review of Agile Scalability for Large Scale Projects* [4]. It is also described further via an example for changing waterfall to agile in a large project at section 4.1.4.

Managing risks and uncertainty in large-scale also affects several projects in Universities [5]. It is also processed by Donals Lessard and Roger Miller from MIT Sloan [6] for large engineering projects. It is worth to see differences in type of risks if we are speaking about an IT or Engineering project [6].

We have studies on R&D project evaluation trying to give a model for valuing options on R&D projects, when future cash flows and expected costs are estimated by trapezoidal fuzzy numbers [7].

On Management framework and toolkit Michael Bloch, Sven Blumberg, and Jürgen Laartz had an article at MSI describing their approaches and methodologies and learning from its experiences [8].

Testing in large scale is also an area of key success. Kristóf Szabados and Attila Kovács discuss their empirical observations related to evolution of a large automated test system in [9].

Development and agile development are very popular subjects also processed by Larman Craig [11] as recommendations for managers as well as for students. Scaling in a large and multisite product development with large scale Scrum is also discussed by Craig Larman and Bas Vodde in [12].

Huge experience in scaling was written down by Robert I Sutton and Huggy Rao in Scaling up Excellence [14]. This book is a deep, thorough look at what it takes to scale an organization and to create an environment where pockets of excellence are emulated

throughout. No matter what kind of organization you're a part of, this book will have relevant advice.

3.3 Scalability and performance

Scalability and performance determine the projects' growth together. The growth is depending on the budget where the performance is a parameter as it can be seen in the following figure:

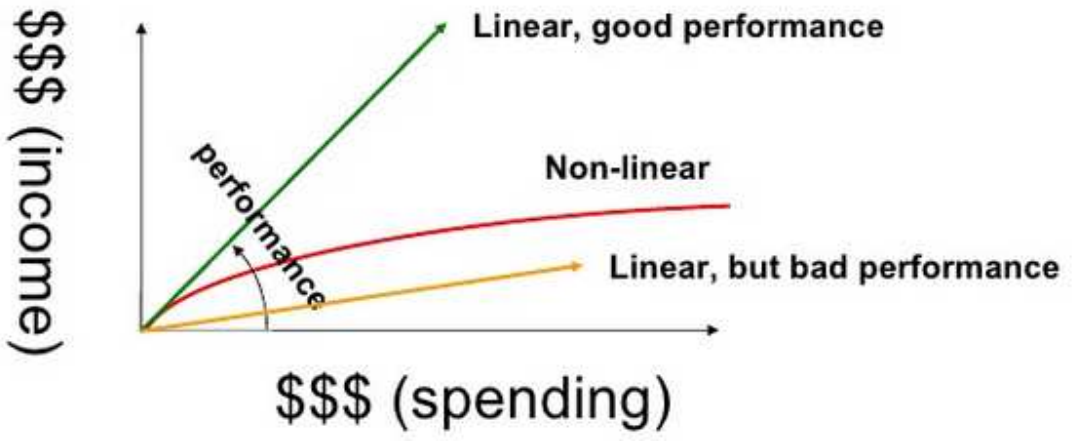

Figure 3-1: Scalability and performance

Regarding scaling we must also take into consideration that we have limits based on our financial results and status. The budget is always given, but that can be influenced by the performance the project shows. In case of good performance the spending is in line with the income and it gives opportunity for scaling up. On the other hand when in case of non-linear or bad performance, options are very limited.

3.4 Milestones of large IT project management

The first large projects were done a long time ago, like the Pyramids of Giza (2570 BC) or the construction of the Great Wall of China (208 BC). We do not know for sure how exactly they were able to develop these amazing buildings, but there were some planning and project management work on their execution for sure. Of course, the development of

IT projects and project management have started much later, in the beginning of 20th century.

Looking at the project management from IT perspective, the first name I would like to mention is Henry Gantt (1861-1919) usually referred as one of the forefathers of project management. He developed a scheduling diagram in 1917 (named after him the *Gantt chart)*, which is still in use today and an important part of the project management toolkit. An example Gantt chart can be seen in the following figure:

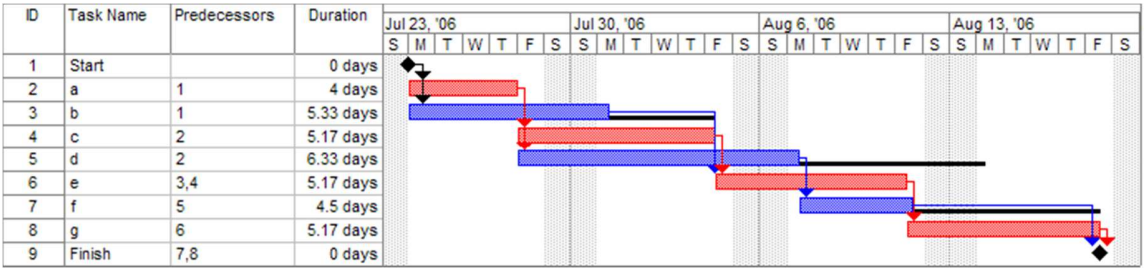

Figure 3-2: Example for a Gantt chart

The American Association of Cost Engineers (now AACE International) formed in 1956. It describes the early project management in terms of planning, scheduling, cost estimating and schedule control. In 2006 AACE released the Total Cost Management Framework which was the first integrated process for portfolio, program and project management.

The Critical Path Method (CPM) developed by the Dupont Corporation in 1957. CPM tries to predict project duration by analysis and show what order of the activities has less and more flexibility. In the very first year of the implementation $1 million is saved for the Dupont Corporation because of this process.

The Program Evaluation Review Technique (PERT) was invented in 1958 for the U.S. Navy's Polaris Project during the cold war. PERT details how to define and give cost estimation for all tasks of the project and how to use this information to identify the minimum time required to finish the project.

In 1962 United States Department of Defense (DOD) published the Work Breakdown Structure (WBS) procedure. It groups the defined deliverables into a hierarchical tree

structure showing the tasks that should be done to complete the project. WBS is still very common and useful project management tool. As an example of WBS please see the following diagram about an Aircraft System:

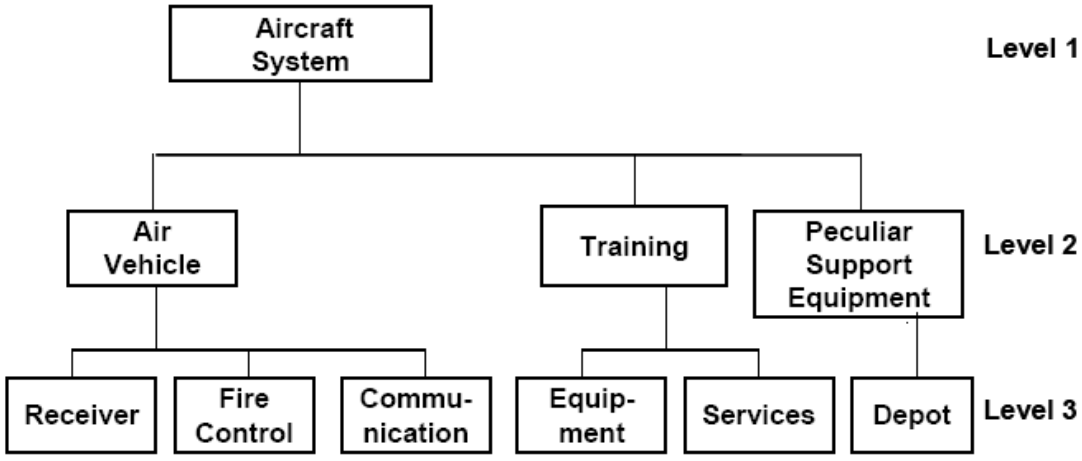

Figure 3-3: A 3 levels WBS diagram of an Aircraft project

The International Project Management Association (IPMA) was founded in 1965 in Vienna and was the first project management association of the world. As a network of project managers it helps to share information, to promote and lead the development of professions. IPMA has more than 100,000 members today.

The Project Management Institute (PMI) was founded by five volunteers in 1969 to favor the Project Management Profession. It is best known as a publisher of PMBOK: *A Guide to the Project Management Body of Knowledge* first published as a white paper in 1987. Since then, it is considered as one of the best tools in project management and has become a global standard for the industry. PMI also offers project management certification on two levels today.

For the crisis of IT projects in 1975 when computer projects were running over on time and budget even ten times the PROMPTII was developed by Simpact Systems Limited as an answer. It was a guideline for the work flow adopted for all IT projects by the UK Government's Central Computing and Telecommunications Agency (CCTA) in 1979.

Brook's law was defined in 1975 in his book *The Mythical Man-Month: Essays on Software Engineering*. Saying: "Adding manpower to a late software project makes it later".

Dr. Eliyahu M. Goldratt introduced Theory of Constraints (TOC) in his novel: *The Goal* in 1984. The Critical Chain Project Management (CCPM) invented by him in 1997 is based on this novel. It helps organizations to achieve their goals and describes a management philosophy to identify constraints that can be used to restructure the organization. It says that the load on the resources should be balanced and they have to be able to switch between tasks quickly.

In 1986 Takeuchi and Nonaka first mentioned Scrum as a project management style in their paper: *The New New Product Development Game* [26].

Earned Value Management (EVM) technique has been around since the beginning of the twentieth century but only became an essential part of program management and elevated to Undersecretary of Defense for Acquisition in 1989. See also in [27].

PRINCE Method was developed from PROMPTII by the UK Government agency CCTA in 1989 and it had been a standard until its revision in 1996. The idea was to assuring progress from tree perspectives. But it was found too rigid and applicable only to large projects.

1994 was the first year when CHAOS report was published on project failures in the IT industry by the Standish Group. The intention of this biennial publication is to improve the success rate of projects.

PRINCE2 was published by CCTA in 1996 and compared to the fist revision it is more generic and usable by any project type.

In 1997 the Critical Chain Project Management (CCPM) came into focus, developed by Eliyahu M. Goldratt as mentioned above together with his novel.

PMBOK became a standard at American National Standards Institute (ANSI) in 1998 and also at Institute of Electrical and Electronics Engineers (IEEE) a year later.

2001 is famous by having the Agile Manifesto written by 17 software developers and defining 12 core principles for software development. This new approach holds the same name today. Some of them established the Agile Alliance organization later.

The *"Total Cost Management Framework"* defined by AACE International in 2006 is known as the first integrated process of portfolio, program and project management.

The 4th edition of PMBOK was born in 2008 with 2 brand new processes and a simplified standard that is easier to understand and follow.

A major PRINCE2 revision was performed by Office of Government Commerce (OGC) in 2009 to make it simpler and customizable. The update provided seven new basic principles for project success and gave a better toolset for the planning and support to deliver the project on time, value and quality.

The ISO 21500:2012 is a standard and guidance for project management from 2012. It can be used by any type of organization and any type of project irrespective of complexity, size or duration.

The 5th edition of PMBOK was released in 2012 to give us the 10^{th} knowledge area about project stakeholder management and four new planning processes.

Nowadays, when the world is changing continuously and we are facing new challenges, there is a constant need to look after new techniques and improvements can be used to manage the project better and better. On the other hand sometimes we are spending more time to change the way of working or to introduce better tools than the time can be earn by them.

4 Scaling On IT Projects

In the following chapters I walk through and discuss the project parts in the structure given by 12207:2008 from ISO [2] and also shown in Figure 2-1. I will focus on growth and measuring what is happening during scaling out from a small garage project to a large multisite project.

The life cycle processes used in a project should be defined and can also be categorized and measured separately. The following reference model [2] shows the grouping of these processes by project perspective:

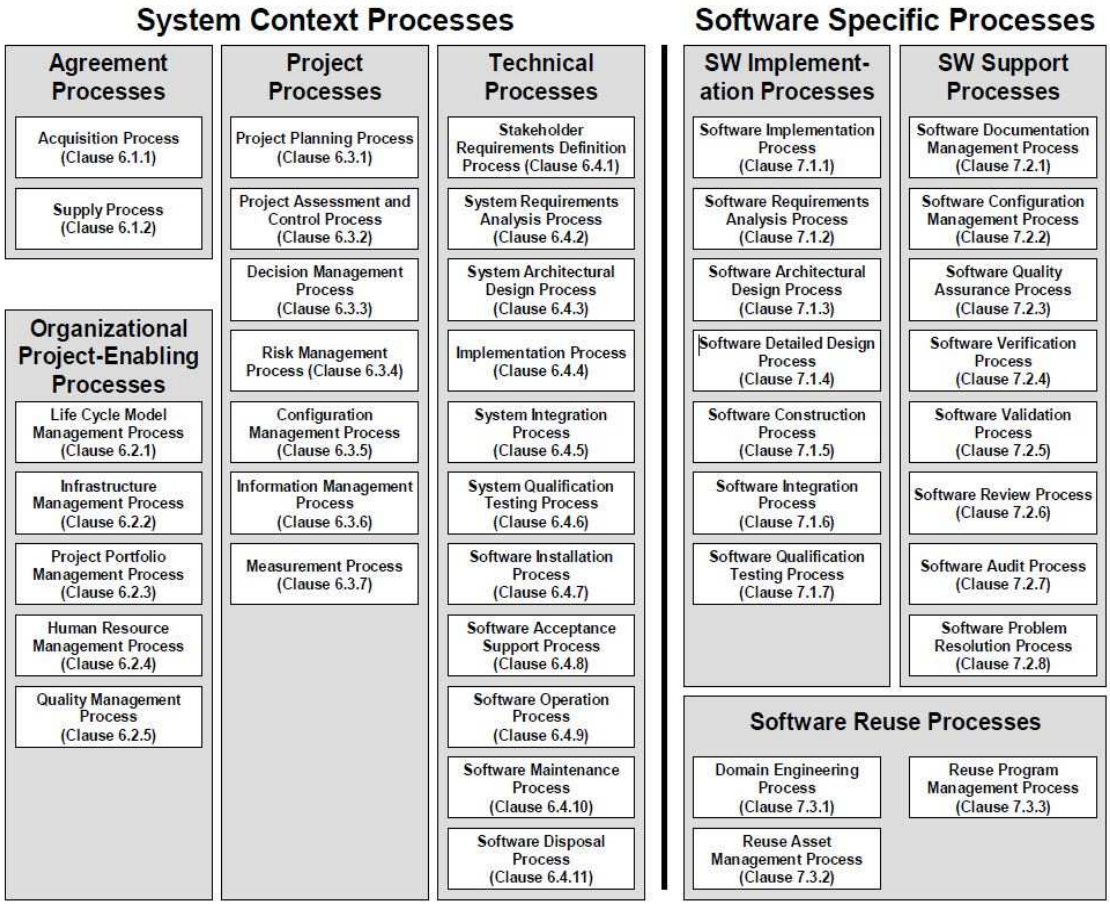

Figure 4-1: Life Cycle Process groups

As we can see several processes should be implemented by a project such as agreement processes, organizational project-enabling processes, through project processes, technical processes, implementation, support and last but not least software reuse processes

intended to be adopted by an organization based on its business needs and application domains. Process outcomes are used to demonstrate successful achievement of the purpose of a process. This helps process assessors to determine the capability of the organization's implemented process and to provide source material to plan organizational process improvement. Why do we need these processes are clarified in chapter 4.1.2.

4.1 Organization

Organization can be measured from different points of view. From size perspective we see that small projects consist of 1-50 participants, a medium sized project has 50-500 and a large project involves more than 500 people. From geographical point of the view a small project has a single location; a middle size project can be geographically spread to a few numbers of sites and countries; a big project is usually wide spread covering the multinational needs and targeting global markets.

Organizations with small projects (so called "garage" projects or start-ups) consists of 1-50 people, are seemingly free of any organizational problems which can be true for a short time in the beginning. As they grow, more and more problems will appear over time. Continue to grow from 50 to several hundred project members leads to a complexity that cannot be maintained without tracing the organization growth and performance from the very beginning. So it is highly recommended to start tracing the project from the early planning phase. Bill Meacham et al. said in one of his study about Estimating Software Projects that, the only way you are going to have a firm grasp on how much effort it takes to develop each component is to keep careful track of the effort you expend as you build a system and record that data for later use. The more often data is recorded, the more accurate estimates can be. This study is a good example of estimation based on the project plan and tracked performance information you can use [13].

In the middle, when size reaches ~50 engineers in a project there will be a separate need for taking care and managing people beside the usual project duty. This is the role of the so called line management. With growing size this management will become a separate organization structured either align/integrated, separated to a division or separated but

orthogonal to the design organization which is often called as operational part of the organization. The latter is the famous matrix structure the companies often use.

A large project's organization often has relation to many other organizations (for example in a multinational company). Most often there are several organizations working on different parts of a complex solution. The structure of the organization is very important and has to be aligned with many aspects like ways of working, life cycle model, organization size, and layers, geological spread and multiple contacts with other organizations or companies and there should be a well-defined border between these products defined by chief architects in the very beginning. The organizations are acquiring and supplying parts to each other. From the scaling perspective it is also very interesting in large-scale. According to Melvin E. Conway in [37], "Any organization that designs a system (defined more broadly here than just information systems) will inevitably produce a design whose structure is a copy of the organization's communication structure".

4.1.1 Communication

The *Organization* is nothing else than the communication architecture of a company. It defines the internal communication channels within and between parts of the company as well as the company's communication with the outside world.

Starting from a one man's project where the only employee develops all the code, test cases and maintains them. He or she takes care of the marketing and sales as well. The usual problematic things like communication, common knowledge and making decision won't cause big headaches. Here there is no need for communication at all as all decisions come from and are aligned with that only one person. With growth, things are getting worse and worse in all growing dimensions. Perhaps if the company does not expand it will never be a great and successful one of the many.

In small projects, consisting of 5-50 people, communication is relatively easy as people can always talk to each other. Usually they are involved into more than one part of the project and taking care of plan, design, test, maintenance, marketing and sales tasks at once. Everybody knows everything. The organizational structure is flat and usually there is

one head of the project and company as the driver. However the communication problems usually starts here at size of 15-20 project members where the need to split for smaller groups appears first. 15 or more people in a team are usually too much as without an organizational structure they will lose focus on distribution of tasks, even if that happens, they will communicate less with each other, loose targets, focus on less important things at the wrong time etc… and thus, they will work inefficiently and in chaos without a well-defined organization. It is also the time to start to think about the common knowledge as there is always need for sharing excellence. Engineers who play several roles like writing the code, building the system, testing the product, installing and maintaining that. It works well in the beginning, because they are aware of everything and communication is not a problem. There are no needs for complicated handovers either. But after a certain size, it becomes very difficult to add and train a new engineer to the project or team and it will become more difficult than performing the actual task itself with a certain size.

In a middle sized project consisting of 50-500 people the need to split for smaller groups is a key for success. Generally there are two ways to define these groups: to use specialization or generalization.

Specialization is to give these groups responsibility for a certain area and tasks belonging to that area. Implementing the specialization is called *the first scale technique*. The specialization when dedicating teams and people to a certain task, like developing subsystem or test cases, test framework and component has advantages and drawbacks. On one hand they will become specialist in their area thus the development of that subsystem will be faster and quality can be better. On the other hand they are moving away from the common knowledge and as a specialized team they cannot be replaced easily any more.

Generalization is the other way when every team should be aware of all parts of the product so they will be kept as close to the common knowledge as possible. They can also replace each other at any time. It is obviously harder to achieve after a certain size than specialization and also has drawbacks. For example after a certain size of a product it is impossible to see through all parts of it and it will take lot of effort form every project

member to fulfill or even makes it impossible to get that common knowledge. From this point it is better to think about specialization or some hybrid solution in between.

A large organization involves more than 400-500 people in different roles. In this case goals of scaling are to choose the least of all evils. The definition of communication paths is extremely important for big companies. These are given by organizational charts usually on different levels. Experience shows that people far down in these charts communicate less and less. Engineers, testers, leaders are split into different groups in the organization on different levels, as they are divided to subgroups and teams and it makes the organization graph complex. Rolling out organizational changes becomes as difficult as defining how often and when one should re-organize. In such a big structure organizational changes are continuous as it is necessary to handle and optimize the communication issues experienced all the time. First of all there are some important questions to be clarified. What needs to be communicated and by whom to who? What knowledge is the most important and who should have it in the organization? What are the most frequent decisions to be made in different project part such as architecture, organization, feature, support, maintenance, hotlists, test, people management? After answers are given, the best is to focus on the most important communication and decision paths and create a group for them. This will solve the actual situation which is continuously changing in the future. If it is done, the re-organization can take place. After the most important communication and decision channels are defined there is a need to pinpoint a suitable manager to run these groups. Naturally there is a need to take care of the down prioritized communication and decision channels as they cannot be ignored entirely. In this way the first issues with the new organizational structure can be identified and preparation can be started for the impending cross organizational challenges to reduce the bureaucratic overhead. As there is continuous change it is crucial to reduce the cost of the change as much as possible.

Organization structure can be varied depending on the complexity of tasks, hosting countries and the company’s culture. One can say that organizational designs are always bad as with any design the communication is optimized over some parts at the expense of other parts. As an example when a portfolio manager is appointed on top of system engineers, he will optimize the communication between portfolio and system guys at the

expense of communication between portfolio management and marketing. When this organization is published the people will recognize the fault sooner or later and they will be right.

4.1.2 Processes

Processes define the way of communication and make the expectation transparent for the whole organization in terms of communication. Processes help to ensure that the communication happens and it happens with quality that makes the collaboration success. Processes can be heavily engineered or can also be a short description about how to sort out a task. The size of a process can vary, but should be aligned with the needs of the communication challenge it describes. Usually the number and the size of processes in use align with the size of the project. The larger the project, the more processes are needed. Also these processes should be scaled up or down to meet the expectation changed by the increased or decreased project's size.

Processes are required even for a small project or company. The first process that needs to be defined is the interview process as it is very important to the success of the company. Processes should be designed by those who are doing the work itself. They are able to describe the parties involved and visualize the communication channels and directions in the best way. By formalizing the process they should make sure to make it scalable. While the project is growing in size, more and more communication channel has to be standardized and moved from point to point communication towards communication channels between different sizes of human groups.

For a middle sized project, process descriptions became mandatory. Focusing on the processes first is always beneficial. It is much easier to add a new colleague to an existing process than replacing an old process to a new one. In the first case the new member should learn the process which is in use with help of the others, in the latter case the mindset of the people should be changed which is always a much longer journey. Training processes are critical to the scaling effort to ensure that the newcomers become productive as fast as possible. This will focus the effort on employees and makes them more effective

and happier in the project. It also has the best chance to preserve the culture throughout the scale in size.

For a large scale project maintaining processes is a usual task. Processes should be measured in each step. The performance of individuals doing these steps should be available and measurable. Depending on the results processes might be subject of reconsideration and actions to be taken to make the process better and more effective. As it is shown in Figure 4-1 there are a branch of process groups to be defined and maintained in a large scale project will be checked in the coming chapters.

4.1.3 Human Resource Management

Hiring new people is necessary to growth. Also it is equally important to keep the existing human resources otherwise the effort to have them productive will be lost. Adding new members to the project is the most time consuming part in a company life. This learn-in time can be reduced with an excellent onboarding package to provide the “fastest” way of introducing a newcomer and have him or her become productive. The way how we are doing it does matter a lot. The human resource department (HR) can be strategic when it takes into account what the mission of the organization is and how that will make it happen. If HR can combine organizational goals with individual and societal goals before making the decision then there is a chance for a right decision.

In a small project or startup most often there is no separated human resource management in the beginning. The head of the company with the help of any project members runs the interview, which is usually a one round undocumented process. With the growth the interview process will be a part of human resource management. The growth usually reaches a limit very early, when the only responsible person is not able to manage the project needs, is unable to find and hire the required number of people. It can be easily detected by the interviewer in advance when interviews start to become a burden beside other tasks. It could also happen that there is a need for a large amount of people in a short period of time intended to be solved by hiring new peoples. At this point a human resource section should be established.

From middle size projects (over hundred employees), a good hiring process is indispensable. The basic recommendation is to have this process as simple as possible and well defined for all parties regarding their roles in the execution. The process defines the requirements for a good candidate for different roles in align with the company's strategy and usually it has 2-3 turns for each candidate to have all the answers for the final decision. Actually human resource management is scaling on people. All successes of the company depending on the HR process which makes it critical to be defined well. Problems with this process could even lead to a bankrupt of the company.

4.1.4 Life Cycle Model Management

In terms of life cycle model management companies should define and use processes to embrace the product life from the beginning till the end in every dimension. Life cycle model should describe the value flow from a business idea to functioning software with all of the stakeholders involved. Without completeness it has to define iterative solutions to coordinate people, processes for all product development state, tool set that holds the information base and coordinate the information processing as well. It describes the iterative cycle of software development activities including planning, change management, requirements management, architecture management, software configuration management, and build-, deployment- and quality management. An appropriate communication of facts based on one core life cycle information system, communication and actions shall be based on this central data.

Definition of the life cycle management is very important to have from the beginning even in a small garage project. Regardless of the size the information should be kept and stored in a central place in every project. Lacking a strict definition of the process could cost the business significantly, lower the productivity and increase the risks if key conversations or decisions are not in the information base and available at any time. Regardless of the project size these processes should be documented, maintained and improved continuously. There are several even free tools on the market to help these processes and able to store information on different purposes during the life cycle.

We can measure the different development life cycle methods from scalability point of view. Generally they can be divided into two main groups to make the comparison easier. The first group has distinct sequential flow model with clear and strict cut-off between phases. These are Waterfall, V-Model, Cleanroom etc… The model of the other group is repetitive with flexible cut-off rules between phases. These are Spiral, RUP, Agile (XP, Lean, Scrum, Kanban) etc…

Let's take an example from both group and check how scalable they are.

The Waterfall as the first process model was founded during the 70th by W. W. Royce in [28] and it is still popular in project development systems. The focus is on the importance of the requirements, design and quality assurance. The model is shown in the following iterative interaction diagram:

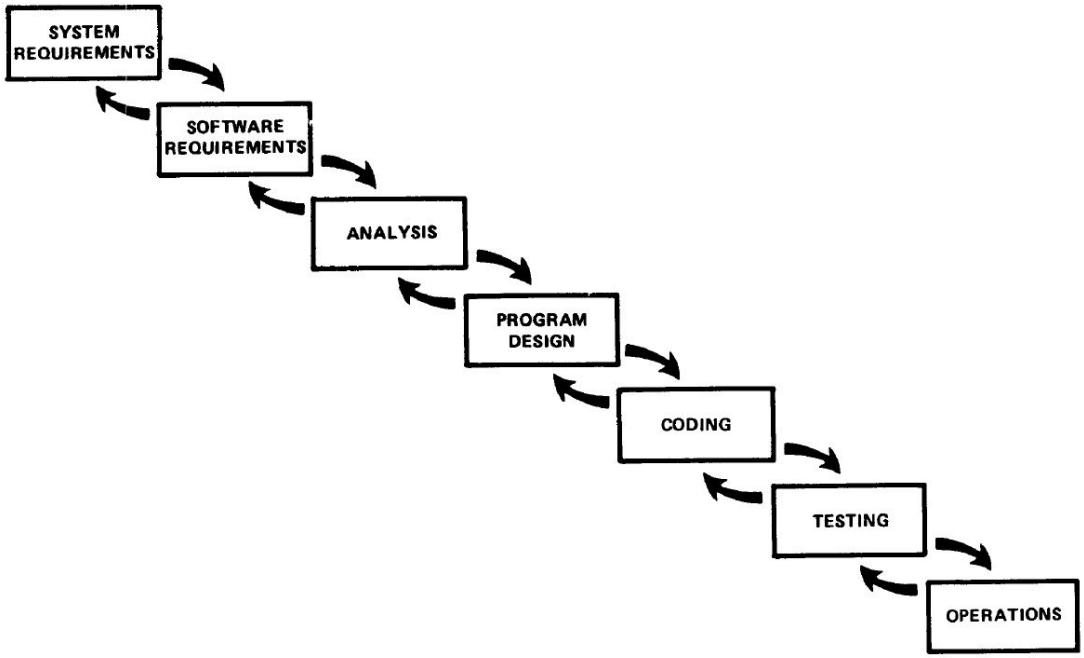

Figure 4-2 - Waterfall Model, Interaction diagram

The model implies that a given stage has to be completed before moving on to the next one, the phases do not overlap. Between the phases there must be a specific verification and validation resulted in a strict indication if the next phase can be started or should step back one and repeat the phase in case of failure. Requirement changes are

better to be finalized before the coding move off. According to the process it is highly recommended to involve the customer into planning to avoid late changes in requirements. The product cannot be used until the entire system is complete. Increments are not built into the process, only a step forward or back can be achieved. To reduce the risk and to avoid late feedback that could cause a complete failure, the author recommends performing a preliminary program design before the analysis begins and document it. It is a kind of mini project that goes through all of the phases just ahead of the project. As a result a prototype can be shown and discussed with the customer to get early feedback. Also it can be used as a base in next phases, to make them faster and better as they can rely on the prototype. We can interpret this theory as “do it twice”. For these facts it can be seen that this process is not so flexible and does not give space for too much scaling. Scaling within the stages can be done perhaps to add or remove resources at a certain point or globally increase the number of participants, but that will also be hard as in a good waterfall-plan the resources should be available in the very beginning of each state for the estimated needs. Also there is a huge gap between the requirements definition and their delivery which makes it rigid and slow to change. Nevertheless Waterfall type models could work for smaller projects where the requirements are very well understood, or initially given. When the project is no time sensitive and end-users can wait for the delivery, this model can also work as it is simple, easy to manage and understand. The main thing missing here is the parallelism. One hand, it could be used to avoid idle time of resources including the work capacity of employees. On the other hand it could help out to improve the time of adapting to changes.

Iterative models, like Agile, hold the flexibility and introduce the concurrent engineering model we also need for scalability. However here we have to make sure about the lifecycle traceability intermediate integration and verification separately. At a scale of middle sized project static processes, like waterfall or V-model, will most probably make the company less effective compared to their competitors using flexible development models. Nowadays the requirements and customer needs are changing much faster than even some years ago, so there is a strong need for being flexible during the whole project in each and every level. The Agile lifecycle is shown in the following figure:

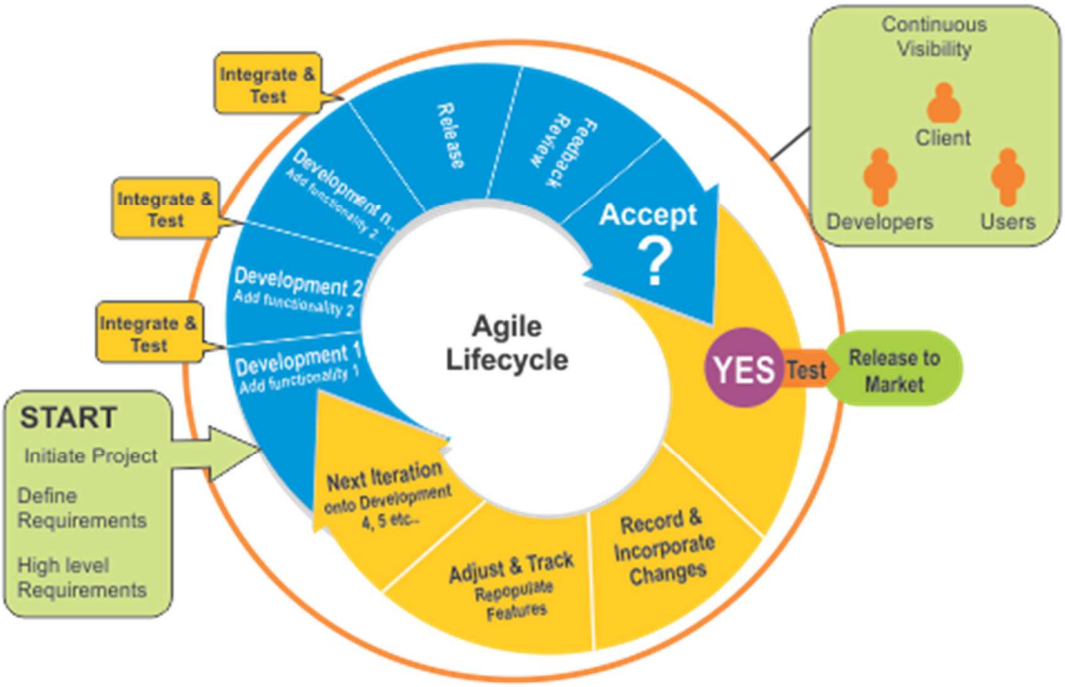

Figure 4-3: The Agile Lifecycle

Agile and iterative methodologies in general brake down the project into small tasks with priority and also cut delivery time from months to weeks. There are parallel groups called “teams”, running this iteration over and over again usually until the project reaches the end of maintenance. One difference between agile and waterfall is the approach of testing. There is always a separate testing phase after a build phase in waterfall. In agile development the testing is usually done in parallel with coding in the same iteration. Iteration develops a small piece of the product, which can be used for testing the legacy and extended to cover the new part. Depending on the results the users will see the value of the updated piece of software and they can make better decisions. Retrospective and continuous software (re-)planning sessions helps the team, project and customer either to update the plans to maximize the value the project delivers. With the iterative practice a new product mindset is building. Software can be seen as a living organism in continuous development. Agile software development also has a strong connection to the Lean startup concept in terms of the short iterations they use.

In agile, here is plenty of space for scaling, as the requirements and belonging tasks are continuously changing and it creates the need for adopting the number of teams or their size or involving new processes, innovation and improvements. Also most companies understand and scale for the actual product stage as every project has limited lifespan:

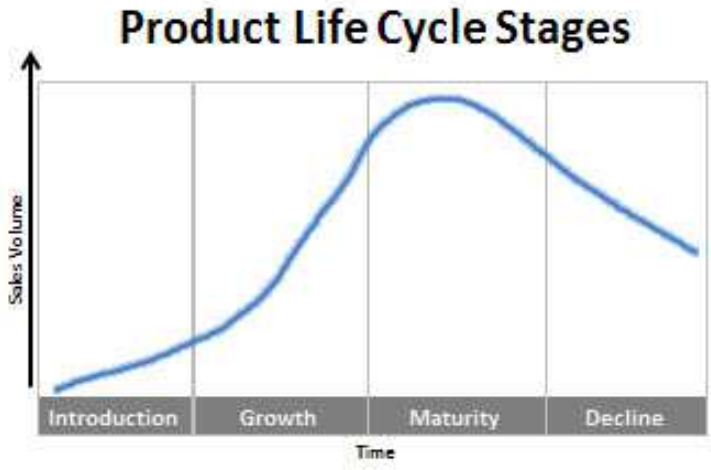

Figure 4-4: Product Life Cycle Stages

When a product reaches the maturity stage, it is time to restructure the investments towards new product development to make sure that the company and business continue to grow. The reorganization will affect all resources including the actual workforce and environment of the certain project.

As for everything, agile style iterative developments also have disadvantages. One of them is for example in relation with the previously defined *Geographic scalability* in section 3.1. If a project reaches a size, it is spread to different sites and countries. Applying the agile approach will introduce new challenges here. As noted in [29] by Martin Fowler, a well-known agile evangelist *"Because agile development works best with close communication and an open culture, agilists working offshore feel the pain much more than those using plan-driven approaches."*. So work assignment needs to be well defined and co-ordination has to be more frequent, detailed and strict.

Changing life cycle management in a running project should be done carefully. It happened about seven years ago in a big project that had been running for 5 years by that time and in which I also took part as a team member. The project had 450 members at that time and divided to ~35 feature teams of ~8-10 members in each. We had used a kind of waterfall model during development and teams worked as mini projects. It had been decided to turn our development process towards scrum as the project performance did not meet the customer needs and management saw the solution in Agile/Scrum methodology. It is a usual and frequent situation in a medium sized and growing project, when the project scale reaches a size requiring organizational change and reconsideration of the processes. Indicators were the huge customer interests, orders for new features, delays in current agreements and quality problems. To handle these changes at once and restructuring a running project in a stretched time schedule foreshadows the mistakes we have made.

Typically the first problem is that most initiatives fail to consider how changes affect the members of the organization. A project with hundreds of members running for five years in a completely different development model develops a strong mindset for the actual ways of working. As the mindset should also be changed it must also be considered and cannot be avoided. How a proposed change will be received by the people will have the biggest impact. Scaling on this we can say that the bigger the project, the longer it takes for the mindset to change. It is better to start the communication at identification of the need for a change to make sure it is well understood in time and embraced by all project levels and people. Also, the scaling cannot be launched via a pure belief or strategy. If people just talk and talk about the change of mindset without proposed actions and commitments then the result will be weak or none. There is a need for constructive actions for successful scaling. As the ancient proverb says: “What I hear I forget, what I see I remember, and what I do I understand.”.

In our teams there were team members who did not understand the change and were not aware of the new way we were going to work in even after months. Good communication, trainings have to be started as early as possible and from both directions, from top to bottom and from bottom to up. This is essential to speed up as much as possible to minimize the time impact on the project. It is also suggested to involve all

people most affected by the change into this implementation. In this case people will have greater ownership of the change thanks to their insights. These people from every project level can help to communicate and spread the information that all project members do need to be aware of, learn and then apply. In our project it took more than 2 years to make it happen.

Second but not the last common problem is that most of the time projects keep several part of the previous model while they are already running the new process. Parts of the old process could remain there forever which decreases the efficiency. To give an example when we have turned towards agile the planning and release processes from waterfall on project level remained. Although teams started to operate according to scrum, the management expected the same reports, documentation and delivery frequency like before. It seemed that we have been running 2 processes in parallel doubling the administration time of teams and management as well.

Third issue was that new tools were involved to support new processes only beside the existing ones. The old ones remained in use. This caused great delays in the project and it took several years to remove some unnecessary parts of the old life cycle process. It was a source of large amount of work and fight between the stakeholders.

Fourth problem was that the changes are introduced mainly on development level. One cannot operate in an agile way without the management behavior adapting to that. Also the actual customers have to change to the new way and actively contribute into an agile process. For example giving the requirements and waiting for the release like in waterfall is not an option any more.

Beside these, there are many good articles that can help to summarize what to consider and note in case of a life cycle intended to be changed. See also in [17].

4.1.5 Infrastructure management

The purpose of infrastructure management is to provide the enabling infrastructure and services to projects to support organization and project objectives throughout the life cycle. This process defines, provides and maintains the facilities, tools, and communications and information technology assets needed for the organization's business [2].

In former times, the infrastructure management meant obtaining and installing the hardware, software and the related network if there were any network at all. Today, because of the rapid pace of growth and more and more challenging competition the infrastructure is redefined and involves the policy and strategy, the plan and the design of environment architecture, the process of procurement, installation and maintenance of tools, facilities, infrastructure expertise, security arrangements, integration management, vendor management beside the usual triumvirate the hardware, software, network.

All IT activities rely on the infrastructure which requires to be managed effectively. Without taking care of, it will impact the organization negatively. Infrastructure might need to be managed as well towards external contacts e.g. customers, business partners, suppliers or during acquisitions to access their networks or even create a common network to support the cooperation on any level.

Although usually the infrastructural needs of a small project or startup are quite low, it is better to create and ensure to have an agreed infrastructure management process from the beginning. As most good processes about the environment, it should be based on the organizational strategic plan. Organization- and project infrastructure needs to be detected by system architects together with portfolio management. If these requirements are well defined and are given, then one can establish, maintain and record the infrastructure. The plan should be reviewed time to time and updated to have it align with the growth, with new requirements and with the continuous change. It is also good to record the indicators of the need for environment change for the future use.

In medium size when an organization reaches some hundreds of developers and the product development reaches a size, IT infrastructure will have been rapidly increasing. This should be monitored and acted upon from time to time. Indicators are visible and

easily detectable in this case and also experiences from the past could help. Most often capacity is not enough in the labs, people start claiming on environment problems and outage, testers are in delay, designers are running out of time and refer to environmental issues in their reports etc… At this stage there is a need for a reliable and cost effective means for maintenance of applications used in the environment, reconsidering and optimizing the infrastructure management processes. It requires significant investment and the creation of a support organization to maintain the infrastructure if it does not exist yet. Pre-planning of the environment updates in time could save lot of headaches and money as scaling out and up takes time and better to think about that in advance. It should be done together with risk and impact analysis to be published to the whole organization in time, to avoid bigger outage of the project, than it requires and surprises it may cause without announcement.

In big projects at large scale most often the IT infrastructure needs to be available in 24x7 with extremely high uptime requirement. To fulfill these expectations it requires complex and reliable solutions in hardware, software and network. To use the latest trends when big projects are moving the infrastructure and infrastructure management to the cloud, that is also an opportunity in scaling of infrastructure. Often companies try to get a slice and provide an own cloud solution for sale, combined with several scalable applications, features and their support. Infrastructure-as-a-Service (IaaS) is one of the three areas in Cloud Computing we differentiate and delivers platform virtualization environment as a service. There are several essays and descriptions about the benefits and pitfalls of moving the infrastructure to the cloud on the internet such as in this interesting article [18].

4.1.6 Acquisition

Scaling on acquisition should be taken into account. Looking at the possibilities for outsourcing different parts of the project to one or more external suppliers have always been an opportunity at scaling. Acquisition can be a cheap and fast way to grow for projects of any size. It is also related to the *Human Scalability* by definition described in 3.1.

Let’s have an example that happened in one of our projects. This project was a medium size project that time and was in a growing period. The product had just gone live and started to earn more and more customers. We experienced a need for significant growth from 250 engineers to 300 as soon as possible. When project is scaling out and should grow to this size within a limited time several important aspects must be considered to avoid future problems. One of them is: What if the market won’t support this performance anymore? Another is: What if there won’t be that many new requests to be managed and requiring this amount of people?

Market peaks can cause big headaches if they are not taken into account. The market is never constant but always waving. Large peaks can be experienced to the top and to the bottom within a short period of time.

How to handle these market peaks? One solution could be acquisition. Mainly there are two problems. In a simple solution one can hire and allocate the required number of people to the project. It is a challenging task itself often impossible within the given time. Even if we succeed and people are available, there is a task to train them for the different roles. The second problem is that the knowledge transfer takes more time. This is also described in 4.1.3 as the most time consuming part of a company life. So what else can we do?

In the above situation we involved an external company offering experienced peoples for the request. There are several companies in the market offering good knowledge base and complex resource kit to be a partner of an IT project. It is an answer for the peaks as these external contracts, signed for a fixed time, can be prolonged in case of need. But if there is no need for extra resources any more, these contracts can be terminated without problems. The responsibility is on the partner to deliver the requested services in time and quality should also be a part of the agreement. In this way we can solve human resource problems instantly and also save on the time, we would have spent for training. Perhaps there should be a learn-in phase as well, but not that significant if newcomers would be hired from the street and have to be trained.

Looking at this from the supplier point of the view, it is also an opportunity for them to use the resources in the contract. From scaling point of the view, for these smaller

companies the project looks like a small one with 30-40 people. It should be considered to group them into smaller teams of 6-8 engineers in different roles in each. However, they should be able to be adopted into the big project, do a knowledge transfer and working efficiently.

We have also involved another supplier from another country and a responsible site was appointed to take the role to be the governing site. With these external resources we have successfully managed the market peaks.

Acquisition can also be used when we cannot have the knowledge for a certain task. Outsourcing these areas to other organizations or external companies is our best chance. These contracts are based on agreement and defined by experts. They are negotiating requirements, the time schedule and plan. This type of growth is often called as “synergies through acquisition”.

In a small project, acquisition is a huge opportunity to scale. As usual, it is in one hand and there is no need for outsourcing a project’s part or itself in the beginning. However, it is an option having pros and cons. A search for a repeatable sales process or customer acquisition model is starting to be developed and experienced. It is hard to know when this process is truly repeatable and the market is open for the product, but it is a key to success to find as soon as possible. Usual questions are: What is the cost of acquiring a new customer? How much time does it require? There is a good summary in an article of UN-Convention at [24]. The technology adoption lifecycle is shown on the figure below:

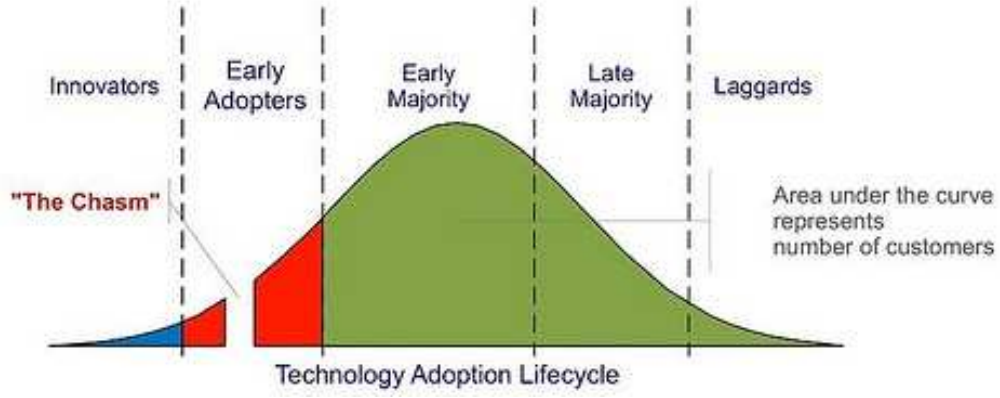

Figure 4-5: Technology Adoption Lifecycle

In medium and big projects, acquisition gives more opportunity, like the purchase of a company or a direct competitor. It can be done for different reasons, like reducing the number of competition, making a better portfolio, building up a stronger product (by joining the companies' experiences and knowledge base), making better advantage with common distribution channels, braking into a new market in a different geographic area etc… As a matter of fact, a big company size cannot be reached without purchasing companies and only a few of the biggest 500 did it without acquiring, see also in [25].

4.1.7 Supply

Supply chain plays a prominent role in a company's cost structure and profitability. It is becoming more and more difficult to rely on traditional supply chain execution systems. Some claim that the day for real-time supply chain practices has come, and is on the verge of being more mainstream thanks to a multitude of cloud data-management tools. Data analytics is the science of examining raw data and to draw conclusions about the examined information to support the companies' supply and making it more and more accurate. A real time supply chain can be used to make faster and better business decisions and also in the sciences to prove/disprove existing models or theories. Please find a summary about in this topic [16].

In small projects and startups there is a need to understand the product side and the market they are targeting. As for every entrepreneur the biggest challenge in scaling is to control the cost and to scale operations accordingly. In this process one can rely on external supply chain professionals. They could help a lot with their experience that was earned from a big company. Their knowledge can make real difference on this level by preventing the small project from the various mistakes. For instance when a new product to be developed and introduced newly into the market. As the supply chain is crucial they can ensure and start to define the supply processes as soon as possible.

Medium companies are usually after their first experiences with market and sales and they have survived. The complexity of the supply chain is increasing with every new project, with every offered new product and newly signed partnerships. The supply processes should be reconsidered time to time following the internal offer list and actual

markets. When a company is involved as a partner into a new project or contract, it is important to check how the technologies are compatible. The selected technology needs to be able to scale and support clear visibility with the partners in the supply chain. Scalability might be required to reconfigurations, acquisitions activity or capacity constraints.

In a large company with many large projects the supply chain is very complex. There are separated sales and supply units on different level to support the sales and marketing. Partners should show insights and advice based upon their mastery of the broader market. As supply chain conditions are changing continuously, the technology, supporting your supply chain, should be backed by expert to react instantly for every conditional change.

The products are in different state in their lifecycle in a large company. When a product's profitability goes down, there will be a need to trim down costs. These processes have to be pre-defined based on the company's and partners' experience on this level. Successful partnerships are based on a foundation of shared expertise as the partners are working on a common goal together. There is a good practice to scale the partnership. Each company, regardless of its size, is well aware of their most important and successful partnerships. Partners are bringing new ideas and innovations constantly, which can be capitalized on by a smart company. On the other hand strategic ownership can be kept even if passing day-to-day execution to the partner.

There we have a few assumptions regarding supply. One of them is that the supply chain can be performed more or less in the same way. Another is that one can drive all changes inside the company walls rather than scale on it. Certainly partners and markets should be measured with open eyes and recognize when an expertise lies.

4.1.8 Project Portfolio Management (PPM)

Portfolio Management is responsible for selecting, qualifying and prioritizing new business opportunities. PPM is also responsible to distribute the selection to the project's resources in alignment with the budget that is allocated for that. Return and benefit must be evaluated and prioritized to determine the most optimal way to invest the capital of the organization and allocate human resources optimally. They draw the line which tasks are meeting the agreement and which ones to be sustained. Parts, below the "meet the agreement" line, can be redirected or terminated.

Accountabilities and authorities should be well defined for each and every project as a role for define and deliver business results. Project selection process has to be adapted to the life cycle model in use and requires continuous update, as the understanding improves. The goal is always to maximize the value and create the best possible product that meets the company's target, with the available human and budget resources. To create a product, that can be sold on the highest price, while minimizing risks. To fulfill these objectives, the performance has to be continuously measured. This allows tracking the degree of the progress and makes it possible to forecast the achievement. So, portfolio is not running the project, but pinpointing which projects to be executed and defines how to fund them. There are several articles cited about these processes. These could give guidance on how to build a successful PPM process [19].

Most of the time, the portfolio is simple in a small startup company if it exists at all. IN the beginning, they usually focus on a single product to make it success. Portfolio management is usually in the hand of an executive who takes the responsibility to define, execute and improve the PPM process. It is very common to forget about portfolio management in small companies but even in larger scale too. Although it leads to the dark, where there is no visibility of what has been done and what is going on. New opportunities could be poorly measured thus bad projects might supersede good ones at selection of execution. Some good examples are shown by Todd Datz in the beginning of his article about *"Portfolio Management Done Right"* [20].

At a middle size company more business values are targeted at the same time. Project and portfolio management strategies need to be introduced either consciously or unconsciously. There are two general ways how companies are implementing and scaling on this. One is to scratch the surface and managing individual projects separately, without a common knowledge, communication and toolset. Projects are isolated and having their own properties, processes and tools. The other way is to build a well-defined project management office, having the role to handle portfolio of projects and build experience into processes and tools.

The system evolution is well described in a PMI essay about project and portfolio management in [21]. In this essay you can find the following figure summarizing the portfolio evolution in general:

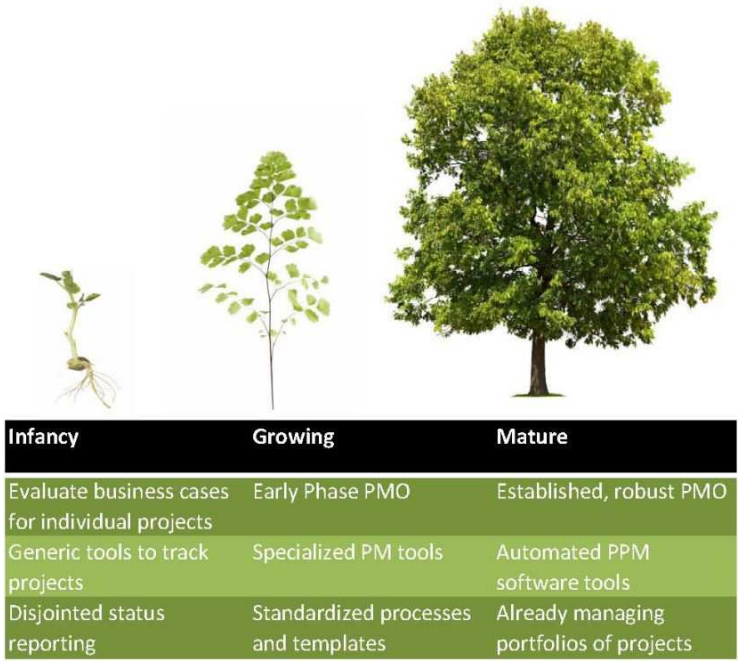

Figure 4-6: Project Portfolio Management Maturity Model

In a big company, that executes several projects in parallel and in large scale, the portfolio management processes are extremely complex. They also have a cyclical and iterative nature. The key players, like stakeholders and their relations, connections and responsibilities, must be well defined as this is critical to the project performance. In this

multitasking system the administration and coordination cost will be much higher than in smaller scales and this should be seriously taken into account. These coordinators, stakeholders and managers should have strong skills, high experience and they should be able to build up, see through and operate such a complex portfolio system in alignment with the company's targets. While a small and simple structured project will survive, a bigger portfolio requires strong governance and more stakeholders delegated into every project unit. The more tasks should be tracked and analyzed, the more chance for faults certainly. A typical problem in large scale is that some projects do not support the larger business goals of the company. It is just left there or forgotten. These are included into a significant part of the simultaneous projects, which do not deliver value. It could happen because of lack of focus on projects or missing key responsible, stakeholder. It could show a problem even with the PPM process. Another typical problem is the delay of several projects, which leads to delay in the return of investment. This could also lead to cost cut and termination of positions as a consequence. Last but not least, there could be bottlenecks in projects created by e.g. lack of resources for a role. Some people become assigned to more projects in parallel to resolve the resource issue. But it is usually resulting in delays.

To avoid these factors of failure, performance management plan can be an answer. In a well-defined performance plan the actual portfolio status can be monitored via key performance indicators (KPIs). These show the actual performance that can be compared to the expected and forecasted results. If they show a bigger gap than allowed, it is time to review the actual portfolio structure, to look after the causes and make the necessary steps to change it back to effective. It is a scaling issue, to have the portfolio be aligned with the company plans again. Perhaps it is not an easy task to define good KPIs for a complex portfolio, but they make the indication of the problem easier and in time.

For big companies having stable cash flow, that is also a good opportunity to buy direct competitors and extend the company's portfolio by that. It has several advantages as, beside the portfolio becomes more complete, there could be new market shares awarded instantly.

There are several ways and suggestions on how to build up a good portfolio system. Many tools are available and ready to help the companies monitoring the process. For example the Quality Software Tool is mentioned in chapter 7 of [22]. This guideline also tries to summarize and give a complete picture on why projects are choosing the wrong portfolio or what metrics are right to find the good frontier. Another tool built for the enterprise from teams of five to more thousand, can be found here [23]. But these are just examples; the task is for the company's decision makers to decide which tool to be used and will support the company's portfolio management in the best way.

4.2 Project

Project management consists of the project- planning, assess and control, decision management, risk management, configuration management, information management and measurement as it is shown in Figure 2-1.

4.2.1 Project Planning

During this phase, the manager or management group shall prepare schedules for the timely completion of tasks, shall give estimation for the effort and adequate resources needed to execute the tasks. The tasks should be delegated to resources and responsible persons should be appointed. Risks should be quantified for each tasks and process. Quality assurance measures have to be defined and costs have to be associated with the process execution. Environmental and infrastructural needs should be identified and provisioned. In addition the definition and maintenance of a life cycle model should be in alignment with the company's defined life cycle models for projects.

Project planning is also important for a small project. Here an action plan can be the first step when all tasks are identified and listed in the right order. This increases the scalability of the project plan. These plans give a framework on how to complete the project efficiently. The key steps should be highlighted in the list not to miss to finish any of them in time. Also as the complete task list is visible that gives opportunity to delegate, outsource or ignore some of them. After the work is completed, the plan can be revised to make it better for the future. Making a note of anything, that could have been done better,

is very beneficial. Quality plan cannot be left out both to show which quality standards are relevant to the project and to determine how to satisfy them.

As the project grows there will be a need to develop more formal management skills. Definition of the project life cycle model takes place to help scaling. The selected model is usually one of the iterative agile models nowadays. A successful agile development on project planning and portfolio level mirrors the agile development on the team level. The transparency, the responsiveness to change and focus on the working software can be applied to project planning and portfolio as well. Communication plan should be defined in the project plan including the contract suppliers and for the cooperation with the customer.

In large scale, these planning processes are complex. Portfolio and project planning office are working together closely and they have to be fully synched. The output of project planning consists of many plans beside the project plan itself. These are the resource plan, financial plan, quality plan, risk plan, acceptance plan, communication plan, and procurement plan etc… Several processes have to be defined as the tender process, the supplier process, the process for issue a statement of work or to request for information, the process to request for proposal and so on.

There are plenty of available processes to use for project planning as it is mentioned in the previous chapters like 3.4, 4.1.2 and 4.1.4.

4.2.2 Project Assess & Control

Projects cannot automatically achieve the desired results even with a good planning and right organizational structure. There should be a mechanism showing the actual status, able to alert deviations and forecasts the success and failure. The continuing follow-up helps to save the wastage of scale resources and gives opportunity to take actions in time.

A good project assess depends on constant monitoring, evaluation and correction of the project. The process is shown by the following figure:

Figure 4-7: Project Control

It is a day to day assessment of the project implementation in relation to agreed schedules, actual progress and achievements to identify deviations. In other words it shows the actual status of the project and providing continuous feedback on implementation to stakeholders. As an outcome, the actual problems can be identified as early as possible to facilitate adjustments to project operation in time. It can also be used detecting project parts that should be improved or reconsidered, learning how different approaches of participation affects outcomes and understanding different stakeholders' perspectives.

Scaling on the company level, a high level evaluation between projects can be done. In medium or big companies, having several projects running in parallel, the assessment shows the projects' relevance, performance, efficiency. The expected outcome can be compared together and required actions can be defined. In order to properly assess project progress, several conditions and understandings are required. Team members must understand the importance of the process of project monitoring, evaluation, and control. Information derived from work breakdown structure is required for the measurement.

4.2.3 Project Support

4.2.3.1 Measurement

The purpose of the Measurement Process is to collect, analyze, and report data relating to the products developed and processes implemented within the organizational unit. The outcome can be used to support effective management of the processes, and to objectively demonstrate the quality of the products. Please find the details of expectations in chapter 6.3.3 of [2]. The collected information is needed for the technical and management processes to have a view on the status of different project parts, identifying the deviations and defining required actions based on them. These measures should be identified and developed during each projects. As projects grow, these indicators are changing constantly and they should be evaluated to the new project's size all the time.

Here should be a "but" for highlighting that the measurement could only give value if we know what to measure and how to use the results of the measurement. Lee Copeland has presented an interesting keynote on this topic at HUSTEF conference in 2015 (can be found here [38]). He emphasized the danger of miss-measure software and then he gave some advice about what is wrong and don't do during measurement. There are some new measurements' indicators which can be used beside the favorite ones as well. For example, I really liked his new indicator, the "surprise", and we are planning to try it out in our project.

In the large scale project I have introduced in chapter 1.1, there are some interesting numbers to show about the growth and measurement. Let's use that project as an example for the measurement. I have taken samples by the growth, when the function test project had become a big project from a medium size. The Titanium tool, a part of the TTCN-3 toolset [40], helped us to analyze the ttcn code for different code smells [39]. The collected data is about different code smells and their occurrences in the system. The basic measurements like number of codes, modules and their relations are also collected. The performed measurement is based on annual samples, which were taken yearly between 2008 and 2016. The dots represent the value for the certain year in a discrete manner. They

are connected together in the diagrams only to show and visualize trends and for better understanding. Otherwise, they do not show a continuous input.

With help of the following charts, I attempt to show that whatever is measured in a large scale system, the graph shows similar distribution in most of the cases and the distribution is logarithmic. The following figures are from the code smell analysis of the latest function test code base for different smells:

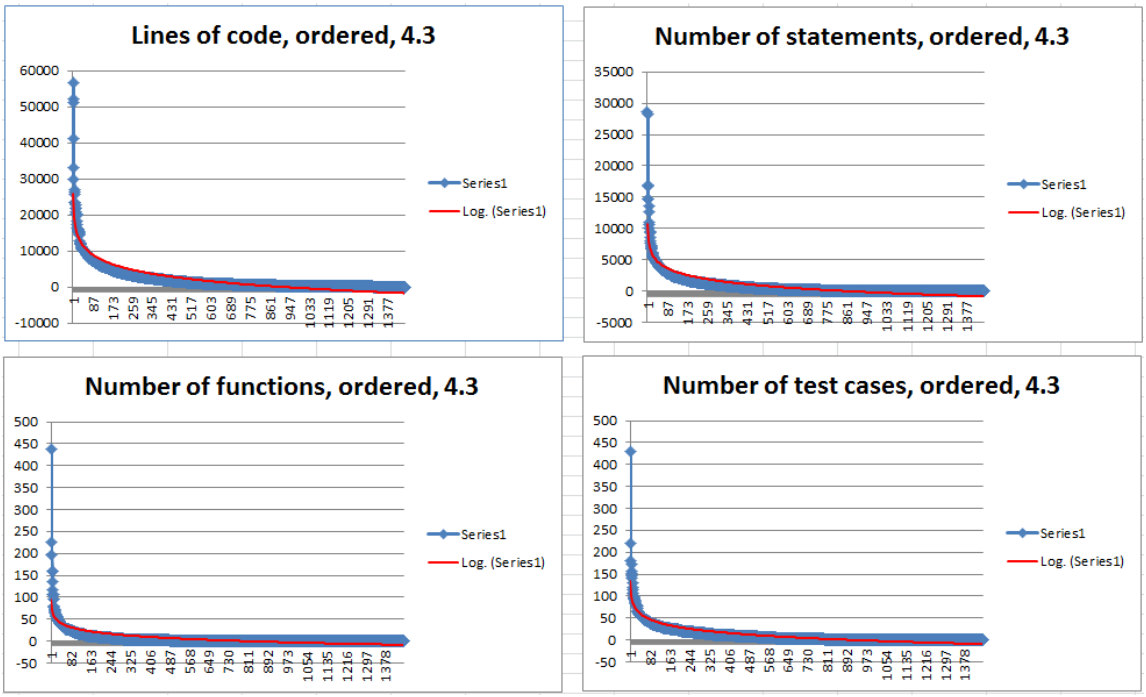

Figure 4-8: Lines of code, Number of statements, functions, test cases in an ordered list from 2016

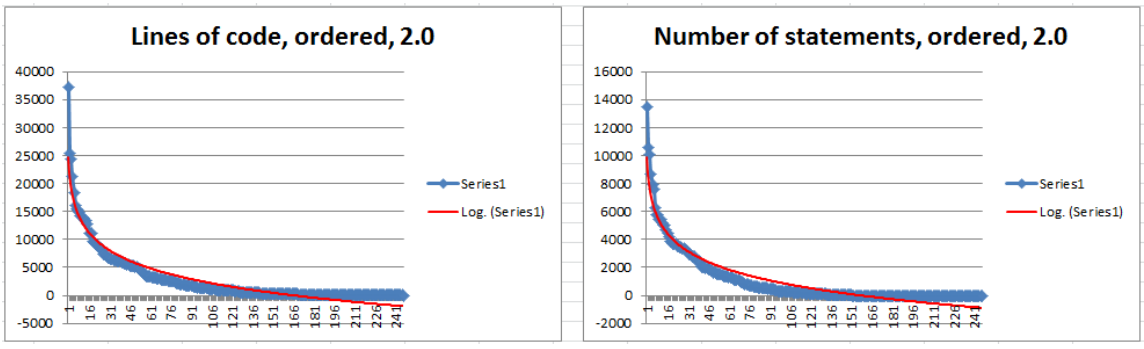

Figure 4-9: Lines of code, Number of statements in an ordered list from 8 years earlier, 2008

As it can be seen in the figures from 2016, the distribution of results for the different code smells are very similar. It is more interesting to see that it was also similar 8 years earlier as well. The red lines in figures are the logarithmic trend lines over the ordered measurement data. In the latter case, in 2008, the size of the test code was at least 6-7 times less compared to the 2016's state. Here the logarithmic trend line does not fit perfectly in the measurement yet, however it is very close. In the 4.3 diagram series from 2016 the logarithmic trend line is nearly perfectly matching the measurement for each case, regardless what has been measured. The measurement can either be a planned measurement, as lines of code or number of function, but can also be for the unnecessary local variable definition that cannot be experienced (it is not a planned measurement usually) without a support of a measurement tool. It is very interesting to see that, when more than hundred function testers have been working on the tests for 10+ years while several restructuring, change of the life cycle management technique and some refactoring of the code has happened, the measurements from the beginning till the end fit to a logarithmic curve. There is a good chance that it could either be predicted in advance from the very beginning, as it is a possibility to predict now for years ahead.

If we take a look on the improvement of the function test system by years, we can also find some interesting points which are worth to be looked at closely. As the system grows, it can be measured via different points of view and see how the distribution of values are changing in scale. I have selected the following years from measurements to reduce the figures to a manageable number and able to show them in one diagram: 2008 (version 2.0), 2011 (11A), 2012 (11B), 2013 (12B) and 2016 (4.3). The series from top to the bottom shows the number of imports and instability values in an ordered way for the above years letting us measure the trends during the test development. I have selected two properties from code smells, the number of imported modules and the instability of modules, because there we can see an interesting trend change. Please find the details, described above, in the following figure:

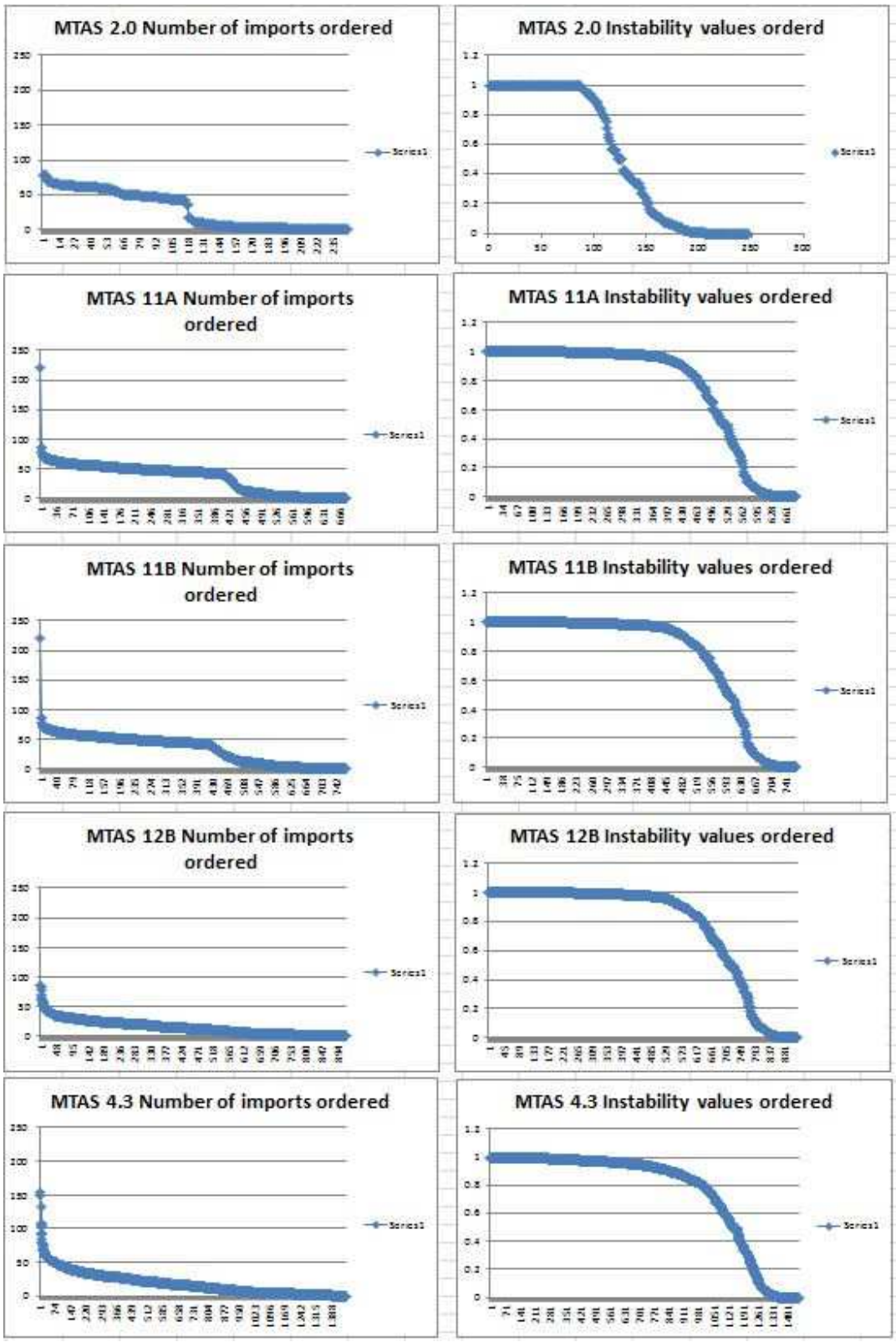

Figure 4-10: Growth trend of imports and instability between 2008 and 2016

Looking at both chart lists in Figure 4-10 from top to the bottom, the trend can be seen clearly shows that, how the graphs are converged by the years towards their current state.

In the left series, about the imports, a change of the curve can be seen between 11B and 12B, which modified the graph towards a final direction. It was a restructuring and refactoring of the code between those years that we will also check separately later on. What we can see here, regarding to imports, is that the unnecessary imports are removed and the remaining ones are reconsidered. Due to that, the curve is corrected to its final shape and now showing a logarithmic distribution, which was not the case prior to 12B.

The graph list on the right of Figure 4-10 shows the instability over the years. The instability of a module is between “0” and “1”. It is relevant for a module and it depends on both the number of imported other modules and on the number of other modules importing the measured one. It shows the chance for any error could happen because of a change in the imported modules. If the value is “0” for a module that means, there is no imported module (it is not built on any other module), so there is no chance for a faulty because of other (imported) modules are changed. On the other hand, they are imported by lot of other modules, so it is hard to change this module in a way that avoiding the cause of any fault. If the value of instability is “1”, that means the opposite. It is an “instable” module importing many other modules, but it is not imported by any other. So there is a big chance, that changes in the imported modules are affecting and changing the module behavior. Looking at the right list of Figure 4-10, we can see that the number of instable modules (with value 1.0) is more and more until 11B. The trend is changed at this point with the restructuring and introduction of new ways of working. The number of maximum instable modules started to decrease from the full test system point of the view. Reason for this could be vary. What I see from test perspective is that, the new test cases more and more rely on other test cases and modules as the system going to be more complex. The required test cases are more for testing new interactions of the existing features and that requires service from other existing test cases and modules from the top layers of the module graph. The testers try to reuse the existing modules to keep the scenarios consistent and also to reduce the development time of the new parts. It is very well shown in the above figure by the less fully instable modules by the time.

Let's take another look at the growing trends of this project. The following figure shows the number of modules in the system and the next one visualizes the connections between them:

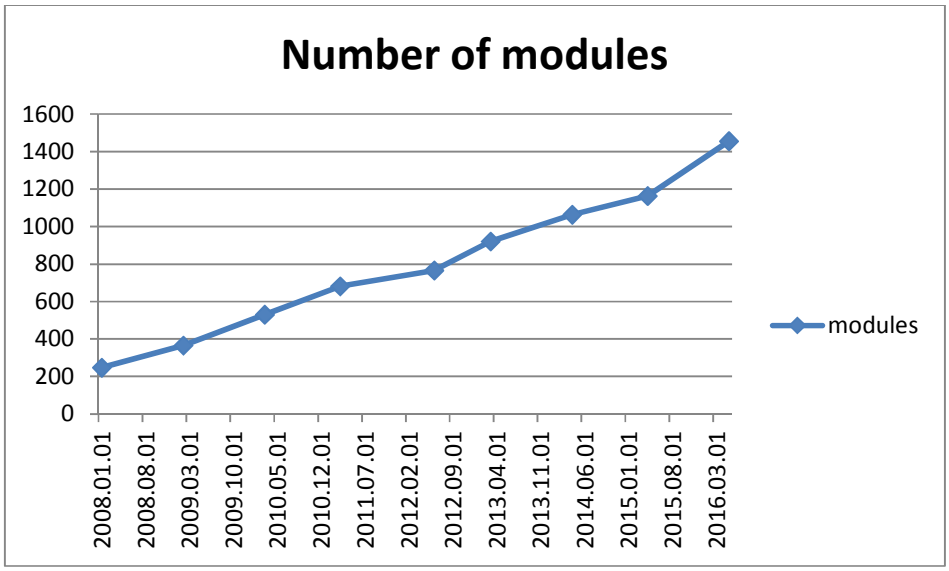

Figure 4-11: Number of modules per year

In 2008, the system had about 250 modules and the growth trend after that was nearly linear. The number of modules is growing with an additional 250 modules per year in average.

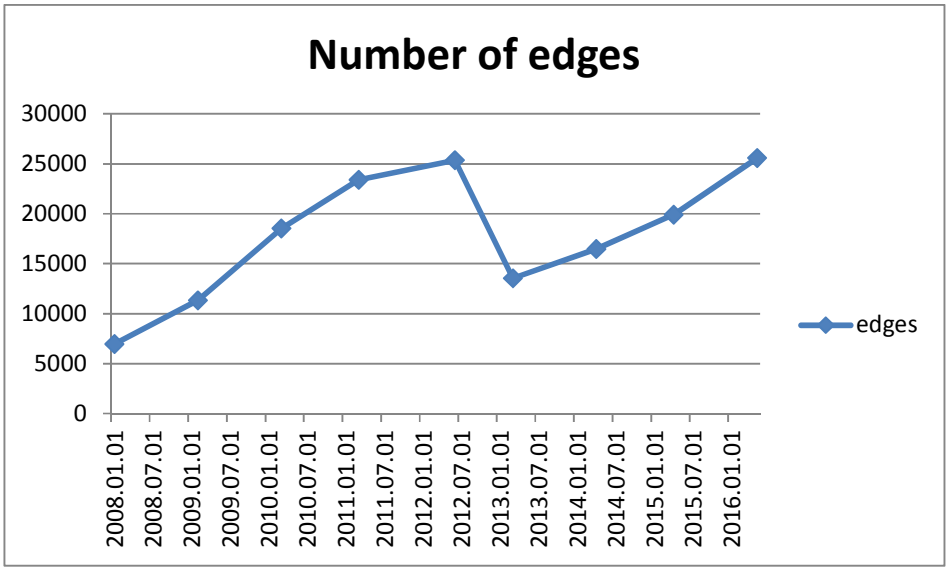

Figure 4-12: Number of edges between modules by year

In Figure 4-12 we can see the changes of the number of edges (an edge represents an import) between the modules. Between 2012 (11B) and 2013 (12B), the diagram shows that, there was a refactoring and restructuring of the modules which affected the import hierarchy of the modules and also reduced the number of edges by ~50%. It gave the modules and codes a little speed up both in the compilation and in the execution. Between 2013 and 2016, the number of edges is reached the same volume as before in 2012, but as I mentioned above, this was mainly a result of the new interaction tests on the top layer. In that layer, the newly added tests required 2-4 other modules into the import, but only from the same layer. It was a surprise to see that the average degree values, which were also decreased by 50% at the refactoring, have not increased with the number of edges. The network diameter, average degree and number of weekly connected components can be seen in the following figure:

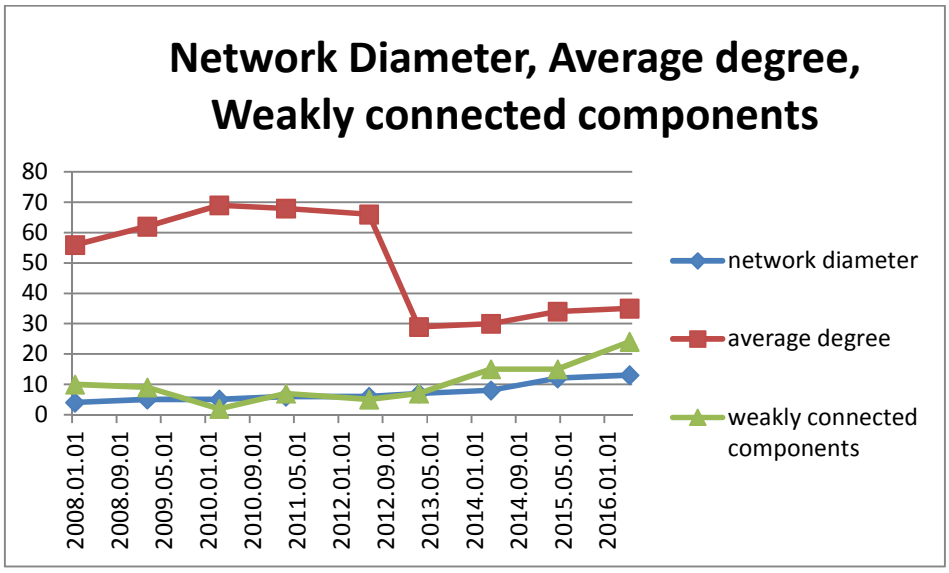

Figure 4-13: The network diameter, average degree and weekly connected components by years

In the interesting interval between 2012 and 2013, about 14000 unnecessary imports and ~3300 local variables were removed (using Titanium, see details about in [40]). The code was restructured, and thanks for that, the average degree of the module graph was simplified. After that, this degree has not increased significantly, but was kept even if the number of edges between the modules increased more rapidly. I assume that, this is because the new edges were introduced in the test case's layer only as a result of the

interaction tests described before. The number of new features to be introduced is typically inversely proportional to the age of the system. This could also be the reason of the less instable modules experienced after 2013.

Measurements could provide information for the various sizes of projects. Certainly measurements are less frequent and rarely used in small scale, but after a while, it becomes essential to be able to monitor the project. In medium and big size, the complexity makes it difficult to draw conclusions and to be able to identify or predict the right numbers. Measurement could also be dangerous if the technique is wrong and we draw false conclusions from bad results. On the other hand, when it is done in the right way, the information can be used to predict the future, keeping the project in align with the budget, notifying different needs (for a change, refactoring or quality deficiency).

4.3 Engineering

4.3.1 Implementation

The purpose, outcomes, activities and tasks of a software implementation process are well described in chapter 7.1 of ISO/IEC 12007:2008 in [2]. As it is discussed before, the applied life cycle model is ideally an organizationally-defined model.

In a small company running one or two projects, it is easy to apply and use a predefined method for implementation that could serve the company's needs. Either a sequential model like waterfall or the two most popular agile frameworks, Scum and Kanban, can be successfully used as it has been discussed in 4.1.4. As the project and the company grow, design groups, often called "teams", appear. They could be allocated for different tasks and purposes as well as there could be several teams playing similar role. The latter case is specific for bigger projects when size of similar tasks is too big for a single team. In case there are teams, then they can be involved into the creation of the implementation process as well. In this way they will have better look and understanding as they are involved.

As the popularity of Scrum and Kanban grew, there is a need to be able to use them in larger scale. Medium or big projects are facing the problem of how to scale on agile frameworks to suit them to larger organizations. There are several methods to emerge this.

One of them is the “Scrum of scrums”, another is called “Scaled Agile Framework” and a third is for example “The Great Wall”. Let’s take a closer look on these techniques.

When several individual teams are working together, as this is the usual case in a medium or big project, Scrum [26] is the most popular agile framework for software implementation nowadays. In case of Scrum starts to work well at team level, the next step for scaling agile could be the scrum of scrums. It is no more than scaling the agile in an agile way. One of the main components of scrum of scrums is the multi-team stand-up meeting. This is not the same as the daily standup meeting which is a status meeting. It is more about coordinating the work of multiple scrum teams. For example, they are discussing important issues that affect the group, defining the required actions and responsible for them. Another misunderstanding is to see it as a meeting for scrum masters to talk about the agile process. But it is not true. However, there should be a selected member from each team, he or she should not necessary be the scrum master. Ideally a person with strong technical knowledge is the best choice to represent the team on this forum. The scum of scrums meeting is usually not held on every day as daily standup, but usually takes place two or three times a week.

Looking at the numbers, a typical Scrum team consists of 4-10 people. Perhaps it can scale up to bigger teams, but the experience shows that bigger teams than 10 people loose from their efficiency. Rather than scaling up the teams, scaling through and having teams of teams is the preferable way. The following figure shows how a scrum of scrums approach allows scrum to scale up and make scrum of scums of scrums and so on:

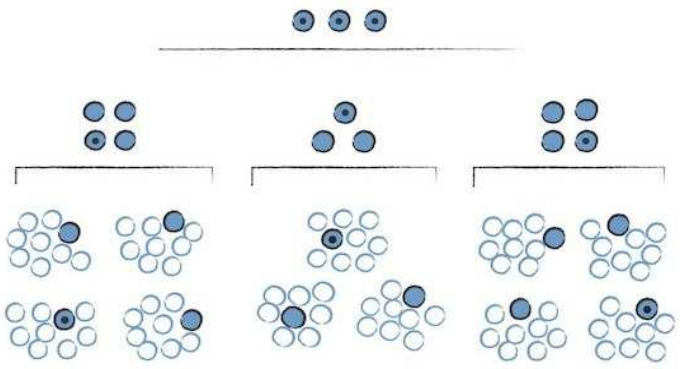

Figure 4-14: Scum of scrums of scrums

Another way to scale agile in large organizations is SAFe pioneered by Dean Leffingwell (see in [30]). The structure of the framework can also be seen on the following picture:

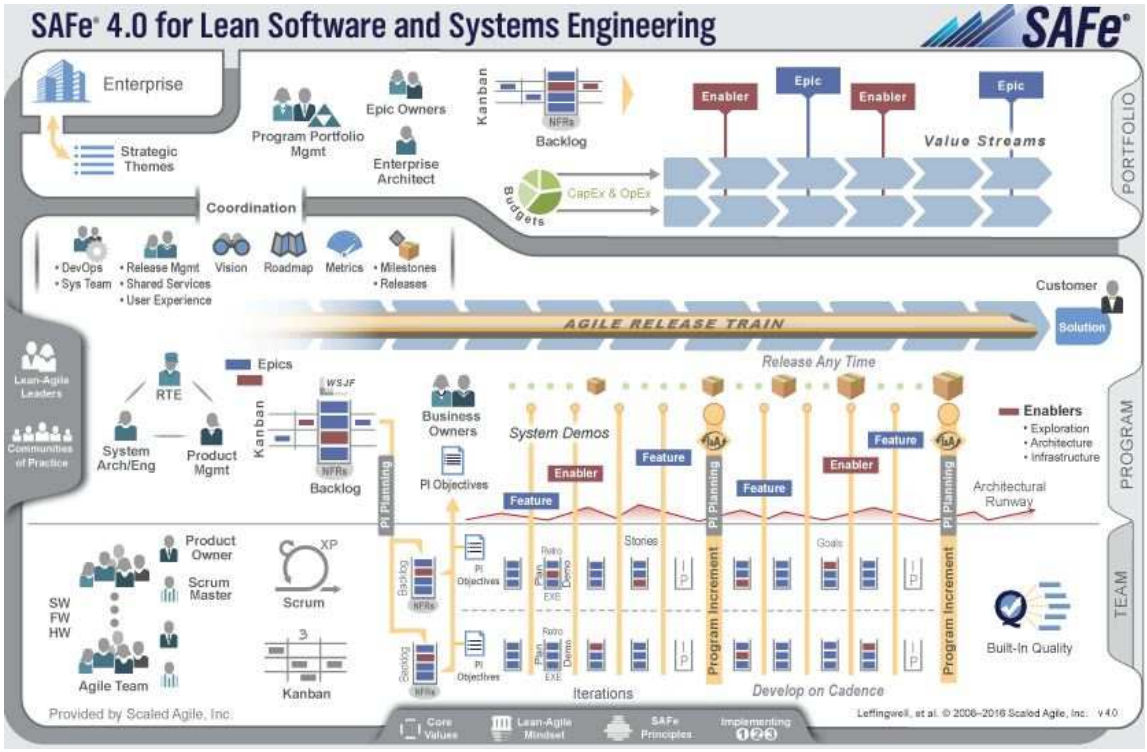

Figure 4-15: SAFe 4.0

As it is shown in the figure above, SAFe defines three levels in the organization: portfolio, program, and team. This structure is more in compliance to larger organizations. The themes are bigger parts of the related work and they are mapped to business epics and architectural epics. Business epics, such as certificate a new product at a customer, describe customer initiatives. Architectural epics, such as replacing the hardware or migration to another operating system, are company technology initiatives. These epics are in the portfolio backlog with a given priority. Each epic becoming an agile program represents an own agile train. Agile teams work together on these trains within the organization. Each program backlog contains several features and architectural work items that lands in a team's backlog for implementation.

The Great Wall says that Scrum of Scrums and SAFe hold a danger: They are optimizing the management's benefit rather than the project's structure to create a usable solution. The need of management is not a question, but the delivery of the value comes first. The Great Wall is a physical wall that holds the User Experience Map of the solution from the customer's perspective on the left towards the User Stories to the right. The map is layered into Minimal Viable Products (MVPs) by the priority of the User Stories. These layers represent the releases of the product. The User Stories are written on color-coded cards indicate the status of the stories. So The Great Wall shows the project's status that should be kept up to date so it can be used at any time. A good summary of this method can be found in this article about The Great Wall [31].

I would like to mention a drawback of these recursive layers are built on each other. When you have a really big project that has to be scaled, the different project layers like team, subsystem, application, and network should either be synchronized during implementation and they are several. The daily standups together with higher level scrum of scrums' meetings can take away all time of the involved persons. Even if the number of these peoples is less and less by going up on the layers, the time expectation is more and more significant. In Agile, the number of meetings, communication and information sharing are always in focus. However, one can think sometimes 'why we are talking so much rather than spend this huge amount of time, or at least a part of it, for creating value. They are partially right. In these agile frameworks, it is easy to pass through to the other side of the horse. It is usually a problem in all of our projects as well. The solution can be various. First of all, we should be strict to the time schedule of these face to face meetings. A daily standup should not take more than 15 minutes, scrum of scums should focus on the main topic. Participants should be well prepared for sprint planning, start, commit, pre-delivery, delivery and done meetings to reduce the time on them. We should try not to bring the whole team to all scrum events, but try to share and balance on these roles and spread them within the team. Sometimes, some of these discussions can be done via mail or chat rooms, where questions can be answered and feedback can be given asynchronously. It could give more time for creating value. In addition, the asynchronous communication is usually kept for the future which can be referred and reused later as well.

As a summary, I advise to keep an eye on the number of meetings and the time has spent on them. It is always beneficial to measure on the communication cost and take action in case that exceeds the allocated budget.

I cannot walk away of the outcome of the meetings. We had several meetings in our big project ended up without any decision. It makes people frustrated and gives a bad feeling about what should be the next. The best is to decide the wished outcome of the meeting in advance to avoid the usual mistake to left a meeting without any decision. A good meeting provides a well-defined action list with pinpointed responsible to each action point and a list called minutes of the meeting about achievements. My other advice is to keep the invited stakeholders on the minimum level to give more room for agreement. A meeting with 5 or more stakeholder could easily become arguing where none of them will say a final word.

4.3.1.1 Software Architecture and Design

Software architecture is often seen as a bad metaphor as the software is not constructed like a building, but rather it grows like a living garden. The software design improves or degrades it day by day as developers add, change or remove lines of code continuously. The software architecture is not static thing, but it evolves and it is always growing better or worse. The first question here, where is the real architecture? Is it in the documents being created and maintained by the architects, or is it in source code files could hold hundreds of thousands of codes? Obviously, the latter is the real design that reflects the true large-scale design or architecture. Every programmer is a kind of architect in this way.

In a small product group with 5-20 people, it is really working like it is described above. Engineers, are developing the code, are either architects in one. There is no division between architecting and programming.

Growing towards medium scale, the code base is going to be bigger and bigger. There will be one or several master programmers start to act like an architect. After a while, they will be accepted in that role and promoted formally as well. As they are close to the code, it could still working well in case the communication with designers remains frequent and

clear. They should continue monitoring and participating in coding, mainly in case of complex problems, as well as being a part of restructuring or refactoring the program.

In a large project of a big company (with 500 or more people involved), there is usually a separation of the architecture and the code. The architects are far or even they do not know anything about the implementation of the architectural design. This is a common mistake, when architects are not in daily touch with programmers. As a consequence, the implementation could really differ from the idea that was architected by system engineers. In big companies, this architecture or system engineer group is in a separated section and the members of this board are usually well experienced people. But moving away from the real program code, they slowly lose touch with the reality and they unintentionally become "architecture astronauts" out of space. After, the master programmers start to believe that architects should not program regularly. The architect astronauts think that the code is not the architecture and also they start to believe that an architecture foundation is a must before start to implement anything. The regular programmers feel that there is a separated group for creating the architecture and coders are not architects. The circle is closed. These together could lead to degrading the real architecture over the time when architects writing the documents for themselves and/or to stakeholders. But after a while there won't be significant connection towards the real code structure and behavior. The programmers will ignore, or do not even aware of, architects. Certainly it is important to have great architecture, the question is, what is the best way to have it? In the next chapter we will see some examples and suggestions about how to create good architecture, how to avoid the above mistakes and how to scale on that.

4.3.1.2 Software Architectural Design

As for most processes, it is better to start the software architectural design with an analysis where the focus on the requirements, resources, needs and constraints which could strongly influence the architecture. This early identification of architecture helps to prioritize the needs requiring for early design decisions. These are the information structure, type of method, programming languages to be used, relation and structure of interfaces and components and so on.

Nowadays in an iterative development, an architectural decision is not a final at all. Changes could happen repeatedly, one or more times per iteration, for example during the architectural planning, review, extension or reconsideration. A good advice could be here that, do not take these decisions final, but try to challenge them to find a better one.

A good forum to solve the cross-team system-level design and architecture issues is the "joint design workshop" in Agile. When several feature and PLM (Product Layer Maintenance) teams are working together on the project's subsystems, feature teams often work cross-component with each other. Other teams are dedicated to a component instead of feature, thus a feature could affect several component teams. There will be a natural need in these teams to get together to discuss and decide on architectural design issues, interfaces, communication and synchronization. Sometimes, the whole team available on the meeting, in other cases it is enough to send a team representative. This should be a real workshop rather than a PowerPoint presentation and it should be repeated until agreement on all questions and details come up. Technical leaders and coordinators are usually invited to theses forums where they can also teach on technical topics to help the teams to make right decisions. About how frequent it should happen, the recommendation is for 3-4 iterations at least. In case there is a separate need, any team can initiate it. If a team representative have participated the meeting and returning to the team, the outcomes of the joint level meeting can be discussed. As a result, the architectural part allocated for the team should be built in to their model/plan and to the code. In this way, the architectural document partly or entirely replaced by a human communication and representative who can bring the details of the decision to the team level and ensuring to make it happen.

In an agile setup, the architects and system engineers are regular team members and should be avoided to separate them to external groups or sections. As the teams are cross-functional and responsible to develop a complete solution, the architectural work is, and should be, a part of that. It is not easy to transform a big organization to agile development as this would mean to terminate other groups, like system engineering here and to join them to the teams. It is often a change in their role as well as they should participate in the coding and being a part of a multi-role team instead of a specialized group. Another factor

to mention here is the problem to change people mindset. But this dimension has been well described in 4.1.4.

I would like to emphasize some pros and cons regarding the above technique that I have experienced about. In one of our big project, I am also participating in, there were similar transition and organizational change towards agile like described above. System engineers became a part of the teams and started to work with them in the cross functional way. In the old model, which was a kind of waterfall, we have documented everything precisely. We have also tried to get rid of all the documents at least on team level that we have seen as not required any more. At some point we recognized that without them, we cannot rely on anything in serious situations with customers. Detailed architectural design documents are one of these. That was hard to decide, if we can follow the above methodology and sit beside the hands-on programmers driving them through the architectural problems or we have to write the documents. On one hand, when we have tried it in the agile way, we were unable to refer to the details to the customer. Project managers experienced that it took significantly longer time to make the handshake even on a single feature. In a large scale project, the customer cannot, or it is better to say won't, sit beside the teams to learn each subsystem structure and behavior. Sometimes, they are far away, other times they do not care or do not wish to be a part of the project in that deep. In large scale, the number of features could make it impossible to do it simply because of the time it requires. On the other hand, plans on the white board or on a wiki page cannot be shown to the customer to discuss about as these formats are not the "official" documents were used before. So, we made a decision that we do need a separated architecture group takes care to coordinate and handle the documentation part of the architectural design but strongly cooperate with the design teams. It is against to the recommendation above. This is also a hybrid solution as we have kept the system engineers inside the teams, but they need to provide information to the architect group about the model of the program that they are developing. Chief architects merge this information into a product documentation library that they maintain and able to use in the customer interactions. Drawback of this solution is the time should be spent for documentation, but as an exchange other teams or even team members can use it for learning the system or other systems by alone or preparing for the daily- or

joint design meetings. It is also usable when developing the code, as it can be a base of that as before. The system engineer can focus more on architectural tasks as an exchange they need to spend less time for mentoring as the document is always a good base for everyone to start with.

The architect group perhaps is being involved into the design, not to move them far from the code. Perhaps, they are not programming directly, but when they make technical decisions or supporting the customer in trouble report handling, they give code level advises to the proposal helping out the designers or promoting the customer to understand it. On the other hand, they are not a part of the teams and thus some of them sometimes tend to become a PowerPoint architect as mentioned before.

These hybrid solutions are not rare. Several articles and essays are written to discuss about this phenomenon. Please find some examples, description and references about in [41].

4.3.1.3 Detailed design

Detailed design could be so various to mention all the possibilities and techniques it could use. Instead, I would like to focus on some advises and highlight a few techniques and possibilities related to scaling.

First problem appears, even in a small scale garage project, is the lack of specific knowledge and design experience. Education is essential. So, there should be at least one master designer who can drive and teach the others in the project. Regardless of the used programming technique and lifecycle model, mentoring people is a key to success from the beginning.

As the project grows, the training process will replace the ad-hoc mentoring system giving a planned structure of educating people. Master engineers take a part of this process by coaching people about the system, used programming languages and techniques. In agile, design workshops are also used for this purpose when technical coordinators, system engineers give trainings to teams on the main topics. Topics could be from a wide range from the general design skills towards more and more specific knowledge (like

subsystems, components, external products or architecture). People, who might need extra attention in the beginning or when the focus changes to a new area, pair programming can be used. It is a good technique to perform the knowledge transfer within and between teams as well. Generally, before writing any code or any design documents, it is better to think over and write down some ideas that can be discussed with other people or technical expert. If it is possible it is good to find out different solution candidates and evaluating them with people having more experience to learn from them.

In a large company, these educating processes are usually centralized and layered. The company level trainings could give the basic knowledge on the specific project areas and on the company level processes. This is a good base to build the upcoming projects on. In case the project reaches a complexity, there will be a need to work out these education processes more precisely on the project level. These are the extension of the company level trainings as they should be aligned with them (e.g. to extend rather than overlap them). For different areas, there should be well-defined training packages that a newcomer expected to receive. This is to ensure they become productive as soon as possible in that project's area. In a complex system, a training package could be quite long, so there should be a schedule and a good priority order of trainings. In agile, the team and forums they are participating could help a lot to become a valuable designer within a short time. As most of the trainings could be performed on the team level, it is a great advantage.

Another key point in the design is the interfaces. We can find interfaces between subsystems, layers, components or towards the users as well. All interaction and communication between different services are going through on them, so there is a need for a strong cooperation and agreement when creating them. They should be as less complex as they could and they must not be sated as final ever. Also the number of layers should be balanced to the size of the system and being align with the necessary separation of the system logic. Using more layers will downgrade on the system performance and using fewer layers than optimal will result the same. It is also recommended to avoid any constraining implementation as by the growth, the interfaces must be able to change and improve. As it is mentioned above, getting feedback on interfaces before implementing is an opportunity to improve the final quality of that. This advice is also stand for every

design activity (may be for other activities than design as well). The good way could be to look at well-designed interfaces and try to use and to learn from the examples.

In case there is user interface (UI) in the product, it is the first thing what people experience from the product. At UI development, it is always beneficial to put yourself in the client position and imagine how to use the user interface. In case of a user interface it is also very important to be able to reconsider and in case re-implement that. A poor UI could prevent the product from success at first seen. Unfortunately, we can see plenty of bad examples for that. However the UI is so important, usually bigger projects do not care much about the UI architecture, but they do care a lot more with any other internal components or subsystem. It can be experienced even more by the growth and it is most often present in all large scale projects. Sometimes, there is no architecture group for UI at all.

In scrum, there is a good way to ensure the design of the UI or any other component. If there are available experts on UI area, then it is worth to spread those into some teams then they can take the UI tasks. If there are no experts, there is a need to start educating some, team members from different teams preferably. In Agile, it is not recommended to separate them into an architectural group and that is not even beneficial. In either way, there will be experts in the teams can emphasize the importance of the user interface. Perhaps as with any other solution here could also be a scaling issue. If UIs are created by more than one teams, who located at different sites and far away from each other, than there could be consistency issues at integration of new parts. It could be simply, because they are too far, cannot communicate enough or they are lacking coordination and so on. So in one hand, yes, that is an advantage to break down the system to smaller tasks. That could give the opportunity to found new development centers all over the word and spread different part of the backlog to them. On the other hand, it will take its price on the communication and coordination time obviously. It is worth to mention that, in the agile way of working, the design workshops and communities are addressing these issues as well and they try to handle this problem there.

When a project reaches a medium size with 50 or more developers, the different design experiences and coding styles make the code hard to understand and impenetrable. In a bigger scale, our solution was to develop style guides that we have called design patterns giving a common shape and guide for coding the several subsystems. Sometimes these guidelines extend the well-known design patterns with project and component specific patterns and they are expected to be followed by the programmers. Following them makes the code unified and also it prevents the design from the usual mistakes they have experienced in earlier stages of the project and extended into the guidelines. We keep them up to date with new findings, ideas or for new components. Also we are revising them time to time with help of the subsystem engineers.

4.3.1.4 Design for growth and scale

In this chapter, I would like to highlight some general rules that can be used at any scale about how to prepare the code and develop for growth without completeness. These advices help to keep the opportunity for scaling the program code when there is a need or requirement.

First of all, the coding depends on the requirements. A good, prioritized requirement base is where the planning, architecting, modelling and coding starts. We can try to anticipate how requirements will evolve and identify the likely next features to count with them at design later on. It is also highly recommended to ensure that the code works if scale changes (on users, call, communication or dimension) by 10 or 50 or more times, as the right solution for a given scale most probably won't be optimal for 100 times as much.

Most of the system is having basic requirements for low latency nowadays. But even if they do not have, it is better to endeavor for that. The other one is the low average response time which is also very important. To think about them there is plenty of time can be saved for other features and components that will be a part of the future system sometime during the development. For example, in external communications there should be as much data as possible being sent at once to reduce the communication time a more request-response pair could cause huge overhead in a less effective solution. Communication is costly. Think about how much data are shuffling around, check the redundancy or if some of them

can be calculated from others. It is a good idea to use timeouts that can also help to cut the tail of the latency. Priorities can also help to process the most important requests first or to make sure that the most critical ones are not delayed. Caching the rest is also an option. The parallelism can help scaling on the performance. For example we can use the Amdahl's law [33] to speed up the program if there is a possibility. Also the Gustafson's law [34], a reevaluation of Amdahl's law, can be used to scale on your system cores in relation to scale up on the data should be processed and so on. This advice list can be continued forever, but the part can be used is different for each project. It should be experienced and learned to find the best solutions.

4.3.2 Software Support

4.3.2.1 Software verification

The purpose of the Software Verification Process is to confirm that the product meets the specified requirements and to detect and fix as much faults as possible. In order to achieve this, a verification strategy has to be developed and implemented. It identifies the criteria for verification and gives detail on the results and measures to be made. During the tests, defects should be identified and recorded. The results of the verification also have to be made available to the customer and to other involved parties. Please find the details about in [2].

There are 2 fundamental approaches to software verification. The dynamic verification, also known as "experimentation" and the static verification, also called "analysis" [35]. Both techniques are good at discovering system faults, but they are usually not enough good to ensure the correctness of that. There is an endless debate on which convention is the better. It is also interesting to see them from scaling point of the view.

For example formal verification is a rigor tool set to ensure about the program fulfills all requirements. Even for a very small code, it could take a huge effort to perform and prove that it is working well. By this fact, it has scalability issues as the method is very costly. Formal verification can be either very expensive if they should be implemented or they will not be automated at all without implementation. These are no options for the software

companies. Most often they cannot afford to spend this amount of time for verification. Also the business is usually over to ensure about the maximum quality. For this reason, usually the dynamic approach is the favorable way for big projects. However, formal verification could be essential sometimes even in large scale. For example, when there are code segments, where complexity metrics indicate potential danger, it is very difficult to detect bugs without formal methods. Otherwise companies usually try to hardly minimize to use it for the reasons above. Nonetheless, we can find some good examples for implementation of formal verification and usage of the formal approach in large projects. One of them is at Facebook, where they have introduced and use it in the basic development at large scale. Please find the details in [36] .

In the dynamic testing approach, a usual question is even in the scaling up of the project, if manual or automatic tests shall be used. In manual software testing, the test cases are executed by a person manually, without help of any tool or script. On the other hand, in automated software testing, as the name implies, the tests are executed automatically by help of different tools and scripts. It is a very common question in software technology: what way is the best to choose for testing the product? The answer is not simple and can be different, depends on various factors including the project status, requirements for the components to be tested, the budget available, resources for test, skills, experience and so on. So, it is better to clarify the difference and being aware of the pros and cons of them to help out the decision. The following figure summarizes the main differences:

Manual Testing	Automated Testing
Manual testing is not always reliable due to human error.	Automated testing is more reliable as it performs the same operation each time, guarded by tools and scripts.
The Manual test execution is not accurate when need to test the same scenario frequently and several times while the product is in continuous change.	Automated tests can be executed at any time when required and help the test even in continuous change. It is very useful at regression testing.
Manual testing requires human resources for every execution. It could be done faster than to develop an automated test. It is useful when test cases have to run only a few times and when frequent repetition is not required.	Automated testing is time consuming once, when developing the tests. After that, it is executed by software tools and only needs rare updates. It is useful when tests are executed regularly over a long time period.
Main cost is to provide human resource.	Main cost is to develop the test cases, testing tools and scripts.
Manual testing allows for human observation, which may be more useful if the goal is user-friendliness or to improve customer experience. It is very good for user interface testing.	Automated testing does not entail human observation and cannot guarantee user-friendliness or positive customer experience. So it is not the best suited for testing the user interface.
Manual tests cannot be done concurrently on different machines. It could require different testers.	Automatic tests can also be done on different operating systems and/or hardware even concurrently.
Manual testing is not suited for build verification or testing repeatedly the same.	Automation testing allows to automate the verification of the build and support to have different loops on different levels.

Figure 4-16: Manual vs. automated testing.

In case we take a look at the different test areas, the manual testing is best suited to the exploratory, usability, end to end and ad-hoc testing.

Exploratory testing requires intuition, creativity and experience from the testers. These tests require human skills to execute them and most of the time it cannot be automated, or not worth to do that. This test is usually about not described scenarios when details are unknown and the test is to find out if it works (e.g. interaction with other vendor system). There is no specification or that is poorly written.

Usability testing is about to test the product from the user point of the view. In this area, the tester checks if the software is user-friendly and convenient. As the human observation is the most important factor in these tests, the manual way is obviously preferable.

End to end testing is a methodology used to test whether the flow of an application is performing as designed from start to finish. It is usually done by manually with real clients to ensure that the right information is passed between various system components and clients.

In ad-hoc testing, as the name also says, there is no predefined specific approach. It is up to the tester how and what to be tried out, so the understanding of the program is essential.

Automated testing is preferred in case of regression-, system- and load testing for example. In general, when testing requires repeated execution, test automation is the preferred way to save time on the human resources.

Regression testing can provide feedback on the constant changing code and doing it manually in large scale is a huge waste of time.

The load testing is also more efficient, if it is done automatically. It requires the simulation of thousands of concurrent users. A static set of the scenarios could make the runs comparable and that could discover performance degradations in addition. Looking at these factors, we can find the best approach according to the situation of the project and in align with the budget and the time plan.

To avoid any misunderstanding about the above analysis of the two techniques, there is no race between them and there is no generally better. As I see both of them are necessary and can complement each other in multiple ways. It is just depending on the above mentioned factors which one can suit better in a certain situation to achieve the goal. Where one of them can identify the faults, the other would miss that, and vice versa. So, it is better to use both of them in the right time. Beside of these, it should be highlighted here that the professional testers and automated test suits are not perfect either. Some bugs always remain in the final product. Anyhow, the perfect product is something we only try

to be closer to. These techniques need to be considered and carefully applied, as the right choice can save a lot of time and can give improved results while a bad choice can take the time and the budget.

In a small scale garage project, the first testing is usually some printouts or manual attempts for testing of the software. It is done by the designer itself and there is no separate tester or tester role is defined. It could work until the project is not so complex, but as the product is growing, there will be a need to think about the test plan and strategy sooner or later. Designers will feel that the manual tests bring more significant time from development than they can afford. Here the test automation comes into the picture and the wish for a separated tester role to reduce the cost of manual testing. When the product complexity reaches the point, designers won't be able to see through the whole code, the preservation of the reliability, speed and reusability become important as well as the long term product quality.

In medium scale, as we keep adding tests, these automated test suits becomes more and more big and robust. A significant human- and cash resource have to be allocated in the project to maintain and develop them. But, there will be advantages of these test suits as they can be used to verify the legacy behavior. If a new iteration or change released, a new run could show that no new bug has been introduced. A continuous integration environment can be built on top of this by executing this test suit on every new build. Perhaps to have it line with the code change, the test suit should be developed to cover the newly added feature or component parts all the time. Without that, the tests may start losing quality as they stop providing meaningful data for the new features. Certainly, it requires resource with good knowledge about both the new features and the test suit itself. To achieve this in agile, these testers are a part of the cross-functional feature team where they are involved into the test planning of the newly developed code parts. In this way, they are able to keep the test suit updated for that feature.

When big companies have a complex product portfolio, the resource allocation is going to be tougher and sometimes critical. Keeping the time to market as low as possible is essential according to get the bigger market share. In this situation, companies that are

running large scale projects, tries to pull resources away from quality assurance and route them towards development. They are doing that on the assumption that they can rely on the test automation in all of their testing needs. Beside of this practice, an ordinary trial here is to introduce the test-driven development. These two approaches could come even together however one of them is enough to leave the project without a tester. They are forgetting about the original reasons they needed testers: to try to ruin the product and find the faults in all possible ways (which could be infinite for a large system) and give feedback from a different point of the view independently from the design.

4.3.3 Maintenance

Software Maintenance is a significant portion of the total cost of a software project from the acquisition during the whole life cycle of the project. It is a general belief that maintenance is only about bug-fixing, or introduced for the need of handling emergency issues. In reality, maintenance is for correcting the bugs that were reported internally or externally, keeping the software relevant to the requirements while modifying it, tracking the goals and trying to prevent the product from reoccurrence of the fixed failures. It could require significant resources even up to 50-80% of the budget depending on the project status, maturity and so on. In maintenance, similar to development, there always has to be a planning and preparation in place in order to get things moving. The process has to be line with development and most of the time, as maintenance and development are usually running in parallel, the maintenance process is a part of the development process. Depending on the product, there could be parts requiring continuous maintenance while others need to be updated only at regular intervals. In either way, the detected or foreseen faults and updates should be tracked and well documented to avoid project failure. Detailed discussion about the maintenance role in projects can be found in several essays like in [42].

In a small project, the software maintenance is handled by the same person usually who have developed the code. From the first experiences in maintenance, there is a need to create a maintenance process that supports the business needs of the company. After a while, one or two persons won’t be enough to take the role of the maintenance tasks as

they can't afford to put focus on several important project areas at once. It is very important to have a person who is able to oversee the whole maintenance structure of the project. In case there is no one, then it is better to start training some candidates in the first place. In this level, the maintenance costs are usually relatively small compared to the design, environmental or other project costs at this level, but it is starting to increase faster with the growth of the project.

In medium size, projects should have the maintenance strategy well-documented. This strategy details the process deeply including how to create, execute, and analyze test scenarios and what actions to be taken depending on the results. Often, the end user should either be involved into the maintenance of the product, which is a good opportunity in small and medium size projects, as the customer is not that far from design than in large project scale. In this level, the maintenance usually becomes separated from development and the cost that it requires, becomes significant. This brings more focus to this process in order to keep the budget low. In agile the maintenance tasks can be dispatched to one or more teams in the project depending on the current workload that project layer maintenance (PLM) can scale. In waterfall, there are designers and testers allocated for the tasks depending on the project phase in the life cycle. Perhaps, in case there are released versions those should be continuously maintained.

In large scale, it is worth to mention what factors could reduce or cover the significant maintenance's cost. Perhaps these are valid in smaller scale as well, but in a big project it becomes essential and a key to success. For example, by including the maintenance cost into the price of the software or limiting the number of annual escalation allowed, a significant saving can be earned.

Organization wise, it is very important how to build up the maintenance organization. Usually, they are separated from development in large projects, although they should work together with the design very strongly. The development delivery strategy will define the milestones when different product versions go to maintenance. If the release achieved, maintenance take over the handling of the product. They should be in sync to be able to

prepare the maintenance structure and resources for the support. Without the product won't survive.

I would like to highlight some important aspects, regarding maintenance, of the project I have been working in and where my examples are usually come from. The maintenance is basically a separated department within our project and that divides the whole organization to two parts. There is a constant balancing on resources between them and about who should take the development and maintenance tasks. In the beginning, there was a rotation in team roles. Maintenance work packages were rotated over the teams, so they received maintenance tasks time to time between feature developments. Teams were cross functional and they had to aware of the whole system. After a while, when trends showed that we are losing capacity to keep the common knowledge on teams, the code were restructured to components and features with appointed responsible. A big problem with this separation in a big system is that, in case of a new feature or a fault, the fix can be done in several ways. Certainly, in an overloaded situation with thousands of tasks in the project and hundreds in the components backlog, every subsystem responsible wanted to get rid of those tasks had a chance to be fixed in another component. Ping-Pong was started. In trouble report (TR) handling, the situation was the same. The occurrences of handing a TR over were tripled instantly. But the chaos was just started.

After the restructuring and foundation of the components and subsystem area teams (SAT), the delivery process has tried to be changed towards one track from the waterfall model. Before, there were several development and maintenance tracks in line with the company's delivery strategy, then it was said to transform that to one track 2-3 years ago. That is still not completed yet, however it is considered done for now, we still have several maintenance and development track. So, people have started to call it "one-tracks" lately.

Maintenance had a problem with the resources in this phase, as the dedicated teams of PLM have become SAT or component teams. TRs, as main PLM tasks, were dispatched to the related responsible teams instead of one PLM team. Coordination became more important as several teams started to deliver to the same PLM package individually. In

addition, a new responsible team was required for delivering PLM packages and guarding all deliveries from both PLM and development side.

The TR handling is interweaves the organization. The updating of TR handling process and TR ways of working (WOW) is a continuous work in large scale. There will always be a changing process or restructuring, reorganizing issue in progress. The number of stakeholders, customers, steering groups, communication channels between them are makes the whole system very heavy and slow in reaction. The first think is usually blamed for any problem is the TR WOW, although the problem is more on the big and cumbersome organization in most cases. Organizational changes should be reconsidered as well in order to speed-up the process. Changing the ways of working which going through the whole organization is a tough work. The advice was mentioned before is standing here as well: try to involve everybody to see their suggestions, but for the review and decision meeting where the agreement has to be made, keep the number of decision makers as low as possible.

Customers need special attention from the maintenance of the company. In a large scale project, where the number of customers are significant, this attention should be provided on the same level for every one of them as they would be a single customer of the company. To achieve this, a good way could be dedicating responsible for each. In large scale even from program management level, there should be a contact person for customers in each layer upwards in the organizational chart. These persons help out the PLM management and put special attention to the problems received form the customer. Naturally there are drawbacks in solution. For example, when there are several responsible trying to push issues from their customer at the same time, there won't be enough priority level can distinguish between the problems. However, a good priority list is mandatory. To achieve this, the customer responsible persons, together with the PLM management, should synchronize between the problems and give them to the teams in an exact priority order. The growing of the complexity of this synchronization overhead is exponential to the number of customers, so it cannot be handled in this way for more than 20-30 customers.

Last but not least, I would like to mention the administration which is a big part of the TR handling in maintenance. This is something that programmers don't like to do but they cannot avoid. In the beginning of our project, main parts of the administration were centralized to reduce the overhead from the programmers. They just did the mandatory steps and the rest was taken by admins. After a while, it cannot be handled by the management or an administrator, simply because the amount of that exceeded the limit they could handle. So, we have tried to distribute some small additional parts down to the teams. In this way, the overhead is spread and takes a little time from every team rather than to allocate more and more resource to handle it centrally. The latter is usually not an option after a certain size of the administration group, as projects do not like to allocate and spend resources for pure administration. Certainly, there must be some control and follow up remain in the management's and coordinator's hand. Communication was very important here convincing the teams to take this unwanted task without major outcry. We have managed this with the trick to involve the teams into the process planning as well. They felt that their opinions are listened and are built in to the process that made them satisfied.

5 Summary

The purpose of this study was to go through of different project parts and projects' methodologies to scale on them from different aspects. Focusing on big projects, examine on advantages and disadvantages of different techniques. Give suggestions for possible solutions for projects from different sizes in different situations. Also, how to improve and what possibilities we have in large scale in various parts of the project.

We have seen the today's view and trends of the project complexity and the different problems it could cause in the beginning of the thesis. We have defined a base project, where the complexity is shown and I have made and presented different measurements with analysis on them. We have seen, how a complex system is growing over 10 years and some interesting measurement and analysis of their outcomes through the figures and their trends.

I highlighted several milestones of the IT project management which helped us to see how old the problem is and how it was developed. With help of the definition of different scalabilities, we went through of different project parts in a structured way, was given by the 12207:2008 ISO [2] standard.

From the organization through the project towards the engineering, we have seen a lot of problem I have broken down to the tree basic project size: small (1-50), medium (50-500) and large (500+).

The study achieved its tasks, as it gives a good view on many problems, through several samples, that we are experiencing in large scale projects in terms of complexity. The complexity, which is growing unstoppable and continuous, that we would like to handle and keep in a manageable size.

6 References

[1] Michael Bloch, Sven Blumberg, Jürgen Laartz: Delivering large-scale IT projects on time, on budget, and on value, article, 2012 Oct., link: http://www.mckinsey.com/business-functions/business-technology/our-insights/delivering-large-scale-it-projects-on-time-on-budget-and-on-value (2016.02.03)

[2] ISO/IEC: Systems and software engineering - Software life cycle processes, ISO/IEC 12207:2008, Patent, link: http://www.iso.org/

[3] Wikipedia: Definition of Scalability, definition, link: https://en.wikipedia.org/wiki/Scalability (2016.02.12)

[4] Hina, Hannan, Mukhtar, Sameer, Fahim: Systematic Literature Review of Agile Scalability for Large Scale Projects (IJACSA) Vol.6, No. 9, essay, link: www.ijacsa.thesai.org, 2015

[5] Sharlissa Moore and R. F.Shangraw, Jr.: Managing Risk and Uncertainty in Large-Scale University Research Projects, Research Management Review, Volume 18, Number 2, Fall 2011, link: http://cspo.org/legacy/library/1201271207F59652378FR_lib_v18n2Moore.pdf, (2016.02.17)

[6] Donald Lessard, Roger Miller: Understanding and Managing Risks in Large Engineering Projects, Sloan Working Paper 4214-01, October 2001, link: https://www.wiwi.uni-muenster.de, (2016.02.19)

[7] Christer Carlsson, Robert Fuller, Peter Majlender: A fuzzy real options model for R&D project, essay, 2005, Link: http://uni-obuda.hu/users/fuller.robert/ifsa05.pdf (2016.02.19)

[8] Management System International: Scaling Up Management Framework and Toolkit, article, link: http://www.msiworldwide.com/approach/tools/scaling-up-framework-toolkit/, (2016.02.19)

[9] K. Szabados, A. Kovacs Developing and Testing at Large Scale, Conference Paper, November 2015, link: https://www.researchgate.net/publication/285025694_Developing_and_Testing_at_Large_Scale

[10] Ipek Ozkaya, Michael Gagliardi, Robert L. Nord, Architecting for Large Scale Agile Software Development, Cross-Talk May/June 2013, link: http://resources.sei.cmu.edu/asset_files/Article/2013_101_001_87865.pdf

[11] Craig Larman, Agile & Iterative Development: A Manager's Guide. Addison-Wesley, 2003, [368], ISBN-13: 007-6092021711

[12] Craig Larman, Bas Vodde: Scaling Agile Development, CrossTalk May/June 2013,link:http://static1.1.sqspcdn.com/static/f/702523/22609354/1367558447003/201305-Larman.pdf?token=29DrpOaTQAkMSJUFUNa%2BAMYadOo%3D , (2016.03.10)

[13] Bill Meacham: Estimating Software Projects, web share, 2013, link: http://www.bmeacham.com/Estimating/Estimating.pdf, (2016.03.12)

[14] Huggy Rao, Robert I. Sutton, Scaling Up Excellence: Getting to More Without Settling for Less, Crown Business, 2014, [368], ISBN 978-0-385-34702-0

[15] Eric Jorgenson, How to Manage Scale, and Operate in Scaling Organizations, Article, Aug 3, 2015, link: https://medium.com/evergreen-business-weekly/how-to-manage-scale-and-operate-in-scaling-organizations-55cd26e382f7#.99u6qyety, (2016.03.13)

[16] Jennifer Baljko, Betting on Analytics as Supply Chain's Next Big Thing, post from 2013.03.05, link: http://www.ebnonline.com/author.asp?section_id=1061&doc_id=262988&itc=velocity_ticker, (2016.03.15)

[17] ETSI, Insider insights, The Change Management Life Cycle, article, 2015, link:http://www.esi-intl.co.uk/resource_centre/white_papers/the%20change%20management%20life%20cycle.pdf, (2016.03.19)

[18] Rackspace Support, Moving your Infrastructure to the Cloud, article, last updated 2014.07.15, link: https://support.rackspace.com/white-paper/moving-your-infrastructure-to-the-cloud-how-to-maximize-benefits-and-avoid-pitfalls/, (2016.03.19)

[19] TechTarget, Definition of PPM (project and portfolio management), article, 2015 April, link: http://searchcio.techtarget.com/definition/PPM-project-and-portfolio-management, (2016.03.22)

[20] Todd Datz, Portfolio Management Done Right, article, 2003 May 1., link: http://www.cio.com/article/2440051/it-organization/portfolio-management-done-right.html, (2016.03.22)

[21] PMI, Systems Evolution, Inc. (SEI) The Natural Evolution of Project Management, Virtual Library, article, 2011, link: http://www.pmi.org/learning/knowledge-shelf/~/media/pdf/knowledge-shelf/systems-evolution_2011.ashx, (2016.03.22)

[22] Lee Merkhofer Consulting, Best-Practice Project Portfolio Management, article, 2015, link: http://www.prioritysystem.com/reasonsabstract.html, (2016.03.28)

[23] Wrike Enterprise, Built for the Enterprise – from teams of five to companies of 50,000, website, Link: https://www.wrike.com/enterprise-project-management/, (2016.03.28)

[24] Unconvention 2016, Scaling-Up Without Screwing Up: 7 Tips On How To Scale, article, May 18. 2015, Link: http://unconvention.eu/scale-without-screwing-up/, (2016.03.28)

[25] Strategic Acquisition, Growth by Acquisition, assay, year is unknown, link: http://www.strategic-acquisitions.com/assay.html, (2016.03.28)

[26] Takeuchi and Nonaka, The New New Product Development Game, article, Harvard Business Review, 86116:137–146, 1986, Link: https://hbr.org/1986/01/the-new-new-product-development-game, (2016.04.15)

[27] Wikipedia: Earned Value Management, definition, Link: https://en.wikipedia.org/wiki/Earned_value_management, (2016.04.15)

[28] Dr. Winston W. Royce, Managing the development of large software systems, IEEE WESCON, preprinted in 1970, published in 1987, [11] ISBN:0-89791-216-0

[29] Martin Flower, Using an Agile Software Process with Offshore Development, article, first published in 2003, Link: http://www.martinfowler.com/articles/agileOffshore.html, (2016.04.23)

[30] Dean Leffingwell, SAFe, webpage, Link: http://scaledagileframework.com/, (2016.04.24)

[31] Tom Looy, The Great Wall-Scaling Agile on a Multi-Team Project, Agile Atlas, Commentary, 2013, Link: http://agileatlas.org/articles/item/the-great-wall-scaling-agile-on-a-multi-team-project, (2016.04.24)

[32] Dean Leiffingwell, Scaling Software Agility, Addison-Wesley, 2007, [384], ISBN 0-321-45819-2

[33] Gene Amdahl, Amdahl's law, computer conference in 1967, Wikipedia Link: https://en.wikipedia.org/wiki/Amdahl's_law, (2016.05.02)

[34] John L. Gustafson, Reevaluating Amdahl's Law, Article from 1988, Wikipedia Link: https://en.wikipedia.org/wiki/Gustafson's_law, (2016.05.02)

[35] Software Verification, Definition, Wikipedia Link: https://en.wikipedia.org/wiki/Software_verification, (2016.05.02)

[36] Facebook Inc., Moving FAst with Software Verification, NASA Formal Method Symposium, February 10, 2015, [9], Link: https://scontent-arn2-1.xx.fbcdn.net/t39.2365-6/10935986_985284008163608_743666691_n/Moving_Fast_with_Software_Verification.pdf, (2016.05.02)

[37] Melvin E. Conway, How Do Committees Invent?, article, first cited in 1968, Link: http://www.melconway.com/research/committees.html, (2016.05.04)

[38] Lee Copeland, TechWell, Mismeasure of software, keynote presentation, HUSTEF, 2015, Link: http://www.hustef.hu/hustef/presenters/DA47ADC6A0D45300C1257EAD005EAC19.html, (2016.05.17)

[39] Wikipedia: Code smell, definition, Link: https://en.wikipedia.org/wiki/Code_smell, (2016.05.07)

[40] Ericsson: TITAN TTCN-3 toolset, open source test toolset via Eclipse Foundation, since 2014, Link: http://www.ttcn-3.org/index.php/tools/tools-noncom/112-non-comm-titan, (2016.05.08)

[41] Reddy Feggins, Scaled Agile – Waterfall, Iterative, Hybrid. Why not go full Agile?, article, Aug 3 2014, Link: https://www.ibm.com/developerworks/community/blogs/c914709e-8097-4537-92ef-8982fc416138/entry/scaled_agile_hybrid_approach_to_devops_projects_why_not_go_full_agile?lang=en, (2016.05.10)

[42] Rajiv D., Sandra A., Project size and software maintenance productivity, conference essay, 1994, [11] Link: http://astro.temple.edu/~banker/Conferences/ICIS1994.pdf, (2016.05.10)