The Importance of Requirement Change Management

Group 7
University of Gothenburg
Gothenburg, Sweden
Georgi Dungarov - dungarov@gmail.com, Patric Lövgren Berg - patriclovgrenb@gmail.com, Plamen Mateev - plamenmtv@hotmail.com, Fjodor Savostin - fjodor.savostin@gmail.com, Hristo Spilkov - hspilko@gmail.com

Abstract — Software development has had enormous and rapid growth in the last three decades. Despite the progress, and the innovation it has reached, software development still has quite a few problems to overcome. One of these problems are issues with the changes in regards to requirement specifications. This document aims to show the importance of clear software requirements in the industry, the problems that occur when changes are made to them and the difficulties for a company to manage them. It describes models that are used to help solve these problems. It also provides theories and models related to Change management and real life examples.

I. INTRODUCTION

Software Requirements as a part of Software Engineering define the needs of the stakeholders and set up expectations on software's behavior. Software Requirements are the blueprints to the software developers from the clients. When developing new software, the importance of clear and accurate software requirements is crucial. Failure to achieve a strong working method that handles changes in requirements will both cost time and money, both of which are precious to a project group or company. This report will address common problems in software requirements changes and how to solve them in an agile manner. These are the questions that we want to answer with this paper: *What processes does a change go through before being implemented? How does changes in the requirement specification affect the project work? What types of changes can there be in the requirement specification? How can a company or project group be prepared for these changes?*

In order to answer the questions above, our group has examined the obstacles a company in the software industry face and the methods they apply to overcome these issues. Furthermore, we discuss the solutions and relate them to Change management. Lastly, we review methods that can be used by the companies in the software industry in order to overcome obstacles related to changes in requirements.

This report covers the software aspect of the software requirement changes and not the business level. This document does not explain how to make good software requirements - it is expected that the user has knowledge about the agile work method when defining the requirements.

II. THEORETICAL BACKGROUND

This paper emphasizes the problems that arise from requirement changes in a software project. The research looks at our industrial partner and compares the solutions they apply in order to fix those problems. The theories and methods which are used have been discussed in Change Management course (DIT035). To give a background regarding the work methods at our industrial partner, we present a description of the reasons behind change in requirements along with processes of how to achieve those changes. In order to adapt to frequent changes in requirements, companies need a flexible approach. Agile development views change as a positive thing which evolves over time. Change in the requirements occurs often in today's software development. Because of that, companies set initial requirements which develop their project scope and develop better schedule. In Agile method the detailed documentation is not important in an early stage.

Processes of Change

On the subject of *Change Management* and how companies handle changes within the requirements, we can see how companies within the industry accommodate their development strategies to fit the requirements, which can be unpredictable - clearly observing Lifecycle theory [8] approach to changes within the organizations. Van de Ven and Poole (1995) talk about change in organizations. They try to explain why and how these changes apply to in organizations by attempting to develop a theoretical framework.

Lifecycle theory is defined as cumulative, which means that the characteristics that exist in early stages are possessed in later stages as well. According to this theory, change is imminent and progressive [8]. The first step in following the *Lifecycle theory* is to take the requirements with the top priority which can be implemented in the next iteration. Each requirement that is going to be implemented is discussed by the team. The next step is to develop working software that meets the initial requirements. The development goes closely with the stakeholders. The final third step is optional and it is to provide a demo of the software. Even though this step is optional it is good to have a working demo that is used as a measurement of progress [8].

According to Scrum, freezing the requirements during iteration provides better stability. The companies that follows this method treats change in requirements that is currently developed, as a new requirement.

Each new requirement is then proceeded to the next iteration. This type of change is defined as *Continuous* [9]. *Continuous change* is part of the theory developed by Weick and Quinn (1999). The authors analyze organizational change contrasting it as episodic or continuous where the first is discontinuous and unpredictable, where the second is evolving and more incremental. The difference between the two changes depends on the point of view of the observer. From a distance, also called "macro level" the event seems like repetitive routine with occasional dramatic change. From a closer point of view, also called "micro level" the event proposes continuous adaptation or adjustment [9].

Continuous change is used as a phrase to indicate evolving,

cumulative and growing change. This change has more of a micro level perspective which suggests that things change constantly and requires adaptation and adjustment because of the small periods of stability. Each continuous change in practice creates conditions for further problematic outcomes and innovations which indicate no beginning or end point. There are three stages – freeze-rebalance-unfreeze. The initial state cause the change making the patterns visible, after that the rebalance is rearranging the patterns and the final state unfreeze which resumes the event with possibility for further changes. The role of the agent is to redirect change [9].

Similarly to the method mentioned above, Lewin (Burnes, 2004) defines a model called *3-Step Model* which perform a change in the current Status Quo [10]. The *3-Step Model* is the key change management model in theory by Lewin (Burnes, 2004). As the name suggests the model has 3 steps – *unfreeze, transition* and *freeze*. The first step of *unfreeze* is to destabilize the status quo by reducing the restraining forces in order the new behavior takes the place of the old one [10]. The second step is necessary in order to bring control and give direction of the change. During that phase new behaviors and values are developed. The third step, *refreeze,* stabilizes the group that has accepted the change. It ensures that people involved in the project know why the performed change is important. It lowers the resistance to innovation if there is any.

Infrastructure of Change in Requirements

Orlikowski and Hofman (1997) discuss organizations that face problematic situations which require change. The authors try to create a model that would facilitate the change process. They see change in requirements as fast and unpredictable, so they base their model on improvisation. To implement the new requirements, two factors are considered – change as a continuing process and unpredictability of related changes [11].
Developers and project managers meet and converse with customers and users to establish the requirements on a regular basis, making sure that the requirements are feasible, and if they aren't - to make sure the customer understands why. The end goal would be to reach a consensus, a synthesis of ideas that both parties can agree on, to ensure the success of the product. Orlikowski and Hofman (1997) develop a framework that try to predict the outcome from a certain change. A company needs to discuss the change in requirements in weekly meetings and prioritize them [11].

Theory by Ciborra, Braa and Cordella (2000) describe the advantages and the problems that come with the modernity of managing technology [12]. The world rapidly changes, and so should the management process. Developers focus on infrastructure connects to data and documents. Even though the infrastructure is hard to be evaluated, the services are not. The variables that one infrastructure can provide to the organization are called *reach* and *scope*. The former is the number of processes ran by the infrastructure. The latter defines the type of processes that are managed. *Reach* and *scope* predict different roles for the infrastructure such as utility, dependence and enabling [12]. The dependence role requires adjustment of the infrastructure. That innovation should be constant, followed by reduction of the cost. That is called *management by maxims.* Another type of management called *management by deals* suggests completing short-term goals and other business factors. This approach is used quite often [12].

Social Behavior in Requirement changes

Klein and Sorra (1996) explain the difficulties and the importance of the commitment to the use of innovation by the organizational members [13]. Innovation implementation begins when a superior member of the organization or a client makes the decision to perform a change. There are two types of stage models – source-based and user-based model [13]. The first one is relying on the perspective of the the developer to do the innovation. It can be a new product or a service. The latter, differs in that it relies on the perspective of the user. In our case, the people who initiate a change are in most cases clients. Organizations, who make a choice to adopt innovation, face the challenge to change their members' behavior [13]. The advantages depend on the overall usage, and not an individual one. An appropriate climate for implementation is needed, meaning that users should be given enough time to experiment with the innovation or be trained by supervisors. Supporting and rewarding the users' behaviour adds to the good climate for implementation [13].

On the employee, or team-based level, dealing with changes in the requirements can be very difficult to cope with, especially since deadlines need to be met, the project/scrum master must be satisfied, as well as the customer. Moe, Dingsøyr & Dybå (2009) [14], presents a case study on introducing self-managing teams (using Scrum), to companies which had never used this type of innovation before. We can clearly see how an erratic customer's requirements keep changing and how that affects the team. The project manager had to set new goals without completing the previous set of deadlines and milestones, the team could never complete their plans, they had to continuously scrap and adapt to the new requirements, which in the end caused a huge loss of resources and time, and the result - a very poor product, a team in disarray and a very dissatisfied customer [14].

III. RESEARCH METHODS AND DATA ACQUISITION

The group started out by defining keywords to search for relevant articles about software requirements, Agile software development, people management and changes. These keywords are as follows:

- Software requirements
- Agile work method
- Change management
- Requirements management in software

- Change in software requirements.

After the keywords were established, we set up a rather simple criteria, to which we stuck to in order to find the most relevant research papers for us. The criteria were:

- The topic of the paper needs to reflect changes in requirements in an agile software development environment
- The research paper must offer problems that stem from, or solutions that tackle changes in the software requirements within a project

For the purpose of this report, we focused on a more qualitative method of data gathering. We utilized a *guided interview*, which had outlined topics to discuss, to extract information from our industrial partner and we conducted a number of *literature reviews and analyses* to provide a more in depth understanding on the subject of change management and how to handle changes in the software requirements.

The basic methodologies for gathering of data where a series of publications and books that were found on the IEEE xplore database and Google Scholar. A popular online literature publication search engine, which helped us get pointed in the right direction when we used our keywords in order to find relevant data.

The overall method for gathering and compiling information for the literature part includes reading and cross checking different publications against each other to avoid redundancies and the use of the case studies that are present with those publications. When gathering information publications would have a 1-2 year gap to see how the field in question has evolved allowing to make a prediction on how it will progress in the future. For the literature part of the paper very little raw data was used unless you count other source material source data.

First, we organized all the research papers that we found *by topic* and how they fit with our criteria. This was the easiest way to determine which research paper fit our criteria and area of interest. Afterwards, we skimmed through the *abstracts* of each relevant paper and made a cross check between all the relevant papers, to see which ones were best for us, with the least redundancy. When the data was cross checked and rewritten a conclusion was drawn based on the overlying theme of the research, the literature and the case study giving a overview of all the empirical data and the results. In order to find the correct papers for us, we scrapped articles which solely focused on "agile work methods", since that was not our top priority, and found research articles and studies which focused on the software requirements, *while* utilizing agile development methods. The research papers ranged from 1985 to 2008 in years of publishing.

In some cases whole sentences were used and were referenced. The references can be found in the appendix [1]-[15].

For the purpose of gathering raw data from a direct affiliate in the field, the team decided to work close with the company "Findwise". In this case we were allowed to ask a series of question that would allow us to get insight in the "Real" functions of an agile software developer and understand the process of resolving the difficulties that may arise in a constantly changing field.

We interviewed "Hans Liljevang", a consultant at Findwise. He has been in the software development industry since the 70s and worked with both Classic waterfall/waterfall and agile. Though leaving the software development role, as in programmer, for some time Hans is currently working as a Project leader at Findwise. We decided to interview Hans for his experience in the software development and management field. It is hard to get a person who have worked with software development for so long so there was no need to interview another person with less experience.

There were two interviews conducted at Findwise. The first interview was mostly about gathering ideas for which kind of topic we would like to investigate while the second one was more in-depth for the topic that we found interesting. The questions for the second interview was structured in a way such that the interviewer could understand them but also so that we could gain in-depth knowledge about our subject. To structure the interview we have analyzed all theories studied in the Change management course. From the information we gathered from each theory we structured individual questions that strongly relate to a particular method from a particular theory. That made our task to evaluate the company and describe their way of solving problems related to requirement changes. We tried to keep the questions short but also informative. The question are about changes in the requirements specification as well as what kind of methodology Findwise has to cope with the changes. These are a few of the themes we have for our questions regarding requirement specification changes:

- Unpredictable changes - How does one cope with them?
- Resistance of changes? - What happens if people resist the change?
- Time span of changes? - How much time does a change take?
- Type of change approval? - How does a change go from idéa to reality?

To accomplish this we had the theories from the DIT035 Change management course as an angle of approach.

IV. ORGANISATIONAL SETTING

The Organization that we worked with is called "Findwise". It started in 2005 and has been a busy organization ever since. They

have 6 offices in total located in Gothenburg, Stockholm, Copenhagen, Warsaw, Oslo and Malmö. Findwise is an it-consultant company that specializes in search engines within a company, so called intranet searches. These searches can be about "information about employees", important documents and similar tokens. With over 800 projects completed it has made a name for itself and have partners like Google, Microsoft, IBM, Oracle and more. Findwise uses agile as a software development model and have always had.[6] We spoke to the department located in Gothenburg so that we could meet employees there in person. Findwise is using an Agile software model and Scrum for all their projects. All their software is also open source.

As an IT-consulting firm, Findwise has multiple customers simultaneously. Findwise also have a role called production manager. This person distributes the workforce available in Gothenburg and is managing the resources needed for each project(Budget, Man hours, Payment etc).

Findwise usually have 2-3 different projects active at one time.

Usually a project team is comprised of:

- 1 Project leader
- 1 Architect
- 1 GUI Designer
- 1-2 Software developers

These numbers can differ depending on the size of the projects and importance. A max of 18 people have been working at the same project at the same time.[7]

The developers are allowed to consider changes in the requirements that can be beneficial to the project but must present these suggestions to the project leader first before making any changes whatsoever. The project leader then speaks with the architect and customer regarding possible structure changes, increasing costs of the software and the overall impact of the change. These impacts can be about response times, different In/Out data, how much time it will take to complete the changes as well as updated sprints and sprint deadlines.[7]

V. FINDINGS CONCERNING CHANGES IN THE SOFTWARE REQUIREMENTS

Based on the data gathered from the research papers, on the subject of Change Management, we have found several prominent theories related to how an organization can operate when enduring change. Be it on the company or employee level. These theories were discussed in section two.

While still on the organizational level, dealing with change in the requirements is something companies can hardly predict and it can be very difficult to think of ways to deal with the problems that arise (such as delays in deployment, bugs in the software, loss of resources, etc.), that's why some organizations such as Findwise do not have a set contingency plan to handle those kind of changes and instead, rely on adapting to the change and the effects that come with it - that's the process of continuous change, illustrated by the article presented by Weick & Quinn (1999), while also supporting the type of strategy planning for the future, presented by Mintzberg & Waters (1985) - which is the *emergent* type of planning [15]. The *emergent* strategy of dealing with problems doesn't focus on a set goal, but instead focuses on tackling problems that pop up over a long period of time [15]. We can see this occurring in many companies within the industry - for example Findwise and their way of tackling sudden changes in software requirements by their customers [9].

The *Lifecycle* development model, discussed earlier, fits well with Findwise's way of developing software. They work closely with their customers to ensure every single software development iteration is to the customer's satisfaction, the work teams discuss among each other which requirements are of top priority and tackle them first, and they try to strictly follow the requirements that the customer has set, in order to meet the end goal [8].

Findwise utilizes the 3-Step model as well[7], specifically in their way of software development iterations. They *unfreeze* the requirements they have set previously, discuss any improvements they can introduce, afterwards they *establish stable requirements* with which the team and the customer agree upon and finally, they *refreeze* them and proceed to finish the development cycle. That way, they make sure the requirements don't change in the process of development, and that's how they keep costs and risks low.

According to Orlikowski and Hofman (1997), and Weick and Quinn (1999) change is fast and unpredictable. Findwise believes that most changes cannot be foreseen [7][9][11]. Different types of changes can occur depending on the people the company is working with. That is because different companies or clients have different style to which Findwise needs to adjust. A company from different country might find a certain problem bigger than a company from Sweden for example [7].

Even though Klein and Sorra (1996) describe that members of the organization have difficult times committing to a change [13], that is not the case with Findwise. The company recognizes the importance of their clients' input and that is why they perform the change in requirements if they are asked to. Even more, Findwise encourage their employees to provide changes that are beneficial to the software [7]. If a developer wants to perform a change, first the client is informed with good arguments. If the client agrees, the new requirement will be done in the next iteration. Findwise adapts to the requirements set by their customers, and try to discuss the changes in case the asked change harms the project or increases the

cost drastically. Klein and Sorra (1996) also describe two types of models - *source-based* and *user-based* [13]. Findwise does not follow strictly any of those model, but they recognize themselves in the latter. They find the interactions with the customers very important for the software development, since they are using Agile development [7].
The approval of the requests is done by the project leader and software architecture. This type of approval is *source-based.*[13] It means that decisions lies on the perspective of senior developers. The senior managers go through the request and analyze risks, cost and outcomes related to the end product. Before perform change, Findwise makes sure the customer understands the requirement specification [7].

The industrial Partner

After the meeting with the industrial partner one could see that good and clear requirements are one of
the most important aspects of a succeeding company or project group. And it is important to see what
changes in the requirement really means for the project group and company. *How does this change affect us? Can we afford to implement this change at the moment? Should we wait for it? The change seems OK but do we really need it at all?* Findwise seems to understand these aspects of software requirements and how to cope with them. "The most common type of changes are the unpredictable ones."[7] It is very cliché to say that one should always be prepared for the unexpected but this is the truth in software development.

Findwise have customized solutions for their customers. It means that there is a pre-developed structural skeletons in which the customers can add or remove functions and features to their liking. Even at this early stage is it very important that the correct features and requirements are clear even at the start of the project so that minimal amount of deviations may occur. Documentation is extremely important due to it can bring up good examples of previous scenarios that are similar to the current project and point out possible risks and problems. Findwise always documents all changes, even suggestions to changes are documented with arguments on why the change did not make it to the final product. This kind of documentation proves a powerful tool which saves time and money since you will have a better understanding and more likely to make the right decisions.

A real life example was given to us by the industrial partner where the customer was not fully satisfied with the requirements:

A customer hires Findwise to create a software/program for their company. Both parties agree on the requirements and the initial price that the software will cost is around "100k swedish kronor"(just an example given by Hans and not the actual value). After the customer had accepted the requirements specification, with signature, the project was ongoing. The customer then had in mind that another feature should be included in the requirement specification, for the same price as they agreed on from the start. So extra functionality for the same price. This obviously caused a detour in the development process since the change would affect the structure of other requirements and it was different than Findwise agreed on earlier. The question arises whether one should continue coding and apply the change later, or apply the change right away to save time? A discussion was held and the following scenarios of solutions came up: [7]

- Findwise initiates the change and the customer pays for it.
- Or Findwise makes a compromise - meaning that both parties pays for the change
- Or Findwise pays for the change entirely.

If an agreement is not possible the entire project can be canceled down but this is very unlikely.

Taken this example in mind, changes in the requirement specification is not a real problem in itself for Findwise. They are a company, and they want to make money. "*The customer is always right as long as the requirement specification is consistent. That means whatever the customer wants us to do and are willing to pay for it, we accept it*".[7] The most common type of requirement changes are often about the GUI. Team members have sessions when they are brainstorming on what is good with the software and what is not good with the software.

The use of Agile software development have helped with alot of the customer interaction. In some cases the customer is present even during coding sessions and is providing idéas and suggestions to the code itself to accomplish the requirements. It is very important to have this special kind of connection with the customer according to Findwise. Making the whole project team interact with each other and other stakeholders is very important to uphold the communication so that misunderstandings does not occur (Hopefully). It is also very important that the customer knows what they are paying for and that they really comprehend the requirements needed to create the software. [7] Misunderstandings between parties is a common reason why changes has to be made in the first place. Inexperienced customers can present bad requirements or even illogical ones. To help the customer the project leader and Software Architect speaks with the stakeholder, presenting strong arguments that the change is not beneficial for the software product.

VI. DISCUSSION

After having conducted the research and review of the literature on **Change management**, it is quite apparent that the theories and ideas presented in those papers are always relevant and always in motion, whether the actors within the organization realize it or not. Comparisons and recommendations of strategies on how to tackle changes in requirements will be provided with the help of the theories from DIT035.

Communication

Communication is one of the more vital parts between the developer and the customer as described by the case study [1]. While it is true that communication has a great impact on the overall progress there is also not always time to communicate often with the customer when the process is well underway.Because of that [1] writes of different types of communication during the different phases in the development process. Having the amount of communication decrease due to time restraints on both the stakeholder and the developers parts. As stated in [1] multiple times physical proximity with the stakeholder is one of the best ways to get direct information from the source in the form of verbal communication. "*Since in traditional development the information travels through a long chain of people, it becomes distorted and some of it is lost."* [1]

Reasoning behind the direct communication is usually feedback from the stakeholder while also seeing the personal focus of the stakeholder and while the practice of on-site customer has not diminished due to its problematic nature "*it enabled effective communication and instant feedback"*[1] prior to the more common video conferencing approach now used. This practice has changed to the now more common *Real Customer Involvement* which does not force the customer to be constantly involved in the development process instead the customer now contributes their feedback on weekly and quarterly basis. Those people are still essential to making sure results are produced based on the requirements set prior. However due to the lack of a full time presence communication and feedback are more essential as stated by [1].

When researching about the agile development process it is impossible to miss the need of customer collaboration. Because of the everchanging pace of the style and constant changes to requirements the definitive feedback of the customer is a must. Considering that the customer requested the given work to be done all steps must be made to avoid incorrect working assumptions or misinformation about the functionalities requested. However it is not only feedback that is required from the customer but also the idea and a well defined structure of the software that is required.[2] In a agile development all team members should have access to the customer during all the development stages. these practices diminish slightly from iteration to iteration see case study [1]. If those conditions are not met it will lead to failure of following the requirements which lead to defects while also preventing new changes in the latter phases of the development cycle. The interview from the industrial partner shows that they have a good customer collaboration and it is working efficiently. They have truly understood that to cope with requirement changes they need to have a strong customer relationship.[7]

Requirement management

"*Without an iterative plan for approaching the development of requirements, the design organization can find itself, months along on the project, developing the wrong software functions."*[5]. While a iterative plan is important it still needs to follow the overall goal of the requirements. The requirements for said system could be different from what the customer really wants as written in [5]. "*When customers present ideas that need system solutions, engineers have an ethical and professional obligation to help them define and simplify their problem. You must build the best solution to the customer's problem, even if the customer does not yet understand how to ask for it."*[5]

This does not imply that customers never know what they need. Instead it shows that for the full functionality of said system requirements other than the original ones have to be used to fulfill the overall goal that the customer requested, its is even better if the customer only provides an overall idea for functions alongside constraints instead of complete architecture for a system, that would allow the engineers to simulate and calculate all the requirements prior to creating the architecture for the project and allowing the engineer to also apply a more fault tolerant system that would work consistently. Findwise has its own solution for such incidents. The solution that our industrial partner uses in such cases is a general skeleton which allows the customer to determine various functionalities and features for their system.[6] This allows Findwise to gain clear software requirements from an early point in the project, since the customer must accept these requirements before moving on to the actual coding of the end-product. To help the customer, a prototype of the software[6] is created within each iteration when needed so that the customer can evaluate the software and see that the features are up to par with the customers idéas and restraints which needs to be accepted before going to the next iteration/sprint. This proves that Findwise have a lot of experiences with the management of software requirements and their intent to make requirement changes more of an option rather than a problem.

The industrial partner also cannot forget that the individuals using the system and the customer who requests the system are not the same person and nothing can be take as obvious. A full explanation of the process should be given in each iteration so that the customer does not feel out of place or lost in the changes however that part has been covered in customer communication.

One of the larger goals of a system reliability. However no matter how stable a system initially is it will start to have a lower and weaker performance over a period of time as show in the appendix. This is why Findwise have an support service that can be bought together with the software. If a stakeholder has this service, Findwise will keep the software updated when big updates comes to operational systems, web browsers, new coding libraries/JDKs. This kind of supportive role is a great way to keep customers informed and be a part of the software even when the project is over. So in a sense, some project never ends.

VII. CONCLUSION

Conclusion communication.

Agile software communication should use face-to-face communication as it's primary communication channel. Due to the ever changing nature of the project and requirements by the customers direct contact will reduce the chances for defects that other communication methods might introduce. While the on-site method is prefered in todays types of development it is not always possible ,because of that the use of *videoconferencing* can be used to lower the chances of defects and emulate a similar environment as constant face-to-face communication.

Conclusion requirement management

When working software requirements the ideas and constraints of the customer should be utilized to create an architectural model. From that model onwards functionalities will be implemented while trying to maintain as much stability as possible to give the customer a lasting system that fits their every need. Findwise knows how to handle changes in the requirements in an professional way[7]. By utilizing a skeleton for the customer to input his ideas and functionalities the system is then tailored to his needs during the various iterations. The change approval process that Findwise uses is effective as well. It relies on that the team has good communication with the stakeholders and presents relevant arguments on whether the change is worth it or not.

Conclusions Change Management

The theories from **Change management** help define different methods and types of changes that conducts the software development and provides better quality. It aims to create standards that capture the software characteristics, the problems that people stumble upon and their respective solutions. Change management strengths the software development. We believe that these ideologies are foundation for the software development in future and necessary supplementary advancement. However, the implementation used from each theory may vary from company to company.

This research presents how and why Findwise change their requirements while developing software. Overall the changes that were mostly had in the requirements for Findwise was the GUI ones. Not that big of a deal but important in its own way. With this study one can say that Findwise is up-to-date to most of the theories conducted in this report and are performing them in their own special way. Findwise is still a young, growing company that has endless possibilities.

VIII. APPENDIX

References:

[1] Korkala M., Abrahamsson P. & Kyllönen P., Case Study on the Impact of Customer Communication on Defects in Agile Software Development. Department of Information Processing Science P.O.Box 3000, FIN-90014 University of Oulu, Finland

[2] Levy M., Hazzan O., Knowledge Management in Practice:The Case of Agile Software Development.

[3] Boehm B., Turner R., Management Challenges to Implementing Agile Processes in Traditional Development Organizations, IEEE SOFTWARE,0 7 4 0 - 7 4 5 9 / 0 5 / $ 2 0 . 0 0 © 2 0 0 5 I E E E, P(30-39).

[4] Kontio J., Höglund M., Rydén J. & Abrahamsson P., Managing Commitments and Risks: Challenges in Distributed Agile Development, Proceedings of the 26th International Conference on Software Engineering (ICSE'04)

[5] L. Bernstein. (2005, Taking software requirements creation from folklore to analysis. ACM SIGSOFT Software Engineering Notes 30(5), pp. 3-5.

[6] Google Docs. 2014. Meeting with Findwise #1. [ONLINE] Available at:https://docs.google.com/document/d/1jmw7R3BL4mCg9wd0UQ61CtQdUztlIYUkDDZe82xedj8/edit?usp=sharing

[7] Google Docs. 2014. Meeting with Findwise #2. [ONLINE] Available at: https://docs.google.com/document/d/1ak0MxDVX-tvgX4OQ8_7a07BA6NlHL34HPrUmFAmdqzE/edit?usp=sharing. [Accessed 19 December 14].

[8] Van de Ven, A. H., & Poole, M. S. (1995).Explaining development and change in organizations. *Academy of management review*, *20*(3), 510-540.

[9]Weick, K. E., & Quinn, R. E. (1999). Organizational change and development. *Annual review of psychology*, *50*(1), 361-386.

[10] Burnes, B. (2004). Kurt Lewin and the Planned Approach to Change: A Re-appraisal. *Journal of Management studies*, *41*(6), 977-1002.

[11] Orlikowski, W., & Hoffman, D. (1997). An Imporvisational Model for Change Managment: The Case of Groupware Technologies. *Inventing the Organizations of the 21st Century, MIT, Boston, MA*, 265-282.

[12] Ciborra, C., Braa, K., & Cordella, A. (2000). *From control to drift: The dynamics of global information infrastructures*. Oxford, England: Oxford University Press.

[13] Klein, K. J., & Sorra, J. S. (1996). The challenge of innovation implementation. *Academy of management review*, *21*(4), 1055-1080.

[14] Moe, N. B., Dingsoyr, T., & Dyba, T. (2009). Overcoming barriers to self-management in software teams. *Software, IEEE*, *26*(6), 20-26.

[15] Mintzberg, H., Waters, J. A. (1985). Of Strategies, Deliberate and Emergent. Strategic

Management Journal, Vol. 6, No. 3., pp. 257-272.

Relevant info:

The reliability of a system varies as a function of time. The longer the software system runs the lower the reliability and the more likely a fault will be executed to become a failure. Let R(t) be the conditional probability that the system has not failed in the interval [0, t], given that it was operational at time t = 0. The most common reliability model is:

R (t) = e-lt,

Where is the failure rate and the Lambda in our requirements protocol. It is reasonable to assume that the failure rate of a well maintained software subsystem is constant with time after it is operational, with no functional changes, for 18 months or more, even though faults tend to be clustered in a few software components. Of course bugs will be fixed as they are found. So by engineering the Lambda and limiting the execution time of software subsystems we can achieve steady-state reliability. Lui Sha of the University of Illinois and I define

Lambda (l) = C/Ee.

This is the foundation for the Lambda Protocols and the reliability model becomes

R = e-k Ct/EeThis equation expresses reliability of a software system in a unified form as related to software engineering parameters. To understand this claim, let's examine Lambda term-by-term.

Reliability can be improved by investing in tools (e), simplifying the design (C), or increasing the effort in development (E) to do more inspections or testing than required by software effort estimation techniques. The estimation techniques provide a lower bound on the effort and time required for a successful software development program. These techniques are based on using historical project data to calibrate a model of the form:

Effort = a + b (NCSLOC)b

Schedule = d (E)1/3

Where a, b and d are calibration constants obtained by monitoring the previous developments using a common approach and organization. NCSLOC is the number of new or changed source lines of code needed to develop the software system. Barry Boehm's seminal books Software Engineering Economics, Prentice Hall, 1981, ISBN 0-13-822122-7 and Software Cost Estimation with COCOMO II, Prentice Hall, 2000, ISBN 0-13-026692-2, page 403 provide the theory and procedures for using this approach.[5]

Table 1. Summary of the case projects.

	Case 1	Case 2	Case 3
Iterations	5	5	5
Iteration span (weeks)	1+2+2+2+1	1+2+2+2+1	1+2+2+2+1
Number of product releases	4	4	4
Product type	Intranet application	Mobile application	Mobile application
Programming language	Java	Mobile Java	Mobile Java
Customer presence	Onsite 80%	Onsite during the beginning and the end of each iteration. Available for face-to-face communication.	Onsite during the beginning and the end of each iteration.
Customer location	Same facility	Same facility	Same city
Number of customers	1	3	1
Face-to-face communication level	High	On demand	Periodical
Mid-iteration communication media used	Face-to-face	Face-to-face, email	Email