Eskindir Abdela

Agile Life-cycle And Innovation Management For Self-Organized Teams

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

Date : 1 May 2014

Author(s) Title Number of Pages Date	Eskindir Abdela Agile life-cycle and innovation management for self-organized teams 48 pages + 1 appendix 1 May 2014
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software engineering
Instructor(s)	Kari Aaltonen, Principal lecturer

The main goal of the thesis is to show how a group of motivated students can develop an innovative solution. The thesis gives a detailed view into different techniques of team organization, theory and applicability of Agile software development model, leadership trends and innovation fostering mechanisms to show that when properly implemented, a small team can have the potential to produce a product worthy of competing in the global market.

The thesis first covers the previously accepted software development models and perspectives and dives deep into the modern school of thought in regards to software development. It first discusses the theory behind the Agile development philosophy. After that it gives a detail process overview of one of the most popular Agile frameworks – Scrum. Then the thesis goes further to see how Agile models help in self-organization and also leadership models and creative management that best suit these development processes. In addition to that, the thesis covers innovation methodologies that a team employs for identifying ideas that make a difference. It covers some of the accepted models that transform an idea to a product.

The thesis focuses on two projects conducted for the Microsoft Imagine Cup. It shows how the team implemented the theory and the application they have managed to produce. In addition, to earning second prize for one of the project, the thesis also discusses the overall success of the projects.

Keywords	Agile, Scrum, innovation management, leadership

Helsinki Metropolia University of Applied Sciences

Contents

1 Introduction

Communication changes everything and it certainly does when our current reality is concerned. It only takes a couple of seconds with Google and the veil of mystery is lifted from our eyes as all information unfolds in a logical manner befitting our expectation. Nothing seems farther from reach in this age where information glides with lightning speed from one corner of the globe to another. One does not even need to sit behind the desk to get work done. Commuter trains and cafes are fast becoming places of work and exchange owing to the enormous leap in hand held technology.

The availability of information and the ability to communicate with lighting speeds has resulted in exponential growth in innovation all around the world. It has not only fuelled innovation but has given a whole new landscape to the perception and ways people collaborate to innovate. Teams need not be working in adjacent cubical workspaces anymore. They can brainstorm and develop systems thousands of miles apart. Such a working atmosphere has benefited highly from the diverse perspectives of individuals coming from different background, all pooling their resources for a common goal.

As the number of applications and their complexities grew, so did the evolution of processes, beliefs and principles that were used to develop these systems. Even though Darwin was not postulating about technology, his theories would hold true in some manner to the ways we have evolved processes used to develop systems and managing teams. The approach that a team leader uses to inspire confidence have gradually improved in time to encompass newer circumstances as teams are becoming more and more virtual and multi-cultural. Additionally, with the ever growing competitiveness it has become utterly important for a team to adopt development principles that foster continuous improvement, retrospection and efficiency.

The cost of turning an idea into a product has dramatically dropped down, especially with the advent of cloud technology and application-stores. Today, it is possible that a team of few members with good idea, having the right tools and the correct knowledge of development principles comes up with an innovative solution worthy of challenging corporate giants with minimal cost.

Goal

The goal of this thesis is to show how a team of self-organized programmers can brainstorm, innovate and efficiently use their time and talent to produce a solution that not only solves an actual problem but also has a potential of success in the market. The team members use modern principles in software development, innovation, team organization and leadership to efficiently pool their resources for a successful completion of their project. The thesis revolves around two projects conducted for Microsoft Imagine cup.

2 Software development models and perceptions

Software has evolved over the past decades to give us luxuries which we casually take for granted. However, the evolution has not been a smooth road rather it has been journey of a continued learning experience and active retrospection.

In the early years of software, great minds of the day had to device a way to address complexity that was fast accompanying softwares. In 1960, Dijkstra presented a paper on structured programming which conveyed methods of software development that addressed complexity by successively breaking it down into abstract layers [1, 47]. According to him, several layers are organized in sequence in such a way that on layer is on top of the other and in a manner in which the bottom layer acts like a virtual machine to the one on top. His theories lay the foundation of today's internet protocols and multi-tiered architectures. He also argued that the way to produce quality software was not by spending time and debugging it in order to track bugs but rather to make sure that bugs do not appear in code in the first place. His methodology of writing a specification to make a certain case pass before writing the actual code is a forerunner to today's popular test-driven development. [1, 47-50]

Even though hierarchical abstractions had addressed complexity, there was one more problem to software development. Maintenance cost of software got higher and higher. David Parnas argued that while Dijkstras' philosophy still holds, he introduced the principle of information hiding whereby, the program flow was decomposed into modules which would render the system easy to maintain. His principles also put forward on how the complexity of softwares can be further simplified by designing modules in such a way that they hide their inner design from other modules and only sharing interfaces with each other. Consequently, Simula, the first Object-Oriented programming software, was developed by the Norwegian computer center in Oslo followed by others soon followed. [1, 50-58]

So far in its history, programming and software development were seen as one and the same. However, in the course of consequent decades, the life-cycle concept had been born and programming had become one part of a larger process. As maintenance costs started to dominate software project, the need for even a better streamlined process become more evident. An IBM publication by Michael Fagan in 1970 shed light on how much effort was spent for non-value adding processes, more than 50%. The paper put forward an agreement to Djikstra's argument of finding errors as early as possible.

However, Fagan’s argument as to how to tackle these issues was different. He perceived and thus elaborated software development as a three-stage process, where each process had a unique responsibility, input and output. Additionally, each system had a set of acceptance criteria by which its completion is validated against. The three staged processes were namely “Design, Coding and Testing”. His ideas propagated the thinking that software could be developed as a series of frames. Barry Boehm took Fagan’s view, improved it and hence the popular waterfall development model was born. [1, 53-54]

The Waterfall model is a plan-driven development model where software development would have different stages as illustrated in figure 1. Analyzing the requirement, designing, implementing etc. are all different stages of development in this model. [4, 1]

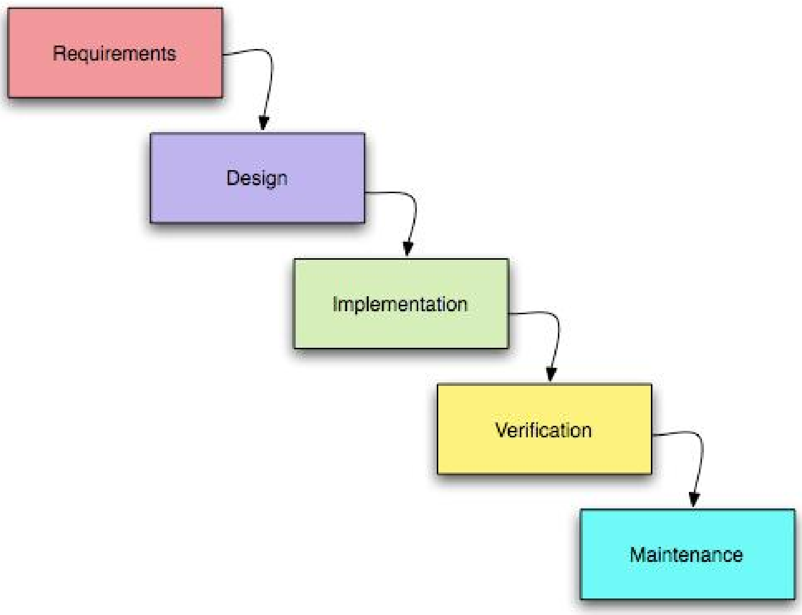


Figure.1 Waterfall software development model [21]

The waterfall model contains several stages which are described as follows:

1. Requirement gathering: This is the stage where requirements for the system in question are collected. This process can be done in several ways including one-to-one conversations with the stakeholders.
2. Design: The requirements which have been previously collected are analyzed in a systematic way. Technical requirements of the system to be built is analyzed on both the hardware and software level. In addition, the technical architecture of the system designed in this phase. This phase also includes the performance

considerations, security parameters and data modeling of the system. Additionally, the UI (user-interface) of the system is also designed in this phase. Upon the end of this phase, the development team has a documentation that would serve as a functional specification for implementing the system.

3. Implementation: This is the stage in the life-cycle where the software is actually coded. The output of this stage is an application built according to the functional specification.
4. Verification: In this phase of the development the built system is tested thoroughly. Testing teams will develop test cases and using those they verify the systems functionality as a whole as well the various components. This phase includes unit testing, integration testing and acceptance testing.
5. Deployment: The system is deployed to its respective target. After this point the maintenance phase continues. [2][3]

One of the main features that contributed to the waterfall model's popularity is the simplicity of the system. The linear development model it employed was easy to understand and it was implemented in several software projects. [22]

One of the main disadvantages that became evident for the Waterfall model was the fact that the customer would have no idea of the product until the very last stages of the cycle. In the real world requirements change often and this model has proven to be rigid for that kind of change. The Waterfall model is by its nature plan-driven and assumes software shares the same characteristics as other engineering fields such as construction where requirements remain the same throughout the development stage. On the contrary, software requirements are very dynamic in nature and it is hard to put a cork on any list of requirements. In addition to that, it is impossible to gather all requirements in the project. Also, the waterfall model does not deliver value to the customer until the very last stage of its development. [4, 2]

Plan-driven methods like the waterfall take several steps to remove the unknowns from the system. Each stage dedicates its resources to resolve any unpredictability that might arise in the development cycle. Because of this, the model does not have the appropriate contingency when unforeseen issues occur in the development. It strives to remove the unknowns and reach a fixed time for delivery. Usually in the real world software will have those issues and the only way the model can address these issues is by using the over-estimated time. The over-estimate is a buffer that is estimated very early in the project and its purpose is to be used to fix unforeseen issues. [4, 2]

The waterfall model has many stages whose main output is documentation. This documentation is intended to give the teams an understanding of the system. However, too much documentation is as bad as having too little documentation. Since the documentation is produced early in the development cycle, lots of things change while nearing the end. By this time synching the document with code is an additional task that will add no value to the customer and can be considered a waste. [5, 6]

Contrary to the staged development like the waterfall model, another school of thought had tried to visualize software development from a new angle. Hence the concept of iterative development was born. The Iterative model is value-driven at its core and it aims delivering it as much value and as soon as possible. It is divergent from previous models both in philosophy and process. Process-wise, it depends on frequent interaction between stakeholders and internal retrospectives to perfect its deliveries. It needs the frequent interaction in order to address the dynamic nature of requirements in software development. In addition to that, the process intends to bring about a quick ROI (return on investment). [4, 3-4]

The past had been more familiar with putting all the unknowns of a project with early stage requirement analysis and design. However, the new iterative concept is more flexible or Agile and it deals with specific chunk of the project at a given time. By doing so, it focuses on small parts of the project and delivering those parts as soon as they are finished. This way, the customer gets a value of its desired product early on and will start to use it and give their feedback which would be essential to the team. The parts of the project that a team does at a time are mainly list of features divided by priority. One advantage of this method is that the team's efficiency in estimating smaller parts of the software gets better as the project goes and there will be fewer unknowns. Retrospection and continuous improvement is at the heart of this flexible or popularly known as Agile methodology. With each delivery, retrospection is held and a team gets to learn and sharpen their practices within the duration of the project. [4, 3-4]

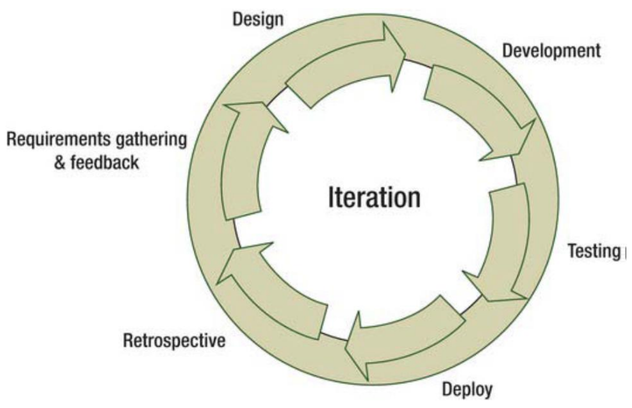


Figure 2. Iterative model [4]

As illustrated in Figure 2, the iterartive model is a cyclical model where each design begins by gathering the requirement of a feature and ends with retrospection. The learned points gathered from the retrospection are going to be used in sucessive cycles shareping the product quality.

3 Agile software development

Early in 2001, a group of well known industry experts going by the name of "Agile Alliance" convened and arrived on a concensus on a set of principles on Agile software development which is popularly known as the "Manifesto for Agile Software development". The manifesto was the result of their motivation to remedy the problems of processes related to software development.

Manifesto for Agile Software development

" *We are uncovering better ways of developing software by doing it and helping others do it.*

Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotitation

Responding to change over following a plan" [6, 4]

This manifesto was thus reached after the industry experts examined the problems associated with the processes in software development thus far. The manifesto holds between their lines a comprehensive philosophy that has become the popular Agile methodology.

Individuals and interactions over processes and tools

The Agile mentality is people-centric and it argues that people are more important to a project than a set of processes. Processes might end up making the best programmer not as efficient as he/she could possibly be. This can also extend to the team. Instead, Agile relies on strong team players as a vital resource for the success of the project. A strong player is someone who could be an average progrmammer but a good communicator and works well with others. Working well with others should boldly be focused. In addition to that, a team should not be using a tool just because it is complex and has the capability to perform several tasks. A team should grow into the tools rather than be forced to use them in the first place. Starting from a piece of paper, a team should demostrate the need and grow into tools that manage complex issues. The idea is to build a team rather than the enviornment and force the team to adopt. [5, 4]

Working software over comprehensive documentation

The Software code is not the best way to communicate it is purpose even among developers. Its essential to document software. However, having a big documentation is also another source for a different kind of problem. Producing and continously synchronized with the changes in code is very time consuming. If not kept in synchronized they become misleading and no one will be reading them. Instead, the idea is to keep documentation as minimum as possible. The documentation should only describe the higher level architechure and concept behind the software. In answering how the code details be described, say for a new team member, is where the human-to-human interaction comes into place. Agile argues that the truth behind any code is best transmitted by a salient documentation and another tem member passing on the information. It is much more comprensive that making a new team member stuck with hundered pages of possibly poorly synchronized documentation. [5, 6]

Customer collaboration over contract negotitation

The idea that software can be viewed as an orderable item with a fixed set of requiremnents has in most cases resulted in failures. In order to have a successful software project it is imperative to have the active participation of the customer. The customers can give feedback as they see the idea unfold into something tangible as their understanding of the system grows. Contracts having elements such as time, schedule and cost of delivery are inherently wrong from the very beginning. Instead, the best contracts are those which focus on how to govern the relationship of the development team and the customer. [5, 6-7]

Responding to change over following a plan

Since the requirements for a software project are quite dynamic in nature, planning way ahead until the project close will likely end up in rework and revision. Agile methodologies propose that the development team requires detail information only for the immediate task at hand. [5, 4-8]

3.1 Agile principles

Agile methodologies has a set of core principles that enable it to successfully deliver projects with a reasonable schedule and high quality. The principles can be broadly categorized into six different areas.

The first of those principles revolves around ‘Variability and uncertainty’. Previous models like the waterfall assume software development as a manufacturing process, where all the inputs are known and all the outputs are precisley predicted whereas, Agile principles see development synonymous with creating a recipe. The product or feature is created only once and further development means more variation is introduced on the already created feature.

Since Agile, as the name suggests is flexible and does not intend to conquer all the unknowns in the planning phase, successful Agile developers understand that they would not get everything running perfectly the first trial. Instead, there is an acceptance for uncertainty and a learning curve within every project and that the product will continue to grow in an iterative cycle. Iterative cycles mean successive revision and rework of a feature until the desired functionality is attained. In additon to the iterative cycles, Agile methodologies also embrace the idea of incremental development. Incremental development simply means building the product into usable brick by brick. Instead of waiting for the whole product to finish, stakeholders like customers can have certain feature delivered to them as soon as they are finished and they can start working with them until other features are delivered. In this way the customer gets to touch and feel the product very early in the project and is able to give useful feedback for the development so that successive features will benefit from this feedback. In additon to that, the customers can have value delivered to them early so that they can start albeit in a limited scope, to use the product. This kind of repetitive cycles are useful to enhance the learning of the team as well as the helping scope the project in a desirable way.

In order for this procedure to be successful, it is essential to carefully design the cycles. Each cycle should produce a feature rather than a phase in the project. Additionally, each cycle should have all the elements of software development, i.e plan, implement, test and deploy. Here, we can plan carefully since the set of features which are expected at the output are wellknown for this cycle. At the end of each cycle there is a period retrosection and successive cycles will continue.

One pitfall to this methodology which is also known as scrumfall is when developers use this cycles to do phases in the project. For instance, cycle 1 could be planning, cycle 2 could be design etc. [6, 29-35]

3.2 Inspection, adaptation and transparency

As can be seen in the Figure.2 there is a period for retrospection. This is one of the core ideas of Agile that sets it apart from other development models. In this phase the stakeholders make a retrospective both on the set of features developed and the processes that were put in place to develop them. The customer tests the delivered feature and fives the appropriate feedback. Feedback might be with the current features or it may also concern reprioitization of other unstarted features as the customer has a better idea of the product. In addition, the development team reflects on their effort and plans on improving the later cycles. In order for there to be a successful end to the project, Agile encourages the flow of information in a transparent manner. This is one of the reasons where communication is at the heart of Agile.

Every development faces some levels of uncertainty. Some projects face uncertainty in terms of what tools and technologies to use in order to deliver the product. some may face uncertainty in discerning the features for the product. Some may face uncertainty with regards to who the actual customers are. Such an uncertainty is quite common with start-ups and novelty products. Instead of addressing each uncertainty discretly, Agile faces them simultaneously with having an integrated perspective. The justification for this approach is that a discrete solution for one may have an adverse effect on the other. [6, 35-37]

3.3 Prediction and adaptation

Agile methodologies are not against making predictions on a project. However, the way predictions and decisions are made is a bit different from plan-driven processes. Plan-driven processses are familiar with making decisions early in the design phase when the collective knowledge of the system to be built is not yet mature. On the other hand, Agile principles dictate that the team should delay making decisions until indecision becomes more costly as illustrated in figure 3. The reason for this is to increase the probability of making an informed decision while not incurring extra cost due to indecision.

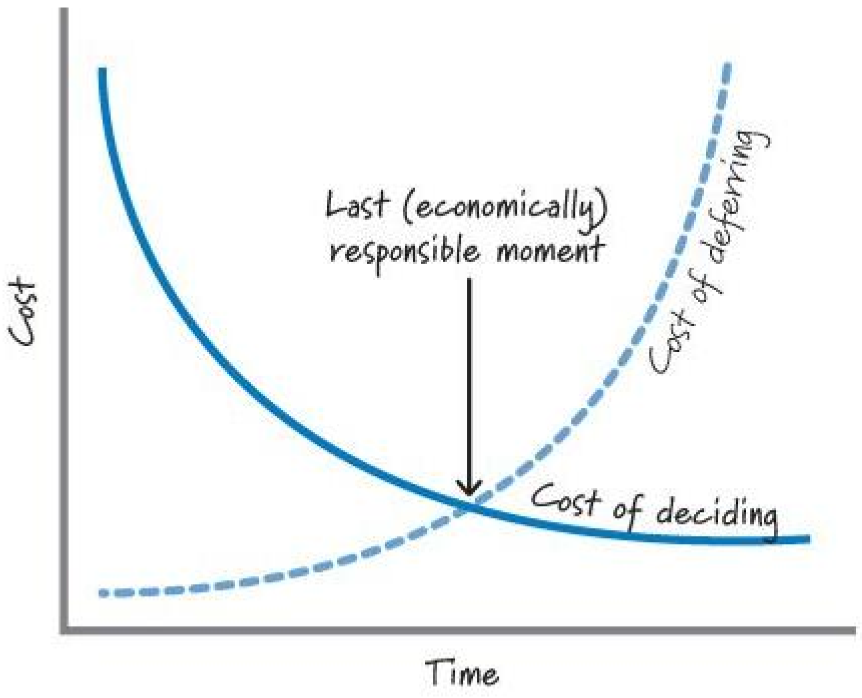


Figure. 3 Decision curve on Agile development [6]

Exploration is also one aspect of Agile methodologies. When faced with uncertaintity Agile favors that the team to engage in developing an exploratory approach inorder to increasing the knowledge of the team about the uncertainty they face. This might include building a prototype.

Agile methodologies accept the inherent character of change in software development. When a change in a requirements happen, the cost of making the change will become significantly higher as we move furhter down the phases of a plan-driven processes like the waterfall.

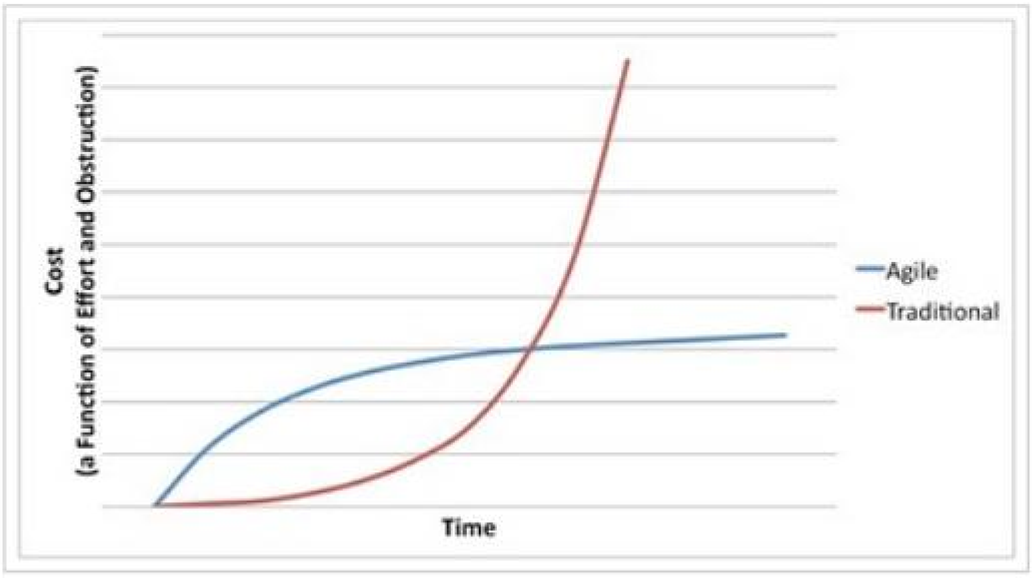


Figure 4. Comparison of agile and traditional development cost function [23]

In order to avoid cost over-runs, plan-driven processes actually spend more resources in trying to reduce possible change in the future. Ironically, such an additional effort only piles the time wasted and the project suffers from an untimley schedule and over-budget. The reason for this is that those activities which are intended to avoid change are based on assumptions rather than feedback from stakeholders. However, Agile embraces this change and has characteristics which will handle this change in an economically saavy manner. Agile methodologies use the principles of JIT (Just-in-time). They instruct the team to create as little software artefacts, such as design documents, as possible. In this way, as illustrated in figure 4, when a change does happen, there will be much less to discard and hence the project suffers a relativley small financial and schedule risk.

Balancing prediction and adaptation to change is very esential in every project. Even though Agile understands that change will come, it does not mean that it will not have any plans upfront at all. Instead, it will plan the project in a way that will faciliate the adaptation when a change comes. Agile will have an overview of the software in question and details for the next successive development cycle. [6, 37-44]

3.4 Validated Learning

Validated learning in Agile is defined as a knowledge that is acquired after carefully testing the assumption. Unlike plan-driven processes Agile development takes is much less tolerant of long-lived assumptions and it does its best to convert them into vaildated learning. This is done as iterated above by iterative and incremental development and low cost exploration.

If an assumption is validated further down the project process there is less chance of using this gained knowledge into a fruitful use. In order to have the validated learning give a meaningful outcome to the project, it is essential to have them in cycles after every iteration. In addition to the already mentioned short development and feedback cycles, Agile methodologies like for instance Scrum (see Scrum framework) have daily stand-up where the team shares what they have been doing the past day every day. Therefore, starting from the daily work of individual developers, Agile encourages feedback and learning processes. [6, 44-48]

3.5 Work in Progress (WIP)

Work in progress is defined as work which has already been started but not yet completed. Plan-driven procedures encourage building sets of similar feature altogether. This principle emanates from the manufacturing where batch manufacturing of items significantly reduces costs. However in Agile, the best way to go about developing in small batches at a time. This approach has the following advantages:

- Small batches mean small inventory that is subject to change.
- Small batches are easy to manage
- Small batches increase the frequency of feedback and validate the learning of the process and therby boost the develoment efficiency.
- Small batches have relativley a lower risk incase to the project schedule and budgjet.
- Small batches deliver value more quickly.

- Small batches enhance the motivation of the team and clarity of the information shared.

Not all manufacturing principles are ill-suited for Agile and this statement becomes true when managing inventory is concerned. Manufacturing processes have an excellent way of handling in ventory which, in software development sense is WIP (work-in-progress). A competent manufacturing manager knows the risk associated with the build up of inventory. Inventory sitting for a long a time not only holds a lot of finance but also is prone to going unusable. In software development too much of WIP can also cause the same problem since the enviornment is very dynamic by nature. Agile methodologies handle this by using well known manufacturing process called Just-in-time. This means the WIP contains only the tasks which are open the current development cycle contrary to the plan-driven approaches where the whole product is in WIP mode.

Another issue which is divergent from traditional thinking in Agile development is its perspective on idle work. Agile puts more focus on idle work rather than idle workers. If there is a deveoper with idle work because his/her next task can only be started if a certain other task if finished, then he/she will have idle time. In a traditonal plan-driven enviornment this developer will be then assigned to another task/project so as to have more billable work. However, Agile focuses more on the idle work rather than the idle time. Instead of preoccupying the developer with other tasks Agile trues to solve the potential bottlenecks so that the develoer can continue with his work and there is as little idle work left on the table. The justification for this diivergent view comes from repeated observation of cost implicaions of idle work vs idle time. This is because when trying to achieve 100% efficiency in work rate it is seldom noticed that the efficienccy of the worker gets lower and lower and therby exponentially adding the idle work which will end p costly for the company.

Progress in Agile is measured, as shown in Figure 5, by the delivered value rather than the schedule or time since the project has started. The delivered value are set of features which have been fine tuned by validated learning and are those which actually deliver deisred functionality. In contrast, plan-driven processes rely on the plan to measure progress and as such might deem a product according to plan if it fullfils the predicted assumptions even though the customer feels that is not really what they are looking for[6, 48-54].

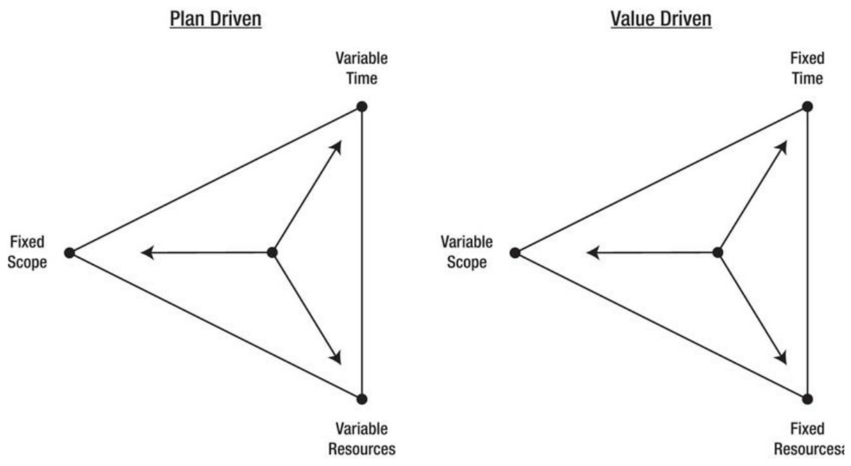


Fig 5. Driving forces in waterfall and Agile methodologies [4]

And hence Agile can be classified as a value-driven process. Every project has three elements. Time, resource and scope. It is only possible to work with either two and their outcome is seen on the third element [4, 15].

3.6 Performance and quality

Speed is an important factor in software development and Agile is no different in that sense. However, it has particular views when it comes to speed. Of course it is splendid to deliver features fast, get feedback and move on with the next cycle with minimal delay and iterativley and incrementally build the desired product. However, speed should not be misunderstood for hurrying. Being in a state of to deliver a feature will wear out the development team in the long run and efficiency will be hurt. In addition to that, hurrying a feature to be delivered will end up with the delivery of poor quality at the end of the cyle.Additionally, it will also decreases the motivation of the team to achieve more.

In plan-driven procedures the quality of a given product will not be known until the last stages of development when testing is actualy done. If this development specially includes simultaneously developed components which are meant to be integrated at thelast stages, and it is found that there is lack in quality, this might unleah an unpredictably long test-and-fix-cycle. However, in Agile development, quality is built within and is finetuned with every deivery and the team does not need to wait to the end of the project to see the verdict of qulaity. [6, 56-57]

3.7 Focus on team

Agile puts people in the center rather than process or documentation. This not only means there is transparency in the information flow but also has fundamental doctrine on the structure of the team. In Agile, teams are meant to be self organised. Responsibility is built-in to the team by means of motivation and other mechanisms(see Leading Agile teams). This means that responsibility for a certain task does not come from a higher command as it is traditional with plan-driven processesm, instead it is communicated across the team and decides the best possible path for a particular scenario. In addition, to that the whole team participates on the whole project. Each individual is free to input his knowhow on a particular subject. There is no team member who is actually responsible for a given type of task. The direct advantage of this is that knowledge and competence grows as a team. It has been observed that wuality architectures and designs emerge from self-organizing teams. [5, 8-10]

3.8 Simplicity

Agile development mentality agrees with the beauty of simplicity and it strives to achieve simple solutions for complex problems. Because the dynamic nature of software development is known from the start, solutions are anticiplated to simple too develop and change. [5, 8-10]

There are several variants of Agile development methodologies in use today. All of them adhere to the core principles of Agile however, they differ in implentation. For the project this thesis is written upon, popular variants of Agile, called the Scrum is used. Some of the other Agile practices include:

- FDD (Feature driven development)
- eXtreme Programming
- Crystal. [4, 8]

4 Applying Agile using Scrum framework

An acronym taken from rugby, Scrum is a development framework that applies Agile principles to successfully implement and deliver a software project. Scrum was first published in the Harvard Business Review citing the success companies like Honda and Canon attained by using an approach that was founded on team and scalability. The way Scrum manages a team was a novel idea at the time, "Teams are self-organizing". [6, 3]

- Satisfy the customer: Customers are satisfied as Scrum delivers features that they actually need rather than those that have been part of the specification.
- Better ROI: Delivery of the product in small and frequent batches improve the return on investment.
- Lower costs: Scrum provides better insight to avoid waste and thereby lowering costs.
- Rapid results: Scrum provides rapid results in form of small batches of valuable product delivered as early as possible.
- Confidence: Scrum develops trust among the customers and the development team.
- Fun: One of the main principles of Scrum is to make all the stake holders have fun throughout the process.

Theoretically Scrum is based on empiricism. According to dictionary.com, empiricism is defined as "*the doctrine that all knowledge is derived from sense of experience*" [7]. Indeed, being an Agile process Scrum relies on knowledge rather an assumption to make decisions. It follows the iterative principle of Agile as well as concepts like adaptation, transparency and inspection which have been explained in previous sections [7, 3-4]. In addition to adhering Agile principles Scrum has sets of core principles. According to Kenneth S. Rubin
"*Scrum is refreshingly simple, people-centric framework based on the values of honesty, openness, courage, respect, focus, trust, empowerment and collaboration"* [6, 13]

4.1 Scrum team

Conway’s law states that:

“*Organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations.*”

A corollary to this law argues that a team which takes collective ownership of the code where members actively participate in every kind of task is likely to produce products of higher quality that the structured teams with delineated responsibility. The justification for this reasoning is that Conway’s law focuses on the relationship and communication within the team, the more transparent the communication and relationship the simpler the architecture and composition of the product. [8]

A Scrum team has new roles and responsibilities which it assigns to certain team members. The team is comprised of Scrum master, product owner, development team which also includes the testers.

Product owner is person is in charge of managing the product backlog. The product owner role is given to a particular person and he/she alone can decide in managing the backlog items. It is important for the rest of the team to understand his/her decision. Additionally, since it is the product owner who is accountable to the backlog items it is also his/her duty to make the development team understands the tasks.

Scrum Master is the role which is the closest thing Scrum has to a team leader. It is the responsibility for the person of this role to ensure the team follows the Scrum practice. Even though he is the leader, the leadership style is different from the traditional command and control. It is more like lead by example (See Leading Agile teams). In addition to that this role is responsible for facilitating the interaction between team members and also the outside environment. The scrum master helps the product owner in backlog management. This role is also responsible for organizing and leading Scrum events and practices. This role is also in charge of coordinating efficiency boost as well as coaching teams with the Agile methodologies. The scrum master is not synonymous with the traditional role of project manager. He/she does not have the authority but they are leaders (See section 7).

0

The development team is the group of people who are responsible in delivering quality increments every cycle which will be the final product at the end of the project. The team consists of various professionals. It is self-organizing and does all the planning, implementation and testing together. The team is self-organizing and also takes collective ownership of the implementation. [7, 5-8]

4.2 Planning

One of the important early tasks of software projects is estimating the work load. How much time and/or resources are needed to satisfy the given requirement? In order to come up with a logical and relatively accurate estimate, Scrum implements certain principles. It is fundamentally different from its predecessors which viewed projects as only successive phases. In Scrum as in other Agile frameworks, the knowledge that a team learns while building a certain product is also taken into account.

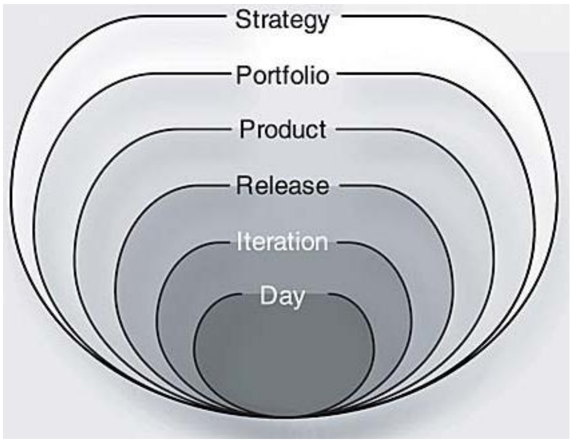


Figure. 6 The Agile Planning onion [9]

There are several scopes of work in a project ranging from daily development, to iteration work to release etc. all the way to strategy as shown in Figure 6. As the scopes grows the level of unknowns decreases increasing the risk of an unrealistic estimation. Scrum only strives to estimate the near deliveries limiting them by the iteration where the level of known requirements in the project is fairly comprehensive. [9, 27-29] However, the techniques of estimating follow a novel approach.

Scrum uses story points to estimate user stories (see section 4.3). There is no hard-line on defining story points, but these points are given to particular user story after analyzing the knowledge needed, complexity and risk associated. It is customary to use Fibonacci series to do the estimation and the idea behind it is that the least difficult task is estimated with a story point 1 and relative to this story the others are estimated. For instance if there is a story which is twice as hard then it is estimated with two. Another important technique involved in Scrum estimation is that it is collaborative. Instead of asking one expert for an estimate on a given story, Scrum encourages the whole team to give an estimate. One reason behind this principle is that Scrum does not assume tasks to individuals in advance and hence it might not be the expert who gets to do the task after all. Additionally, the relative points of team members are also taken into consideration as Scrum estimates by analogy. Another consideration in estimation is the story size. If the story is thought to be big, then it will be broken down into many other stories and these stories may be grouped as a theme. (see section 4.3). [9, 36,51-53].

Velocity is a new concept in Agile development frameworks like Scrum. It is defined as “the rate of progress” [9, 36,51-53]. Empirically, it is the total number of story points delivered in a single iteration. For instance, if 2 user stories, each of 5 and 8 story points respectively delivered in 2 weeks, then the team’s velocity is considered 13. Velocity is used to estimate the team’s capacity, and is based on the historical data of the team, velocities can be estimated. In circumstances where there is no historical data, a team might need to forecast the velocity of a new project based on skill and complexity of the tasks ahead. The main advantage to velocities is that they take away the burden of re-estimating and go hand in hand with dynamic nature of Agile development. For instance, if the team estimates a velocity of 20 story-points/ iteration and deliver 15 story points of work, then they can adjust the estimates for future deliveries with 15 story points. Thus, Scrum estimations avoid the need to re-work the estimations. [9, 38-39]

There is usually an initial constraint to a project. For instance, that constraint may be a date of delivery and a scrum team creates a release plan. A release plan contains the estimated sets of features that are to be delivered or alternatively the time needed to deliver a set of features by using velocity and story points it is possible to release a plan for the project. [9, 133-134]

4.3 User stories

User stories are one way of expressing a feature specification in Scrum. They are lightweight mechanism of building a small set of statements to express the feature requirements. The user stories are understandable by both the development team and the other stakeholders in the project. The level of detail in user stories may differ but user stories are not intended to have all the details, instead they are there as conversation starters. This conversation between the development team the stakeholders will go on throughout the development of the story.

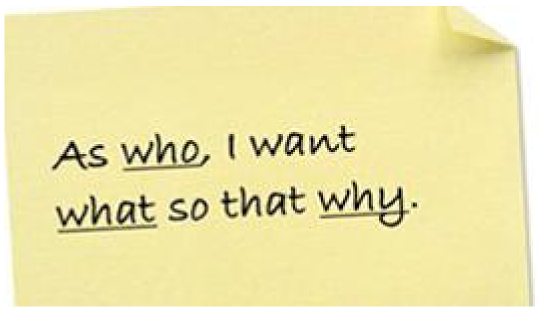


Figure.7 A sample user story for a login [24]

A typical user story specifies the role (who), the function performed (What) and the expected results (why) as shown in Figure 7.Some user stories may come with sets of conditions that must be fulfilled in order that this story is accepted as done. These conditions are often treated as test-cases which the development team confirms the implementations. In development patterns like Test-Driven development (TDD) and Acceptance-Test-Driven development (ATDD), the tests are written first and followed by the piece of code which pass those tests.

User stories are good for a specific iteration. However, they fall short in giving a team the useful higher level insight needed to enable release plan. For this purpose there needs to be a hierarchy in user stories with levels of abstraction. An epic is a user story which is considered too big for a specific iteration. And hence, epics contain fewer details but can include many user stories inside. They give an abstract description of the sets of user stories. Using epics teams can be able to visualize the end product albeit abstractly. [6, 83-88]

4.4 Sprints

The core part of Scrum is the sprint. A sprint is a time-boxed development cycle in Scrum. One month is considered as the maximum duration of a sprint with bi-weekly sprints becoming the norm. During this period the team plans and implements a valuable increment to the product. It is not advisable to change the goal of the sprint once it has started but scope however can negotiated with the help of the product owner. A sprint normally consists:

- Sprint planning
- Daily scrums
- Sprint review
- Sprint retrospection

Sprint planning is the process where the team decides what to deliver in an upcoming sprint. Sprint planning for instance, for a two week sprint should not exceed 4 hours. Sprint planning is organized and coordinated by the scrum-master.

During the sprint planning two core questions are answered. What can be done and how? For the first question, the product backlog, the latest increment if any and teams capacity serve as the inputs for the team. The product owner elaborates on the product backlog (See section 4.9) and the priorities of the items contained within. Based on the complexity of the user stories, the teams known or estimated velocity and capacity the team then decides what to include in the sprint. In regards to the second question the team decides on the technical issues and obstacles in order to convert the sprint items into finished product. The development by nature is self-organized and hence collaborates amongst one another on how to develop the solution.

4.5 Daily Scrum

A daily scrum is another time-boxed event with duration not greater than 15 minutes. During this time the team inspects the work of the past 24 hours and plans the next 24. Each member answers what they have been doing, what they plan to do and what obstacles if any is hindering their work. The daily scrum informs the whole team of the progress of the sprint. Usually the team members will have detailed discussions with each other after the daily scrum to further collaborate on items so as to stay in pace with the sprint goal.

4.6 Sprint review

A sprint review is an event organized and coordinated by the scrum master and held at the end of every sprint. This meeting is time-boxed with duration of 4 hours. During this meeting, the development team, the stakeholders, the product owner and the scrum master are present. The product owner goes through the product backlog, particularly on the done tasks. The team demonstrates on what has been done and also will start to go through the future tasks by going through the backlog items. In addition to that, other issues are also discussed such as over-all deadline, budget or revised scope.

The outcome of this meeting will serve as an input for the upcoming sprint planning. Sprint review serves as the adaption and inspection part of an Agile software development.

4.7 Sprint retrospective

Sprint retrospective is a time-boxed meeting held at the completion of every sprint by the team members. During this meeting, which is also organized by the development team, the team reviews the past sprint from the process perspective. Each team member shares what was particularly good about the last sprint and what was bad. The team, based on this retrospection, focuses on optimizing their efficiency based on what has already been learnt. This is the validated learning part of an Agile development. The process and competence of the team improves every sprint and this reflects positively on the quality of the product [7, 7-11].

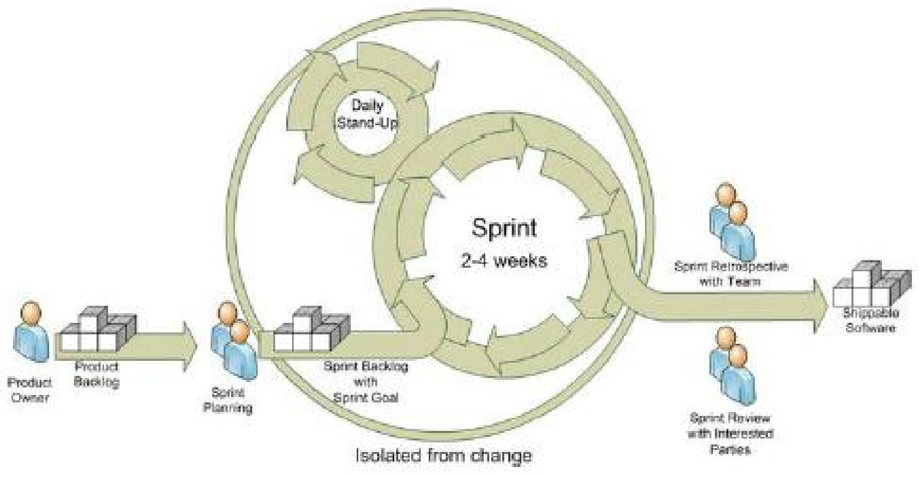


Figure 8. Scrum process [4]

Figure 8 illustrates the cyclical nature of Scrum, the process contained in every cycle and how product backlog items are converted into a shippable product.

4.8 Technical debt

A Technical debt in general means the several problems a software systems faces due to some steps taken by the development team or the other stakeholders. As the financial debt has to be paid, so is technical debt or else it compromises the quality or even the fundamental functionality of the software product. The technical debt can be broadly classified into three main sub-groups.

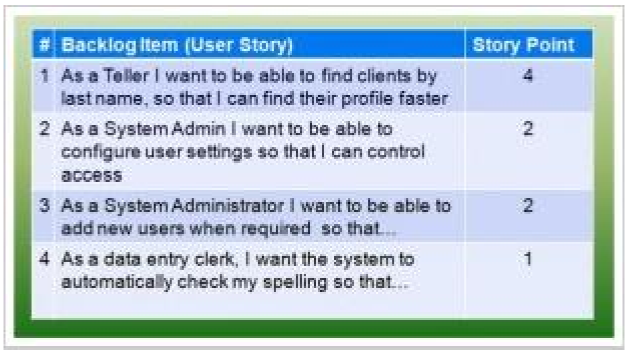
- Naïve debt: This is the technical debt the development team accrues due to the usage of bad development practice. For instance, a badly written code can cause a technical debt in the future when a problem arises from that area. In addition to that, a bad design, not enough test-coverage in the system and poor integration. are some of the reasons for shipping technical debt into the system.
- Unavoidable debt: This is the kind of debt that gets accrued by the change that happens later and compromises the already shipped code. Since this is the kind of debt that reveals itself in time, there is no way of predicting it.
- Strategic debt: This one accrues due to the business decisions. Sometimes the business stakeholders of the software in development might decide to allow some development shortcuts in order to meet a specific deadline.

Mature product owners understand the implication and source of a technical debt and handle it systematically before it becomes damaging. Some of the damages a technical debt causes are increasing the delivery schedule, increasing the number of bugs introduced in the system, rising the cost of production, customer dissatisfaction and decrease in team performance.

In order to avoid technical debts, there are sets of steps that are laid out in the Scrum framework. The first and foremost is to implement the system in using good technical methodologies and best practices. This includes using of well-known design patterns, refactoring the code and also writing automated testing. It is immensely difficult to add a new functionality on poorly implemented software. Refactoring it alone can be a very time consuming task. Secondly, Scrum teams use DOD (Definition of Done) for each task. DOD are a set of criteria that must be met for a task to be deemed complete. Such a practice helps in tightening loose ends from which software bugs might arise. In addition to tightening the screw on development practices, product owners must strive to manage the debt accrued in a proper manner. Not only do they need to manage it but they must be able to make it visible for everyone in a way that also shows the economic costs of these debts. Such a practice would lead to a planned servicing of the system which would positively reflect on the product. [6, 144-162]

4.9 Product backlog

A product backlog is a list of TODO items for the development of a software product. The product backlog is available to all to promote transparency. Each item in the product backlog is a user story for a specific functionality. Since Scrum is an Agile framework, the product backlog is dynamic. As long as the software is undergoing development the product backlog is continuously being updated. A good product portrays DEEP (Detailed, Emergent, Estimated and Prioritized). Each PBI (Product backlog item) should have detailed information about what is needed to accomplish the task. In addition it should also have the estimated work effort needed to produce this functionality as well as the priority of developing this task. The product backlog should always be dynamic which refers to "Emergent". A sample product backlog provides the user stories and their story points as shown in Figure 8.



#	Backlog Item (User Story)	Story Point
1	As a Teller I want to be able to find clients by last name, so that I can find their profile faster	4
2	As a System Admin I want to be able to configure user settings so that I can control access	2
3	As a System Administrator I want to be able to add new users when required so that...	2
4	As a data entry clerk, I want the system to automatically check my spelling so that...	1

Figure.8 A sample product backlog [24]

The process of managing a product backlog so that it portrays DEEP characteristics is called Product backlog grooming. Product backlog grooming is a collaborative that involves the development team, the product owner, the scrum master and other stake holders with the ultimate decision power resting on the product owner. Mature product owners however, emphasize on collaboration to arrive at a DEEP product backlog. [6, 99-105]

5 Innovation Management

Nova is a Latin word for new and it is incorporated in the word innovation which, according to MIT professor Ed Roberts Innovation is “*the embodiment, combination, or synthesis of knowledge in original, relevant, valued new products, processes or services*” [11, 2]. Innovation is not just the introduction of a new idea in a certain area, but it is also exploiting it to a successful end.

Innovation can be broadly classified into two main groups, incremental and radical innovations. Incremental innovations are those which take the advantage of an existing product or service and adding an increment or value to it, whereas radical innovations, is the creation of a new product or service that changes the previously established system. For instance, the jet propulsion engine and the transistor are radical innovations that changed the world. Radical innovations change the established technology and provide a new landscape for business. In industry, rather than being polar opposites both incremental and radical technologies are part of a process. Incremental innovations build upon radical innovations. An Innovation can be in the introduction of a new product or a service or even in the process where we produce a certain product or deliver a service [11, 2-11]

Innovation is a continuous process and demonstrates certain behavior in time. The S-Curve framework is a mathematical model that is used in various disciplines to express an entity’s behavior in time. [12] In innovation the S-Curve is used to show the introduction, growth and maturity. The plot on Figure 9 shows how two innovations in the same field behave in time. Innovation-A, which is an established technology vs innovation B which is a new innovation. Innovation A has had enough time to perfect its flaws and as seen in the plot its performance starts rather slowly but has an immense growth. By the time the new innovation comes into the market innovation A is enjoying growth and the market is yet to turn to the new product which yet has to perfect its flaws and processes. Meanwhile as time progresses, increments become more expensive and innovation A reaches its maturity while innovation B starts to enjoy growth. The pattern between these two innovations portrays the behavior of innovations and their common interactions with one another.

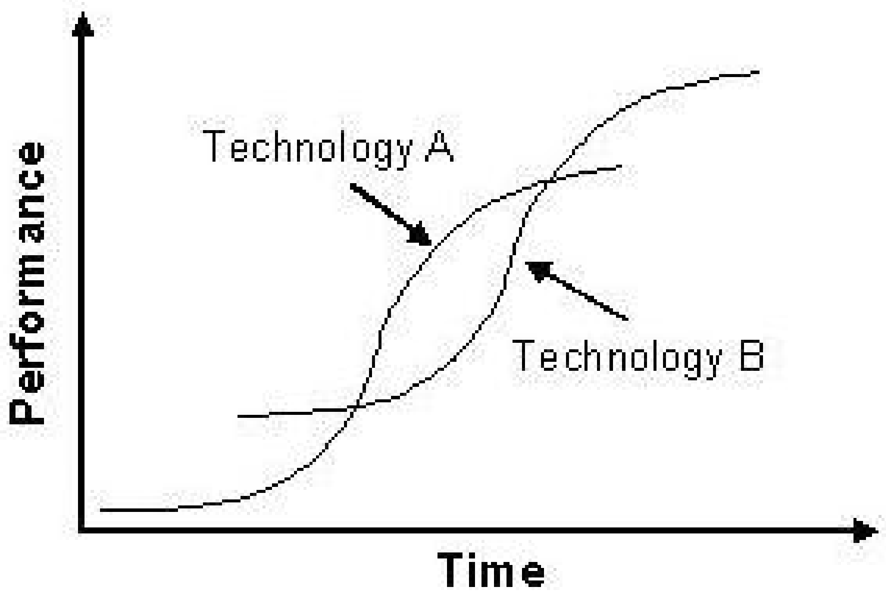


Fig 9. Technology S-Curve (source: http://innovationzen.com/)

In today's world of cutthroat competition, start-ups need more optimized techniques in order to stay afloat or even thrive in the world of innovations. There are several methods that can be used to foster innovation, as there are many schools of thought in this area.

A famous quote from Louis Pasteur "Chance favors the prepared mind" [11, 39] and indeed one of the core starting steps is mental preparation. In order to find a solution for an existing problem, would-be innovators should prepare themselves for generating fruitful ideas. They should dive into the literature, view the problem from different angles and be ready to refuse the accepted wisdom. Idea generation is the most important early step of the process and teams use different techniques in order to achieve that.

5.1 Brainstorming

Webster's online dictionary defines brainstorming as:

"A group problem-solving technique that involves the spontaneous contribution of ideas from all members of the group; also: the mulling over of ideas by one or more individuals in an attempt to devise or find a solution to a problem" [16]

In order to efficiently conduct brainstorming, there are five key principles:

- Focus: Brainstorming should put focus on a problem and spend the team's energies in solving it.
- Suspended judgment: Brainstorming sessions should be devoid of judgment on ideas presented no matter how odd they might seem.
- Personal safety: Brainstorming members must be guaranteed for their safety against any kind of provocations or judgment based on their idea.
- Serial discussions: Brainstorming sessions should limit the conversation to one focused topic.
- Build on ideas: Brainstorming sessions encourages team members to take the ideas of others further. [11, 45]

Brainstorms can be broadly categorized into three groups, visioning, modifying and experimenting. Visioning is a session dedicated to coming up with a breakthrough solution to a given problem. Envisioning encourages imaginative thought with the goal of breaking free from the status-quo. Modifying, on the other hand is concerned with modifying the current product or service or in general the status-quo. Experimenting is trying to come up with some novel ideas by combining the already existing one in different fashions.

As ideas are coming in thick and fast, a new challenge is faced by the team. The challenge is in regards to selecting the ones which would have a commercial success. The ideas need to be screened using techniques that would enable the selection of an idea with the most with potential of success. [11, 15-48].

5.2 Rational stage model

According to Steve Conway and Fred Steward, the rational stage model is a combination of steps, decision gates and feedback loops that would enable a screening of ideas. [10, 282] The model serves as a funnel which takes the bulk of ideas are screened the using some predefined criterion. As illustrated in Figure 10, an idea must pass different levels of screening in order to be ready for the market.

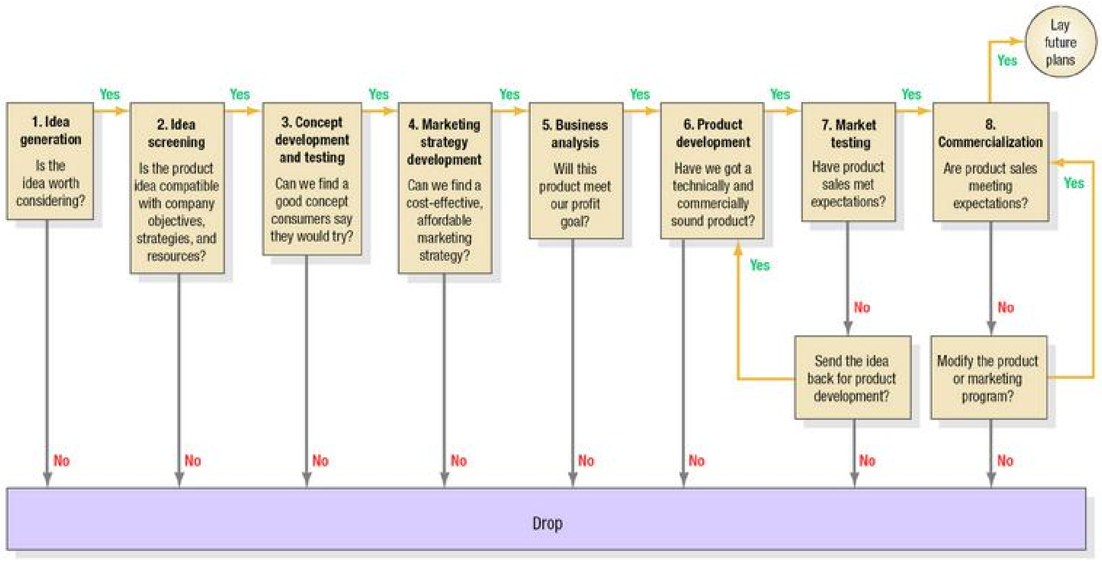

1. Idea generation
Is the idea worth considering?
2. Idea screening
Is the product idea compatible with company objectives, strategies, and resources?
3. Concept development and testing
Can we find a good concept consumers say they would try?
4. Marketing strategy development
Can we find a cost-effective, affordable marketing strategy?
5. Business analysis
Will this product meet our profit goal?
6. Product development
Have we got a technically and commercially sound product?
7. Market testing
Have product sales met expectations?
8. Commercialization
Are product sales meeting expectations?
Lay future plans
Send the idea back for product development?
Modify the product or marketing program?
Yes
No
Drop

Figure. 10 Rational stage model on innovations [10]

Once ideas are generated, they are subjected to pass through this logical funnel. The set of criterion and decision gates might be different in relation to the product or service being developed. However, the most common gates are the marketing strategy and business model. The marketing strategy focuses on answering some key questions: the size of the potential market and its behavior, the potential position this product or service will acquire once launched, what makes it different from its key completion, long and short- term marketing strategies. Upon analyzing the idea through this set of questions the idea is tested and if it passes it goes to the next test and so on.

Sometimes this process might take a long time to complete so there are other techniques which are devised in order to accelerate this process. Some of them include conducting multiple stages in parallel. The need for accelerating the process arises from the requirement to market the innovation at the right time [10, 280-294].

6 Leading Agile teams

21st century has been referred to as the century of complexity by the renowned physicist and mathematician Stephen Hawking [13, 5]. Unlike the linear systems theory which argues that for every effect E there is a specific explainable and/or predictable cause C, the complexity theory argues that the interaction of entities has outcomes which are unpredictable. The expectancy that the world is made up of clear cause and effects is known as causation fallacy.

Because of the human nature to be in control, scientific management has found favor with many. At its core, it’s about planning to determine every possible effect and employing a command and control structure to govern the process. Such has been the case with waterfall style of development. However, the creative and dynamic arena of software development has proven too complex or unpredictable to this style.

Agile management is inherently complex, accepting the fact that not everything can be planned for and that it is by an iterative and retrospective approach that a close perfection of an implementation can be possible. Concepts such as self-organization and emergence are at the heart of Agile management. In order to manage Agile teams it is important to understand the type of leadership that best suits it. There are many leadership styles that are best suited in to a particular area. In order to understand what kind of leadership style and how to employ it is necessary, one should have a painted picture of the scenario at hand.

The project which this thesis focuses on is an Agile and innovative one. Therefore it is safe to assume that it is creative and that creative management is one of the needed features. In addition to that, since the team is a self-organized group of students motivation and self-organization play an important role to create the cohesive force needed to make the team a success. Since the team is self-organized, the leader would have to show extreme care towards the employment of orders and other related issues. It is also important to understand that the team does not have experience of working together and hence the creating chemistry is also another big task. Another important factor in the nature of the team is that the leader is also a team member and might not be the most talented one at that. Hence, he/she should employ means to get the best out of the team members.

Delegative or “Laissez-faire” leadership style is one of the most common leadership styles. It gives complete control to team members to make their own decisions [14].

Leaders provide the tools necessary for the teams to do their job. In the Agile management world an adaptation of delegation is used. Congruent to delegation is the concept of empowerment and the level of delegation. In Agile management, there is no black and white in delegation where one either is free to make a decision or not. Instead, there is the level of freedom a team member is given and this is known as empowerment. Many researches on the corporate world have shown that this technique boosts efficiency in teams.

6.1 Creative management

In order to manage creative teams it is first important to understand the nature of creativity and how we develop it as part of our biological process of growth. Human beings exhibit three phases of creative process, the pre-conventional, conventional and post conventional creativity. The pre-conventional creativity, which is seen in young children under the age of seven, include spontaneous and emotional involvement aided with visual conceptions, whereas conventional creativity, which is common in children between seven and eleven, happens with the involvement of actual thinking. However, the constraints of less developed skills are seen at this age. The post-conventional creative stage is usually seen in adults and it involves creating something with the awareness of the constraints. [13, 69-72]

Even though these are stages in growth of creativity, research has shown that most people are stuck with the second level and have a problem of going beyond. This becomes a challenge for the leader to help his teammates to grow to a post-conventional creativity and become innovative. [13, 72]

One of the most important tasks for a manager is to make sure there is enough available information, knowledge and several motivation of members. In addition to that, it is the leader’s job to create an ambiance that best suits creativity. One of the steps in this regard is to make sure there is enough freedom of expression and safety of ideas from the team members. Alongside the freedom, the leader should create a playful environment and take a playful approach to discussing ideas and somehow inject playfulness into the team. [13, 72-73]

Nothing kills creativity than routine work and so the leader has to make sure there is enough variation in the tasks of the team mates and also on the way the processes are handled.

6.2 Motivation

According to dictionary.com, motivation is defined as the means of providing a reason in order to act in a certain way [26].ö Motivation is used by leaders in order to get the best out of their teams. There can be two kinds of motivation in software development teams, Extrinsic and Intrinsic motivation. Extrinsic motivation is the kind of motivation that a leader (manager) employs in terms of merit pay, salary raises, bonuses etc. Even the casual praise for one’s good work is considered extrinsic. However, Jürgen Appello [Ref. Management 3.0] argues that in complex systems extrinsic motivation falls short of attaining their goals. According to him, extrinsic motivation is a child of the causal deterministic fallacy and that it does not address the complex world we live in today. Different authors have identified several side-effects of extrinsic motivation. These include unnecessary competition among colleagues and lower problem solving efficiency. [13, 75-76]

Intrinsic motivation on the other hand is the kind of motivation that comes from the need to see the job done well. It is the kind of motivation that arises by instilling a sense of purpose on the job at hand and that its accomplishment is its own reward. There is a widely accepted view that intrinsic motivation is actually the foundation of a creative team. The advantage of this kind of creativity is that it works well with complex systems, rather than having an unpredictable side effect and also helps creativity. [13, 78]

6.3 Demotivation

According to the Motivator-Hygiene theory put forward by psychologist Fredrick Herzberg, satisfaction is independent of dissatisfaction [13, 79]. For instance an employee with a low salary or working in a difficult environment would not be motivated if those issues are solved rather he/she will be motivated if he/she is given additional responsibilities, a chance to make decisions and also to create the feeling that one belongs to a group. Herzberg explains his theory using two simple factors:

Motivators: These are factors that actually induce motivation. Challenge in ones work, increased responsibility, opportunity for developing competence etc.

Hygiene: Job security, health insurance, good salary, benefits etc.

Herzberg used the word hygiene because according to him these factors are like hygiene. It is their absence that creates a problem. However, their presence only makes us healthier and not happier. Taking away these problems will bring one into a state of neutrality. In order for a team to be in a motivated state, the leadership must do more than just taking away the demotivators. [13, 79]

6.4 Self-organization and emergence

Self-organization is the property of a group of entities to emerge in patterns without any central authority. Self-organization is an inherent behavior with complex-systems and hence considered as a default for Agile teams. Command-and-control type of organization is common with previous models like the waterfall. In Agile development, teams organize by themselves so as to choose the best way to deliver value to the stakeholders. Schedules, estimations and delegation of tasks, are all done collectively with agreement with no central authority steering the process. The pressure from the external systems such as deadline, play as an organizing attribute for the team. [13, 99-100]

Self-organization does not mean a leaderless wild-west where everyone is free to do what they want. There would still be the need to control but the way that control is implemented is sufficiently different from command and control. The main role of leadership in this kind of situation is to have a birds-eye view of the dynamics of self-organization inside a team and provide coaching and mentoring, so that the team organizes in ways that it can deliver value to the stake-holders [15]. In addition to that, the leader should also make sure that the team evolves retrospectively and develops its efficiency through continuous learning. In such situations, a leader in Agile development is compared to a Gardner. His/her main objective is to make sure that the team grows. Jürgen Appello [13, 115] uses an analogy that we build code and assemblies and we grow team chemistry, relationships, and social networks. We grow teams and software systems. [13, 100-115]

7 Discussion

The first project conducted was for Imagine Cup 2010. The team's name was Team#. At the start of our project we were four students with a high level of motivation and much less idea of the road map we needed to follow in order to be successful. The first step that we undertook was to analyze our situation and the requirement of the Imagine Cup competition.

The Imagine Cup software design category required a team to come up with an idea that could change the world. It could affect a small group specifically or a lare group of people broadly. It was also a competition where hundreds of thousands of students took part and therefore technical excellence was required. The four main judgement points were value of the idea, presentation, technical maturity and business potential. In addition to that there were key Microsoft technologies that needed to be incorporated in the product.

From the requirement we understood that the team needed to deliver an innovative solution. The team was new and we did not have a track record of developing together. There was the need to self-organize in a way that would bring the result. In addition to that, we were not assigned to develop a set of technical requirements, but instead the team was given the freedom to create and change those requirements on its own. Naturally, our choice was Agile. Our very first step was to choose an Agile framework that would best suit us. In addition to Agile, however, we needed to implement techniques to foster innovative ideas and means of screening them so as to be able to select the right one.

From the infrastructure point of view, we did not have a central office and also our work times were different. In addition to that we had different technical specialites as well as some areas of needed technical requirements where we had less knowledge. Therefore, competence development was also another challenge we needed to overcome.

Once we had agreed on scrum, we started to analyze our current status and see how we could best utilize it. Since we did not have an office space and our schedules were different, we decided on having three hours of working together every Saturday. In addition, we started to utilize the Google code and agreed to post our daily development in our tasks there. In practice this was our daily scrum. At this point it was very important to have the motivation really high, because the steps agreed needed a disciplined approach from every team member in order to be successful.

Our very first approach to the project was to see in what ways we could develop our technical expertise in some needed areas such as web development. Our team’s first task was to research some of these areas such as Silverlight, web development, .NET and server management and have a means of sharing that knowledge by means of presenting our findings every Saturday. The knowledge sharing not only gave the team more light but it also created a fun and creative atmosphere. Before we started the hard parts, we were able to establish chemistry and sense of freedom in the team. These are highly valued attributes of a successful Agile team.

Next we started the ideation process. During our weekly meetings we started to have brainstorming exercises. We started one idea per minute per session basing ourselves on the imagine cup innovation requirements. The brainstorming sessions we carefully organized and coordinated so that the team felt maximum freedom in suggesting new ideas no matter how wild. After the session was over we ran catch-ball kind of trick where we tried to further grow on ideas thrown by a team mate. After the session was complete, the most promising are selected the team takes the responsibility of researching on them so there will be an informed screening process.

The next session was the screening session, the candidate ideas were screened based on the imagine cup requirements. Each team member first presented a research on the candidate ideas and then the team analyzes the idea. In order to have a successful idea, the candidate idea must pass a series of questions. Does it solve a problem? Do we have the technical expertise to pull it off? Does it have a viable business model?

The role of the leader during the ideation and screening level is very important. The leader is responsible for creating a culture of creativity and freedom within the team. One of the steps to achieve that is to maintain judgment-free brainstorming sessions. Not only is the leader in charge of the atmosphere making sure that the sessions are judgment free, he/she must also enforce the strategic direction of the team. The strategic direction helps the team to focus their resources on fruitful efforts. In addition to enforcing the strategic direction the leader should also be involved in the creative process. He/she should not be an onlooker but rather an active player in the creative process. [11, 121-122]

The ideation process that was followed was cyclical, few idea candidates were chosen from each brainstorming session. Research would be conducted on those and then they were subjected to the screening process. This process was repeated a few times and then the a clear idea was chosen.

The chosen idea was telemedicine. According to the American Telemedicine Association, telemedicine is defined as using ICT tools and technologies to better a patient’s medical status by enabling remote patient-doctor communications [17]. The area chosen was indeed quite broad and therefore the team needed a more specific telemedical solution that solved a problem. LiveMED was born.

7.1 Ideation scenario

LiveMED is a concept that aims to provide a technical solution to enable hospitals of the developing world to make use of expatriate doctors via a telemedical portal. Research has shown that many developing countries are continuously losing doctors towards developed countries for better pay. In addition to that, those doctors who are living in those countries are residing in city areas leaving the remote areas with a higher scarcity of medical professionals.

On the other hand the current trend on telemedicine is that the giver and receiver are both health facilities which form a VPN (Virtual Private Network) so as to conduct their services. For instance, if a hospital decides to give telemedical assistance to a remote clinic, then this hospital creates a VPN with that clinic and starts its services. Using this model, the hospital can only use in-house doctors to achieve its means.

Imagine a distributed application. Imagine there is a doctor in Europe who is willing to give medical assistance to developing countries or even remote locations in Europe. Imagine there is a hospital which has a telemedical department using LiveMED telemedical product. This hospital gives medical assistance to clinics in remote towns and villages using this product. The doctor checks a central website (say LiveMED.com) which provides a list of hospitals using the LiveMED platform. The doctor contacts the desired hospital. Once his/her credentials are confirmed, he/she will be given the access right to be a member of the telemedical department. Then, the doctor can publish his/her free dates, confirm appointments and from the comfort of his/her sofa he/she will save the life of a child thousands of kilometers away.

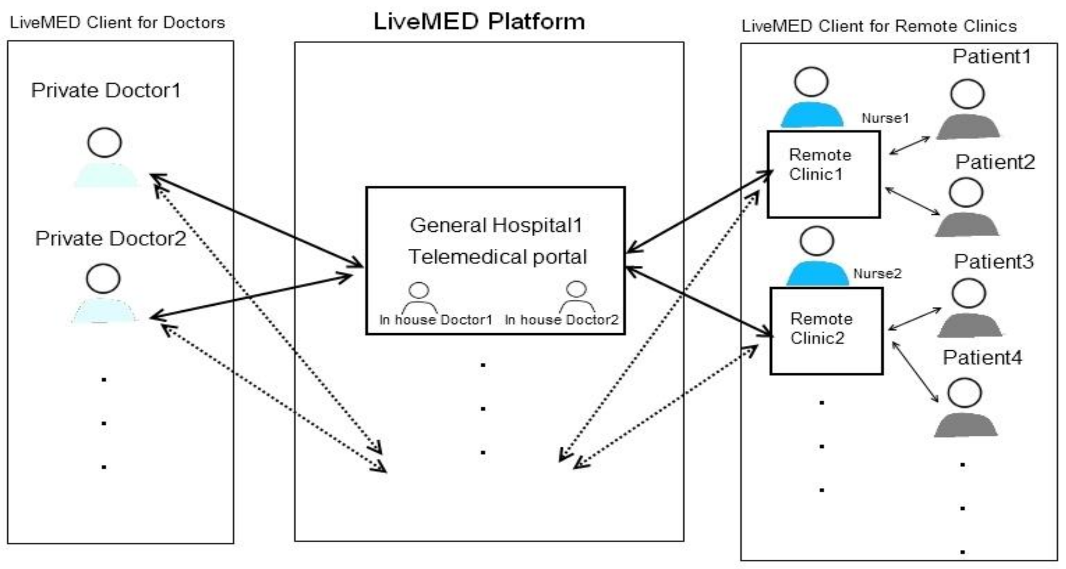


Figure.11 Platform Topology: Showing the distributed nature of LiveMED

As seen from Figure.11, it is understandable that one doctor can give medical assistance to many patients through multiple general hospitals. One general hospital can give assistance to multiple remote clinics and one remote clinic can receive aid from multiple general hospitals. The distributed nature of the application gives rise to a permutation which allows thousands of doctors to give medical assistance and information to thousands of patients around the globe.

After a thorough analysis of the requirements, Team# (name of our team) anticipated to come up with an iterative and incremental development model. In this model version, LiveMED version 1.0 would incorporate a web-based appointment scheduler and video conferencing environment. In this version, actors would use LiveMEDs’ web portal to schedule appointments and have a video conference (see section 8.1).

The application would be first installed in a general hospital in some urban area which would like to extend its help to remote clinics in far away towns and localities. The hospital will authorize its own personnel and also medical professionals from abroad as members of its telemedical department. The hospital will have a link on its home page to its own internal LiveMED portal. Authorization will be done through the hospital’s own workflow and the hospital administrator would give access rights to those who are eligible. Therefore, all outside doctors who would want to render services will be given access rights and they will do so through the general hospital’s telemedical department. The administrator of the site will decide of how many departments of medicine or how many remote clinics can be covered by the telemedical department. LiveMED can then be used to book appointments and have a video conference.

As the project began, one important issue needed immediate attention. Security of patient medical information and legal issues involved is a sensitive matter. For instance, the Finnish law demands that any medical information of a patient shall not be shared with anyone unless the patient explicitly consents. In this regard, the application imagined a secure storage for sensitive information, and a disclaimer which appeals the patient’s willingness to allow exchange of the medical data for the medical professionals involved.

Furthermore, an authentication and authorization process is needed bearing in mind that this is a medical-ICT product. Doctors should be confirmed that they are indeed doctors and thus manual secure authentication is necessary when using this application. Another important issue is the initial cost a health center has to incur in order to have this application given that most institutes who welcome this application are from developing countries.

7.2 Technical scenario

LiveMED is an application intended to create a telemedical platform which hospitals can customize to their own specific needs. Hence, LiveMED focuses on an evolving system capable of being an independent platform with SDKs and APIs for customization. Starting from a lobby where doctors meet remote patients, it will evolve by adding functionality by analyzing further requirements and feedback. Following this plan the technical design revolves in coming up with a generic model which will evolve in time and avoid prohibitively expensive system upgrades.

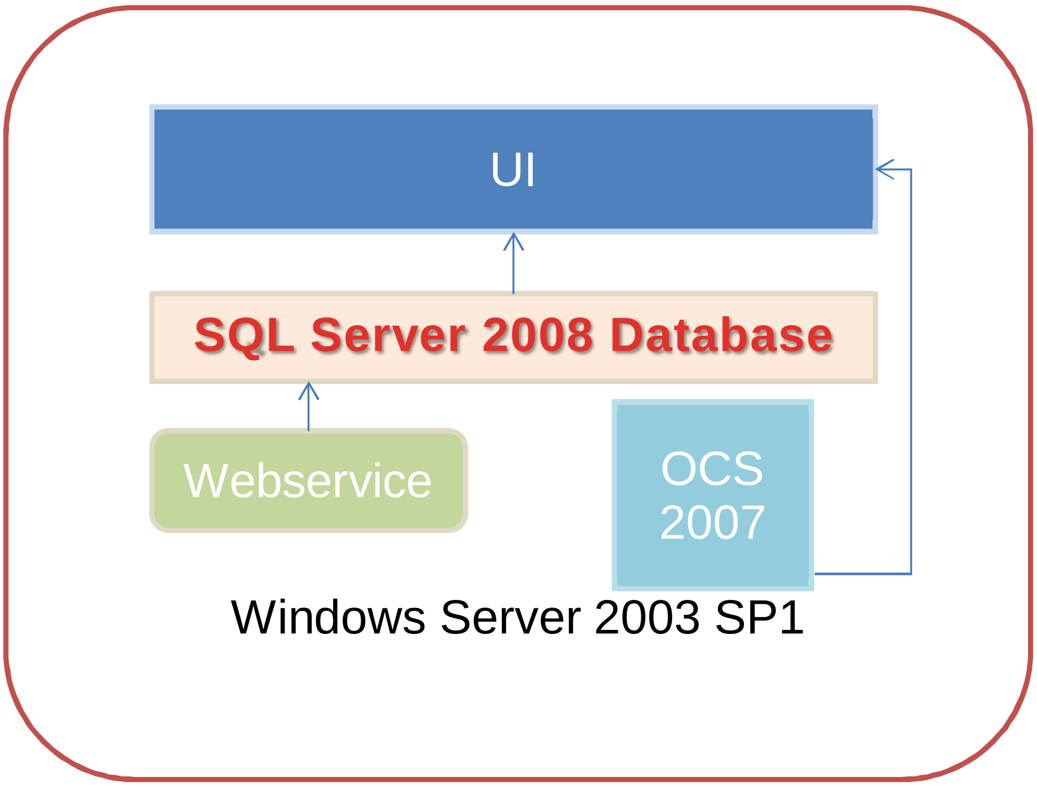


Figure. 12 Technical architecture of LiveMED for a given general hospital.

As seen in figure 12, the application will be operating in a Windows Server 2003 SP1 environment which will have the SQL Server 2008 database and Office Communication Server 2007. The data model of the application is object relational where the database can expand using inheritance structure and it has the support of SQL-server UDT to hold customized data types. The first version of LiveMED will only have scheduling appointments and video conferencing and hence there will be no need to have customized patient data types.

The video conferencing will be built using Unified Communication Client API for one to one audio video chat. Later versions of the product will have secure multi-party conferencing and collaboration. A webservice will be used for billing appointment sessions. The appointment scheduler uses a 20 min mark to book appointments (see User Scenario). However, the session limit is configurable. The webservice will then conduct a web method to calculate billing info for every confirmed appointment.

LiveMED's central website will have a listing of general hospitals and remote clinics using LiveMED. Moreover, it will have a rich Silverlight interface containing version revisions, news feeds, updates etc. This site will evolve in its functionality as LiveMED evolves.

7.3 User scenario

The application will have two components. The application deployed on the hospital's premise will be a web portal interface built using ASP.NET connected to an SQL Server 2008 Database. The remote LiveMED client will be a desktop application installed on the user's workstation. The application will have three main actors namely Doctor, Nurse and Hospital Administrator.

Authentication of the web application defines three user roles, namely doctors, health officers (nurses, trained assistants) and patients; each having different access rights to the services of the site. Authentication against doctors and health officers will be done manually since this involves high risk of fraud and requires extra security measurements. Patients will not directly be able to access the site but use health officers as a proxy.

The administrator will administer the LiveMED application in the hospital and does not need to be a medical professional. He/she authorizes, registers or prohibits personnel to use this application. He/she also decides how many clinical departments (Cardiology, Dermatology etc.) can be supported. The database takes these arguments and enables the scheduling to support these departments.

The doctor will have an interface that will enable him/her to publish free dates. Upon registering the system converts the input time to discrete 20 minute timespan appointments. He/she can confirm pending appointments and also view pending appointments. After appointments are requested from the nurse's interface the doctor confirms or rejects the request. Once a doctor confirms an appointment, he/she will be liable for missing that appointment.

The nurse will register incoming patients in their own specific clinical departments. The nurse then views the available doctors on the desired date from the department of choice. Upon selection the nurse will view available time slots and then the nurse will choose a patient from that department and book an appointment.

The application will have a notification system. A notification email is sent to a doctor when a nurse books an appointment. Similarly, it is sent to the nurse when the doctor confirms or rejects an appointment. Due to the distributed nature of the application all time slots are handled in a GMT format but displayed in the local time for the end users.

Desktop Client Application: the desktop application is used for the video conferencing. The doctor will choose which general hospital to use. After the authentication, the doctor can then view the appointments within the next 24 hours. Once an appointment time comes, a video player will be active for the video session. On the other hand, the doctor can just post his/her presence information. If the doctor's presence is available, a remote clinic can ask for assistance directly without the scheduling. This way, if a doctor has a free time he can just login and make himself/herself available for clinics and answer their call and conduct sessions. Figure 13 shows the view of a new doctor's portal with an empty schedule.

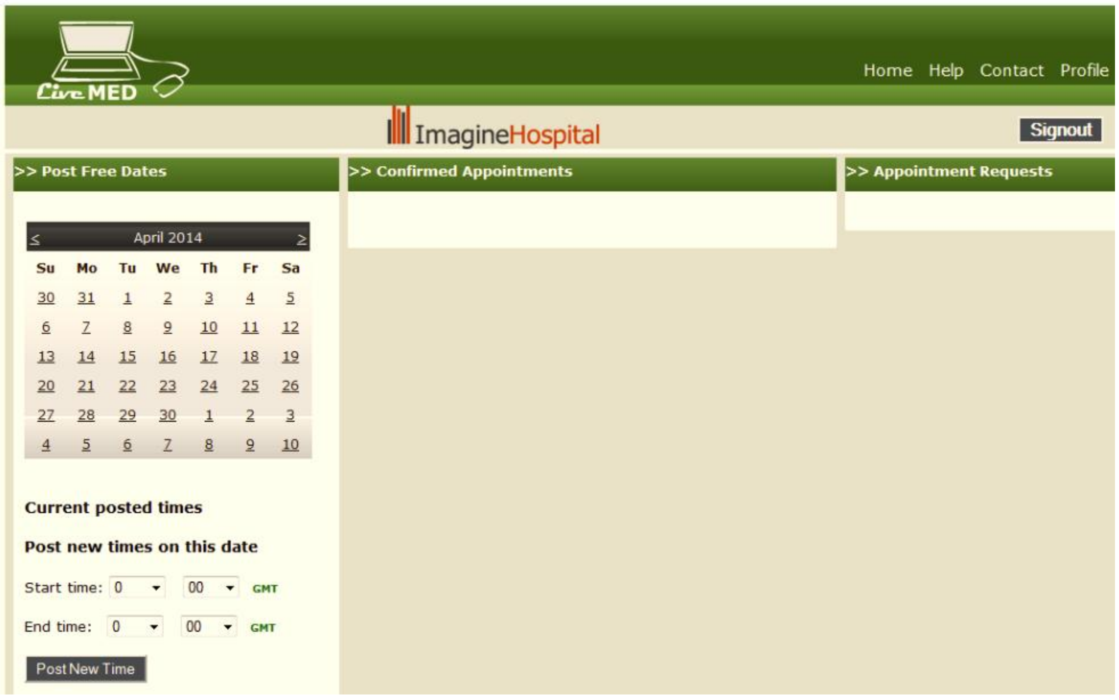


Figure. 13 Doctors view of LiveMED

7.4 Business Viability

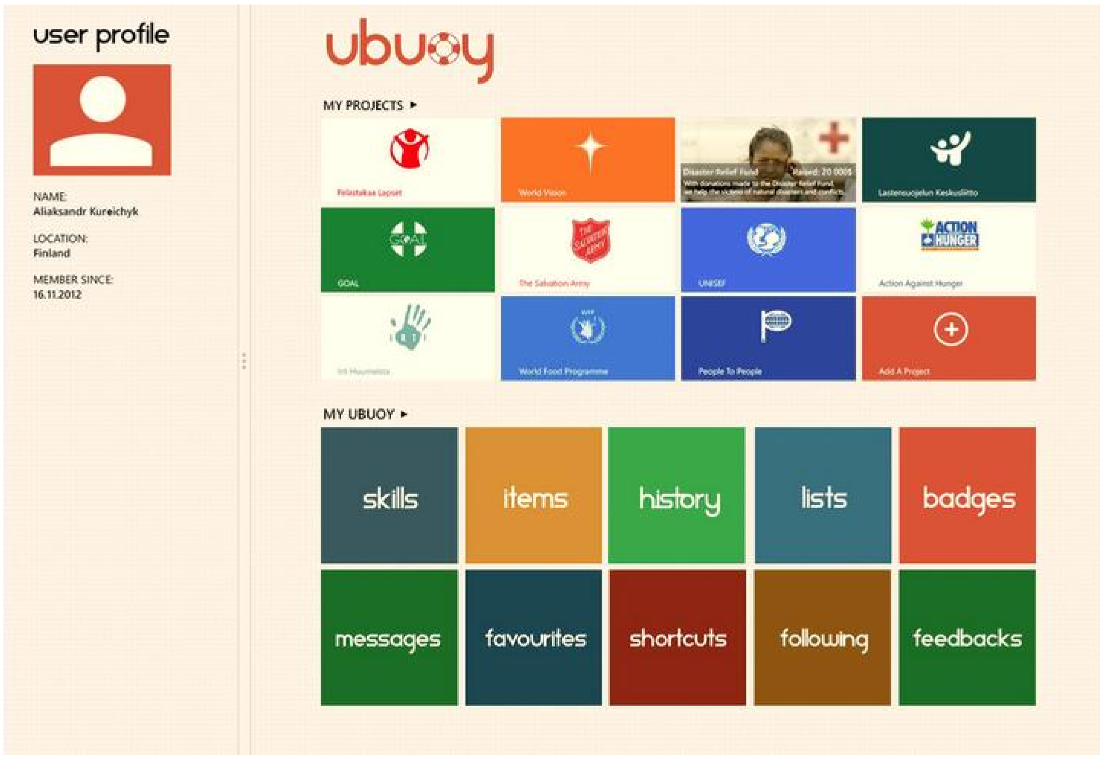
Even though we have a business plan that is more than 50 pages, this thesis follows a Lean approach on explaining the basics. Ash Maurya argues that the core idea behind a business plan should be risk mitigation and gaining trust, trust from investors, customers etc. He further argues that any idea is implementable given the right amount of resources unless of course it is a cure for cancer. The first approach is making sure that the UVP (unique value proposition) is clear and simple to understand. The next step is to choose a few prospective customers and engage in a long interview and explore the idea further from the view point of these prospects. Next, we should analyze the competition [18, 3-75].

LiveMED understands that even though the idea is good the initial customer cost for infrastructure might be prohibitively expensive and thus we have opted to using cloud and making LiveMED a subscription service where the customer can pay on the go for the usage. In addition to that, the growing number of mobile phone users in remote areas of developing countries opens doors for many other monetization schemes.

7.5 Ubuoy

In 2012, another group came together under my leadership/coaching for the imagine Cup competition. The group like the one before had all the characteristics of the previous one with added complexity of being multi-cultural. However, on the plus side there was valuable learned experience which could be utilized well. Apart from putting into place the processes which had proved successful in the previous campaign, the leader had first taken steps to measure the level of motivation the members had. Although motivation is a necessary input expected from the leader, it is best that the team members show a minimum level of commitment. In addition to that, strict rules were set for not wasting other team member’s time and effort by not lifting one’s own share. As a result of this, one team member was let go from the team and was replaced by another.

Once the team chemistry was achieved and the environment properly set, ideas started flowing and selection process also followed rigorously. It was during this ideation stage the Ubuoy was born. The name came from joining a buoy which is the circular inflated balloon used to save lives at sea and the letter “U”. Ubuoy is a service where people exchange their talents for donation. For instance, if someone has the talent of fixing a computer and a desire to donate to UNICEF, he/she creates a profile at Ubuoy and inputs his/her list of skills. On the other end a person with a broken computer sees this user, contacts him/her via the service and the payment for fixing the computer goes straight to the charity of the user’s choice, in this particular scenario to UNICEF. As the user increases in using the service several badges or certificates will awarded from the service which gives him/her more prominence.



NAME:
Aliaksandr Kureichyk

LOCATION:
Finland

MEMBER SINCE:
16.11.2012

Figure. 14 Profile view of a user

As can be seen in the figure 14 the team opted for a user interface design that was similar to the Windows 8 metro style. It was a result of experimenting and an explorative ambiance we created inside the team.

7.7 Vision and minimum viable product

In today's innovative product development scene vision is something more than a list of features to be built. A vision should be the same picture of the product in every team member's head. Having the same vision about a product would lead to a more creative success in developing the product. A vision is not a list of features but rather serves as a guideline as to the direction of development of the product, also taking into account the dynamics of the surrounding environment. According to the elevator test, a clear and concise vision should be explained in a short elevator ride. A vision should have a very clear value proposition that makes it unique.

Once a vision is set the next step is to have the minimum marketable product. Given the fact the only certainty is that nothing is, the minimum viable product should not try to encompass all the competitive features of the existing alternative but rather focus on the main value to be presented and leave room for further development in the future. The same approach was followed when Apple launched iPhone for the first time. Even though remarkably successful, it did not ship with copy and paste, developer kit or even the ability to text to several contacts at once. Apple, however, gained good grounds on time against its competitors by launching a superb experience and then added all the features in later increments.

The same approach was used with Ubuoy when deciding the minimum viable product. At first, significant time was spent in discussing our way to a common vision and then we decided on delivering the key value “Donate to your favorite aid using your talents”. We also decided to ship all the other increments that increased user engagements, social features etc. on later increments. However, features which were planned to be delivered on the first release were to be subjected to an iterative development. [20, 23-31]

7.8 Technical scenario

One of the important issues in Agile development is to know that situations are bound to change in the future. In such scenarios the code should also be built in ways that it is flexible enough to change. Separation of Concerns (SoC) is a principle that makes a code base more adaptable to change. It dictates that the implementation should be divided up of loosely connected layers which are responsible for a single action. For instance, a model should hold the domain objects, a repository should be responsible for data retrieval and persistence, a business layer should be concerned with the logic of the application and then there is the UI. In following such a pattern the business layer does not care if the UI is a mobile application, a web interface or even a desktop client. [19, 37-54]

There are several proven design patterns in the software development discipline that help in better maintainability and agility. One of them used in Ubuoy is the template design pattern which was used to implement a generic repository. The template design pattern is a behavioral pattern used when several classes share the skeletal algorithm. In Ubuoy a generic repository was designed so that any new entity can adopt the functionality. In addition, lambda expressions and delegates available in .NET were used to further encapsulate the type of queries (See Appendix 1). [19,105]

8 Conclusion

This thesis aimed to elaborate that in order to have a creative and Agile software development team, the main focus should be on people. Processes will be fine-tuned in retrospection, but still it is the people who will benefit from a continuous learning path. I would like to conclude by stating the gains in people that these two projects have brought.

I conducted a round of interviews with past team-mates and did an analysis of what the position the team members were in before and after joining Imagine Cup. The first main characteristic that they all exhibited was a lack of conviction on the professional path they wanted to follow. They lacked the vision of themselves a few years from now. In addition to that their technical capacity was at best at a junior level. Even though they knew the value of self-learning they had not pushed their limits to see what they could achieve.

Results should be measured in what was gained by the people and six of the eight team members are now professionally employed. The other two are actively looking for employment. Six have graduated and one is on the way. Two of them have teamed up again for another Microsoft-Aalto University competition. All of them agree that it had been the best technical project at school and would not hesitate for a minute to work together. In addition to the people results, LiveMED was awarded the second prize in the Finnish finals of Microsoft Imagine Cup in a closely contested competition. It is all about the team, if we concentrate on growing the team’s potential and spirit, innovation and success will be inevitable.

References

1. Mary and Tom Poppendieck. Leading Lean software development. 1st ed. Indiana: Addison-Wesley; 2009.
2. SDLC Waterfall Model [Online]
   http://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm
   Accessed 11.03.2014
3. Understanding the pros and cons of the Waterfall Model of software development [Online].
   http://www.techrepublic.com/article/understanding-the-pros-and-cons-of-the-waterfall-model-of-software-development/#
   Accessed 12.03.2014
4. Jerrel Blankenship, Matthew Bussa and Scott Millett. Pro .NET Agile development with Scrum. 1st ed. New York: Apress; 2011.
5. Robert C.Martin and Micah martin. Agile Principles, patterns, and Practices. 6th ed. Boston: Prentice Hall; 2010.
6. Kenneth S. Rubin. Essential Scrum: A practical guide to the most popular Agile process. 1st ed. Michigan: Addison-Wesley; 2012
7. Ken Schwaber and Jeff Sutherland. The scrum guide: The definitive guide to scrum: Rules of the game [Online]
   URL:https://www.scrum.org/Portals/0/Documents/Scrum%20Guides/2013/Scrum-Guide.pdf
   Accessed 28.03.2014
8. Building real software [Online]
   http://swreflections.blogspot.fi/2012/10/should-you-care-about-conways-law.html
   Accessed 28.3.2014
9. Robert C. Martin. Agile Estimation and Planning.8th ed. Boston: Prentice Hall; 2008
10. Steve Conway and Freed Steward. Managing and Shaping Innovation. 1st ed. New York: Oxford University press; 2009.
11. Harvard Business Essentials. Managing creativity and innovation. Boston: Harvard Business School Publishing Corporation; 2003.
12. Understanding the S-Curve [Online]
    http://innovationzen.com/blog/2006/08/17/innovation-management-theory-part-4/
    Accessed 6.4.2014
13. Jurgen Appello. management 3.0: Leading Agile developers, developing Agile leaders. 2nd ed. Indianna: Addison-Wesley;2011.

14. Laissez-faire leadership [Online]
http://psychology.about.com/od/leadership/f/laissez-faire-leadership.htm
Accessed 28.4.2014
15. Leadership of Self organized teams [Online]
http://www.infoq.com/news/2013/04/leadership-self-organized-teams
Accessed 28.4.2014
16. Brainstorming definition [Online]
http://www.merriam-webster.com/dictionary/brainstorming
Accessed 6.4.2014
17. Telemedicine definition [Online]
http://www.americantelemed.org/
Accessed 6.4.2014
18. Ash Maurya. Running Lean. 6th ed. California: O'Reilly press; 2013.
19. Scot Millett. Professional ASP.NET Design Patterns. 1st ed. Indianapolis: Wiley publishing; 2010.
20. Roman Pichler. Agile product management with Scrum, Creating products that customers love. 1st ed. Boston: Addison-Wesley;2010
21. Waterfall development methodology [online]
http://learnaccessvba.com/application_development/waterfall_method.htm
Accessed 12.03.2014
22. Waterfall model advantages and disadvantages [Online]
http://www.buzzle.com/articles/waterfall-model-advantages-and-disadvantages.html
Accessed 12.03.2014
23. Comparing costs and effort between agile and traditional development [online]
http://blog.scottbellware.com/
Accessed 14.03.2014
24. User stories [online]
http://www.sphereinc.com/approach-user-stories/
Accessed 17.03.2014
25. The product backlog [online]
http://www.plannowtech.com/blog/?p=335
Accessed 21.03.2014
26. Motivation definition [Online]
URL: http://dictionary.reference.com/browse/motivation?s=t
Accessed 25.3.2014

Appendices

Appendix 1. Generic repository for Ubuoy

```
namespace Ubuoy.UserAuthentication.DataLayer
{

  public class UbuoyDataRepoditory<T> : IUbuoyRepository<T> where T : class
  {

  private ObjectContext _context;


  private IObjectSet<T> _objectSet;


  public UbuoyDataRepoditory()
    : this(new Ubuoy_DB_ModelEntities())
  {
  }


  public UbuoyDataRepoditory(ObjectContext context)
  {
    _context = context;
    _objectSet = _context.CreateObjectSet<T>();
  }

  public IQueryable<T> Fetch()
  {
    return _objectSet;
  }

  public IEnumerable<T> GetAll()
```

```
{
    return _objectSet.AsEnumerable();
}



public IEnumerable<T> Find(Func<T, bool> predicate)
{
    return _objectSet.Where<T>(predicate);
}



public T Single(Func<T, bool> predicate)
{
    return _objectSet.Single<T>(predicate);
}

public T First(Func<T, bool> predicate)
{
    return _objectSet.First<T>(predicate);
}

public void Delete(T entity)
{
    if (entity == null)
    {
        throw new ArgumentNullException("entity");
    }

    _objectSet.DeleteObject(entity);
}



public void Delete(Func<T, bool> predicate)
{
    IEnumerable<T> records = from x in _objectSet.Where<T>(predicate) select x;
    foreach (T record in records)
    {
        _objectSet.DeleteObject(record);
    }
}


public void Add(T entity)
{
    if (entity == null)
    {
        throw new ArgumentNullException("entity");
    }

    _objectSet.AddObject(entity);
}


public void Attach(T entity)
{
    _objectSet.Attach(entity);
}


public void SaveChanges()
{
    _context.SaveChanges();
}


public void SaveChanges(SaveOptions options)
{
    _context.SaveChanges(options);
}
```

```
        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }


        protected virtual void Dispose(bool disposing)
        {
            if (disposing)
            {
                if (_context != null)
                {
                    _context.Dispose();
                    _context = null;
                }
            }
        }

    }
}
```