SOFTWARE DEVELOPMENT IN PRACTICE:
A QUALITATIVE STUDY IN TURKISH DEFENSE SECTOR

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS INSTITUTE
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

BAŞAK DİLBER

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
INFORMATION SYSTEMS

MAY 2013

SOFTWARE DEVELOPMENT IN PRACTICE:
A QUALITATIVE STUDY IN TURKISH DEFENSE SECTOR

Submitted by BAŞAK DİLBER in partial fulfillment of the requirements for the degree of **Master of Science in Information Systems, Middle East Technical University** by,

Prof. Dr. Nazife Baykal _____
Director, **Informatics Institute**

Prof. Dr. Yasemin Yardımcı Çetin _____
Head of Department, **Information Systems**

Assoc. Prof. Dr. Sevgi Özkan _____
Supervisor, **Information Systems, METU**

Examining Committee Members:

Prof. Dr. Semih Bilgen _____
Electrical and Electronics Engineering, METU

Assoc. Prof. Dr. Sevgi Özkan _____
Information Systems, METU

Assist. Prof. Dr. Erhan Eren _____
Information Systems, METU

Assist. Prof. Dr. Banu Günel _____
Information Systems, METU

Prof. Dr. Soner Yıldırım _____
Computer Education and Instructional Technology, METU

Date: 29.05.2013

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Başak Dilber

Signature : _____

ABSTRACT

SOFTWARE DEVELOPMENT IN PRACTICE:
A QUALITATIVE STUDY IN TURKISH DEFENSE SECTOR

Dilber, Başak
M.Sc., Department of Information Systems
Supervisor: Assoc. Prof. Dr. Sevgi Özkan

May 2013, 80 pages

Since management is an abstract concept, modeling or simulating software development projects is not exactly possible. Thus successful software project development and the factors affecting it have been popular fields of research for the last half century. The aim of this qualitative grounded theory study is to bring out the essence of the experience of people working as software developers in a specific organization operating in Turkey; to come up with a set of principles explaining the main success factors for managers and software development teams.

KEYWORDS: Software Development Projects, Software Development Success Factors, Software Projects Management, Grounded Theory.

ÖZ

YAZILIM GELİŞTİRME ÜZERİNE:
TÜRK SAVUNMA SANAYİİNDE NİTEL BİR ARAŞTIRMA

Dilber, Başak
Yüksek Lisans, Bilişim Sistemleri Bölümü
Tez Yöneticisi: Doç. Dr. Sevgi Özkan

Mayıs 2013, 80 sayfa

"Yönetim" kavramının soyut bir olgu olmasından ötürü yazılım geliştirme projelerini modellemek veya simülasyon ile kurgulamak tam anlamıyla mümkün değil. Bu sebeple başarılı yazılım geliştirme ve bunu etkileyen faktörler geçtiğimiz yarım asır boyunca popüler bir araştırma alanı olagelmiştir. Bu niteliksel gömülü teori çalışmasının amacı, Türkiye'de faaliyet gösteren bir organizasyonda yazılım geliştirme projelerinde görev alan kişilerin tecrübleri doğrultusunda yazılım geliştirme projeleri yöneticileri ve ekipleri için temel başarı faktörlerini açıklayan bir ilkeler seti geliştirmektir.

ANAHTAR KELİMELER: Yazılım Geliştirme Projeleri, Yazılım Geliştirme Başarı Faktörleri, Yazılım Projeleri Yönetimi, Gömülü Teori.

ACKNOWLEDGEMENTS

First of all, I would like to thank to my supervisor Assoc. Prof. Dr. Sevgi Özkan for her guidance as well as her kind and friendly attitude that was what really kept me going.

I simply could not complete this thesis without Prof. Dr. Soner Yıldırım, I am really grateful for his invaluable support to me and to this study. In addition I can not ignore the contributions of Prof. Dr. Semih Bilgen, Assist. Prof. Dr. Erhan Eren and Assist. Prof. Dr. Banu Günel, I want to thank them each for their time and valuable opinions.

I also want to thank to ASELSAN Inc. not only for supporting my education but also for giving me the chance to meet marvelous people beyond just colleagues to work with. It would take a number pages of names if I tried to express my gratitude to all of them in person, but the unlucky ones that were really there even when I was whining are Yasin Gökpınar, Eda Göksoy Kalaycılar, Çağla Ateşalp, Esra Eroğlu Keskin, Güliz Kaya, Umut Şekerdağ, Yücel Sezer and Elif Erdem & Melih Günay.

Last but not least, I will never forget how my one and only, my dearest fiancé Arda Yücekayalı never gave up on me. His enthusiasm completed mine, even at times when I didn't have any left. I am most grateful for his existence in my life.

DEDICATION

To my lovely mother and kindhearted father;

for giving me wings and the strong urge to fly.

TABLE OF CONTENTS

LIST of TABLES

LIST of FIGURES

CHAPTER 1

INTRODUCTION

1.1. Background

The term "software" is defined basically as a computer program that models a real world problem along with the solution; which has abilities such as controlling hardware, computing, and creating an interface for human-computer interaction (Mei, Cao, & Yang, 2006).

Since the birth of it, software has spread into every part of our lives; including the ATM's, cellular phones, cars and the airplanes (Charette, 2005). The expansion of the usage of software led to the development of "software engineering" concept (Mei et al., 2006). Software engineering is briefly described as the engineering area that is concerned with the methodologies and management principles used for development of software (Althoff, Birk, Wangenheim, & Tautz, 1998; Sommerville, 2007). This study is concerned with the issues that were defined under software engineering term in order to develop successful software which means software that meets the quality, cost and time requirements.

However widely used in human beings' lives, development of software is still an unsolved mystery and an unstructured issue in the IT field (Wright & Capps III, 2010; Adams, 2007; Martin, Pearson, & Furumo 2007; Aladwani, 2004; Yeo, 2002). Many research in the literature tried to come up with solutions to the so called "chaos" that has been created by software development projects (Remus & Wiener, 2008; Sharp, Woodman, & Hovenden, 2005; Ewusi-Mensah, 1997; Brooks, 1986).

For the last few decades development of software systems projects and the factors that influence the fate of those have been tempting topics of research for many professionals studying on different subjects. Thus, a sea of literature that includes huge numbers of different and similar findings has been created on this subject whereas still, remarkable portion of software development projects are being laid off despite causing disastrous economical loss as Charette (2005) estimated (McLeod & MacDonell, 2011; Ahonen & Savolainen, 2010).

The already complicated nature of software is becoming even more difficult to manage as the technology changes rapidly where the expectations and needs on what software is capable of doing grows, so does the search for more flexible and modern approaches to successful software development projects. (McLeod, MacDonell & Doolin, 2011). In the software engineering area the common tendency of research was mostly depending on studies made by quantitative methods in the first decades of the problem however, many researchers and authors claim that being a highly human interaction involving field, studies on software development project should be evaluated by conducting qualitative studies (Clarke & O'Connor, 2012; Dybå, Prikladnicki, Rönkkö, Seaman & Sillito, 2011; McLeod & MacDonell, 2011; Virili & Sorrentino, 2010; Aladwani, 2004; Seaman, 1999).

The aim of this study is to propose a set of principles on the main issue of concern when executing a software development project by conducting a qualitative research in a certain company operating in Turkish defense industry, while exploring the failure and success factors that affect software development projects.

1.2. Statement of the Problem

The actual problems faced in software development efforts were discussed officially and internationally first at the North Atlantic Treaty Organization (NATO) Science Committee Conference in October 1968; naming the issues arising during software development as the "software crisis" (Mei et al., 2006). With the Information Technology growing inevitably, development of even more complex software ended up with having even bigger problems in 1980's (Adams, 2007). Those problems creating the crisis were basically caused by development of low-quality software that does not meet customer requirements or fails too many more times than the tolerable amount or even if it does satisfy the end user by means of its operation, does not fit into the budget and/or time limitations (Adams, 2007).
While The Standish Group Chaos Report published in 1994 revealed only 16% of the analyzed software development projects were completed successfully within cost, time and quality constraints, 53% of them were labeled as challenged projects and a total of 31% as completely failed. After 15 years in such a fast growing organism –the IT field- the numbers have not been changing in a very promising way considering that Chaos Report claimed still in 2009 that only 32% of all software projects were successful, while 44% were classified as challenged and the remaining 24% failed (Wright & Capps III, 2010; Evelens & Verhoef, 2008).

Savolainen et al. (2011) claimed that despite the huge success of software applications in a wide range of areas in human life they are still mostly associated with "failure" in the literature.

Brown (1990) in his article about the UK standards on software development for defense systems, points out the fact that, as the "*level of design complication" increases because of the expansion of "the number, complexity, scope, and interconnection of system functions that can be implemented economically under software control*" so does the possibility of "*catastrophic design errors*" occurring during software development. He especially draws the attention to the point that these errors mostly cannot be detected until the systems are operated on the job. One of the some disastrous examples to this statement is the "Patriot Missile Defense System" failure happened in 1991 in a US army base located in Saudi Arabia. The miscalculation of a threat missile's location caused by a software design error resulted in twenty eight soldiers dying and approximately a hundred others getting injured (Zhivich & Cunningham 2009). While the "missile software", because of the nature of the subject is already open to such disasters, another horrible incident caused by a software error took place in US and Canada between the dates 1985 and 1987. A cancer treatment device massively injured cancer patients by giving them 100 times more than the needed amount of radiation because of a design error made in its software (Zhivich & Cunningham 2009).
Within the many outcomes of research on how to prevent failure of software projects that may even cause those kinds of fatal errors; one of the very significant findings discussed many times by different researchers is that software development is highly dependent on human aspects (Adolph, Hall & Kruchten, 2011; Martin et al., 2007; Sharp et al., 2005; Leung, 2001). For many years; the problems behind software development projects were thought to be caused by technical issues, whereas the social issues affecting the projects have been gaining importance over the years (Sharp et al.; Aladwani, 2004; Leung; DeMarco & Lister, 1999). As the time passed by, researches even started to point out that the people who are involved in any software development project and the social environment that the

projects are built within are much more important than the technical aspects and also the methods and tools used for the projects (McAvoy & Butler, 2009; Aladwani).

Seaman (1999) pointed out that the software engineering researchers can overcome the difficulty of studying software development issues in real life environment by collecting as much data as possible and since qualitative data offers "richer" content than qualitative data, qualitative methods should be used to increase the depth of information on the field. Even after twelve years, researchers Dybå et al. (2011) address that there are many unique managerial and organizational problems faced in software development and imply the human interaction in the process brings out the need for focusing on the "non-technical issues".

To summarize, the subjects that are evaluated as problems in the field of research are:

- after many years of research and experience still software projects fail often, and there are debates on the main factors for successful software development projects – and some agree on the conclusion that "there is no silver bullet" and success factors may vary for different environments-,
- there is a strong emphasis on the lacking of qualitative research conducted on software development.

1.3. Research Question

This study is conducted in order to develop a set of principles on how to achieve success on software development projects. The main question to be answered as the result of this qualitative grounded theory study is presented below:

Main research question: "What are the main factors that are affecting the success of software development projects?"

It should be noted that the research environment is limited in the context of a company operating in defense systems industry in Turkey. Thus, the research question is answered within this scope. This and other limitations and delimitations are given in detail in Chapter 5.

1.4. Purpose of the Study

The basic purpose of a software development project is delivering the actually needed and wanted software within the cost, time and quality requirements (Leung, 2001). Those projects, almost a %100 of time are executed and organized by means of several plans, methods and techniques, which are very basically defined as project management efforts (McBride, 2008).

As easy as it seems to make a plan, make estimations and forecasts for the sake of a project; for example a construction project, it has been very problematic to do that for software projects because of their unique nature and their being too complex to predict many issues beforehand (Morgenshtern, Raz, & Dvir, 2007; Brooks, 1986). Over 50 years of study has come up with many different methodologies to solve this problem and eliminate uncertainty in software development projects (Remus & Wiener, 2008; Daniels & LaMarsh II, 2007; Rehman & Hussain, 2007; Wang & Liu, 2006; Boehm, 1988; Brooks, 1986).

Boehm (1988) reveals the primary function of a software project model as being a framework that answers the following questions:
"*What shall we do next?*" and "*How long shall we continue to do it?*".

However, the findings of the present study shall not be evaluated as a model or a methodology as Boehm suggested. Rather than showing the steps to follow while executing software development projects, the purpose of this study is to come up with a set of principles that points out the main success factors for such projects in the defined context.

Kroenke (2007) describes information systems as a composition of five basic elements that operate interpedently: "*software, hardware, data, processes and people*". Fernandez and Lehmann (2011) states that these 5 interrelated pieces all together are creating a "*package*" that involving both technical and social issues within. They argue that "*the Grounded Theory Method (GTM) was at its inception explicitly developed for research about the interactions of individual human actors in predominantly social settings and that applying the method to IS research with its mixture of human and technical constituent elements is therefore needed*".

This qualitative research was conducted in a Turkish defense systems company context for developing an hypothesis to draw attention to main success factors and to help lessen the difficulties and failures faced during software development projects. Within the context of this study, the failure of software may mean either a cancelled software development project or a project not meeting either one of the time, budget or quality requirements.

The purpose of this study is as Fernandez and Lehmann (2011) suggested, investigating the software domain by using one of the most appropriate methods, Grounded Theory methodology. Grounded Theory is simply described as "*building theories from data about the social world such that theories are 'grounded' in people's everyday experiences and actions*" by Knigge and Cope (2006). The methodology uses mainly observations and/or interviews as raw data and step by step systematically brings out the essence of the expert experience and events on the research area, described by means of words written on paper. The methodology process is presented in detail in Chapters 3 and 4.

1.5. Significance of the Study

As Aladwani (2004) stated in his paper; the researchers in the Information Systems field have been continuously trying to find out answers to their questions at "*either system or organizational levels*". These efforts brought out several different approaches to the problem and many different models and methodologies to achieve success at developing quality software (Remus & Wiener, 2008; Geetha, Kumar, & Kant, 2007; Sharp et al., 2005; Ewusi-Mensah, 1997; Boehm, 1988; Brooks, 1986).

It is understood from the literature that previous researches and studies that human beings' interaction and social environment issues are very important to consider while conducting such a research on software development projects (Morgenshtern et al., 2007; Aladwani, 2004). No matter which methodology or model is being used for managing a software project, it is considered crucial to include not only quantitative data to the decision process but also qualitative approaches even to make better, more realistic forecasts on the manpower needed for the project or the schedule (Morgenshtern et al., 2007; Sharp et al., 2005; Aladwani, 2004). The intended findings of this study should support management to make better plans and estimations.

Sharp et al. (2005) also stated that the revealing of social issues faced during software development projects can be achieved by using qualitative methods.

Lehmann (2010) drew attention to the dominance of quantitative methods even while conducting social science studies all along the literature whereas Adolph et al. (2011) supported that claim by revealing that "*a survey of the software engineering research*

literature (Glass et al., 2002) discovered that a very small percentage (in many cases less than 1%) of research papers explored organizational issues or employed research methods useful in understanding social behavior".

Mcleod et al. (2011) states that "*a recent meta-review of prior empirical research on software systems development project outcomes reveals that quantitative factor-based studies form the basis of much of the current knowledge in this area*". They claim that quantitative studies are good for identifying the "*causal influence of a set of contingent conditions or variables on outcomes*" however; they are not enough by themselves to explain the underlying problems in the software development field that is blended a lot with human-nature related outcomes and there is a need for methodologies revealing a deeper set of data about what is really going on.

Despite many researchers' conclusion on the need for more qualitative studies conducted about software development processes, still it's often reported that, qualitative methods are labeled as relatively rarely preferred compared to quantitative methods in software development research (McLeod et al., 2011; Dittrich, Rönkkö, Eriksson, Hansson & Lindeberg, 2008; Glass et al., 2002). The reason that quantitative methods are relatively more frequently adopted for such research is explained by the fact that software development is an engineering area and especially researchers coming from technical backgrounds are more comfortable with quantitative data (McLeod et al., 2011; Dittrich et al., 2008; Seaman,1999).

However, as Seaman (1999) described it software development is an "*awkward intersection of machine and human capabilities*" and involves "*central role of human behavior*" thus, should be investigated using methods that can reveal out that complex behavioral and human related issues (Dybå et al., 2008; Glass et al., 2002).

The aim of this qualitative study is to come up with a set of principles for software project shareholders indicating what factors and events should be considered before and during the software development processes. As described briefly in Chapter 1.4., one of the widely used qualitative methods, Grounded Theory Methodology is preferred to help analyzing and coding the raw data gathered from real life settings, and finally bringing out the hypothesis on the subject of interest from those analyses (Fernandez & Lehmann, 2011; Virili & Sorrentino, 2010).

Grounded Theory was introduced first by Glaser and Strauss in 1967 (Kulkarni, 2012). The methodology offers mainly shaping raw data into a meaningful hypothesis by conducting the open, axial and selective coding phases; where open coding is the categorization of paragraphs to sentences and comparison of the findings in order to come up with all the themes buried in textual data, axial coding refers to the narrowing down of those themes into categories and defining a "core category" which is the main influencer of the subject and selective coding phase is the one where the researcher analyze the whole raw data purely by his/her perception and codes it again from the beginning in accordance with the "core category". To sum up, the open and axial coding parts reveals out the main interest area (the core category) of the research and the final part, selective coding shapes the raw data into hypothesis by keeping that main interest area as the focus of the study.

Strauss and Corbin (1998) suggest that Grounded Theory Methodology takes the following assumptions –which are all adoptable to software issues or even matching with the very definition and the nature of the software development practices– as a baseline to conduct researches:

- *The need to go to the field to discover what is really going on.*
- *The complexity and variability of phenomena and of human action.*

- *The belief that persons are actors responding to problematic situations.*
- *The assumption that persons act on the basis of meaning.*
- *The understanding that meaning is defined and redefined through interaction.*
- *An awareness of the interrelationships among conditions (structure), action (process), and consequences*.

Since the researcher of this study is an employee of a well known defense industry company in Turkey that has been operating for the last few decades and has been deeply involved in software development projects, the preferred method is to gather experience of the people engaged in software projects in this organization by means of questionnaire and semi-structured interviews. The questionnaire is preferred for collecting numeric data about software success factors in order to help supporting and validating the findings of this study as depicted in Chapter 4. A semi-structured interview was chosen in order to keep interviewees within some sort of a boundary of the subject while also letting them run free around that field.

CHAPTER 2

LITERATURE REVIEW

2.1. Introduction

Living in the information age, software is everywhere in human beings' lives (Mei et al., 2006; Wang & Liu, 2006; Charette, 2005; Aladwani, 2004). Yet, problems still occur in management and completion of software development projects (Evelens & Verhoef, 2008; Wang & Liu; Aladwani).

The literature review aims to provide a complete understanding and review of previous findings about success and failure factors affecting the software development projects.

This section of this study reviews the literature under 3 headings:

- Software Development Projects Failure Factors
- Software Development Projects Success Factors
- Synthesis of the Related Literature

2.2. Software Development Projects Failure Factors

The Standish Group's periodically updated Chaos Report claimed in 2009 that 32% of the examined software projects were successful, while 44% were classified as challenged and the remaining 24% failed (Wright & Capps III, 2010). Despite studies can be found in the literature that discusses the reliability of Chaos Report findings such as the paper written by Evelens and Verhoef (2008), it is a common knowledge now that software project failures are there and are making life difficult for many people and organizations participating in this area (Martin et al., 2007; Wang & Liu,2006). Even the opponents of Standish Group figures realize the widely acceptance and usage of those figures as references in the literature (Eveleens & C. Verhoef, 2008).

The findings of Ewusi-Mensah (1997) indicate failure factors for software projects as follows:

- First very critical factor is misinterpreted goals or purposes that the stakeholders did not reach a conclusion.
- Another issue arises by forming a software development team composed of problematic members.
- Lack of ability and experience of the project management and control mechanisms; a management that cannot evaluate the progress of the project and weak on critical decision making does not work well for the project.
- Low capability of the development team members; the experience and technical know-how of the team is important.
- Whether the technical adequacy and infrastructure of the organization that manages the software development is sufficient does matter.
- Lacking of top management support and insufficient involvement in the project causes problems about the control of the project and important managerial decisions in this case are made by incapable people.
- Delays in the completion of the project or exceeding the budget of the project because of problems in planning and control.

Wang and Liu (2006) on the other hand focused on the "uncertainty" problem faced during software projects. The so called uncertainty is attributed to the factors:

- "Size of the project" (as the size grows, problems grow along with it)
- "Project team skill" (it is hard to control or define the technical and social abilities, as well as the experience of development team members)
- "User-client experience" (the end users' being experienced and knowledgeable in the area is important in order to both identify software requirements well and provide necessary and useful feedback during the project, which is usually not the case)
- "Technological complexity" (as the technological complexity of the software increases, which is inevitable with the increasing technological need and demand, the difficulties to develop the software also increases)

Morgenshtern et al. (2007) emphasize the importance of the estimation errors made on task duration and effort. They conclude that the duration and effort estimations are used for many different and important reasons such as control and monitoring of the project, team members selection, project and development team performance evaluation and many other issues; plus since the estimations are made at the very beginning of a software development project, they affect the whole process.

Other studies tend to point out the absence of an approach that takes human related issues into consideration. Aladwani (2004) declared lack of attention to the social environment wherein the development is made and emphasizes that without looking at the contextual structure and characteristics, one cannot define the factors affecting the development efforts and act accordingly.

According to DeMarco and Lister (1999) within any software development project organization since the beginning of the Information Systems era, there was never a technical issue to explain a project's failure, and that it has always been human related problems that caused projects fail. Leung (2001) also emphasizes the effects of social-human related issues on software development projects' failures.

Verner, Sampson and Cerpa (2008) in their study conducted to find out most common software development project failure factors, first identified the most famous failure factors discussed in the literature as follows:

- unsuccessful or inappropriate organizational structure,
- unrealistic or unarticulated goals,

- software that fails to meet the real business needs,
- badly defined system requirements, user requirements and requirements specification,
- the project management process, poor project management,
- software development methodologies, sloppy development practices,
- inadequate scheduling and project budget,
- inaccurate estimates of needed resources,
- poor reporting of the project status,
- inability to handle project complexity, unmanaged risks,
- poor communication among customers, developers and users,
- use of immature technology,
- stakeholder politics conflicts,
- commercial pressures,
- customer dissatisfaction,
- product not meeting quality requirements,
- unsuccessful leadership, lacking upper management support,
- personality conflicts,
- inappropriate business processes and resources,
- poor, or no tracking tools.

After the research was conducted; they came up with some similar but more specific findings which are:

- "The delivery date of the software impacted the development process; 93% of the failed projects;"
- "The project was underestimated; 81% of the failed projects;"
- "Risks were not re-assessed, controlled, or managed through the project; 76% of the failed projects;"
- "Staff were not rewarded for working long hours.; 73% of the failed projects;"
- "Delivery decision was made without adequate requirements information; 73% of the failed projects;"
- "Staff had an unpleasant experience working on the project; 73% of failed projects."(Verner et al., 2008).

Similarly Charette (2005) listed the common failure factors as;

- Unrealistic or unarticulated project goals
- Inaccurate estimates of needed resources
- Badly defined system requirements
- Poor reporting of the project's status
- Unmanaged risks
- Poor communication among customers, developers, and users
- Use of immature technology
- Inability to handle the project's complexity
- Sloppy development practices
- Poor project management
- Stakeholder politics
- Commercial pressures

According to Martin et al. (2007) software development projects are becoming more and more important for organizations because of the growth in technology and the effects of globalization; and to ensure success in those projects, organizations should take responsibility and further research especially on human related issues regarding software projects is necessary.

Brown (1990) in his article which was written especially for safety-critical systems software, focuses on the well established software development standards in order to be able to prevent disastrous software errors.

Ahonen and Salovainen (2010) conducted a case-study research where they analyzed five different cancelled software development projects (four of them were canceled even before the requirements definition phase) in their study in order to reveal the failure factors. The findings are as follows:
Case 1: Software supplier failed to understand the actual needs of user/client.
Case 2: The software development team was consisting of inexperienced members.
Case 3: The supplier took a great risk by agreeing on unrealistic scheduling requested by user/client.
Case 4: The project was an in-house software development project; the cancellation was not until the coding phase where an architectural design mistake was made.
Case 5: The project was upgrading of existing software, the supplier tried to complete the project without fully understanding the technical problems and shortfall of the old version.

2.3. Software Development Projects Success Factors

The success of a software project is usually evaluated according to three factors; cost of the project, the time taken to complete the project and the quality of the software that are typically determined in terms of requirements where the quality of software can be defined by its properties such usability, reliability, maintainability, integrity and reusability (Subramanian, 2007; Martin et al.,2007; Aladwani, 2004). In order to meet these constraints; the communication and tight collaboration of all the stakeholders involved in the project are very important (Procaccino, Verner, Overmyer, & Darter; 2002). Defined by Ewusi-Mensah (1997) the so called stakeholders getting involved in a software development project are the software development team, management team and the client or end user of the software.

Another factor that is highlighted many times in the literature is the size of the project; as the project size gets bigger, not only because of the technical issues like actually writing the code, but also because the coordination within project stakeholders is harder, the problems also become bigger and more complex (Morgenshtern et al., 2007). Yet; despite the suggestions that the size of the project has a negative impact on the success of software development projects not only because of the technical issues but also because the project team gets bigger and harder to coordinate (Morgenshtern et al., Wang & Liu, 2006). Aladwani (2004) claims the opposite defending that several studies prove to indicate success on problem solving within larger project development teams.

A study conducted by Wang and Liu (2006) suggests that the software development projects' performance is improved by means of "planning and control, internal integration, user participation and top management support". The study also suggests that these factors and other important factors affecting the projects are influencing the project at different steps of the development process; for example the user participation is stated to be most effective during the implementation of software.

According to McAvoy and Butler (2009) the very first thing to consider for the success of a software project is "the individuals" rather than the methods and tools, whereas the study conducted by Geetha et al. (2007) emphasizes the importance of performance evaluation during and especially before the software development process on producing successful software. Also risk management efforts are considered key factors enabling successful software development (Subramanian et al., 2007; Boehm, 1988).

Another set of key influencing inputs for successful software development is suggested by Aladwani (2004); "the use of support technology, project team size, clear goals, expertise of staff, and management advocacy". In his research, Aladwani proposed an integrated model to increase software development performance; including these key influencing factors and added "However, there is still a need for subsequent effort to refine the proposed theory until it results in a set of guidelines that can be transferred to operational settings".

Sharp et al. (2005) suggested that software development projects' success depends highly on quality management systems of organizations, and defines the project environment as consisting of the development team on one side and the quality management on the other side. They propose the following statements for successful quality management process and thus successful projects:

- it is operating in an atmosphere of mutual respect, friendship and trust.
- the process is comprehensive and its operation is not reliant on any one individual or group of individuals.
- a suitable balance of power is maintained between the process and the developers, i.e., neither one is in charge at all times, but both are seen as having authority.
- the quality process and the developers are both entrusted to achieve what they are best suited to, i.e., the quality process should allow developers to make decisions for specific situations, while the quality process should maintain suitable rules and regulations.
- the quality process is not founded exclusively on logical, rational grounds, but integrates pragmatism as well.

Another study by Subramanian et al. (2007) evaluates four software development strategies that are developed by Alter in 1979 which are;

- *"Keep it simple"*
- *"Executive (or top management) participation and commitment"*
- *"Training"*
- *"Prototyping/evolutionary development"*

The outcome of the study in summary supports Alter and also other findings in the literature according to the researchers. "Executive (or top management) participation and commitment" and "Prototyping / evolutionary development" strategies are said to have significant positive effects on both increased software quality and project performance constraints. Whereas the "Keep it simple" strategy has an impact on better project performance only, "Training" of the project team strategy is claimed to provide higher software development project performance.

Wright and Capps III (2010) suggested that commitment by management to the software project is another important factor on success. They claim four types of factors that define the project commitment concept:

- *"Project":* The project factors include the pros and cons that are perceived by management, in this case the benefit of software versus the cost of it.
- *"Psychological":* The psychological factors are the ones that make managers mistakenly believe that the project will end up successfully anyway; depending on the previous experiences and feelings of responsibility of management.
- *"Social":* Social factors affect management particularly in situations where there is competition within or out of the organization; causing project management a need or wish to prove themselves.
- *"Organizational":* The organizational factors define management commitment by means of the internal politics and culture.

Mei et al. (2006) suggests software reuse -that is defined by producing some standard parts of code and combining them within different programs according to the need- is a promising

and successful method for increasing software development performance rather than developing completely new codes each time.

Lu, Xiang, Wang and Wang, (2011) states that of many other aspects, the role of software development team is the most critical one to take into consideration because of the nature of software development process and because it is highly affected by the people in the development team and inner team dynamics.

Tesch, Kloppenborg and Frolick (2007) also partly focuses on the importance of software development team by declaring that team members should have clear responsibilities and goals supported by a "roles matrix", the outcomes and the success criteria should be defined beforehand. They also add that top management support and user/client involvement to the project are important factors for the success of a software development project. Whereas Rahman, Rahim and Chen (2012) focuses their efforts on risk management in order to achieve success on software development projects.

Most of the studies conducted about software development processes concluded that a certain type of management methodology or a model such as Waterfall Model, Incremental Development Model, Evolutionary Development Model, Spiral Model or application of principles such as CMMI, TQM, Six Sigma helps development of quality software meeting the time and budget requirements (Adams, 2007; Rehman & Hussain, 2007; Subramanian et al., 2007; Martin et al., 2007; Brooks, 1986). Yet, many agree on the conclusion that there is no such thing as a one and certain salvation for every software project, but every project according to its nature should be treated with a different management approach (Adams, 2007; Rehman & Hussain, 2007; Brooks, 1986). In fact; aside from supporting the use of different management mechanisms and tool for different needs, McBride (2008) depending on the findings of his study, indicates a certain fact that project managers with more experience in the software development area prefer to use multiple methods and tools even within the same project for monitor and control purposes.

2.4. Synthesis of the Related Literature

Software is simply everywhere as part of our lives. Its growing importance despite huge problems and even disasters happening because of failing software or software projects lead many academicians and scientists to conduct research in this field. However, becoming even more and more important, the software area is still full of mysteries and unsolved conflicts.

The problems with software development projects are obvious and becoming more and more important with the growth of technology (Wright & Capps III, 2010; Adams, 2007; Martin, et al., 2007; Yeo, 2002). There are many studies and research conducted in this area to find out success and failure factors but still there is not a common place that every research ends up on (Evelens & Verhoef, 2008; Wang & Liu, 2006; Aladwani, 2004).

Many researchers and academicians agree that after more than 50 years of study and field experience on software development, still there is not a clear, certain way how to do things right (Savolainen, Ahonen & Richardson, 2011; Wright & Capps III, 2010; Adams, 2007; Martin et al., 2007; Aladwani, 2004; Yeo, 2002; Gibbs,1994).

While some researchers focus on well established standards, others emphasize the importance of software development team or successful risk management whereas many others combine these with other factors that are the outcomes of their studies.

The famous article of Brooks published in 1986 with the name "No Silver Bullet: Essence and Accidents of Software Engineering" reveals that a single best approach to develop

software does not exist and each software development project is unique in nature, thus should be treated accordingly, with different techniques and tools. This statement still finds support by other younger researchers (Adams; Rehman & Hussain, 2007; Mei et al., 2006). After conducting the literature review the researcher of this study also concludes that, while each of those factors that were determined through the years have some sort of impact on the course of events while developing software; software development practices and the main success factors may –and should- vary according to the nature of software developed. Nature of the project and its level of complexity may change according to the industry branch it is being developed for, according to being a purely new or an already existing technology and such. Actually, though each software itself is "unique" in nature, because of the high human interaction within software development process, it may even be affected by the culture of the organizations, and the region or country they are originating from.

The very basic finding of the literature review is that As McLeod and MacDonell (2011) suggested there are many researchers to claim similar and different factors that affect software development processes, and none of them can be labeled as completely right or wrong. In fact none of the researchers declares one factor being the only to affect the software development process or another factor being not relevant to it; but most of the studies are trying to figure out the most important factor within the environment the research was conducted. These studies (three books and forty three articles in the related literature which were published within the years 1990 and 2012) are analyzed for the success and failure factors. They are retrieved and combined in a list to conduct a questionnaire that is represented in Table 2-1 along with their definitions as perceived by the researcher - in order to measure their importance in the organization of research. Also the questionnaire was used as a template for the semi-structured interviews, to be able to keep the subject together while reminding the possible important factors to the interviewees.

Table 2-1 : Factors Retrieved From Literature

Factors Affecting Software Development Projects	Definition
top management support	: both moral and material support to project team by upper level management
capable decision makers	: successful decisions given not only by project manager but also by team members
existence and success of project control mechanisms	: tools and mechanisms developed to inspect the project evaluation
performance evaluation	: measuring the team members' performance by means of predefined criteria
technical infrastructure	: the tools and infrastructure used for developing software
estimations on duration and task effort	: the pre-defining of time and effort needed to complete the project
software reuse	: the reuse of any software development process such as code blocks, requirements definitions or test documents through various projects
company principles or standards	: a group or map of standards institutionally adopted by the company such as CMMI or TQM
the use of a certain development methodology	: a methodology such as waterfall, incremental or evolutionary development models
quality management all along the SW development process	: the monitoring of project steps by people other than the development team
team members' personal attributes and attitudes	: social characteristics of development team members that will affect relationships between them
experience and technical know-how of team members	: the years spent on software development and level of technical know-how of team members
training of team members	: additional training programs for team members other than their educational background
rewarded, happy staff	: keeping development team members satisfied and motivated
user/client experience	: the depth of know-how of user/client about the software development
user participation, communication and collaboration among them and the developers	: constantly receiving feedback from user/client and involving them to the progress
requirements definition	: solid and clear definition of requirements either by user/client and the development team
user/client patches	: user/client's additional requirements popping up throughout the project
size of the project	: the size of the project regarding its context
technological complexity of the project	: the technological complexity of the project regarding its context
different goals or purposes of stakeholders	: project shareholders (such as user/client, top management, development team members) having different aims throughout the project

CHAPTER 3

RESEARCH METHODOLOGY

3.1. Research Model Definition

This part of the study focuses on the research methodology and the principles of the methodology.

This study is conducted for transforming the tacit knowledge developed in one of the main players in Turkish defense and systems/software development sector, into explicit knowledge.

In order to conduct this study, a qualitative methodology is chosen. The reason to specifically conduct a qualitative method lies under the findings of the literature in the area. The subject of this study is proven to be highly affected by human related issues, and a qualitative method is considered to be the best way to discover human-hand effect on the researched subject (Clarke & O'Connor, 2012; Dybå, et al., 2011; Fernandez & Lehmann, 2011; Virili & Sorrentino, 2010; Aladwani, 2004). It is also observed from the literature that qualitative studies have been – and should be – gaining more and more importance in the software field (McLeod & MacDonell, 2011; Sharp et al., 2005; Seaman, 1999).

Seaman (1999) describes the consistence of conducting qualitative research methods for software engineering very simply and nicely with his words: "*Qualitative results often are considered 'softer' or 'fuzzier' than quantitative results, especially in technical communities like ours. They are more difficult to summarize or simplify. But then, so are the problems we study in software engineering.*".

Dybå et al. (2011) and Seaman (1999) declares that the main advantage of conducting qualitative research is that "*they force the researcher to delve into the complexity of the problem rather than abstract it away. Thus, the results are richer andmore informative. There are drawbacks, however. Qualitative analysis is generally more labor-intensive and exhausting than quantitative analysis*".

The preferred method to conduct this qualitative research is Grounded Theory Methodology. Adolph et al. (2011) suggest that especially while conducting qualitative research, compared with other methodologies Grounded Theory is more efficient in order to produce theory that reveals the contributors' attitude as an integrated system of actions. Such an explanation of the research method leaded the researcher of this study to realize how appropriate it is to use

Grounded Theory for producing data from the experience of people engaged in software development projects for many years.

The qualitative grounded theory methodology aims improvement of previous theories and models, and development of new theories depending on the findings captured from the people and organizations that have experience in the study field. (Glaser & Strauss, 2007). The researcher of the present study is employed by a defense systems company which has been involved in software development projects for many (approximately 30) years and she had easy access to the people that have experience on the field. Thus, along with the appropriateness of Grounded Theory Methodology for investigating software development efforts; also the proximity to the source of raw data leaded the researcher to conduct the methodology by making interviews.

Strauss and Corbin (1998) suggest that Grounded Theory Methodology takes the following assumptions –which are all adoptable to software issues or even matching with the very definition and the nature of the software development practices– as a baseline to conduct researches:

- *The need to go to the field to discover what is really going on.*
    - The aim of this research is to develop a set of principles on what the main success factor is for a software development project.
- *The complexity and variability of phenomena and of human action.*
    - Software development is after 50 years of study, still considered "messy" and "complicated" and involves high human interaction.
- *The belief that persons are actors responding to problematic situations.*
- *The assumption that persons act on the basis of meaning.*
- *The understanding that meaning is defined and redefined through interaction.*
    - The three above assumptions are adopted also by this research, meaning that people engaged in software development projects do not ignore problems on the way but try to solve them, they do what is defined as logical and right to solve those problems; and finally there is a common sense and knowledge on what is logical and right.
- *An awareness of the interrelationships among conditions (structure), action (process), and consequences*.
    - This assumption is also considered true for this study meaning that software development projects consist of a chain of actions and reactions where the environment along with people's decisions and actions cause results and affect situations.

Grounded Theory Methodology was first introduced by Glaser and Strauss in 1967 (Hoda, Noble & Marshall, 2010). The name of the methodology defines the process itself which is systematically bringing the "theory" out that is "grounded" within the raw data (Kulkarni, 2012).

One of the fathers of the methodology, Strauss in his study along with Corbin (1998) suggested that the goals listed below can be achieved by grounded theory:

"1. Build rather than test theory.
2. Provide researchers with analytic tools for handling raw data.
3. Help the analysts to consider alternative meanings of phenomena.
4. Be systematic and creative simultaneously.
5. Identify, develop, and relate the concepts that are the building blocks of theory."

The three main steps of the method are: open, axial and selective coding. The first two steps are the ones where for this study; the researcher digs into the interviews to identify

categories since the raw data consists of what the interviews told about the subject. First in the open coding part, the concepts buried in the interviews are categorized and labeled. An example of the open coding phase is presented below:

> **Interviewee:** *"bir projenin böyle sağlıklı ilerlemesi için öncelikle işte proje yöneticisi, işte sistemin başındaki adam, yazılımın başındaki adam, kimler tarafından yapılacaksa bu iş, bunların oturup birbirlerine dependencylerini de göz önüne alarak bir plan çıkartmaları lazım bir. Sonra bu planı sürekli monitor etmeleri ve update etmeleri lazım çünkü hiçbir zaman özellikle bizim projelerde tam olarak hani, inşaat projesi değil şu kadar zamanda olur bu iş diyemiyorsun, beklenmedik sıkıntılar olabiliyor vesaire o yüzden de bu planın sürekli update edilmesi ona göre de önlemler alınması gerekli."*
>
> **Code derived from the paragraph:** A living plan, updated at all phases

After applying this principle to whole data, the researcher has a list of codes which may be seen as a summary of the interviews. In the next phase, axial coding, the researcher takes all these codes and compares them constantly with each other in order to eliminate duplications and create categories from the codes until no new concepts or categories can be gathered out from the codes. The code derived in the above example, with other codes similar to it by means of what they symbolize, are gathered together and build up the final category as "Processes" of the software development project.

After completing the axial coding, the researcher now has a saturated summary of pages of written data buried in a few or more categories. Before conducting selective coding, the "main" category of the research should be identified which is called the "core category" that should come out as the hidden main concern of the interviewees and has a direct relationship with all the other categories defined (Kulkarni, 2012 Hoda et al., 2010; Virili & Sorrentino, 2010; Glaser & Strauss, 2007; Strauss & Corbin, 1998). To choose the category, the criteria that were defined by Strauss and Corbin (1998) were used. According to those criteria the core category should have the following characteristics:

- *"It must be central; that is, all other major categories can be related to it.*
- *It must appear frequently in the data. This means that within all or almost all cases, there are indicators pointing to that concept.*
- *The explanation that evolves by relating the categories is logical and consistent. There is no forcing of data.*
- *The name or phrase used to describe the central category should be sufficiently abstract that it can be used to do research in other substantive areas, leading to the development of a more general theory.*
- *As the concept is refined analytically through the integration with other concepts, the theory grows in depth and explanatory power.*
- *The concept is able to explain variation as well as the main point made by the data; that is, when conditions vary, the explanations still hold, although the way in which a phenomenon is expressed might look somewhat different. One also should be able to explain contradictory or alternative cases in terms of that central idea."*

The final step is the selective coding phase where at the end of this phase the hypothesis is built. In this part the researcher reviews the interview again from the very beginning, but this time keeping the core category in his/her mind. That is to say, the coding process shall be made again but the interpretation of the data should be related with the core category at all times. This process will shape the data according to the core category which is determined as

the most important factor in the interviews and bring out the words to build the set of principles.

According to Avison and Myers (2005) grounded theory approaches have been becoming quite popular in the IS research literature especially "because the method is extremely useful in developing context-based, process-oriented descriptions and explanations of the phenomenon under study". Another researcher, Lehmann (2010) agrees on this idea and claims he is very surprised that here are still debates on the question whether Grounded Theory Methodology is suitable for information systems field research; while it obviously is because information systems should be identified as a social discipline.

In order to achieve data collection to conduct the Grounded Theory research, a semi-structured interview is used for the qualitative grounded theory research. Before conducting the interviews, to support the informants by leading them to think about the subject beforehand, success and failure factors in software development projects is derived from the reviewed three books and forty three articles in the related literature which were published within the years 1997 and 2012. As presented in Table 3-1, the factors that are affecting the success of software development projects are given to the informants (including the interviewees) of this study as a questionnaire.

Table 3-1 Factors Affecting Software Development

Factors Affecting Software Development Projects	Importance (1-10) (1 being the least and 10 being the most important)	Importance (1-10) 2nd Iteration
1. top management support		
2. capable decision makers (strong project management)		
3. existence and success of project control mechanisms		
4. performance evaluation		
5. technical infrastructure		
6. estimations on duration and task effort		
7. software reuse		
8. company principles or standards (CMMI, TQM...)		
9. the use of a certain SW development methodology (waterfall, incremental, evolutionary)		
10. quality management all along the SW development process		
11. team members' personal attributes and attitudes		
12. experience and technical know-how of team members		
13. training of team members		
14. rewarded, happy staff		
15. user/client experience		
16. user participation, communication and collaboration among them and the developers		
17. requirements definition		
18. user/client patches		
19. size of the project		
20. technological complexity of the project		
21. different goals or purposes of stakeholders		
OTHER		

The previous list was introduced to 20 voluntary participants by the researcher to conduct a Delphi research; in order for them to rate those factors according to how important they are, regarding the success or failure of software development projects. The importance of the factors change between 1 and 10 points, where 1 point means the least and 10 point means the most important. The results of the questionnaire will provide numerical data to compare with the findings of Grounded Theory methodology.

Since the rankings were stable at the end of the second round of the Delphi study, it was considered enough to gather saturated information. The results of the Delphi survey are given in Appendix A.

The Delphi methodology is suggested quite appropriate for masters and PhD studies where the aim is to develop a new understanding of current phenomena and the iterations on the survey are proposed to be stopped when saturation occurs within the data gathered from different participants (Skulmoski, Hartman & Krahn, 2007).
Figure 3-1 displays a sample of 3 rounded Delphi process (Skulmoski et al.).

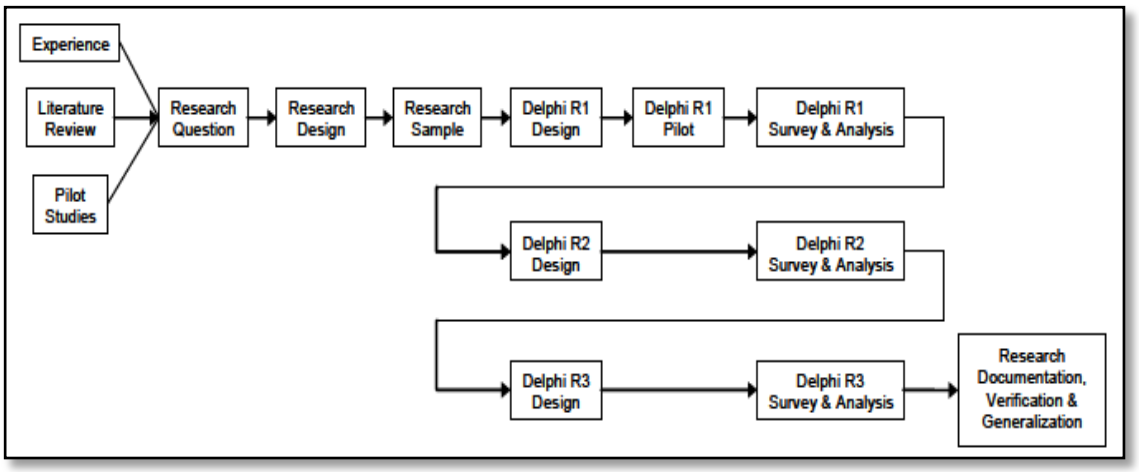

Figure 3-1 Delphi Process (Skulmoski et al., 2007, p.3)

In the research, the factors gathered from the literature review were used for preparing a questionnaire where the participants ranked all factors from 1 to 10 (1 being the least and 10 being the most important). The list and the factor rankings were given in Appendix A. The survey was made in 2 iterations, the data saturation was assumed to be achieved and participants claimed no necessity for further iterations. The interviews were made both depending on the research questions and the results of the Delphi study. The interviews (made and written in Turkish) are presented in Appendix B.

Skulmoski et al. (2007) gives out other Information Systems/Information Technology studies that were conducted by using the Delphi survey in Table 3-2.:

Table 3-2 IS/IT Studies That Used Delphi Method (Skulmoski et al., 2007, p.6)

IS/IT Study	Delphi Focus	Rounds	Sample Size
Niederman, Brancheau, & Wetherbe, (1991)	Survey senior IS executives to determine the most critical IS issues for the 1990s.	3	114, 126 & 104
Duncan (1995)	Identify and rank the critical elements of IS infrastructure flexibility.	2	21
Brancheau, Janz, & Wetherbe (1996)	Survey SIM members to determine the most critical IS issues for the near future.	3	78, 87 & 76
Nambisan et al. (1999)	Develop a taxonomy of knowledge creation mechanisms.	3	11
Scott (2000)	Rank technology management issues in new product development projects	3	20
Wynekoop & Walz (2000)	Rank the most important characteristics of high performing IT personnel.	3	9
R. Schmidt, Lyytinen, Keil, & Cule (2001)	Identify and rank software development project risks: an international comparative study.	3	Finland: 13, 13, & 13 Hong Kong: 11, 11 & 9 USA 21, 21 & 9
Keil, Tiwana, & Bush (2002)	Rank software development project risks.	3	15, 15 & 10
Brungs & Jamieson (2005)	Identify and rank computer forensics legal issues.	3	11

The purpose of the researcher by conducting a semi-structured interview after the questionnaire is to let the interviewees talk independently while also keeping them within the borders and the scope of the study and reminding them the possible success factors.

3.2. Participants

The targeted participants of this study are software engineers, computer engineers who participate in software development projects and software project managers in a particular organization engaged in defense systems industry in Turkey. The company has a CMMI Level-3 certification. The participants involved in both the Delphi process and the interviews are kept anonymous in this research.

In grounded theory research, the researcher seeks for data saturation to stop collecting or evaluating any more input (Creswell, 2008). The sample size that the data shall be gathered from is usually small in number (Creswell; Suzuki, Ahluwalia, Arora, & Mattis, 2007). Thus, the invitations for the Delphi research and the unstructured interview are sent to the chosen departments and people in the selected organization, summing up to a total of 45 people who are involved in software development projects. The participation to the study is voluntary. Exactly 20 of those 45 senior and junior software engineers, team leaders and managers accepted to be part of the Delphi survey and 5 of them accepted to join the interviews.

All participants' job descriptions and their years of experience are given in Appendix A along with their rankings and are summarized in Table 3-3. The profile of interviewees among the participants is represented in Table 3-4:

Table 3-3 Participants' Profile

Job Description	No. of Participants	Years of Experience
Program Manager	1	15-20 years
Software Engineering Manager	1	15-20 years
Senior Software Engineer	5	10-15 years
Senior Software Engineer	2	5-10 years
Software Quality Engineer	2	10-15 years
Software Quality Engineer	1	5-10 years
Software Development Team Leader	1	15-20 years
Software Development Team Leader	2	10-15 years
Junior Software Engineer	5	1-5 years

Table 3-4 Interviewees' Profile

Job Description	Years of Experience
Interviewee 1 (Program Manager)	15-20 years
Interviewee 2 (Software Engineering Manager)	15-20 years
Interviewee 3 (Senior Software Engineer)	10-15 years
Interviewee 4 (Junior Software Engineer)	1-5 years
Interviewee 5 (Software Quality Engineer)	10-15 years

3.3. Instruments

In order to conduct this qualitative grounded theory research; using a semi-structured, face-to-face interview is preferred. The interviews with the informants are recorded to an audiotape. Choosing this approach provides both advantages and disadvantages. The biggest advantage of a face-to-face, semi-structured interview is that the researcher has the opportunity to ask clarifying questions and to expand the content of the subject, whereas this may as well turn into a disadvantage because of the need for expertise for the researcher herself.

The main topics included in the semi-structured interview are:

- What factors (including technological/human aspects and inside/outside the project development team) lead to software development project success?
- What factors (including technological/human aspects and inside/outside the project development team) lead to software development project failure?

Also, before the interview, a list of factors that affect software development efforts are retrieved from the literature (from three books and forty three articles in the related literature which were published within the years 1997 and 2012) and presented to the interviewees in order to keep the interviews within the boundaries of the subject while preventing missing any important factor that may be experienced by the interviewee but forgotten during the interview.
The researcher's knowledge on the subject depends on the Masters Program of Information Systems that she engaged in Middle East Technical University, 5 years of partial involvement in software development projects management in the organization that is chosen as the sample organization and the knowledge retrieved from the literature review.

3.4. Data Collection and Analysis

This section explains the methods used for data collection and analysis.

3.4.1.Data Collection

An invitation for the questionnaire and semi-structured interview was sent to a group of 45 people consisting of software engineers, computer engineers who participate in software development projects and software project managers in a particular organization engaged in defense systems industry in Turkey. The participation to the study is voluntary and kept anonymous.

Among the 45 people, 20 agreed on conducting the questionnaire by means of a two iteration Delphi research and 5 of these experts with different job descriptions and experience levels accepted to join the interviews.

The interviews with the volunteers were conducted face-to-face in the work place, during lunch time in their offices. The whole speeches are recorded on an audiotape. Following the completion of the interviews, the audio records were transferred to written documents for further analysis which are also presented in Appendix B.

3.4.2.Data Analysis

The factors affecting software development projects were retrieved from the literature in order to conduct a Delphi study

The recorded data, after documented on paper, was analyzed according to the principles of grounded theory. The grounded theory methodology follows three steps to transform the unstructured data into meaningful findings, in other words create knowledge from raw data. The three steps are open coding, axial coding and selective coding (Glaser & Strauss, 2007). At the first step, open coding; similar data are determined and labeled under different concepts (Rodrigues, 2010). Following the first step, in axial coding phase, the categorized and labeled concepts are compared with each other in order to form even more essential categories and within those refined categories, the core category is determined which represents the main concern of the source data (Rodrigues, 2010). Finally at the selective

coding phase, the interviews are coded by relating the interviews to the core category in order to come up with an hypothesis (Rodrigues, 2010).

The open and axial coding processes, in order to eliminate researcher bias was made separately by two other researchers; one of them was involved in software development projects for 5 years working in another defense sector company and the other one was an employee of the company that the research was conducted in and involved in project management, but not necessarily software projects for the last 7 years. Again for validation purposes, the open and axial coding process conducted by the researcher of this study were given to the interviewees as a list of codes and they were asked to match the codes with the themese and categories defined by the researcher. The comparisons are presented in section 4.2 Validity of Data Collection and Instruments.

3.5. Definition of Concepts and Terms Used in the Study

The definitions and terms listed below are the key concepts used in this study and may be used for meanings specific to this study.

Challenged project: "Challenged projects are completed and approved projects that are over budget, late, and with fewer features and functions than initially specified" (Dalcher & Benediktsson, 2006, p. 51).

Failed project: A project that is "canceled before completion, never implemented, or scrapped following installation" (Dalcher & Benediktsson, 2006, p. 51).
Software project: "A unique and integrated product that represents collections of realized objectives, solved problems, and obtained results of activity that satisfy the requirements of the customer of the project" (Lavrishcheva, 2008, p. 327).

Successful project: A project that is completed within predefined time, budget and quality constraints.

CHAPTER 4

RESULTS AND DISCUSSION

4.1. Findings of the Study

The Delphi and Grounded Theory study findings of this research are represented in order in this chapter.

Delphi study results were, compared to the literature review, not very surprising. The results for all twenty two factors (twenty one of them retrieved from the literature and one added by the research participants) are presented in Appendix A. The top five ranked factors for a successful software development project management are listed below in Table 4-1 and the least important ranked factors by the participants are listed below in Table 4-2:

Table 4-1 Factors Ranked As Most Important

Factors Affecting Software Development Projects	**Importance** (1-10) (1 being the least and 10 being the most important)
requirements definition	9,4
harmony (teamwork)	8,7
team members' personal attributes and attitudes	8,4
rewarded, happy staff	8,4
capable decision makers (project management as well as within the development team)	8,1

Table 4-2 Factors Ranked As Least Important

Factors Affecting Software Development Projects	**Importance** (1-10) (1 being the least and 10 being the most important)
different goals or purposes of stakeholders	6,6
performance evaluation	6,5
technical infrastructure	6,4
estimations on duration and task effort	6,4
size of the project	6,4

Delphi questionnaire was conducted for only two iterations because of the variances of rankings were too small to continue the rounds. The average results are represented below which involves rankings by 20 voluntary participants for both iterations in Table 4-3:

Table 4-3 Delphi Findings

Factors Affecting Software Development Projects	**AVG 1st round**	**AVG 2nd round**
1. top management support	7,1	7,2
2. capable decision makers (strong project management)	8,0	8,1
3. existence and success of project control mechanisms	6,8	6,8
4. performance evaluation	6,6	6,5
5. technical infrastructure	6,4	6,4
6. estimations on duration and task effort	6,4	6,4
7. software reuse	7,9	7,8
8. company principles or standards (CMMI, TQM...)	7,3	7,3
9. the use of a certain SW development methodology (waterfall, incremental, evolutionary)	6,7	6,9
10. quality management all along the SW development process	7,1	7,2
11. team members' personal attributes and attitudes	8,4	8,4
12. experience and technical know-how of team members	8,0	8,1
13. training of team members	7,9	7,9
14. rewarded, happy staff	8,3	8,4
15. user/client experience	6,7	6,8
16. user participation, communication and collaboration among them and the developers	7,1	7,0
17. requirements definition	9,4	9,4
18. user/client patches	7,0	7,1
19. size of the project	6,4	6,4
20. technological complexity of the project	6,8	6,9
21. different goals or purposes of stakeholders	6,7	6,6
OTHER		
Harmony (teamwork)	9,8	8,7

The raw data buried in the study was manipulated into meaningful concepts by using Grounded Theory Methodology. The findings gathered from open and axial coding processes conducted by the researcher are presented in Appendix C and depicted in Figure 4-1:

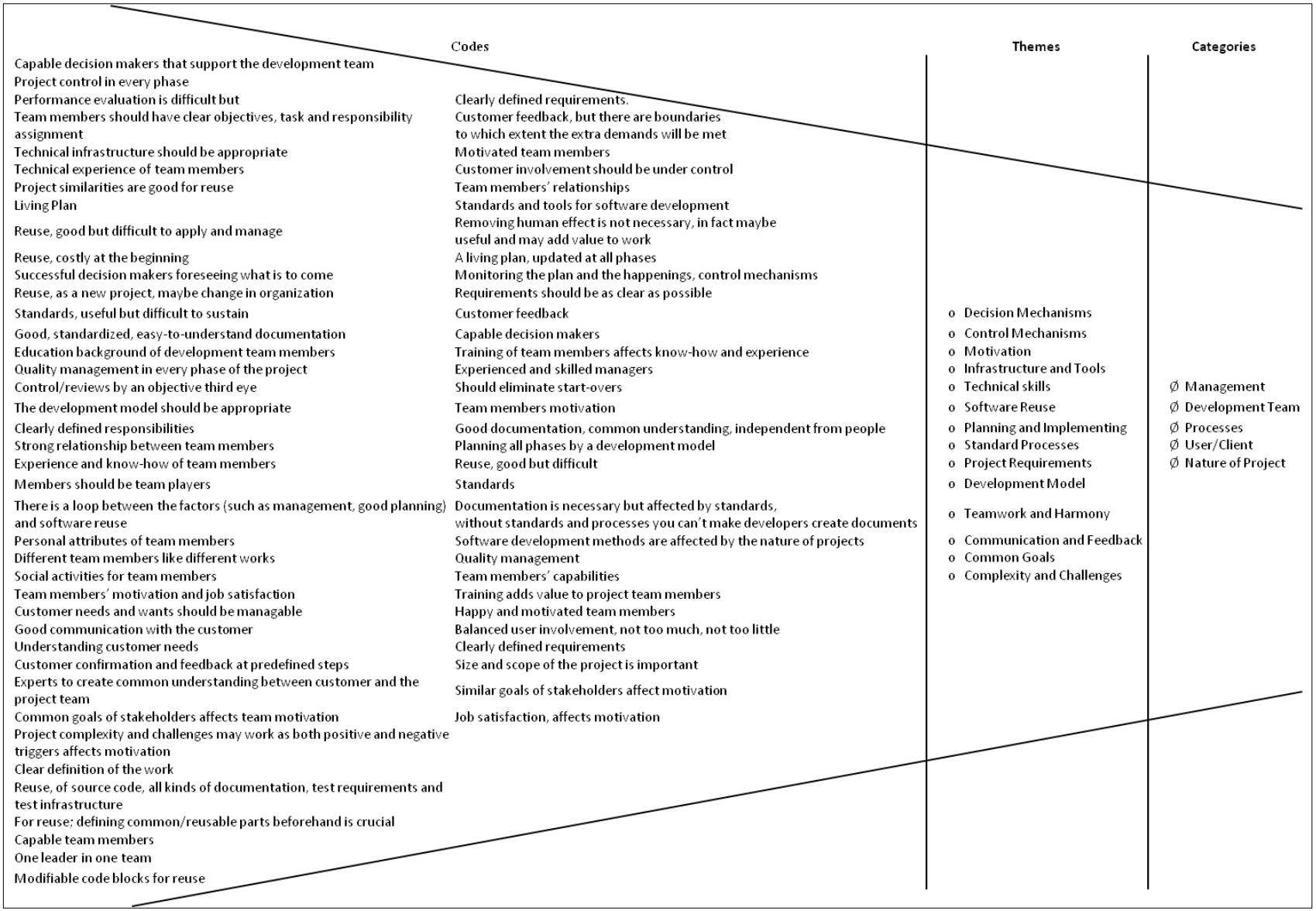

Figure 4-1 The Open and Axial Coding Process and Findings

For the software reuse concept, all interviewees agreed on how difficult it is create a good basis for reuse and it may be both expensive and exhausting at the beginning, yet it worth the burden and the system will pay back in a short while. Looking at the interviews, the reuse word is suggested as a successful tool for not only the source code but also for requirements definition, documentation, tests requirements and test infrastructure. The reuse concept is concluded to be meaningful only after a solid and healthy software development environment is created, thus beyond the scope of this study.

4.2. Validity of Data Collection and Instruments

Like Seaman pointed out in her study (1999) the software field and qualitative methods involve "fuzzy" terms and no matter how appropriate qualitative methods are for studying software efforts, they are also that difficult to validate and depict as research findings. There are ongoing debates on how to validate findings of such studies whereas some methods are concluded to be effective by many (Kulkarni, 2012; McLeod et al., 2011; Seaman, 1999).

Kulkarni (2012) declares that the main advantange of quantitative methods by means of validation is that the questionnaires and numerical data can be given in Appendice or within the study itself to illustrate the study but it is difficult to provide complete evidence for the study in qualitative ones. Thus he suggests that it is important to use diagrams and drawings to explain the essence of the study. The interviews, the coding processes followed by the researcher and two other experts, the matchings of codes, themes and categories by the interviews are all represented in the Appendix chapter or within the main body of this study. But in order to create a full understanding of the study, the summary of the processes in order to conduct this study, involving each step followed is represented in Figure 4-2:

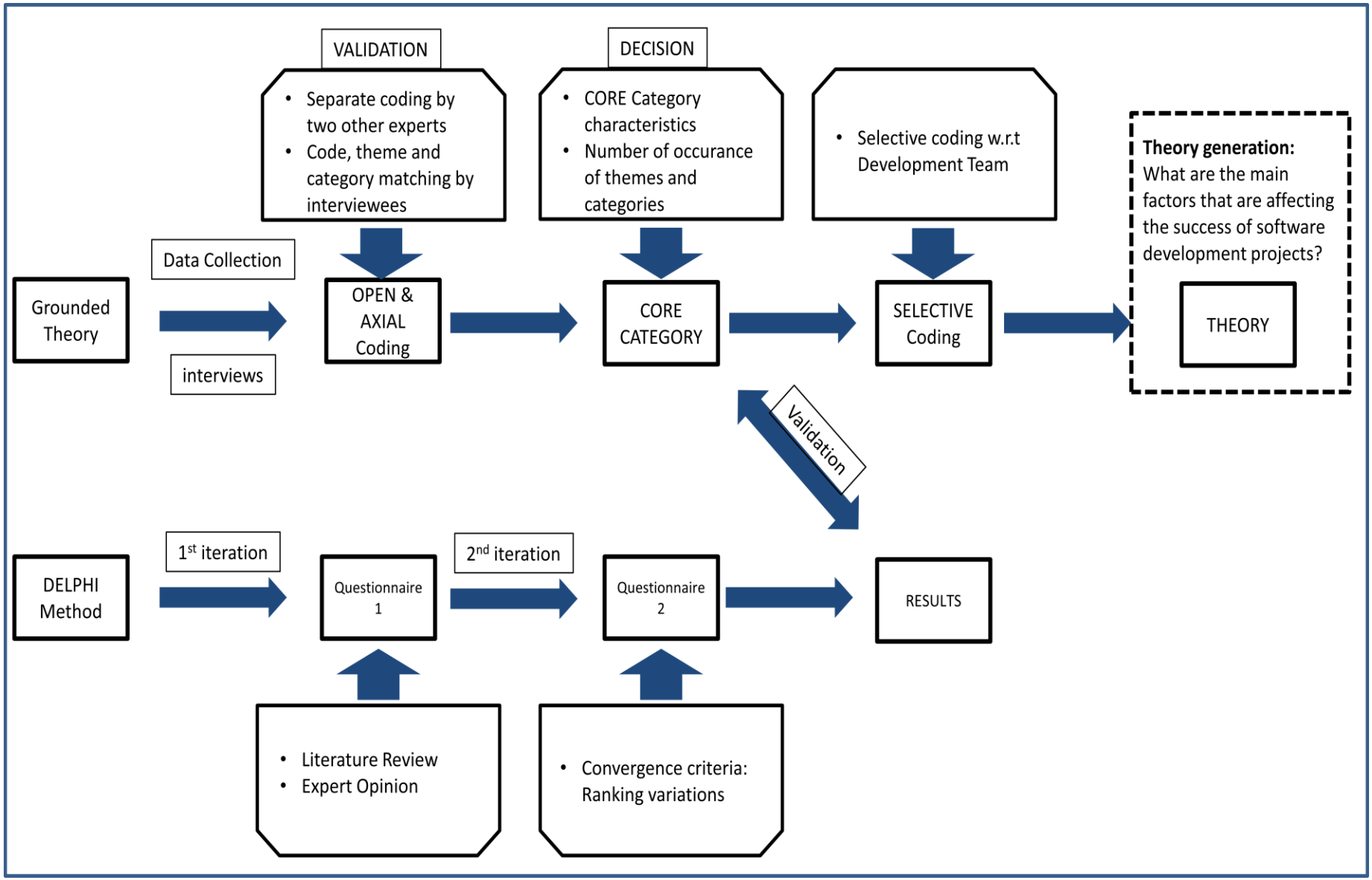

Figure 4-2 The Road Map of the Study

Ahonen and Savolainen (2010) suggest that one of the best ways to increase data validation is using multiple sources of data. In order to achieve that, the Delphi questionnaire was applied to 20 people each engaged in different roles in software development projects and the interviews were made with 5 different people among those 20 who represent the highest variety regarding both the roles they have and the years of experience in the research field.

The validity of data construction depends on the assumption that the interviewees share their true opinions and experiences in the research area. Other than that, considering the validiation of data analysis, the open and axial codings were conducted by two other people engaged in software development and project management activities in order to see whether the interviews give out the same –or similar- core category. The comparisons of the open and axial codings made by two other experts are given in Table 4-4 and are presented in detail in Appendice C.

Table 4-4 Coding Processes Comparison

Researcher		**Expert #1 (Engaged in Software Development)**		**Expert #2 (Engaged in Project Management)**	
Development Team	22	Core Team	26	Software engineering group	26
Processes	13	Company infrastructure, control mechanisms and profile	10	Company literature	10
User/Client	12	Customer relations	5	-	
Management	11	Low Level Management	7	Project administration	12
		Top Level Management	3		
Nature of project	4	Project Goals	2	Project characteristics	8
Reuse	9	Reuse	2	Software reuse	4

The open and axial codings were also shared with the interviewees along with the definitions of the codes that were represented in Table 2-1, for them to match the codes-themes-categories in order to detect if there were a common sense of understanding between them and the researcher of this study. The results and comparisons are are given in Table 4-5 and are presented in detail in Appendice D.

Table 4-5 Comparisons of Code-Category Matching

Categories	**Researcher**	**Int. 1 (Program Manager)**	**Int. 2 (Software Engineering Manager)**	**Int. 3 (Senior Software Engineer)**	**Int. 4 (Junior Software Engineer)**	**Int. 5 (Software Quality Engineer)**
Development Team	22	23	25	23	22	29
Management	11	9	17	18	18	7
Processes	13	13	8	10	13	20
User/Client	12	14	9	8	6	3
Nature of project	4	3	3	3	3	3

The comparisons reveal that "development team" has the highest occurrence along other categories, thus act as a validation material for core category selection. The findings differ mainly around "management", "processes" and "user/client" concepts.

Seaman (1999) recommends using "triangulation" for the validity of both qualitative and quantitative studies. The triangulation concept is defined by her as collecting various kinds of evidence form different resources and by different ways (interviews, questionnaires, observations and etc.) and analyzing those by different methodologies. As different methodologies, she suggests combining and comparing quantitative and qualitative conclusions. To conduct triangulation, the literature review supplied a list of possible success and failure factors in software development, the Delphi questionnaire provided numerical data and finally the main outcomes of the study which qualitative in nature is derived from the Grounded theory methodology. Since both the Delphi questionnaire and the interviews are conducted within the same organization (and evaluated by the same people - interviewees- plus other 15 in-house participants) the hypothesis derived from the interviews were expected to be somehow parallel to the findings of the Delphi. This parallelism is considered to be a part of the validation of the set of principles built by the researcher.

Seamann (1999) offers *"member checking"* as another way to validate qualitative studies. This means sharing the steps and findings with the participants of the study who are the main sources of raw data and receiving feedback from them. But Glaser who is one of the fathers of the Grounded Theory methodology strictly conflicts with that idea (2002). He suggests that since the hypothesis achieved after selective coding part is actually not seen and felt by the participants but is rather "buried" within their experiences, and the researcher's job is to bring that essence out. He declares that they may not like or approve the findings and they may not even understand the study. Kulkarni (2012) supports this view and says that the participants may be "unaware" of the hypothesis and that *"grounded theory is not their voice; it is a generated abstraction from their doings and their meanings that are taken as data for the conceptual generation"*. Thus, the feedback form the participants by matching codes, themes and categories; was received for the open and axial coding phases to eliminate researcher biases on narrowing down the codes retrieved from the interviews but not for selective coding part which is the hypothesis generation phase of the study.

CHAPTER 5

CONCLUSION

5.1. Conclusion

The conclusion part reveals the steps and the results of the final coding phase; selective coding. The core category was chosen according to the criteria that were defined by Strauss and Corbin (1998) and were given in section 3.1 Research Model Definition. Also, the codes retrieved from the interviews are counted according to the times they occurred during the interview as represented in Table 5-1:

Table 5-1 Number of Occurrences of the Codes in the Interviews

Code	# of Occurence
Development Team	22
Processes	19
User/Client	13
Reuse	9
Management	4
Nature of project	4

As mentioned in section 4.2 Validity of Data Collection and Instruments, Delphi findings are used as a validation tool for the choosing of core category of the interviews. Among the most important five factors that affect software projects, four factors are focusing on development team attributes which are: harmony (teamwork) team members' personal attributes and attitudes rewarded, happy staff capable decision makers (project management as well as within the development team)
With the core category in hand, that is "development team", selective coding was implemented as given in Figure 5-1:

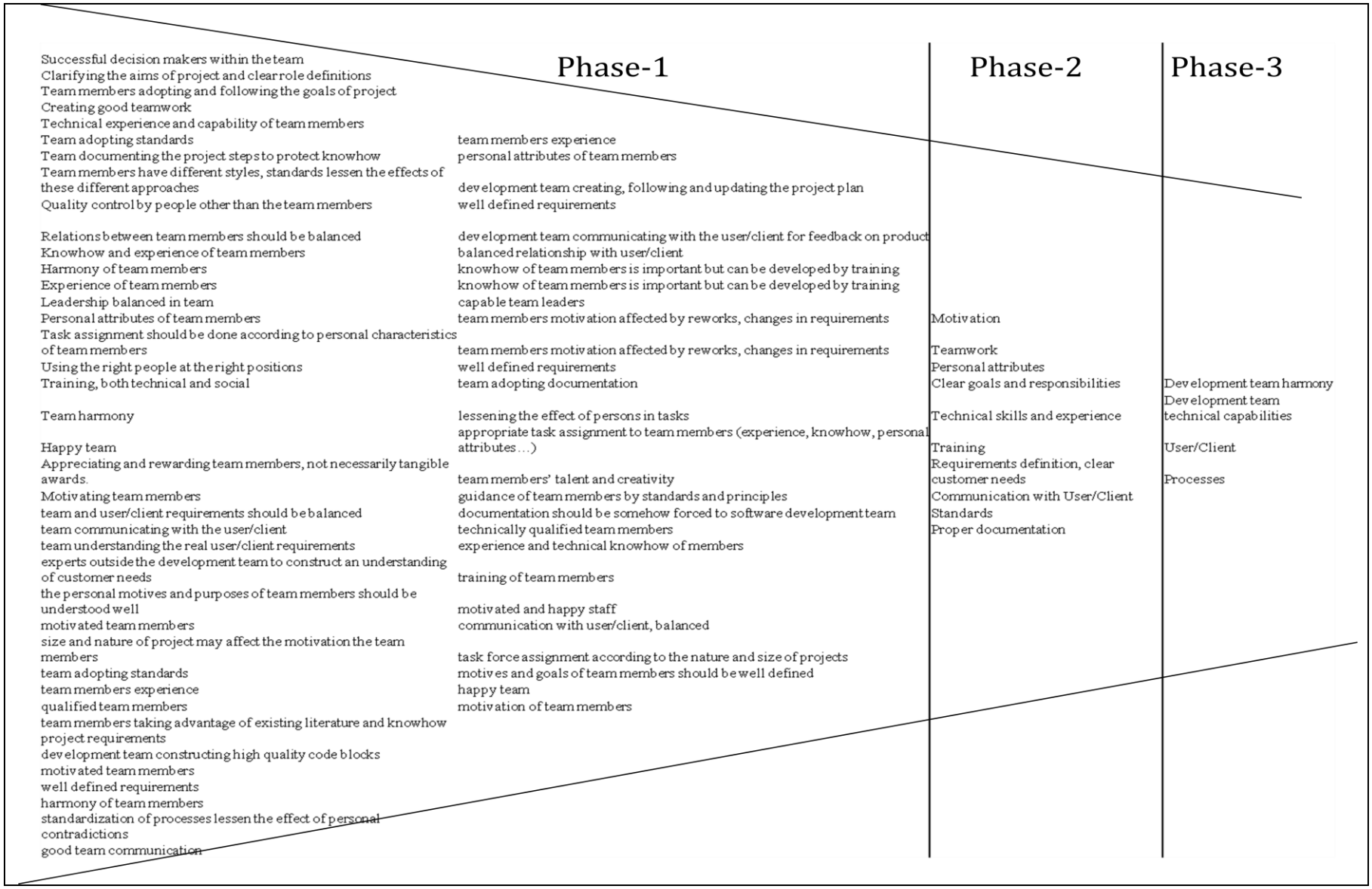

Figure 5-1 Selective Coding Process

The set of principles proposed is: *For a successful software project development the development team should be; motivated and have a good sense of teamwork where the goals and the responsibilities of team members are clearly defined in accordance with their personal attributes. Team members' years of experience and technical skills do affect the success of the project and should be developed through training. Development team should clearly define and understand the requirements and needs of the user/client and keep receiving feedback through adequate communication with the user/client. Finally the team should be working by following predefined processes that include efficient documentation of know-how.*

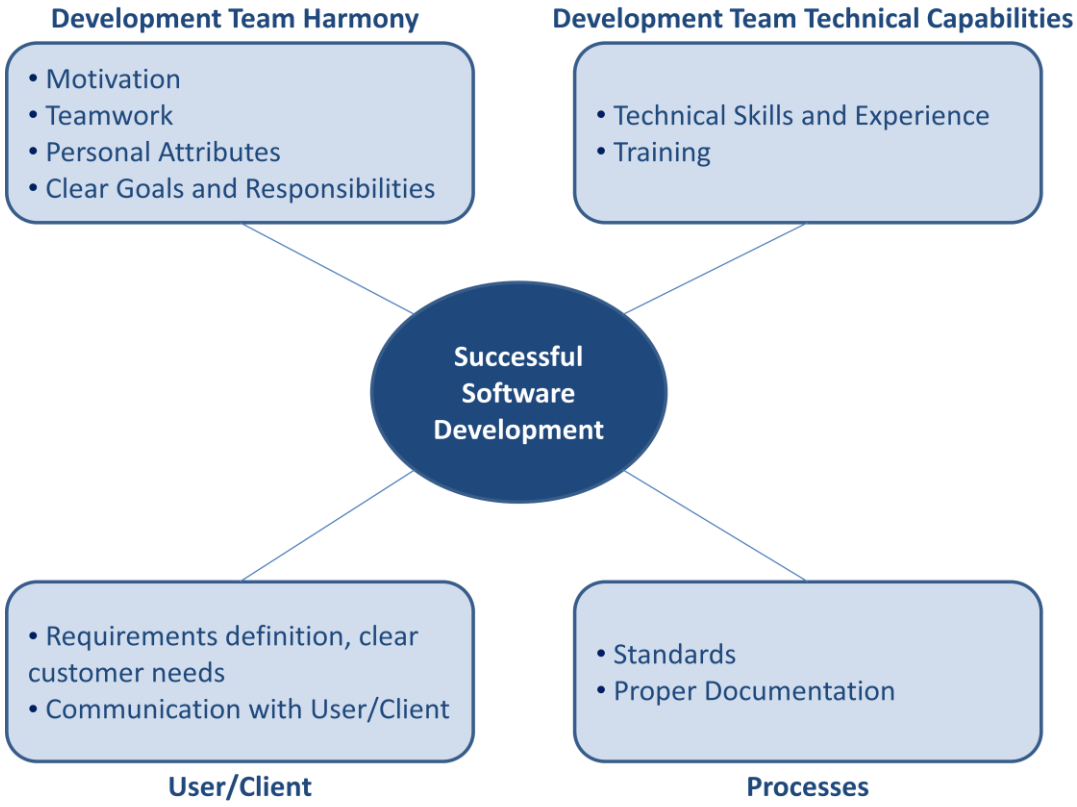

Figure 5-2 The Final Set of Principles

Still, after many decades of research and standarts generation, software field has its unique problems. The present study assumes that these problems are natural outcomes of the suggestion that software field is highly human dependent. So the problems may be handled by the sharing of expert knowledge. The rationale of the qualitative study was to share the tacit knowledge that was gathered from a well known, successful firm that employs qualified software men, with other reserarhcers, software developers and team members.

As the technology grows inevitably, so does the technical tools and methodologies in software engineering field. However, the conclusion of this study reveals that still the success of a software development project depends very highly on human aspects and especially on the development team –at least within the research environment of this study-.

5.2. Comparison of Findings with Studies from Literature

Chapter 2 Literature Review of this study reveals findings of some of the studies conducted on the very subject of this research while this section includes certain outcomes retrieved from the literature which appears to be similar or contradict with the findings of this study.

Whereas the results of this study should be taken into consideration within the research environment which is a company that engages business in Turkish Defense Sector, different studies that focused on the success factors while developing software within many other environments, regarding both civilian and defense software development projects are included in this section.

Sung, Moon and Kim (2010) in their study on the development of "Fleet Anti-air Defense Simulation Software" concluded that in order to come up with a successful simulation software, *"a collaborative work process"* is a must and this can be achieved only by *"assigning the right participants with the right expertise in the right place and time"*, which is a claim that shows similarity with the main components of the set of principles proposed in this study.

Another researcher, Jones (2002) in his study where he investigated the success and failure factors of defense software development focusing especially on different sizes of defense software projects developed in different organizations with CMM ratings, came up with the following conclusions:

- *"Organizations at or above CMM Level 3 are more likely to be successful on large systems [larger than 10,000 function points or 1,000,000 source code statements] than those at Levels 1 or 2."*

- *"For small applications below 1,000 function points or 100,000 source code statements, formal processes are not as significant as the experience of the development team. This is because teams are small so competence – or incompetence of even one person tends to be visible and significant."*

- *"For small applications below 1,000 function points or 100,000 source code statements, the level achieved on the CMM by the development team does not lead to major differences in successes or failure rates."*

At the end of Jones's article, he summarized the outcomes of his study by his sentences *"Overall, there are hundreds of ways to cause software projects to fail, and only a few ways to make them succeed. The highest odds of success will be found where capable teams use formal quality control and formal project management methods.*" which show parallelism with the findings of this research through emphasizing the importance of "capable" development teams as well as solid quality and project management.

Rather than coming up with a likewise outcome, Lu et al. (2011) based their study on the assumption that the quality of software developed is mainly related to software development team and investigated what factors affect team performance. They conducted a questionnaire survey in three different information systems development firms established in China, where they claimed to have very high software failure rates related to low quality development teams. Their study revealed the idea that *"management support, team characteristics, communication quality, knowledge sharing, and clarity of mission"* are the main important factors that affect success of software development teams and concluded that when these are achieved team performance increases significantly in terms of developing desired quality software.

Basri and O'Connor (2011) suggested that the software development team is the main factor to affect the success of software developed and they based their study on identifying team dynamics. The results indicate that "working and social relationship, willingness to share, having a good interpersonal skill and work closely with each others" create the best team dynamics. Whereas Beaver and Schiavone (2006) also focused on the development team as a success factor; their study comes up with the idea that what makes a development team really compatible and succesful is experience.

Another qualitative study was conducted by Siau, Tan and Sheng (2010) in order to figure out the most important characteristics of an information systems development team. The study yielded very similar findings when compared to the findings of this study. They concluded that "*working/cognitive ability, attitude/motivation, knowledge and interpersonal/communication skills*" are the most important attributes of a good IS development team.

There are also other studies such as the one conducted by Aguirregoitia et al. (2010) where the deveopment team is not taken into consideration as a success factor but the focus is on the processes followed during software development. They determined that the most important aspects are as follows:

- *Scheduling and project estimation tracking*
- *Requirements changes management.*
- *Risk identification and analysis.*
- *Failure identification, classification and correction*

Silveira et al. (2010) also suggest that developing quality software is "*directly related to the quality of the processes through which software is developed*". They recommend maturity models such as CMMI and SPICE which enlighten software organizations and claim that "measurement" is a main influencer of those processes. Similarly, Brown (1990) and Marsh (1997) based their studies especially on predefined standards for safety critical and defense software developed in the UK, for successful running projects.

5.3. Limitation of the Study

Because of the preferred research approach, the results and findings of this study may not be generalized to other organizations, industries or countries (Creswell, 2008).

Another limitation of the research method may occur because of the data collecting method. The interviews are conducted on a face-to-face basis, where interviewees may be misled by the researcher. Since the participation is voluntary, possible affects of pressure is eliminated to some extend and the interview is conducted during lunch time or after work, when the informants have their free time and are not expected to perform their duties.

The coding process itself which is the main tool used for developing the hypothesis may create another limitation because it depends on the researcher's perception of the data. To be able to eliminate this limitation, the coding process was conducted separately by two other people (one of whom is experienced in software development and other in system development projects) and the findings are compared. Also the matching of researcher's codes, themes and categories were done by the interviewees themselves, to see if there are considerable variations within the understanding of them and the researcher of this study.

But for the hypothesis generation part; In Lehmann's (2010) words, Grounded Theory Methodology is a very appropriate way for creating new theories in the field of Information Systems however; "it is, not formulaic – it requires creativity" thus may bring risks created

by the unique perception and experience of the researcher, where high risk may as well return with high outcomes. In this case, the researcher has an undergraduate level of Business Administration and has taken the courses of Information Systems graduate program along with working as a member of the project management team in the defense company which the study was conducted at for the last five years.

5.4. Delimitation of the Study

Since the research is a qualitative grounded theory study, in order to be able to generalize the findings and set of principles proposed by the study, further research should be conducted in other organizations or countries.

Another delimitation is that the respondents, in this case the questionnaire participants and interviewees are perceived as to provide accurate and reliable data. The data construct validity is depending solely on the assumption that the participants of the study have and given honest responses to the questionnaire and been truthful during the interviews.

Finally, nature of the project requires anonymity thus, only basic data related to the job descriptions and years of experience of the participants are provided. The names, age, gender, division of the participants are not exposed.

5.5. Implication for Future Research

The research was made based on the different levels of experiences of computer and software engineers who work in a certain company that has a CMMI Level-3 and operates in defense industry. Future research may be conducted for organizations or multiple organizations that engage in different industries where the nature of software is expected to differ substantially as well as in organizations that have different CMMI certification or that are managed by processes and standards other than CMMI. Also a purely quantitative and more complex methodology than just conducting a Delphi questionnaire can be developed to compare findings.

Furthermore, the people engaged in software development projects with different roles, different backgrounds, and years of experience may have certain patterns and variances by means of their perception of the factors that affect software development compared to each other. The effect of those aspects may also be analyzed for future work.

Since the findings of this study claim to come up with the factors that create a successful software development environment, those may be verified by a project which has already proven to be completed with success. The comparison of the set of principles of the present study to the real world occasion should be hopefully verifying the conclusion of the research.

Finally, the conclusion of this study refers to the factors that should be taken into consideration in order to complete a software development project successfully, such as motivating the development team members. However, it doesn't address how to create scuh an environment. Further study may be conducted on how to extend the capabilities of software development teams.

REFERENCES

Abdul-Rahman, H., Mohd-Rahim, F. A., Chen, W. (2012). Reducing Failures in Software Development Projects: Effectiveness of Risk Mitigation Strategies. *Journal of Risk Research,* 15:4, 417-433, doi: 10.1080/13669877.2011.634520

Adams, T., L. (2007). INTERPROFESSIONAL RELATIONS AND THE EMERGENCE OF A NEW PROFESSION: Software Engineering in the United States, United Kingdom, and Canada. *The Sociological Quarterly,* 48**,** 507–532.

Adolph, S., Hall, W., Kruchten, P. (2011). Using Grounded Theory to Study the Experience of Software Development. *Empirical Software Eng,* 16:487-513, doi: 10.1007/s10664-010-9152-6.

Aguirregoitia, A., Dolado, J. J, Presedo, C. (2010).Applying the Metro Map to Software Development Management. Proceedings from SPIE 7530, Visualization and Data Analysis, 7530 (0I), doi:10.1117/12.837752.

Ahonen, J. J., Savolainen, P. (2010). Software engineering projects may fail before they are started: Post-mortem analysis of five cancelled projects. *The Journal of Systems and Software*, 83, 2175–2187, doi: 10.1016/j.jss.2010.06.023.

Aladwani, A., M. (2004). An Integrated Performance Model of Information Systems Projects. *Journal of Management Information Systems,* 19 (1), 185–210.

Althoff, K., D., Birk, A., von Wangenheim, C., G., Tautz, C. (1998). *Case Based Reasoning Technology, From Foundations to Applications.* London: Springer-Verlag.

Avison, D., Myers, M. (2005). *Qualitative Research. Research in Information Systems: A Handbook for Research Supervisors and Their Students.* Oxford: Elsevier Butterworth-Heinemann.

Basri, S., O'Connor, R. V. (2011). A Study of Software Development Team Dynamics in SPI. *Communications in Computer and Information Science*, 172, 143-154.

Beaver, J. M., Schiavone, G. A. (2006). The Effects of Development Team Skill on Software Product Quality ACM SIGSOFT Software Engineering Notes, Volume 31 Number 3, doi: 10.1145/1127878.1127882.

Berterö, C. (2012). Grounded theory methodology - has it become a movement? *International Journal of Qualitative Studies on Health & Well-Being*, 7 (0), 1-2, doi: 10.3402/qhw.v7i0.18571.

Boehm, B. W. (2006). A view of 20th and 21st century software engineering. Proceedings of the 28th International Conference on Software Engineering, 12-29, ACM, New York.

Boehm, B. W. (1988). A Spiral Model of Software Development and Enhancement. *Computer*, 21(5), 61-72.

Brooks, F., P. (1986). No Silver Bullet: Essence and Accidents of Software Engineering. Proceedings from IFIP 10th World Computing Conference, 1069-1076, NL, Amsterdam.

Brown, M. J. D. (1990). Rationale for the Development of the UK Defence Standards for Safety-critical Computer Software. *IEEE AES Magazine*, November 1990, 31-37.

Charettte, R. (2005). Why software fails, *IEEE Spectrum*, September, 2005, pp.42-49.

Clarke, P., O'Connor R.V. (2012). The Situational Factors That Affect The Software Development Process: Towards A Comprehensive Reference Framework, *Journal of Information Software and Technology*, Vol. 54 (5), 433-447.

Coleman, G., O'Connor, R. (2008). Investigating software process in practice: A grounded theory perspective. *Information and Software Technology*. 81 (2008) 772–784, doi: 10.1016/j.jss.2007.07.027.

Coleman, G., O'Connor, R. (2007). Using grounded theory to understand software process improvement: A study of Irish software product companies. *Information and Software Technology*. 49 (2007) 654–667, doi: 10.1016/j.insof.2007.02.011.

Creswell, J. W. (2009). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches (3rd ed.).* USA: SAGE.

Creswell, J. W. (2008). *Educational research: Planning, conducting, and evaluating quantitative and qualitative research* (3rd ed.). Upper Saddle River, NJ: Pearson.

Dalcher, D., Benediktsson, O. (2006). Managing Software Development Project Size: Overcoming the Effort-Boxing Constraint. *Project Management Journal*, 37(2), 51-58.

Daniels, C. B., LaMarsh II, W. J. (2007). Complexity as a Cause of Failure in Information Technology Project Management. Proceedings from IEEE International Conference on System of Systems Engineering. San Antonio, TX.

DeMarco, T., Lister, T. (1999). *Peopleware: Productive Projects and Teams.* New York: Dorset House.

Dittrich, Y., Rönkkö, K., Eriksson, J., Hansson, C., Lindeberg, O. (2008) Cooperative Method Development: Combining Qualitative Empirical Research with Method, Technique and Process Improvement. *Empirical Software Eng.* 13(3), 231–260.

Dybå, T., Prikladnicki, R., Rönkkö, K., Seaman, C., Sillito, J. (2011). Qualitative Research in Software Engineering. *Empirical Software Engineering*, 16, 425–429, doi 10.1007/s10664-011-9163-y.

Ewusi-Mensah, K. (1997). Critical Issues in Abandoned Information Systems Development Projects: What is it about IS development projects that make them susceptible to cancellations? Communications of the ACM, 48(9), 74-80.

Evelens, J., L., Verhoef, C. (2008). The rise and fall of the Chaos report figures. *IEEE Software*, 27, (1), 30-36.

Fernandez, W. D., Lehmann, H. (2011). Case Studies and Grounded Theory Method in Information Systems Research: Issues and Use. *Journal of Information Technology Cases and Applications*, Vol 13 (1), 4-15.

Geetha, D. E., Kumar, T. V. S., Kanth, K. R. (2007). Predicting Performance of Software Systems during Feasibility Study of Software Project Management. Proceedings from 6th International Conference on Information, Communications and Signal Processing. Singapore, SINGAPORE.

Gibbs, W., W. (1994). Software's Chronic Crisis. *Scientific American*, September 1994, 72-81.

Glaser, B., Strauss, A. (2007). *The discovery of grounded theory: Strategies for qualitative research*. Piscataway, NJ: Aldine Transaction.

Glaser, B. (2002). Conceptualization: On Theory and Theorizing Using Grounded Theory. *International Journal of Qualitative Methods*, 1(2), 23-38.

Hoda, R., Noble, J., Marshall, S. (2012). Developing a Grounded Theory to Explain the Practices of Self-Organizing Agile Teams. *Empirical Software Engineering*, 17(6), 609-639. doi: 10.1007/s10664-011-9161-0.

Hoda, R., Noble, J., Marshall, S. (2010). Using Grounded Theory to Study the Human Aspects of Software Engineering. Proceedings of Human Aspects of Software Engineering (HAoSE'10). ACM, NY, US. doi:10.1145/1938595.1938605.

Huysegoms, T., Snoeck M., Dedene, G., Goderis, A.(2011). Requirements for Successful Software Development with Variability: A Case Study. *Communications in Computer and Information Science*, 219, 238–247.

Jones, C. (2002). Defense Software Development in Evolution. *The Journal of Defense Software Engineering, November* 2002, 26-29.

Knigge, L., Cope, M. (2006). Grounded Visualization: Integrating the Analysis of Qualitative and Quantitative Data Through Grounded Theory and Visualization. *Environment and Planning,* 38 (11), 2021-2037.

Kroenke, D. (2007). *Experiencing MIS*. Upper Saddle River, NJ: Pearson.

Kulkarni, V. V. (2012). Unexplored Theory Of Social Research Grounded Theory and Coding Issues in Grounded Theory. *Review of Research Journal*, Vol. 2 (2), 224-238.

Kulkarni, V. V. (2012). 'Grounded Theory' : An Emerging Theoretical Perspective of Social Work Research. *Review of Research Journal*, Vol. 1 (12), 1-4.

Lavrishcheva, E. M. (2008). Software Engineering as a Scientific and Engineering Discipline. *Cybernetics and Systems Analysis*, 44(3), 324-332.

Lee, R. M., (2011). The Most Important Technique …": Carl Rogers, Hawthorne, Andthe Rise and Fall of Nondirective Interviewing in Sociology. *Journal of the History of the Behavioral Sciences*, Vol. 47(2), 123–146, doi: 10.1002/jhbs.20492.

Lehmann, H. (2010). Grounded Theory and Information Systems: Are We Missing the Point?. Proceedings of the 43rd Hawaii International Conference on System Sciences, Kauai, Hawaii.

Leung, H. (2001). Organizational Factors for Successful Management of Software Development. *Journal of Computer Information Systems*, 42 (2), 26-37.

Lu, Y., Xiang, C., Wang, B., Wang, X. (2011). What affects information systems development team performance? An Exploratory Study from the Perspective of Combined Socio-Technical Theory and Coordination Theory. *Computers in Human Behavior,* 27, 811–822, doi:10.1016/j.chb.2010.11.006.

Marsh, W. (1997). Harmonisation of Defence Standards for Safety-Critical Software. *Microprocessors and Microsystems,* 21 (1997), 41-47.

Martin, N. L., Pearson, J. M., Furumo, K. (2007). IS Project Management: Size, Practices and The Project Management Office. *Journal of Computer Information Systems,* 47 (4), 52-60.

McAvoy, J., Butler, T. (2009). The role of project management in ineffective decision making within Agile software development projects. *European Journal of Information Systems*, 18, 372–383.

McBride, T. (2008). The mechanisms of project management of software development. *The Journal of Systems and Software*, 81, 2386–2395.

McLeod, L., MaCDonell, S. G. (2011). Factors that Affect Software Systems Development Project Outcomes: A Survey of Research. *ACM Computing Surveys*, Vol. 43, No. 4, Article 24, doi:10.1145/1978802.1978803.

McLeod, L., MaCDonell, S. G., Doolin, B. (2011). Qualitative research on software development: a longitudinal case study methodology. *Empirical Software Engineering*, 16:430–459, doi: 10.1007/s10664-010-9153-5.

Mei, H., Cao, D., Yang, F. (2006). Development of Software Engineering: A Research Perspective. *Journal of Computer Science and Technology*, 21 (5), 682-696.

Mills, J., Bonner, A., Francis, K. (2006). The Development of Constructivist Grounded Theory. *International Journal of Qualitative Methods*, 5(1), 25-35.

Morgenshtern, O., Raz, T., Dvir, D. (2007). Factors Affecting Duration and Effort Estimation Errors in Software Development Projects. *Information and Software Technology,* 49 (8), 827–837.

Opdenakker, R. (2006). Advantages and Disadvantages of Four Interview Techniques in Qualitative Research. *Forum Qualitative Sozialforschung*, 7 (4), Article 11.

Procaccino, J. D., Verner, J. M., Overmyer S. P., Darter, M. E. (2002). Case Study: Factors for Early Prediction of Software Development Success. *Information and Software Technology,* 44 (1), 53-62.

Rehman, A., Hussain, R. (2007). Software Project Management Methodologies/Frameworks Dynamics "A Comparative Approach". Proceedings of International Conference on Information and Emerging Technologies. San Diego, CA.

Remus U., Wiener, M. (2008). A multi-method, holistic strategy for researching critical success factors in IT projects. *Information Systems Journal*, 20, 25–52. doi:10.1111/j.1365-2575.2008.00324.x.

Roach, C., Menezes, R. (2011). Using Networks to Understand the Dynamics of Software Development. *CompleNet*, 116, 119–129.

Rodrigues, S. (2010). *Using Analytical Frameworks for Classroom Research: Collecting Data and Analysing Narrative*. Routledge: Oxon.

Roulston, K. (2010). Considering quality in qualitative interviewing. *Qualitative Research 2010.* 10: 199, doi: 10.1177/1468794109356739.

Savolainen, P., Ahonen, J. J., Richardson, I. (2012). Software Development Project Success and Failure From the Supplier's Perspective: A Systematic Literature Review. *International Journal of Project Management*, 30, 458–469, doi:10.1016/j.ijproman.2011.07.002.

Seaman, C. B. (1999). Qualitative Methods in Empirical Studies of Software Engineering. *IEEE Transactıons On Software Engineering*, 25 (4), 557-572.

Sharp, H., Woodman, M., Hovenden, F. (2005).Using Metaphor to Analyse Qualitative Data: Vulcans and Humans in Software Development. *Empirical Software Engineering*, 10 (3), 343–365.

Siau, K., Tan, X., Sheng, H. (2010). Important characteristics of software development team members: an empirical investigation using Repertory Grid. *Information Systems Journal*, 20, 563–580, doi:10.1111/j.1365-2575.2007.00254.x.

Silveira, P. S., Becker, K., Ruiz, D. D. (2010). SPDW+: A Seamless Approach for Capturing Quality Metrics in Software Development Environments. *Software Quality Journal*, 18:227–268, doi: 10.1007/s11219-009-9092-9.

Skulmoski, G. J., Hartman, F. T., Krahn, J. (2007). The Delphi Method for Graduate Research. *Journal of Information Technology Education.*Vol (6), 2007.

Sommerville, I. (2007). *Software Engineering.* Harlow: Pearson Education.

Strauss, A., Corbin, J. (1998). *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory* (2nd ed.). Thousand Oaks, CA: Sage.

Subramanian, G. H., Jiang, J. J., Klein, G. (2007). Software Quality and IS Project Performance Improvements from Software Development Process Maturity and IS Implementation Strategies. *The Journal of Systems and Software*, 80 (4), 616–627.

Suddaby, R. (2006). "From the Editors: What Grounded Theory is Not". *Academy of Management Journal*, 46 (4), 633-642.

Sung, C. H., Moony, I., Kim, T. G. (2010). Collaborative Work in Domain-Specific Discrete Event Simulation Software Development: Fleet Anti-air Defense Simulation Software. Proceedings from Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), 19th IEEE International Workshop, Larissa, GREECE.

Suzuki, L. A., Ahluwalia, M. K., Arora, A. K., Mattis, J. A. (2007). The pond you fish in determines the fish you catch: Exploring strategies for qualitative data collection. *The Counseling Psychologist*, 35(2), 295-327. doi:10.1177/0011000006290983.

Tesch, D., Kloppenborg, T. J., Frolick, M.N. (2007). IT project risk factors: The Project management professionals perspective. *Journal of Computer Information Systems*, 47(4), 61–9.

Urquhart, C., Lehmann, H., Myers, M. D. (2010). Putting the 'Theory' Back Into Grounded Theory: Guidelines for Grounded Theory Studies in Information Systems. *Info Systems Journal*. 20, 357–381, doi: 10.1111/j.1365-2575.2009.00328.x.

Verner, J., Sampson, J., Cerpa, N. (2008). What Factors Lead to Software Project Failure? Proceedings from 2nd International Conference on Research Challenges in Information Science. Marrakesh, MOROCCO.

Virili, F., Sorrentino, M. (2010). The enabling role of Web services in information system development practices: a grounded theory study. *Information Systems e-Business Manage*, 8, 207–233, doi: 10.1007/s10257-008-0097-x.

Wang, Q. Z., Liu, J. (2006). Project Uncertainty, Management Practice and Project Performance: An Empirical Analysis on Customized Information Systems Development

Projects. Proceedings from IEEE International Engineering Management Conference. Bahia BRAZIL.

Warren, C. (2002). Qualitative Interviewing. *Handbook of Interview Research.* USA: SAGE.

Wright, M., K., Capps III, C., J. (2010). Information Systems Development Project Performance in the 21st Century. ACM SIGSOFT Software Engineering Notes, 35 (2), 1-9. doi: .1145/1734103.1734121.

Yang, H., Tang, J. (2004). Team Structure and Team Performance in IS Development: A Social Network Perspective. *Information & Management*, 41, 335–349, doi: 10.1016/S0378-7206(03)00078-8.

Yeo, K., T. (2002). Critical Failure Factors in Information System Projects. *International Journal of Project Management*, 20, 241–246.
Zannier, C., Chiassob, M., Maurer, F. (2007). *Information and Software Technology* 49 (2007) 637–653, doi: 10.1016/j.infsof.2007.02.010.

Zhivich, M., Cunningham, R. K. (2009). The Real Cost of Software Errors. *Security and Privacy*, 7 (2), 87-90, doi: 10.1109slashMSP.2009.56.

APPENDICES

APPENDIX A: QUESTIONNAIRE DELPHI RESULTS

Table 6-1 Delphi Survey Results 1st Iteration

Factors Affecting Software Development Projects	Senior Software Engineer 10-15 years of experience	Senior Software Engineer 10-15 years of experience	Senior Software Engineer 5-10 years of experience	Senior Software Engineer 5-10 years of experience	Software Quality Engineer 10-15 years of experience	Software Quality Engineer 5-10 years of experience	Senior Software Engineer 10-15 years of experience	Software Development Team Leader (Software Engineer) 10-15 years of experience	Software Development Team Leader (Software Engineer) 15-20 years of experience	Senior Software Engineer 10-15 years of experience	Software Development Team Leader (Software Engineer) 10-15 years of experience	Junior Software Engineer 1-5 years of experience	Junior Software Engineer 1-5 years of experience	Junior Software Engineer 1-5 years of experience	Junior Software Engineer 1-5 years of experience	Interviewee 1 (Program Manager) 15-20 years of experince	Interviewee 2 (Software Engineering Manager) 15-20 years of experince	Interviewee 3 (Senior Software Engineer) 10-15 years of experience	Interviewee 4 (Junior Software Engineer) 1-5 years of experience	Interviewee 5 (Software Quality Engineer) 10-15 years of experince	AVG 1st Iteration
	Importance (1-10) (1 being the least and 10 being the most important)	Importance (1-10) (1 being the least and 10 being the most important)	Importance (1-10) (1 being the least and 10 being the most important)	Importance (1-10) (1 being the least and 10 being the most important)	Importance (1-10) (1 being the least and 10 being the most important)	Importance (1-10) (1 being the least and 10 being the most important)	Importance (1-10) (1 being the least and 10 being the most important)	Importance (1-10) (1 being the least and 10 being the most important)	Importance (1-10) (1 being the least and 10 being the most important)	Importance (1-10) (1 being the least and 10 being the most important)	Importance (1-10) (1 being the least and 10 being the most important)	Importance (1-10) (1 being the least and 10 being the most important)	Importance (1-10) (1 being the least and 10 being the most important)	Importance (1-10) (1 being the least and 10 being the most important)	Importance (1-10) (1 being the least and 10 being the most important)	Importance (1-10) (1 being the least and 10 being the most important)	Importance (1-10) (1 being the least and 10 being the most important)	Importance (1-10) (1 being the least and 10 being the most important)	Importance (1-10) (1 being the least and 10 being the most important)	Importance (1-10) (1 being the least and 10 being the most important)	
1. top management support	7	6	7	8	6	5	9	8	4	9	7	8	8	8	5	8	8	5	8	8	7,1
2. capable decision makers (strong project management)	9	7	9	8	9	8	6	8	9	7	9	7	9	6	8	10	8	6	9	7	8,0
3. existence and success of project control mechanisms	8	6	7	8	9	7	6	4	5	8	7	6	5	8	7	6	7	8	9	5	6,8
4. performance evaluation	6	5	6	5	7	8	5	6	9	8	7	5	6	7	8	6	7	9	8	3	6,6
5. technical infrastructure	6	4	3	7	8	6	3	7	8	5	6	7	4	8	7	7	8	10	7	6	6,4
6. estimations on duration and task effort	8	6	8	7	5	4	5	4	7	8	9	6	6	4	8	8	7	4	9	5	6,4
7. software reuse	9	8	10	5	8	9	6	10	7	7	8	9	7	6	8	9	10	9	7	5	7,9
8. company principles or standards (CMMI, TQM...)	7	8	5	6	9	8	9	8	7	7	8	6	7	5	9	8	8	6	7	7	7,3
9. the use of a certain SW development methodology (waterfall, incremental, evolutionary)	6	8	9	5	6	7	8	9	5	4	7	6	8	9	5	7	6	4	8	6	6,7
10. quality management all along the SW development process	8	7	9	8	9	8	5	8	7	9	6	5	8	7	9	7	7	3	9	3	7,1
11. team members' personal attributes and attitudes	10	9	9	10	6	9	7	8	10	7	9	8	9	8	7	8	8	8	7	10	8,4
12. experience and technical know-how of team members	9	9	7	8	9	8	5	8	6	9	10	7	8	8	5	10	9	9	6	10	8,0
13. training of team members	7	8	9	8	7	8	6	8	9	10	7	8	6	10	9	6	7	7	9	8	7,9
14. rewarded, happy staff	9	10	9	9	9	7	8	9	10	8	5	8	6	9	10	8	7	9	5	10	8,3
15. user/client experience	5	6	8	7	9	4	7	8	9	5	5	6	4	7	8	9	8	7	8	3	6,7
16. user participation, communication and collaboration among them and the developers	6	7	8	9	5	6	4	8	9	10	9	7	8	6	5	10	8	4	8	5	7,1
17. requirements definition	9	10	9	10	9	10	9	9	10	9	9	10	10	9	10	10	7	9	10	9	9,4
18. user/client patches	7	5	7	6	8	4	5	7	7	8	9	7	8	9	4	8	8	8	5	9	7,0
19. size of the project	5	6	6	8	7	5	6	8	5	7	4	6	9	7	5	7	7	6	5	8	6,4
20. technological complexity of the project	6	7	5	6	8	5	4	6	9	8	8	9	7	5	7	8	8	8	5	7	6,8
21. different goals or purposes of stakeholders	7	4	7	8	9	6	5	8	9	6	7	6	6	8	5	8	8	4	5	8	6,7
OTHER																					
Harmony (teamwork)	10			10			10	9								10		10			9,8

Table 6-2 Delphi Survey Results 2nd Iteration

Factors Affecting Software Development Projects	Senior Software Engineer 10-15 years of experience	Senior Software Engineer 10-15 years of experience	Senior Software Engineer 5-10 years of experience	Senior Software Engineer 5-10 years of experience	Software Quality Engineer 10-15 years of experience	Software Quality Engineer 5-10 years of experience	Senior Software Engineer 10-15 years of experience	Software Development Team Leader (Software Engineer) 10-15 years of experience	Software Development Team Leader (Software Engineer) 15-20 years of experience	Senior Software Engineer 10-15 years of experience	Software Development Team Leader (Software Engineer) 10-15 years of experience	Junior Software Engineer 1-5 years of experience	Junior Software Engineer 1-5 years of experience	Junior Software Engineer 1-5 years of experience	Junior Software Engineer 1-5 years of experience	Interviewee 1 (Program Manager) 15-20 years of experince	Interviewee 2 (Software Engineering Manager) 15-20 years of experince	Interviewee 3 (Senior Software Engineer) 10-15 years of experience	Interviewee 4 (Junior Software Engineer) 1-5 years of experience	Interviewee 5 (Software Quality Engineer) 10-15 years of experince	AVG 2nd Iteration
	2nd Iteration	2nd Iteration	2nd Iteration	2nd Iteration	2nd Iteration	2nd Iteration	2nd Iteration	2nd Iteration	2nd Iteration	2nd Iteration	2nd Iteration	2nd Iteration	2nd Iteration	2nd Iteration	2nd Iteration	2nd Iteration	2nd Iteration	2nd Iteration	2nd Iteration	2nd Iteration	
1. top management support	7	6	7	8	6	5	9	8	6	9	7	8	8	8	5	8	8	5	8	7	7,2
2. capable decision makers (strong project management)	9	7	9	8	9	8	6	8	9	7	9	7	9	6	8	10	8	9	9	6	8,1
3. existence and success of project control mechanisms	8	6	7	8	8	7	6	6	5	8	7	6	5	8	7	6	7	8	8	5	6,8
4. performance evaluation	6	5	6	5	7	8	5	6	9	8	7	5	6	7	8	6	7	9	7	3	6,5
5. technical infrastructure	6	4	3	7	8	6	3	7	8	5	6	7	4	8	7	7	8	10	8	6	6,4
6. estimations on duration and task effort	7	6	8	7	5	4	5	4	7	8	9	6	6	4	8	8	7	4	9	5	6,4
7. software reuse	9	8	10	5	8	9	6	9	7	7	8	9	7	6	8	9	10	9	7	5	7,8
8. company principles or standards (CMMI, TQM...)	7	8	5	6	9	8	9	8	7	7	8	6	7	5	9	8	8	6	7	7	7,3
9. the use of a certain SW development methodology (waterfall, incremental, evolutionary)	6	8	9	5	6	7	8	9	5	7	7	6	8	9	6	7	6	4	8	6	6,7
10. quality management all along the SW development process	8	7	9	8	9	8	5	8	7	9	6	5	8	7	9	7	7	3	9	5	7,2
11. team members' personal attributes and attitudes	10	9	9	10	6	9	7	8	10	7	9	8	9	8	7	8	8	8	7	10	8,4
12. experience and technical know-how of team members	9	9	7	8	9	8	5	8	6	9	10	7	8	8	5	10	9	9	7	10	8,1
13. training of team members	8	8	9	8	7	8	6	8	9	10	7	8	6	10	9	6	7	7	9	8	7,9
14. rewarded, happy staff	9	10	9	9	9	7	8	9	10	8	5	8	6	9	10	8	7	9	7	10	8,4
15. user/client experience	5	6	8	7	9	6	7	8	9	5	5	6	5	7	8	9	8	7	7	3	6,8
16. user participation, communication and collaboration among them and the developers	6	7	8	9	5	6	4	8	9	9	9	7	8	6	5	10	8	4	7	5	7,1
17. requirements definition	9	10	9	10	9	10	9	9	10	9	9	10	10	9	10	10	7	9	10	9	9,4
18. user/client patches	7	5	7	6	8	4	5	7	7	8	9	7	8	9	4	8	8	8	8	9	7,1
19. size of the project	5	6	6	8	7	5	6	8	5	7	5	6	9	7	5	7	7	6	5	7	6,4
20. technological complexity of the project	6	7	5	6	8	5	4	6	9	8	8	9	7	5	7	8	8	8	7	7	6,9
21. different goals or purposes of stakeholders	7	4	7	8	9	6	5	8	9	6	7	6	6	8	5	5	8	4	5	8	6,6
OTHER																					
Harmony (teamwork)	10	7	9	10	9	6	8	9	10	9	8	10	10	9	10	5	8	10	9	8	8,7

APPENDIX B: THE INTERVIEWS

Interviewee 1 (Program Manager):

Şimdi asıl olarak beklediğim yazılım projelerinizdeki tecrübenize dayanarak ve elinizdeki ankette verilen başlıklara da bakarak hakikaten projede şu olmazsa olmaz, ya da bu olursa yürümez proje diyebileceğiniz ne var yani yazılım projeleri neden yürüyor/veya yürümüyor?

Şimdi her bir kalem üzerinden teker teker geçelim, üst yönetim desteği tüm projeler için geçerli, burada da projeler açısından önemlidir. Önemli olan işte üst yönetimin projeye kaynak ayrılması için gerekli talepleri yapmış olması ve taleplerin karşılanarak projenin yönetilmesi.

Şimdi projede karar vericilerin yeteneği çok önem kazanıyor çünkü üst yönetim şunu yapalım şeklinde yaklaşırken esas yapılacakların kararlarını daha alt seviye yöneticiler veriyor ve işi yürütüyor. Ve orada hiç karar vericinin olmadığı, şunu da müşteriye soralım diye yaklaşıldığı zaman müşterinin farklı yaklaşımları nedeniyle istenen noktaya gidilemiyor. Bu nedenle projede sorumlu ve karar verme yeteneğine sahip birinin olması…

Şart diyorsunuz…

Bunun illa proje yöneticisi olması gerekmiyor. Yani güvendiği bir teknik elemanın bu karar verme yeteneğini uygun görüyorsa ona bu görevi delege edebilir. Burada önemli olan şirketin hedefleriyle alınan kararın uyumlu olmasını sağlamak ve verilen kararlarla projenin başarılı olmasını sağlamak. Proje kontrol mekanizmasının varlığı ve başarısı proje üzerinde etkin ama bunu yanıltabiliyoruz.

Evet, ölçümler konusunda her zaman başarılı olunamıyor özellikle yazılım projelerinde…

Burada baktığımızda proje kontrol mekanizmalarını kurduğumuzda buraya sağlanan bilgilerin de doğru olduğuna yönelik analizlerin yapılması gerekli. Aslında ham veriyi kullandığın zaman ham veri bize doğru bilgiyi vermiyor. Bu işi yapmak için ne kadar zaman kullandın dediğinde veriyi sağlayan kişi brüt zaman da verebiliyor net zamanda verebiliyor. Yani aynı işi yapan iki kişiden biri iş üzerinde geçirdiği net zamanı biri de brüt zamanı verebiliyor. Bu yüzden karşılaştırmakta da zorlanılıyor. Bunlara yönelik bazı otomasyon araçları var işte team software process veya personal software process gibi bu süreçlerin kullanılması için ama hala burada bu kişilerin bu araçları aynı amaçla kullanacağını deklare etmesi ve yönetimin de bunu sağlaması gerekli. Diğer konu da aslında bağlantılı, personelin performansını değerlendirme kısmı da 2 şekilde, hem objektif hem de subjektif değerlendirme yöntemiyle yapılabiliyor. İşte olabildiğince objektif herkesi eşit kantarla tartmaya çalıştığınızda da her zaman kolay olmuyor. O nedenle bunlar uygulanmadan da başarılı olmuş projeler var. Yönetimin burada aslında önemli olan husus doğru hedefleri net şekilde belirleyebilmesi ve proje ekibinin de bu hedefler doğru anlaması ve benimsemesi, projede kendi görevlerini yerine getirmesi. Eğer ekip çalışması oluşturabilmişsek projede başarıyı en çok etkileyen etkenlerden birisi olarak ben ekip çalışmasını görüyorum.

Süreçlere geldiğimiz zaman burada süreçlerde de yönetimin bazı yönlendirmeleri oluyor özellikle technical infrastructure konusu. Bir takım yatırımlar yapmanız gerekiyor ama bir yandan da projenin bütçesini aşmamanız gerekiyor. Bir diğeri de projeler içindeki ortaklıkların dikkate alınması ve bunların yönetilmesi. Şimdi altyapı yazılım projeleri

açısından baktığımızda kullanılabilecek araçlar belli ve herkesin elinde az çok var hani çok büyük yatırımlar yapılmasını gerektiren araçlar olmadan da başarılı yazılım projeleri yapılabiliyor.

Zaten aslında artık teknolojik anlamda da "ram yavaş kaldı, bu bilgisayar çalıştı çalışmadı" gibi çok temel problemleri pek yaşamıyoruz.

Iııı teknolojik altyapı konusunu ikiye ayıralım bir kullanılacak araçların sağlanması ki bu kolay olan kısmı elde ediliyor ikincisi işgücünün yeterli teknik tecrübeye sahip olarak o tecrübenin sağlanması. Ki bence ikincisi bence projenin başarısında daha önemli… İlk defa o teknik tecrübeyi kullanarak işi yapan bir ekiple, daha önce 8-10 kere o tecrübeyi kullanarak işi yapmış olan bir ekip arasında başarı farkı oluyor.

Projenin boyutlarına bağlı olarak süresini ve harcanacak emeği tahmin etmek oldukça zorlu. Eğer siz aynı işi tekrar tekrar yaparak ilerliyorsanız orada iş daha net olarak tariflenebiliyor ama her proje ayrı bir karakteristikte gelip benzerlik içermiyorsa o zaman projeyi tariflemekte zorlanıyoruz. Daha projeyi kontrol edebilmek ölçüm sonuçlarını daha iyi takip edebilmek için tahminleri sağlıklı yapmak önceden önemli ama bu tahmine göre de faaliyetin yerine getirilebilmesi lazım. Genelde yazılım projelerinde baştan tahmin ettiklerinizle daha sonra gerçekleşenler hem gereksinimler açısından hem de ortaya çıkan ürün açısından farklı oluyor o da yönetilmesi gereken bir faaliyet yani.

Aslında estimation, önden plan yapmanın yazılımda çok büyük etken olmadığını iddia eden yorumlar da var, inşaat projeleri gibi şu kadar çimento bu kadar demire ihtiyaç var denemediği için. Hatta önceden kestirim yapmanın projede ayak bağı olabildiğini öneren görüşler de var. Siz ne düşünüyorsunuz bu konuda?

Yazılımın aşamaları belli, uygulayacağınız süreç seçiliyor. Yazılımı yaparken bazı kurallar da uygulanabiliyor mesela analizi yapacağınız zamanla tasarım dönemine harcadığınız zaman yazılımı kodlayacağınız testini yapacağınız zaman arasında yazılımın doğasına göre önceden belirlenmiş bazı oranlar var tahminde bu oranlar da kullanılabiliyor. Şimdi asıl sıkıntı işin boyutunu tahmin etmeye ne kadar emek harcadığınız. Gelen işi bir saatte planlayıp sonra bir sene boyunca projeyi o tahmine göre yönetmeye çalışıyorsanız orada bir hata var. İşin boyutunu tahmin etmek için de efor sarf edilmesi gerekiyor. O eforu harcadıkça aynı zamanda veri tabanından bazı verilere ulaşıp işte demin bahsettiğim bir birim zamanda analiz yapıyorsam tasarım için de x birim zaman gerekir kodlamayı da bu şekilde iki birim zamanda yaparsam teslim için de en az bir birim zaman harcamalıyım diye yaklaşıp bunun da pratikte sonuçlarına ulaştıysak pratikte daha önce yaptığımızı yeni projede tekrarlayabiliyoruz. Ama her ekibin farklı yaklaşımına bağlı olarak mesela analizi daha uzun zaman harcayıp tasarım ve kodlamayı daha hızlı yapabilen ekipler olduğu gibi analize hiç zaman harcamayıp doğrudan tasarımdan başlayıp daha sonra kodlayıp test aşamasında çok daha fazla zaman harcayıp burada analize yönelik geriye dönük işler yapan ekipler de oluyor. Bunlar süreçler net olmakla birlikte ekiplerin yaklaşımına üst yönetimin ekiplerden beklentilerine göre faklılık gösterir. Bir senelik proje aldığınızda bir ay geçtiğinde hadi bir şeyler gösterelim dediğinizde o zaman analiz süreçleri atlanıp hızlı bir şekilde geçtiğinizde proje normal bir süreç yaşamayıp sadece müşteri odaklı üst yönetimin müşteriye göstermek istedikleriyle ilgili bir hal alıyor ve o zaman da başarısız olma ihtimali artıyor. Eee tabi müşterinin girdilerinin sağlanması da önemli olduğu için bu iki faaliyet bir arada götürülmeli yani analiz aşamasında müşteriye yazılımı değil ama prototip yazılımlar üzerinden göstererek gitmeli. Bu ana yazılımı etkilememeli.

Yazılımın yeniden kullanılabilmesi herkesin istediği, kesinlikle kullanıldığı durumunda üzerimizdeki yükü eforu azaltabilen bir şey ama uygulanması zor. Bunu uygulayabilen kurumların ekiplerin oldukça başarılı olduğunu görebiliyoruz ama…

Benzer işlerin alınıyor olması mı gereklidir?

Benzer işlerin haricinde de şeyi görebilmek lazım, yani anlık problemi çözmek yerine daha geniş bakarak belki ilk yapılan projede bir birim maliyetle yapılacak bir işi 3-4 birim maliyetle yapmayı gerektiriyor. Oradan edinilenlerle daha sonra gelecek işleri daha düşük maliyetlerle tamamlamak proje sayısı arttıkça bunu sağlayabilmek önemli. Bunu yapabilmek için benzer alanlardan projelerin gelmesi önemli bir de bu konuda çok demokratik de olunmaması gerekiyor. Yeniden kullanım önemliyse birinin şu şu konularda ben bunu yeniden kullanacağım denmesi gerekiyor ve başka bir iş geldiğinde ben bunu başka şekilde yapacağım dediğinde biri ona izin verilmemesi gerekiyor. O da bir ilk başta daha yüksek maliyet gerektirmesi nedeniyle yönetimler tarafından desteklenmiyor ikincisi yönetimin daha sonra gelen projelerde daha düşük maliyetlere yapabilme olanağını ortadan kaldırıyor. Ve her seferinde bu farklı bir trade off olarak ortaya çıkıyor. İlk projeyi bir yerine 3'e yapmayı kimse kabul etmiyor diyelim ki kabul etti, diğer proje geldiğinde o bire değil de diyelim ki 0,8'e yapılacak ama şimdi ben bunu 0,8'e yaparım ama istenilen kalitede olmaz sonuca ulaşmaz müşterinin isteğini karşılamaz diye yeniden müşterinin isteğine göre yapmalıyız diyip o geçmişte yapılanları kullanmayıp çöpe atıp yeniden yapılması her seferinde yeniden kullanım için altyapının oluşmasını engelliyor. Bu dediğim gibi hani…

Sistemi kurmak zorlayıcı diyorsunuz.

Yeniden kullanımın kurulması için hatta organizasyonun değişmesinin de gerektiğini söyleyen yapılar var. Literatürde benzer uygulamalar var oralarda da hep söylenen belli bir zaman içerisinde yeniden kullanım daha baskın hale geliyor. Bu kullanılacaktır deniyor ve yönetimin de orayı desteklemesi gerekiyor. Bunun için de yeniden kullanımın da ayrı bir proje gibi ele alınması ve şirketin kendi iç projesi gibi desteklenmesi gerekiyor.

Standartlar aslında her yönetimin CMMI TQM gibi… Bir tanesi şu, bir yazılım ekibi kurdun, o ekip standartları kullanmadan da zaman içerisinde tecrübe ederek de süreçleri oturtabilir ama zaten pek çok yazılım şirketinin kullandığı ve onların tecrübelerini kullanarak oluşturulmuş yöntemlerin kullanılması bize zaman kazandırıyor. Yani bazen aynı noktaya insanlar deneme yanılma yöntemleriyle gelebiliyor ama yüzlerce firmanın kullandığı bir standart aslında başka insanların daha önce çektiği acılarla kurulmuş.

En güzel yanı da knowledge management, aslında o ekibin tecrübeleriyle oluşturduğu bilgi o ekip ayrılınca yok oluyor, ankete de süreçleri yazmaktaki bir amacım da dokümantasyonu akla getirmek aslında standartlar olmayınca dokümantasyon da yapılmıyor pek yazılımlarda anlaşılan.

Yazılımların dokümantasyonu deyince orada da enteresan bir nokta var dokümanlar yazılımları yaşatmak için hazırlanıyor ancak o dokümanları sonrasında kim kullanıyor dersen o da farklı bir konu.

Çok faydasını görmedik diyorsunuz…

Dokümanın yazılan kodla uyumlu olması ve birbiriyle linklerini kaybetmemesi gerekiyor aslında yazılım geliştirme araçlarının da ulaşmaya çalıştığı şey, dokümanların içlerine yazılım ile ilgili notların da alınması gerekli ve yazılımla birlikte o notların devam etmesi. Burada bizim için daha önemli olan yazılmış olan kodla ilgili bilgi birikiminin bir şekilde korunabilmesi. Geçmişte hiç doküman yazılmadığı zaman oturup o kodu gözle trace edip ne yapıldığını anlamaya çalışıyorduk o zaman yakalamakta zorlanıyorduk. Şimdi yazılım geliştirme araçlarının grafik tabanlı araçlarla yapılıyor olması bir avantaj haline geldi. Bu takibi kolaylaştırdı. Ama hala bu tür yapılan işlemi anlatan notlara ihtiyaç var. Bu notları psedo kod olarak değil de önemli noktaları anlatması gerekli. Ama yüzdeye vuracak olursak kodu yazan kişi veya tasarımı yapan kişi onun dokümanlarını hazırlıyor özellikle tasarımı anlattığı dokümanları ama daha sonrasında başka birisinin o koda müdahale etmesi gerektiğinde ilk gördüğümüz dokümanları okumak yerine kodu açıp anlamaya çalıştığı oluyor. O nedenle dokümanın kodla ilişkisi çok önemli. Diğer yöntem, yazılımcı başka

birinin koduna baktığında çoğunlukla onu beğenmiyor. Orada bir standart olmadığı için diyebiliriz ya da yapış tarzını anlayamadığı için de oturup o kodu yeniden yazıyor.

Yani aslında yazılımı resim gibi görebilir miyiz her ressamın farklı bir tarzı yaklaşımı var, yazılımda da böyle bir sıkıntı oluyor diyorsunuz.

Evet, standartlaşmaya geçtikçe bundan kurtulabiliriz işte grafik tabanlı yazılım geliştirme araçlarıyla bunlar bir nebze engellenebiliyor ama yazılımın içerisinde birisinin yaptığını diğerinin beğenememesi de çok büyük bir problem olarak görünüyor. Çok büyük bir yazılım da olsa ya ben bunu baştan yapsam daha iyi yaparım yaklaşımı var.

Yazılımlarda yani yazılımın geliştirme süreci boyunca kalite yönetimi yapılması hem yazılımın takvimi açısından hem de çıkan ürünün kalitesi açısından olumlu etkileri olduğunu düşünüyorum. Bunun da sebebi geçmiş projelerimize baktığımızda yazılım ekibinden bir kişiye ya da iki üç kişiye siz bu yazılımı yapacaksınız diye verdiğimizde müşteriye teslim edilene kadar o yazılımla sadece onlar ilgileniyorlar, dokümanları var yok kontrolü yapılıyor içerik kontrolü yapılmıyor ve müşteriye çalışır halde sistem teslim ediliyordu. Ve müşterinin çalışıyor veya çalışmıyor şeklinde yorumları oluyordu. Şu anda yapılansa her bir aşamada çıkarılması gereken iş ürünü istenilen şekilde ve doğru zamanda çıkarılmış mı onlar kontrol ediliyor ve CMMI ile birlikte peer reviewlar gündeme geldi bu konuda faaliyetler gösteriliyor ve bu da her seferinde aslında yazılımcıların görmediği kendi işinde bulamadığı hataların bulunmasını sağlıyor. Kalite bölümünde kalite güvencesi konusunda uzmanlaşan arkadaşların olması onların birden çok yazılım projesini gördükten sonra orada yaşanan ortak sıkıntıları gördükten sonra düzeltici ve önleyici faaliyetleri göstermesi yazılım mühendisliğinde yapılan işlerin daha kaliteli hale gelmesini sağlıyor.

Geliştirme ekibine bakacak olursak…

O kısma geçmeden ben bir şey merak ediyorum. "Extreme programming principle"ları hakkında ne düşünüyorsunuz, bunun kullanılıp kullanılmamasına projenin doğasına göre mi karar vermek gerekir ne diyorsunuz?

XP'nin ilk manifestosu yayınlandığında hepimiz bayıldık işte dedik biz de bunu diyorduk. Şimdi oradaki durumda yazılımları uygulamaya çalışırken süreçlere baktığımızda mesela şelale modelinde her şey dokümante ediliyor her şey kontrol ediliyor herhalde batı kültürü için daha uygun bir yöntem ama bize baktığında dokümantasyon odaklı yöntemler pek sevilmiyordu. Bak işte xp konuşarak çözüyorlar beraber bakıyorlar

Happy hour yapıyorlar:)

Minimum doküman çıkarıyorlar hoşumuza giden bir yöntem olmuştu ne kadar doğru bir yöntem diye bakmıştık… Uygulandığı zaman baktığımızda ise aslında ancak daha küçük projelerde başarılı olabiliyor.

"Size" önemli diyorsunuz yani yazılım projesinde…

Bir veya iki kişilik bir yazılım projesinde ortaya çıkacak ürünün kalitesini katlarca arttıracaktır. Bir de ıı tabi bunu yaparken yan yana oturttuğunuz kişilerin anlaşabilmesi, birbirlerinin görüşlerine olumlu yaklaşabilmesi önemli, birisi patron birisi amele şeklinde olursa ve sürekli kodu yazan kişinin işinde hata ararsa diğeri bu ilişki bir zaman sonra kapanabilir.

Tamam.

Ekibe geçtiğimizde gerçekten en yüksek puanı ekiplerin bilgi birikimine vermişim ve tecrübelerine. Bunun bir sebebi aslında ekip haline gelebilmeleri için ekiplerin birbirleri ile

uyum içerisinde çalışabiliyor olması ve bir işi de beraberce her aşamasını yaşayarak bitirmiş olmaları lazım.

Burada ama takım olarak diyorsunuz, takımın uzun süredir bir arada çalışmış olmasından bahsediyorsunuz. Bunu ayrıca "teamwork harmony" olarak ekleyebilirim.

Takımdaki kişilerin de tecrübesi de önemli. Takım içerisinde eğer liderlik yapabilecek, teknik altyapıya sahip biri varsa diğer etrafındaki kişileri yöneterek iyi sonuçlar alabilir ama yaptığımız yazılım projeleri bir kişinin bütün her şeye hakim olmasına uygun projeler değil oldukça büyük projeler o nedenle farklı alanlarda tecrübe kazanmış kişiler kendi tecrübelerini yansıtarak ve açık bir şekilde ekip içerisinde paylaşarak ortak kararlar alabilirse ekip içinde o da önemli oluyor.

Bir diğer tecrübe bazen takımlar oluştuğunda doğru takımlar oluşturamıyoruz. Yani bir ekipte liderlik ruhuna sahip 2 adam olduğunda çatışma yaşanabiliyor.

Bir koltukta iki karpuz hikayesi…

Veya tam tersi ekipte hiç liderlik ruhuna sahip olan biri olmadığında herkes işini yapıyor ama herkes kendi bildiği yöntemle yapınca bütüne baktığınızda eksiklikler kalıyor.

Burada biraz "management"a bağlayabilir miyiz işi? Yani kaynak atamasını sonuçta üst yönetim ya da proje yöneticisi yapacağına göre biraz da takımın oluşturulması iyi yönetimle ilgili.

Evet, burada takım oluşturulurken bir kere orada kullandığı insanların kişisel özelliklerini biliyor olması lazım proje yöneticisinin.

İnsan faktörü hep önemli, pek çok şeyi insan faktörüne bağlıyoruz.

Takım üyelerinin birbiriyle ilişkisinin proje üzerindeki etkilerini biliyor olması lazım. Bazıları olumsuz bazıları olumlu etkiler de oluyor. Tedbir alınması lazım... Bir de insanlar sevdikleri işi yapmak istiyorlar. Mesela çok yaratıcı bir adama yaratıcı işler verdiğinizde bir anda çarpan artarken o adama daha sıradan işler verdiğinizde bir anda o adam o işi sahiplenmiyor. Tersi de geçerli daha sıradan işler yapmayı seven birine de daha yaratıcılık isteyen bir iş verdiğinizde o yaratıcılığı göremiyorsunuz. Hatta çalışma ortamında bile tecrübelerini aktarmak insanlara yansıtmak ama lider olmak istemeyen birine ekip liderliği verildiğinde iş bitmiyor. Tersine o adama kendi başına yapacağı bir iş verip onun diğer ekip üyelerini destekleyebilecek yardım edecek bir duruma getirdiğinizde çok daha iyi oluyor. Herkesin problemini çözen bir adam haline gelebiliyor.

Kimi nerede nasıl kullanacağımız önemli.

Aslında bir madde olarak ekip üyelerinin eğitimi vardı. Bir tarafta teknik eğitimler varken bir tarafta da ekip olma eğitimleri gerekiyor. Genelde biz ekip olma eğitimlerini vermedik ama bazı şirketlerde uygulandığını biliyoruz. İş hayatı haricinde dışarıda sosyal etkinliklerle daha iyi ekipler haline gelinebiliyor. Benim gözlemim insanların iş hayatı haricinde bu tür faaliyetlere yönlendirilmeleri ve bir proje sonunda yaşayacakları başarı hissini paylaşımı, daha önce yaşamaları mesela bir paintball maçında yaşamış olmaları bir anda ekip kaynaşmasını sağlayabiliyor.

Özellikle tek başına bir işi yapmaya kalktığınızda zorlanacağınız ama ekip haline gelindiğinde daha kolay yapılacak konularda farklı sportif konularda olabilir bunu gösterip kaynaşmayı sağlayabilir. Hatta ekip içerisinde liderini bulamamışsa bu aktivitelerde kendiliğinden lider bile belirlenebilir.

Belki içerde çürük yumurtalar yani ekibe uyum sağlayamayacak kişiler varsa onların da ortaya çıkmasına yarar mı?

Evet, görmek açısından iyi olabilir ama bu geri planda. Ödül ve çalışanların mutluluğu… Aslında çalışanlar çok, yani maddi anlamda anlaşılıyor çok ama mesela yaptıkları işin takdir görmesi bile,

Tabi insanların sırtının sıvazlanması bile girebilir bu konuya…

Motivasyonu arttırıcı oluyor zaman zaman hani ters politikalar da kullanılabiliyor onun yerine baskı oluşturmak yerine motive edici politikalar bence daha etkili olabilir. Kullanıcı ve müşterilerle olan konularda aslında ekibin tecrübesi çok etkili. Müşterinin de yaklaşımıyla ilgili sonuçta ilk baştaki ilişki kazan kaybet şeklinde kuruyorsa takım da sürekli kaybediyorsa takım artık problemlere daha kötü yaklaşmaya başlıyor. Daha zorlayıcı oluyor ama müşteri ve takımın ihtiyaçlarını dengeleyip her iki tarafın avantajına işletilebiliyorsa bu tabi proje açısından önemli. Müşteri açısından da baktığımız da ne istediğini bilen müşteri avantajdır ama her şeyi isteyen müşteri aslında her şeyi tariflemekte zorlandığı için çatışmalara sebep oluyor projede.

Bir diğer konu kullanıcıların geliştirenlerle iletişimi, onlarla işbirliği yapması konusu. Buna da katılıyorum çünkü projeyi geliştirenler müşteri gereksinimlerini okuyup işlere başladığında kendi akıllarındaki işi yapıyorlar ama müşterinin gereksinimlerini yazarken ya da müşteriden gereksinimler alınırken her şey net bir şekilde alınamıyor zaman içerisinde müşteriye sorulması gereken yeni şeyler çıkıyor. Müşterinin isteklerinin ana kaynağını bilmek ve onun ihtiyacını bilmek ürünün doğru ürün olarak çıkmasını sağlar. Bu işimizi de kolaylaştırır daha sonra tasarımcı kendi kafasından müşterinin ne istemiş olduğunu yorumlayarak işini yaptığında müşterinin istediğinin çok ötesinde ve kompleks bir yapı ortaya koymaya çalışıyor dolayısıyla müşteri ile konuştuğunda bunun o kadar kompleks olmadığını görüyor. O nedenle değişik aşamalarda müşteriye göstermek mesela yaşam döngüsü olarak mesela evoloutionary yöntemleri tercih etmek önemli çünkü müşteri de projeyle beraber gelişiyor. Dolayısıyla her aşamayı gördükten sonra müşteri bu yapılsa daha iyi olur ya da bu yapılmasa da benim ihtiyacım karşılanıyor gibi çözümleri müşteri de bulabiliyor.

Gereksinimlerin tanımlanması çok önemli aslında burada bankacılık sektörü biraz da önem kazanıyor işi yapanla yaptıran arasında etkileşim var ve bu aşamada neden olduğunu birbirlerine anlatıyorlar ve gereksinimleri birbirlerine aktarıyorlar. Askeri projelerde ise müşteri kullanıcı arayüzü üzerinden ihtiyacını anlatmaya çalışıyor bu nedenle iş akışını anlatmada bazı problemler olabiliyor. Bu nedenle gereksinimleri bizim olabildiğince çok açmamız, detaylandırmamız gerekiyor. Bu gereksinimleri eğer mühendis algısıyla yazılmış bir detaylandırmaysa kullanıcı anlamıyor, kullanıcının detayları da mühendislik tarafında çok yer bulamıyor. İşte bunların ikisini de anlamlandıracak alan uzmanlarına ihtiyaç duyuyoruz işte hem mühendislik dilinden anlayabilecek hem de müşterinin ihtiyacını anlayabilecek kişilere ihtiyacımız var. Bunları kullanan projelerde ortaya çıkan ürünün müşteri ihtiyacını karşılamada daha başarılı olduğunu görüyoruz.

Projenin doğasına geldiğimizde 3 maddemiz var. Boyut, karmaşıklık ve paydaşların farklı amaçları... Farklı amaçlar olan bir yapı için, müşteri ne derse onu yapmaya çalışan bir ekip örneğin müşterinin ağzından çıkanı gerçekleştirip onu mutlu etmeye çalışır. Müşteri ile ekip arasında bir çıkar çatışması da yok. Böyle bir durumda hedefler ortak konmuş oluyor. Diğer tarafta kurumsal olarak yaklaşan firmada birden çok müşteri var, amaç müşteri ihtiyacını minimum eforla karşılamak. Sonuçta başka projeleri de hızlı bitirip minimum kaynakla diğer projeleri de hızlı bitirmek amaç. Müşterinin isteklerine olumlu cevap verse de karşılamada biraz daha sıkıntılı olur bu senaryo. İlk örnekte çıkarlar çatışmadığı için daha kolay. Büyük kurumda ise çalışan arkadaşlar hem işini bitirmek istiyor aynı zamanda kendisi açısından ne kadar değerli olduğunu da ölçüyor, o proje ona ne katacak ileride nasıl bir yere gelmesini sağlayacak gibi. İkincisi projeyi yaparken kendisini hangi konularda geliştirecek. Bunların her biri motivasyonu etkiler. Aynı yönde faaliyetler varsa daha başarılı olur motivasyon artar.

Yine aslında temelde işi yapan adamın şahsi motivasyon etkenleri proje başarısını etkiliyor.

Evet, kurumsal yapıda müşteri ihtiyaçlarının yanı sıra başka etkenler kurallar da var gözetilmesi gereken. Ama her iki yapıda da projeye göre iyi işler çıkabilir. Projede görev alanlar ile müşterinin amaçları çatıştığı durumda projenin başarısı etkilenir ama tabi.

Projenin büyüklüğü ve karmaşıklığı… Yazılımın türüne göre bir başarı farkı olur mu?

Teknolojik olarak karmaşık bir proje bazen insanların başarılı olma hedefleri nedeniyle daha bile motive edici olabilir diye düşünüyorum. Daha sıradan işler yerine daha gizli, zor işleri yapmak daha cazip gelebilir insanlara.

Projenin teknolojik karmaşıklığının nasıl bir sonuç yaratacağı, olumlu mu olumsuz mu, belli olmaz diyorsunuz.

Evet, ayrıca bir problemin daha karmaşık olması kaynak sayısının da fazla olmasını sağlayabilir. Projenin boyutu büyüdükçe işlerin zorlaşmasını bekliyoruz ama aslında işin kapsamının genişlemesinin işi çok da etkilediğini düşünmüyorum.

Peki, var mı case study niteliğinde olumlu ya da olumsuz sonuçlanan bir tecrübeniz, şöyle bir proje oldu ki şu sebeple istediğimizi elde edemedik ya da şöyle bir insan çıktı ve projeyi çok daha başarılı tamamladık diyebileceğiniz?

Biz iş geliştirme dönemlerine çok dikkat ettiğimiz için hep tanımlı işlere girdik, bitiremediğimiz bir iş olmadı. Kaynak atamasında da tecrübeliyiz. Bugüne kadar öyle bir örnek görmedim.

Çok dramatik sonuçlara sebep olan bir olay olmadı yani?

Şöyle bir şey oldu; standartları yeni yeni kullanmaya başladığımız dönemlerde gereksinim analizini çok fazla önemseyip uzatan ve 2 senesini sırf gereksinim analiziyle geçiren projeler oldu. O projelerde ürün çıktı ama takvim sapmaları yaşandı. Bu da şeyin etkisiydi yani standartlara yeni geçilirken adaptasyon sebebiyle analize ağırlık verildi, ve yeni mezun arkadaşlar analiz dönemini yürüttü, başarısızlıklar yaşandı.

Peki, çok teşekkürler zaman ayırdığınız için.

Interviewee 2 (Software Engineering Manager):

Şimdi öncelikle bence yazılım dünyasında süreçler, standartlar filan işte Extreme Programming filan bir ara çok gözdeydi ama artık biraz software reuselar'a dönüldü süreçler önemli olmakla beraber.

Peki sizce neden böyle oldu, yazılım mühendisliğinde belli bir doygunluğa mı ulaşıldı mesela?

Yok, yani etkisi vardır mutlaka ama teknoloji ile birlikte artık yazılımların boyutları da büyümeye başladı. Kod satır sayıları bile yüz katlarına bin katlarına çıktı. İnsanın sınırları da belli tabi o yüzden büyük yazılımları artık ortaya çıkarabilmek için önceden oluşturulmuş blokların kullanılmasına ihtiyaç duyuluyor. Sizin çalıştığınız bir sahada birden fazla ürün teslimatınız varsa önce bunlar üzerinde çalışıp, ortak noktalarını filan önceden belirlerseniz hem daha hızlı hem de daha kaliteli ürünler çıkarabiliyorsunuz. O yüzden şimdi artık tüm dünya buna yönelmeye başladı gibi. Bir product line üzerinden çalışılmaya uğraşılıyor artık. Reuse denince aslında insanların aklına sanki sadece source code geliyor ama aslında gereksinimlerden tutun da test kaynaklarına kadar reuse edilmesi söz konusu source code da aslında çok önemli ama onu tekrar kullanıyorsun ama test tanımlarını tekrar yazıp, test ortamlarını tekrar yaratıyorsan yine bir anlamı yok bence.

O zaman işlevli olmuyor…

Evet, o yüzden baştan çok dikkatli olunmalı ortak noktalar gereksinim aşamasından başlayarak belirlenmeli.

Hızlıca daha önce verilen anketteki faktörler üzerinden bir kere daha geçebilir miyiz? Hali hazırda iki kere puanladınız ama…

Hı hı… Ben puanlarımı değiştirmem de☺

Önceliklendirebileceğiniz maddeler veya eklemek istediğiniz bir şeyler olabilir belki.

Ya yazılım tekrar kullanımının arttırılması bence avantajlı olmakla beraber çok da zor bir konu literatürde bu iş de çok araştırılıyor. Bunda başarısız sonuçlanan girişimler de çok çok fazla oluyor. Bunun nedeni de nedir, bence şudur; yönetimsel destek, planlamanın iyi yapılması, geliştirme ekibinin buna inanıyor olması… Burada bir kısır döngü oluşuyor, en başta yazılımın oluşturulmasında bu etkenler çok kritik, sonra da reuse'a yönelirken aynı etkiler devreye giriyor. Bunun haricinde pek diyebileceğim bir şey yok.

Tamamdır. Bir şey daha sormak istiyorum. Sizce bir yazılım projesinde olmazsa olmaz nedir? Başarılı olması için…

Bu şirket özelinde mi?

Yo, sizin tecrübeniz genel olarak yazılım diyince size ne gösterdiyse, şirket özelinde değil. İçinde bulunduğunuz projeleri düşünün.

Hepsi başarılı bittiği için tabi bir şey diyemiyorum☺ şaka bir yana takvim, bütçe bazen oynamıştır projelerimizde ama her zaman istenen ürünü çıkardık. Kapatmadık yani hiçbir projeyi. Çünkü sonuçta belli bir kalitede işgücü var burada her şeyi çözer demiyorum ama kaliteli insan olunca tabi farklı oluyor. Yazılım dediğin de software işte soft bir şey yani çok elle tutulur bir yanı yok çözemeyeceğin de bir şey yok ama önemli olan ne kadar kaliteli çözdüğün sen bir ürünü teslim edersin mutlaka ama iki üç yıl sonra idamesi nasıl olacak orası ayrıdır işte o kalitesidir. Normalde tasarımınız güzelse daha sonra yaptığınız bir müdahalede kolayca uyum sağlar yazılım ama en ufak müdahalede dağılacak yazılımlar da

çıkarılabiliyor refactor edemezsin yani onu. Bu nerede düzeliyor, ta ki siz reuse edebilecek kadar baştan çalışmayı yaparsanız tekrar düzenleme ihtiyacı kalkıyor tam tersi elindekini korumaya oluşturduğun yapıyı tekrar kullanılır halde tutmaya çalışıyorsun bu sefer. Sonuçta bu donanımlar gibi mekanik gibi değil ki bir şey bir şeye girmiyorsa girmiyordur ama software de bir şekilde bir şey bir şeye uydurulur. Hataları mümkünse yazılım üzerinden düzeltmeye çalışırız biz mesela, daha kolay olan mekaniği değiştirmek değil yazılımın içinden yapmak şeklinde ele alırız. Yazılım kendi içinde kendi düzeltmelerini de yapar.

Biraz yama yapılmış olunmuyor mu o zaman yazılımda?

Evet, yama oluyor bazen ama o yüzden baştan gereksinimleri ve yapıyı iyi belirlemek gerekiyor. Yazılımın kalitesini sağlamanın yolu doğru bir ekip ile doğru bir süreç işletilmesi baştan gerekli.

İnsana bağlanıyor yani olay?

Ya evet insan eli tabi ki ama literatür ve bu işin geçmişi çok önemli sadece kişilerde değil çözüm dünyası, var olan bilgiyi kullanmak gerek. Benim diyeceklerim bu kadar…

Peki çok teşekkür ederim.

Interviewee 3 (Senior Software Engineer):

Top management ile başlayalım…

Top management derken üst yönetimin maddi manevi desteğinden bahsediyoruz...

Ben hala 5 diyorum top management'a.

Top management köstek olsa da adamın niyeti varsa yapar diyorsun.

Yapar tabi ben yazılımdayken, top management'ı çok hissetmediğimiz için ben böyle düşünüyorum. Yaptık yani çünkü bizim için önemli olan YGÖ'ydü, gereksinimlerdi biz YGÖ'yü elimize aldık gereksinimler doğrultusunda işimizi yaptık. Capable decision makers ve strong project managemet'ın işte bu da buna bağlı olarak düşük…

Ama top management ile Project management arasında bir fark var öyle düşün.

Top management ile kastedilen burada Yönetim Kurulu falan, şirket yani.

Tabi tabi. En üst seviye… Halbuki burada kararı veren, ekip lideri mesela.

Bunu o zaman değiştirmem lazım ben farklı düşünmüşüm. Ekip lideri baya etkin çünkü. Ben buna 9 veriyorum hatta çünkü ekip lideri bir şey dedikten sonra sen oturup yapıyorsun.

Bu her projede farklıdır diye mi düşünüyorsun? Neden 4 ?

Ben aslında biraz bütüne bakmaya çalıştım, yazılımda biz hep parçaları oluşturduk, en altta bizler, sonra o parçalar entegre edildi. O yüzden en alttaki yazılımı kalitesiz yaparsan ister incremental yapmışsın ister waterfall yapmışsın o parçalar kalitesizse hiçbir anlamı yok. Bunlar ancak şunu getirir yani bazılarının bazılarına üstünlüğü vardır. Mesela bazı modellerde incremental geri dönüşleri azaltır, o yüzden ben yöntemleri çok önemsemedim alt seviyelerdeki birimin, yazılımın doğru kodlanmış oluşturulmuş olması ön planda, daha kritik. Yani bu da çok etkisini hissetmediğim bir olaydı…

Özellikle yazılımda demoralize olma olayı çok dramatik oluyor. Ya da bir yazılımcıyı motive ettiğin zaman aklına hayaline gelmeyecek işler çıkarabiliyor. Ama keyif alınmadığında hakikaten çok baştan savma oluyor. User participation'da ortalamanın altında kalmışım. Müşteri değil mi bu user?

Evet, son kullanıcı... Sen biraz uzak tutmuşsun kullanıcıyı, ihtiyaç neyse gereksinimleri versin, işe çok karışmasın demişsin.

Evet.

Kimisi de şey olarak düşünüyor, sürekli kullanıcıdan feedback almak daha doğru ürün çıkmasını sağlar diye mesela.

Bu biraz yazılım projesinin türüne bağlı bir şey bu soru. Eğer yazılım projesinde çok miktarda kullanıcı arayüzü varsa, özellikle bizim savunma sanayii için söylüyorum, daha çok müşteri gördüğü şeylere görüş verir, görmediği şeye görüş veremez dolayısıyla biz yazılımdayken gömülü yazılım yaptığımız için kullanıcı da bir görüş veremiyordu, kullanıcının etkisi olmuyordu aslında. Ben buna daha bile düşük verirdim ama kullanıcı arayüzünü etkiler diye düşünerek vermedim.

Yani ilk aşamada user participation olsun, gereksinimler belirlensin diyorsun.

Evet, ilerleyen aşamalarda aşağıdaki kodu etkileyecek bir görüş veremez müşteri bizim projelerde… Benim göz önünde bulundurduğum hep yapı taşları bizim projelerde… Yapı taşları kalitesiz, iyi oturmamış olursa ürün de kalitesiz oluyor, yaşadık biz bunu.

Stakeholders' goals açıklar mısın bir aynı şeyi mi anladık?

Burada örneğin sen yazılımcı olarak işini bitirip çıkmak isterken, müdürün fazla mesai yapmanı bekliyor, üst yönetim karı arttırmaya çalışırken ise müşteri daha az maliyetle daha etkin ürün satın almak istiyor. Paydaşların çıkarları birbirinden farklı yani…

Hıı… Ben bu puanı biraz yükseltmek istiyorum, farklı değerlendirmişim bunu ancak yine de şöyle bir denge var burada, kendi çıkarına çalışsan daha hırslı istekli çalışırsın ama sonuç çok kaliteli ürün olur diyemem. O yüzden yine de çok etkisi olmaz diyorum ben... Yazılım Gereksinim Özellikleri dokümanı varsa elinde çıkar mıkar fark ettirmez bence.

Peki, dramatik bir olay örneğin var mı? Bir kararlarla bir insanla gidişatı değişen bir proje mesela?

Vereyim. Çok var… Yazılım projelerinde hazırlamakta olduğu birim kodun başka bir kişiyle tanımlanan arayüzü varken, 2 kişi ayrı iki kod yazıyor ve bu yazılımlar birbiriyle haberleşecek, bir arayüzleri olacak. Eğer bu iki kodu yazan 2 kişi iyi anlaşıyorsa bu yazılımlar da iyi anlaşıyor, gerçek söylüyorum, bunu yaşadık. Kodları gördük. İtiş kakış olan iki kişinin kodları arasında ne yokuşlar ne itiş kakışlar oluyor… İnsanlar birbirini uğraştırmak için bile inadına bilerek kötü yapı oluşturdukları oldu. Farkında olarak öyle yapmıyor. Ve bu fark ediliyor, adam yine de inadına yapıyı değiştirmiyor, ben böyle yaptım diyor.

Peki, bunların yöneticisi yok mu, bu böyle yapılsın şu şöyle olmasın diye müdahale edecek biri yok mu?

Yani bu tip iletişimler biraz daha kurallara bağlı olsa, yapısı önceden kurulmuş olsa yöneticiye de gerek yok. Kurumsallaşma için oluşturulan yazılı kurallar olsa daha farklı olabilir. Büyük yazılımların birbiriyle haberleşmesi için "Kobra" türü "DDS" türü hazır yazılımlar var yani o ayrı onlar da faydalı mesela. Aynı yazılım projesi içindeki küçük paketlerin haberleşmesi için diyorum… Ara yüzleri iki kişi oturup tanımlayacaksa zor.

Peki, bu yazılımları kodlayan iki kişinin arası kötü, ne oldu iş nasıl bitti?

Son ürün pek de kaliteli olmadı, en ufak değişiklik ihtiyacında göçüyor, çok daha iyi kurgulanabilir, daha kaliteli olabilirdi.

Nasıl çözülebilirdi bu olay?

Bu iki kişinin arayüzü olmayan başka işlere vermek gerekirdi, ya da bir adamı değiştirmek. Adamların arasını düzeltmekle uğraşacak zaman yok, yapı değiştirilebilirdi.

Ya da ayrı bir kişi sadece o ara yüzü tanımlamak için görevlendirilebilir miydi? O 2 kodu alıp üzerinde çalışacak?

Tabi o da olurdu. Ama ne yaparsan yap ekip içi iletişim iyi olacak, şart.

Takım ruhunu ben ayrıca ekleyeceğim zaten. Peki, iş dışı sosyal aktiviteler de faydalı olur mu sence yazılım projelerinde?

Tabi tabi, ama bazı insanlar da iş dışında ve iş üstünde çok farklı davranabiliyor. Bu sorunu kesin çözer diyemiyorum.

Acıklı ama şu ana kadar konuştuklarımızdan benim çıkardığım sonuç şu; ne kadar insandan bağımsız bir hale getirirsen ne kadar mekanikleştirirsen o kadar iyi sonuç çıkar yazılım işinde.

Evet, ama iyi ilişkiler de çok iyi iş çıkarıyor insanlar arasında. Bir de eski kötü tecrübelerden çok ders aldık, bir projede yaşadığımız sorunlardan, bir sonraki projemizde hiç sorun yaşamadık. Orada da ekip içi iletişim kuvvetliydi artı bu eski kötü tecrübeyi bildikleri için çok dikkatliydiler, çok başarılı oldu.

Olumlu bir örnek var mı peki? Gidişatı değiştiren bir şey…

Evet, bir projemizde bir takılma yaşadık, çok yavaşladı iş, aylarca sebebi bulunamadı bir türlü. Bir arkadaşımız oturup çok detaylı çalıştı sadece kendi kodu üzerinde değil tüm yazılım üzerinde çalıştı. Yazılımın kullandığı daha alt seviye bir yazılımın bir ayarıyla ilgili bir sıkıntı olduğunu keşfetti, o ayarı değiştirince sorun çözüldü.

Peki bunu yapmaya onu iten neydi sizce?

Onu iten kişisel araştırmacılık ruhuydu, yani burada bir sorun var deyip geçmedi, ben bunu biraz daha detaylı araştırırsam olur dedi, literatürü araştırdı, akşamları kalıp ekstra çaba sarf etti yani ve kişisel çabasıyla bu işi yaptı. Eskiden bu yazılımı biz 1 senede çıkarız dedikleri işleri bu yazılımın alt birimleri hazır olduğu için birkaç saatte çıkarabiliyorlar. Çok modüler bir yazılım yarattı. Böyle büyük bir iş yaptı tek bir kişi yani.

Peki, çok teşekkürler röportaj için.

Interviewee 4 (Junior Software Engineer):

Şimdi evet diyorum ki ben bir projenin hani böyle sağlıklı ilerlemesi için öncelikle işte proje yöneticisi, işte sistemin başındaki adam, yazılımın başındaki adam, kimler tarafından yapılacaksa bu iş, bunların oturup birbirlerine dependencylerini de göz önüne alarak bir plan çıkartmaları lazım bir. Sonra bu planı sürekli monitor etmeleri ve update etmeleri lazım çünkü hiçbir zaman özellikle bizim projelerde tam olarak hani, inşaat projesi değil şu kadar zamanda olur bu iş diyemiyorsun, beklenmedik sıkıntılar olabiliyor vesaire o yüzden de bu planın sürekli update edilmesi ona göre de önlemler alınması gerekli.

Yaşayan bir plan diyorsun…

Kesinlikle evet bence birincisi yaşayan bir plan olması lazım ve monitör edilmesi lazım hani buna, burada ne vardı use of certain development, yok o değil, quality managment ya da performance evaluation da diyebiliriz, hani onlara ben o yüzden yüksek verdim.

Kontrol mekanizmaları demişsin…

Ha evet, bu tür, bunların yüksek olması lazım çünkü bu farkındalığı arttırıyor, hani anında çözüm üretiyorsun ve hani olası problemleri önceden görüyorsun. Ondan sonra hani en önemli diyeceğim şey, baştan requirements'ların güzel bir şekilde define edilmesi. Gereksinimlerin karşılıklı, customer'ın da işte feedback'iyle birlikte gereksinimleri iyi belirlemek,

İyi bir plan yapabilmek için de şart zaten belirlemek önceden ne istediğini, 10 puan da vermişsin…

Evet, aynen öyle ona da ben çok yüksek verdim aynen öyle,

Genelde de yüksek çıktı zaten ortalama…

Evet ve onun dönem dönem de aslında kullanıcıya gösterilmesi ve feedback alınması da aslında hani en son ulaşacağı o müşterinin memnuniyetini arttıran bir şey bence. Ama bu hani şey de değil, sürekli patch şeklinde bir şey vermesinden bahsetmiyorum, yani birbirine bak sen şunda anlaşmıştık, ben de şöyle bir şey yaptım, mutabıkız değil mi'yi yani onaylatmak için, yani aynı şeyi doğru şeyi mi anlamışız diye,

Müdahale edip sürekli değiştirmesi değil de,

O değil kesinlikle, hatta onu yapmaması için gerekli bütün önlemlerin alınması, hani yapılabildiği kadar aslında oradaki detay seviyesi de önemli ama çünkü çok fazla detay verirsen bu sefer o düşündüğün şekilde olmayabilir bu sefer hani ona uymadığın için sıkıntı yaşayabilirsin o anlamda değil ama ne yapman gerektiği konusunda hani detaylı bir şekilde belirleyip ne yapman gerektiğini, onu da onaylatarak… Benim öyle bir proje deneyimim olmuştu beraber çalıştığımız askerler bize hani bizimle beraber feedback vererek aslında hani biz yaptıkça işte çok extreme olmayacak hani tamamen değiştirmeyecek mantığını ama hani onun da işini kolaylaştıracak vs feedbackler vererek beraber çalışarak çok daha,

Mesela şey gibi mi hani sistemin şurası şöyle olsun şöyle çalışsından ziyade şunun rengi yeşil olsun, kırmızı değil de yeşil olsun gibi daha küçük feedbackler, hani kodu pek değiştirmeyecek türde?

Gibi, ya da mesela belki birazcık işleyişini de değiştirecek ama temelde yaptığın işi değiştirmeyecek şeyler hani yazılımın ana üç tane fonksiyonu vardır buna 4.'yü ekletmekten bahsetmiyorum ya da hani "o üçüncüyü bunu böyle çıkaralım"dan değil ama biraz daha detay verecek hani ufak isteklerde bulunacak ya da mutabık kalınacak gibi… Bu arada dediğim gibi customer'a şey yapmak hani onaylatmak kendini safe side'a almak gibi çünkü ürün çıktıktan sonra adam bu benim istediğim ürün değil derse geri dönmek çok daha zor

oluyor, maliyetli oluyor. O yüzden arada bir customer'dan şey almak şart. Dolayısıyla ben teknik altyapıdır mutlu çalışanlardır eğitimdir, eğitim biraz önemli bir şey tabi. Ama know how'ın o kadar önemli olması da şart değil sonuçta eğitim sağlarsan o konuda ve kontrol edersen ve baştaki şeyleri belirlersen o kadar sıkıntı olmaz gibi geliyor bana.

Ama hep quality management yapılması kontrol edilmesi baştan planlanması hani bak şey de çok önemli capable decision makers bizim en çok sıkıntı çektiğimiz. Bir şöyle yapılsın deniyor ondan sonra bir yer geliyor mesela biri bambaşka bir şeyden bahsediyor. Hani mesela daha önce konuştuğumuz bir şeyi tekrar konuşuyoruz. Halbuki biz ona göre bir karara varmış ve onun için çalışmaya o yönde ilerlemeye başlamışız, başka bir yöne kayıyor ve hani çok da mantıklı elzem nedenleri yok. Orada gerçekten duruma hakim birinin buna gerek olmadığını ve orada doğru kararı almış olsa zaten bir daha öbür yöne dönme gereği olmayacak. Hani elbet bazı kararlar değişecek ama önemli kararlardan bahsediyorum, projenin teslimatını geciktirecek olabilir. O tür durumlarda öngörülü birinin, ben o durumu çok yaşadım burada daha önce mesela çalıştığım ekip liderlerinde çok daha öngörülü olup yaşanabilecek problemleri önceden öngörüp ona göre kararlar veren birisiyle, hani bunları öngöremeyip, o zaman başımıza bir şey gelince bakarız duruma göre düşünürüz diyip ondan sonra da aldığın yolu tamamen geri gidip baştan yol almak onlar çok zor şeyler…

Sen önünü görerek iş yapmak istiyorsun, hem plan hem karar, spontane iş yapmayalım diyorsun…

Aynen öyle o bize yazık, o sürekli yeniden iş yap demek o zaten motivasyonu da düşüren bir şey o an yaptığın şeyi de özenmeden yapmana sebep oluyor bu nasıl olsa son şey olmayacak diyorsun nihai bir şey değil diyorsun ona göre davranıyorsun sen de baştan savma yapıyorsun o yüzden bence baştan ne yapacağın belli olmalı ona vakti daha çok harcamalısın işte neyi ve nasıl yapacağını gereksinimi belirlemeye ve tasarlamaya yapmaya başlamadan önce daha çok vakit harcaman lazım bizde tam tersi oluyor önce haldır huldur girişiyoruz ondan sonra zaten eciş bücüş bir şey oluyor onu zaten düzeltmesi adamın değişen istekleri desen dinlemediğin için doğru düzgün onu o değişen isteklere göre değiştirmek çok zor oluyor. Hani sen çatıyı sağlam kurmayınca böyle yap boz yapmış oluyorsun.

Peki, dokümantasyon için ne düşünüyorsun, bizde örneğin çok mu fazla?

Hayır bizde dokümantasyon filan yok!

Bu genelde yazılımcıların çok şikâyetçi olduğu bir konu, özellikle bizde gereksiz dokümanlar yazıldığı iddia ediliyor. Yazılımcılar kod mu yazacağız doküman mı dolduracağız, süreç gerekleri ağır diyor.

Şimdi gereksiz dokümanlar vardır ama ben hiçbir zaman doküman yazılmasına karşı değilim, gereksiz dokümanlar olabilir, dokümanlar sadeleştirilebilir.

Ama şart diyorsun…

Tabi zorundasın ya da en azından doküman değil de dokümante etme de istersen resim çiz yani bir yöntemini bul bir şekilde... Çünkü mesela yazılımcılar, biz test yaptık ya, şimdi ben mesela yazılım testini yapacağım yazılımı yazan arkadaş sakatlandı rapor aldı gitti ben testini yapacağım sadece elimde ne var, System Requirements Definition dokümanı var. Ben insanlar olmadan da onlara bakıp sistemi anlamam lazım öyle değildi mesela dokümanlar,

Bir de onları anlamak gerekecek…

Evet, yani kişiden bağımsız işlerin yürüyebilmesi için o dokümanlara ihtiyaç var. Hani altını boş şeylerle doldurmak ıvır zıvır şeylere takılmak yerine, işe yarayacak bilgilerle doldurmak lazım. Hani dediğim gibi şekli içeriği değiştirilebilir.

Başka söylemek istediğim bir şey var mı diye bakıyorum.

Şuradan en önemli bulduğun 5 başlığı, maddeyi seçebilir misin?

Management benim için önemli, sonra requirement definition, şimdi technological complexity of the Project, buna ne diyeyim çok önemli değil,

Yani birinde sıradan bir komuta kontrol yazılımıdır, diğeri karmaşık bir füze yazılımı…

Eyvallah yani ama ona göre adam almak lazım vesaire…

Orada da know-how gibi başka şeyler devreye girer,

Evet yani bu tabi ki önemli de, o zaman bu kapsamda karşılaştırmış olamıyoruz. Ben benzer complexity'deki projeler için düşünüyorum.

Savunma sanayi sektöründeki projeler diye ben ayrıca da ekleyeceğim zaten çalışmaya. Ama hani yine de aynı sektörde de olsa karmaşıklığı çok farklı olabilen projeler var. Bir sistem 2 temel yazılım ile çalışırken bir diğer sistemde pek çok yazılım arka planda haberleşiyor mesela…

Anladım.

En önemlilerine şimdi ben capable decision makers dedim, estimation yapmak dedim ve Project control mechanisms. customer'ın kapasitesi çok belirleyici değil hem quality de bir nevi customer'dır bence. Böyle…

Peki, çok teşekkür ederim.

Interviewee 5 (Software Quality Engineer):

Şimdi, üst yönetim desteği… Üst Yönetim desteği firmadan firmaya değişir yani mesela benim daha önce yazılımcı olarak çalıştığım firmada yazılım, sistem entegrasyonu yazılımıydı, satış ağırlıklıydı. Yazılıma üst yönetim desteğini çok önemsemiyorduk. Tamam mı o yüzden problem olmuyordu. Capable decision makers, dediğim gibi buradaki bütün notlar genelde yüksek, ben burada da fikrimi koruyorum. Ya yazılım geliştirmede genelde verilecek kararlar, strong project management özellikle, çok sevimli bir laf değil yani aslında ben düşe de bilirim hatta burada neden çünkü…

Bu ama şöyle de algılanabilir, müşteri ile arayüzün senin project management'ındır ya, müşteri bir şey istediğinde bunun karşılığını cevabını verecek, yapabiliriz yapamayabiliriz, planları değiştirebilecek adam.

Evet, ben kendimi sanki burada bir havuz varmış da onu dağıtıyormuş gibi davranıyorum, aslında bazıları birbirine baskın çünkü bence. Existance and success of control, ya yazılım işini bence kontrol edemezsin yazılım işi adama bağlıdır. O yüzden ben bunu önemsemiyorum. Performans değerlendir, çok zor. Adamın performansını değerlendirmek çok zordur. Yazılımcı adam o gün bir recursive kod yazar, çözer işi o adama doyum verir o gün daha fazla çalışmaz. Yazılımcının performansını nasıl değerlendirebilirsin, yazdığı satır sayısıyla mı? Kesinlikle hayır. O yüzden fazla puan vermem.

Tamam.

Teknik altyapı artık kritik bir şey değil, eskiden olsa farklıydı fakat artık teknoloji çok ilerledi teknik altyapı önemini yitirdi.

Yani yine de sunulan imkanlar, yalnızca donanım anlamında değil yazılım programları vs. de farklı olabiliyor.

Ya ben geçen hafta sırf bir sosyal paylaşım sitesine fotoğraf yükleyebilmek için dünyanın parasını verip ... marka laptop aldım. Yani artık teknik imkanlar çok kritik değil bence, her şey erişilir durumda.

Hı hı…

Estimation on duration and task effort, çok zor… Doğru yapabileni, tutanı ben görmedim. Mesai saatleri dışında çalışma yazılımda zorunluluktur, işin başında gevşek başlayıp sonlara doğru sıkıştırmak yazılım işinin doğasında var mesela. Senin bugün işin ne kadarını yaptığını bile kestirmen bu kadar zorken daha işe başlamadan bunu iyi, doğru planlayabilmek neredeyse imkansız bence. Cocomo'yu Function Point ile kullanan örnekler gördüm, cocomo 2'yi gördüm, hala da tutmadı. O yüzden burayı da geçiyorum. Ya demem o ki, yani keşke yapabilsek ama yapamıyoruz, keşke yapabilsek faydalı olurdu ama zor…

Pekala.

Software reuse. Ya o da zor… Software reuse yapmak kolay bir şey değil ya. Özellikle hazır yazılım geliştirme alanında 3-4 yıl çalışan bir insan olarak söylüyorum, reusable kod yazmak da onu kullanmak da kolay değil.

Bazı durumlarda reuse yerine sıfırdan başlamanın daha kolay olduğunu iddia edenler var, ne düşünüyorsunuz?

Şimdi ikisinin de avantajları dezavantajları var. Ama yani şunu demek istiyorum, şimdi biz bu yazılımda %50 reuse yapacağız dendiğinde ooo yaşasın yarısı reuse dememek lazım. Reuse bela bir şey tamam mı? Tutmuyor çünkü kolay kolay.

Company principles and standarts, buna katılıyorum çünkü yazılımcılar dağıtmaya meyilli adamlardır bir yandan, bir yandan da süreçlere prensiplere ihtiyaç duyan adamlardır. Çocuklar gibi. Mesela çocuk da öyledir ya anne babanın mutlaka biraz guidance göstermesi gerekirmiş. Şimdi bu adam oturuyor on bin satır kod yazacak kalkıp da 30-40 sayfa doküman yazdırmak adam için dert değil aslında ama yaz demezsen de yazmaz. Mesela donanımcılarda bu dert değildir çünkü adam için de işi kontrol altında tutmak için faydalıdır ama yazılımcıda öyle değil.

Evet evet sanırım bizde de donanım dokümanları daha düzenli çıkıyor.

Kesinlikle, dolayısıyla bunun projeye mutlaka etkisi vardır, iyi tanımlanmış standartların ve iyi yürütülen süreçlerin etkisi vardır. The use of certain development methodology… Yani şimdi bu metotların hangisi iyidir, hangisi kullanılmalıdır kesin bir şey yok, her biri farklı durumda başarılı olabilir. Ama çok da kritik değil bence.

Hı hı…

Quality mangement, burada genel ortalamaya en yakın çıktığım yerlerden biri ama yine de uçurum var ☺ Bu etkili aslında, ben buna verdiğim puanı arttırmak istiyorum.

Development team… Evet yani adam iyi olacak ben bunu derim. Bunu yaşamış birisiyim ben. Daha önce çalıştığım şirkette benim, başta adamları seçerek alıyorduk birkaç üniversiteden. Daha sonra o üniversitelerden mezun olup geleni seçmeden aldık. Daha da ilerledi, aynı üniversitelerden mezun olup iş bulamayanı da işe aldık. Son noktada artık o okullardan değil başarı ortalamaları daha düşük okullardan adam almaya başladık. Kalitenin nasıl güme gittiğini ben gördüm. Kesinlikle yazılım işinde en önemli şey bu bence…

Experience and technical know-how… Ben yukarıdaki ile birlikte düşünüyorum. 10 puan☺ Training, eğitim şart☺ Happy staff… Evet, happy staff ya bu şart. Bu iş daha böyle sanal bir iş ya, kafanın daha rahat olması gerekiyor. Benim hep acayip iyi fikirler aklıma duşta gelir, defalarca eşimden duşta kağıt kalem istedim ciddi söylüyorum. Bu çok önemli, kafa rahat olacak.

User client experience, avantajları ve dezavantajları var. Çok bilince de iyi değil ki çok karışıyor. Hiç bilmeyen de aşırı şeyler isteyebiliyor. Bu iyi yönetilmeli, önemli yani. User communication, aynı hikaye…

Requirements definition çok önemli çok. Bir keresinde şöyle bir requirement vermişlerdi bize, bordro yazılımı geliştirilecektir. 3 kelime ya. 5 adam 3 sene çalıştı buna. O bordro diye elimize aldığımız kağıt var ya, onun altında binbir tane iş var. O yüzden requirements çok önemli.

User client patches.. 9 vermişim biraz düşürmek istiyorum, 8.

Size da önemli, projenin büyüklüğü işleri değiştiriyor. Technological complexity. İşte ben daha önce bordro yazılımında çalıştım, şimdi radar yazılımı. Nereden nereye ama işte kullandığın kaynaklar da ona göre değişiyor mutlaka. Olanaklar farklı olduğu için yine de bir denge var, ama karmaşıklık da size kadar etkin bence.

Different goals or purposes of stakeholders. Bu ne demekti hatırlatır mısın?

Yani projede sen belki o gün işini erkenden bitirip eve gitmek istiyorsun, üst yönetim maksimum kar sağlamak, müşteri ise daha ucuza daha az maliyetle daha iyi bir ürün ve hizmet al.mak istiyor, yöneticin sen daha çok çalış istiyor.

Ama bu sadece yazılıma özgü bir şey değil, hayatın her alanında var bu yani bunu bir faktör olarak yazılımda almamalı bence.

Peki şöyle düşünsek, maaşlı çalıştığın bir iş yeri değil de 2 arkadaş sahibi olduğunuz bir yazılım şirketi, kendi işin olarak düşünsen…

Hıı bak bu iyi bir örnek oldu, o zaman motivasyon farklı olur. Puan iyi bence... Ya sonuç olarak yazılım sosyal bir iştir ben böyle görüyorum.

Bunlar hariç aklınıza gelen olumlu/olumsuz bir faktör var mı?

Burada şöyle bir şey var. Belki bunu different goals or purposes of stakeholders altında düşünebiliriz ama. Benim eski işyerimde 2 ayrı ekiptik, biri müşteri için biri ise şirket içi ERP için yazılım geliştiren ekiplerdi. Biz iki ekip kavga ederdik hep şöyle "requirements" lar var bunları kim karşılayacak diye. Savaş çıkıyordu aslıdan iki ekip de yapabilirdi ama burada eksik iyi tanımlanmış gereksinimlerdi. Şirketin bunu iyi yönetmesi lazım... Bu kavgalar çalışanları yıpratıyor. Happy staff çok önemli işte o illa ki piknik, güzel yemekle olmaz, iş tatmini olarak düşünüyorum onu ben aslen. Ben mesela eskiden IT sektöründe yazılımcıydım şimdi burada yazılım kalite sorumlusuyum şimdi yaptığım işten bir mühendis olarak daha çok keyif alıyorum. En kötü şey, yaptığın bir şeyi çöpe atmaktır, bu motivasyonu performansı filan mahveder. Gerçi sonuç olarak requirements definition'da düşünmüşüm ben bunu oraya bağladık☺

Var mı başka eklemek istediğiniz bir şey, örneğin Google gibi bir ortamda olsak daha iyi yazılımlar çıkar mıydı? Kaydıraktan kaysak mesela☺

Ya yapılan işin doğasına göre değişir. Mesela senin köpeğin yanında olsa ben rahatsız olacağım için o iş kötü olur☺ O işlerin olayı ilk önce çıkmak, bir fikir ile öncü olmak. Yazılımları zor ve büyük olduğundan değil. Google'da kişi başına ciro vesaire çok fazla, yaratıcılık da daha önemli çünkü orada, olay adamın aklına gelen fikir yani, iş farklı.

Peki, çok teşekkür ederim zamanınız için.

APPENDIX C: THE CODING PROCESSES

CODING of RESEARCHER

Codes (Open Coding)	Themes (Axial Coding)	Categories (Axial Coding)
Capable decision makers that support the development team	Decision Mechanisms	Management
Project control in every phase	Control Mechanisms	Processes
Performance evaluation is difficult but	Control Mechanisms	Processes
Team members should have clear objectives, task and responsibility assignment	Motivation	Team
Technical infrastructure should be appropriate	Infrastructure and Tools	Processes
Technical experience of team members	Technical skills	Team
Project similarities are good for reuse	Software Reuse	REUSE
Living Plan	Planning and Implementing	Processes
Reuse, good but difficult to apply and manage	Software Reuse	REUSE
Reuse, costly at the beginning	Software Reuse	REUSE
Successful decision makers foreseeing what is to come	Decision Mechanisms	Management
Reuse, as a new project, maybe change in organization	Software Reuse	REUSE
Standards, useful but difficult to sustain	Standard Processes	Processes
Good, standardized, easy-to-understand documentation	Project Requirements	User/Client
Education background of development team members	Technical skills	Team
Quality management in every phase of the project	Control Mechanisms	Processes
Control/reviews by an objective third eye	Control Mechanisms	Processes
The development model should be appropriate	Development Model	Processes
Clearly defined responsibilities	Motivation	Team
Strong relationship between team members	Teamwork and Harmony	Team
Experience and know-how of team members	Technical skills	Team
Members should be team players	Teamwork and Harmony	Team
One leader in one team	Teamwork and Harmony	Team
Personal attributes of team members	Teamwork and Harmony	Team
Different team members like different works	Motivation	Team
Social activities for team members	Teamwork and Harmony	Team
Team members' motivation and job satisfaction	Motivation	Team

Codes (Open Coding)	Themes (Axial Coding)	Categories (Axial Coding)
Customer needs and wants should be manageable	Project Requirements	User/Client
Good communication with the customer	Communication and Feedback	User/Client
Understanding customer needs	Project Requirements	User/Client
Customer confirmation and feedback at predefined steps	Communication and Feedback	User/Client
Experts to create common understanding between customer and the project team	Control Mechanisms	Processes
Common goals of stakeholders affects team motivation	Common Goals	Nature of Project
Project complexity and challenges may work as both positive and negative triggers affects motivation	Complexity and Challenges	Nature of Project
Clear definition of the work	Project Requirements	User/Client
Reuse, of source code, all kinds of documentation, test requirements and test infrastructure	Software Reuse	REUSE
For reuse; defining common/reusable parts beforehand is crucial	Software Reuse	REUSE
There is a loop between the factors (such as management, good planning) and software reuse	Software Reuse	REUSE
Capable team members	Technical skills	Team
Modifiable code blocks for reuse	Software Reuse	REUSE
Clearly defined requirements.	Project Requirements	User/Client
Capable decision makers	Decision Mechanisms	Management
Motivated team members	Motivation	Team
Customer involvement should be under control	Communication and Feedback	User/Client
Team members' relationships	Teamwork and Harmony	Team
Standards and tools for software development	Infrastructure and Tools	Processes
Removing human effect is not necessary, in fact maybe useful and may add value to work	Motivation	Team
A living plan, updated at all phases	Planning and Implementing	Processes
Monitoring the plan and the happenings, control mechanisms	Control Mechanisms	Processes
Requirements should be as clear as possible	Project Requirements	User/Client
Customer feedback	Communication and Feedback	User/Client
Customer feedback, but there are boundaries to which extent the extra demands will be met	Communication and Feedback	User/Client

Codes (Open Coding)	Themes (Axial Coding)	Categories (Axial Coding)
Training of team members affects know-how and experience	Technical skills	Team
Experienced and skilled managers	Decision Mechanisms	Management
Should eliminate start-overs	Planning and Implementing	Processes
Team members motivation	Motivation	Team
Good documentation, common understanding, independent from people	Standard Processes	Processes
Planning all phases by a development model	Planning and Implementing	Processes
Reuse, good but difficult	Software Reuse	REUSE
Standards	Standard Processes	Processes
Documentation is necessary but affected by standards, without standards and processes you can't make developers create documents	Standard Processes	Processes
Software development methods are affected by the nature of projects		Processes
Quality management	Control Mechanisms	Processes
Team members' capabilities	Technical skills	Team
Training adds value to project team members	Technical skills	Team
Happy and motivated team members	Motivation	Team
Balanced user involvement, not too much, not too little	Communication and Feedback	User/Client
Clearly defined requirements	Project Requirements	User/Client
Size and scope of the project is important	Size and Scope	Nature of Project
Similar goals of stakeholders affect motivation	Common Goals	Nature of Project
Job satisfaction, affects motivation	Motivation	Team

CODING of EXPERT #1 (Engaged in Software Development)

Codes (Open Coding)	Themes (Axial Coding)	Categories (Axial Coding)
top level management support	Top level management support	Top Level Management
low level management shall take responsibilities	Low level management capability	Low Level Management
low level management decision capabilities are crucial	Low level management capability	Low Level Management
parallelism between management decisions and company goals	Goals shall be clear and directive	Company infrastructure, control mechanisms and profile
more detailed control mechanisms are required	Control mechanism effectiveness	Company infrastructure, control mechanisms and profile
staff or developers shall be well informed on control mechanisims	Control mechanism effectiveness	Company infrastructure, control mechanisms and profile
low management shall dictate usage of control mechanisms	Control mechanism effectiveness	Company infrastructure, control mechanisms and profile
goals of the project shall be clearief by low managment	Goals shall be clear and directive	Company infrastructure, control mechanisms and profile
team work affects project success	Team harmony	Core Team
no need to expensive tools	Infrastructure	Company infrastructure, control mechanisms and profile
workforce experienve is critical	Worker experience	Core Team
project estimations (time, workplan, financial) is hard, but necessary to manage	Low level management capability	Low Level Management
succesfull project planning depends on experience	Low level management capability	Low Level Management
low level management project planning is crucial	Low level management capability	Low Level Management
workforce approach/characteristics on project process affects project success	Worker approach/character	Core Team
customer shall clearly specify project requirements	Clear understanding with customer	Customer relations
reuse is desiable but very hard, a succesfull and experienced development team is required for a successfull reuse	worker experience	Core Team
top level management decision on reuseis crtitical	Top level management support	Top Level Management

Codes (Open Coding)	Themes (Axial Coding)	Categories (Axial Coding)
reuse of the project gets harder by customer strict/different and unique requirements	Customer requirements	Customer relations
reuse shall be supported by top level management	Top level management support	Top Level Management
standard are benefical to decrease the time required to gain experience	Company/Literature standards	Company infrastructure, control mechanisms and profile
code developer shall bind the software and software document successfully	Worker perception/commitment	Core Team
code developer shall use coding standards and documentate the code smartly	Worker experience	Core Team
development team shall work together for along time for succesfull developement projects	Team harmony	Core Team
standard applications, investigation, quality check increases the success	Related standards	Company infrastructure, control mechanisms and profile
background of the software developers are important and low level management shall consider it	worker experience	Core Team
good humour and relationships between the workers are important	Team harmony	Core Team
team experience as a team is required	Worker experience	Core Team
experienced and participative team members	Worker experience	Core Team
leadership inside the team is required	Team harmony	Core Team
developer characters are important at their responsibilities	Worker perception/commitment	Core Team
relationship between team members are important and shall be supported	Team harmony	Core Team
good relations within the team	Team harmony	Core Team
worker motivation and happiness	Team harmony	Core Team
team experience on customer relations are important	Worker experience	Core Team
customer relations, support and feedback during the project is important	Customer relations	Customer relations
a mediator shall be present in team negotiate with customer	Customer relations	Customer relations
company profile affect project nature	Company profile/infrastructure	Company infrastructure, control mechanisms and profile
individual motivation of team members affects project nature	Worker perception/commitment	Core Team

Codes (Open Coding)	Themes (Axial Coding)	Categories (Axial Coding)
benefits of team members, company and customer shallbe parallel for success	Project requirements	Company infrastructure, control mechanisms and profile
project scope affects motivation of project workers	Project requirements	Project Goals
roles shall be given according to member experiences	Low level management capability	Low Level Management
reuse is key factor for successs	Reuse	Reuse
project requirements and goals should be clear	Project requirements	Project Goals
for a proper reuse of a software, management and workers play critical roles	Reuse	Reuse
quality workforce brings success or easiness for a software project	Worker quality	Core Team
for a proper reuse, experience of the team members are important	Worker experience	Core Team
literature, experience and good development team	Worker experience	Core Team
motivated team members	Worker perception/commitment	Core Team
team leader is important	Team harmony	Core Team
team member motivation is critical	Worker perception/commitment	Core Team
user feedbacks may not always be possible, but if possible, they affects nature of project	Customer relations	Customer relations
base of the software is keypoint, a good workplan is required	Low level management capability	Low Level Management
relationship between developers affects the software quality	Team harmony	Core Team
members should get with each other well	Team harmony	Core Team

CODING of EXPERT #2 (Engaged in Project Management)

Codes (Open Coding)	Themes (Axial Coding)	Categories (Axial Coding)
top management support	top management	project administration
existence of successful decision makers	decision makers	project administration
Reliability and comparability of data used for project control mechanisms	control mechanisms	project administration
Determining goals and making sure that the project team understands the goals	top management	project administration
teamwork is important	teamwork and coherence	software engineering group
reuse of common baselines within different projects	reuse of software	software reuse
existence of experienced project team	experienced team	software engineering group
it is difficult to make time and effort estimation for new kinds of projects	planning	software engineering group
	similar projects	company literature
estimation of the size of the project beforehand is critical but difficult	planning	software engineering group
prototyping the end product may be useful	prototyping	company literature
reuse of software is difficult but important	reuse of software	software reuse
the main obstacle for software reuse is the first development costs	reuse of software	software reuse
using standards helps overcome the mistakes made when setting the processes	standards	company literature
documentation helps protecting the knowhow developed on software	documentation	company literature
taking notes to explain the work is important	documentation	company literature
standardization is necessary to lessen the effect of persons on software	standards	company literature
quality management through the software development process	quality management	project administration
xp brings success only for small projects	size of the project	project characteristics
teamwork and coherence is important, a team that worked for long together	teamwork and coherence	software engineering group
experience of the project team	experienced team	software engineering group
task assignment in the team in accordance with the experience of team members	task assignment	software engineering group
team leader in the project	leadership	software engineering group
task assignment in the team in accordance with the personal attributes of team members	task assignment	software engineering group
training of project team	training	software engineering group
social activities help increasing the cohesion within project team	teamwork and coherence	software engineering group
motivating the project team	team motivation	software engineering group
client that can explain his or her real need	requirements	project characteristics

Codes (Open Coding)	Themes (Axial Coding)	Categories (Axial Coding)
determining the client need right	requirements	project characteristics
on different levels of project, the project team should communicate with client, and determine the changing needs	communication with client	project administration
when applying new standards, adaptation problems may create delays in projects	standards	company literature
larger software brings the need to create reusable sub blocks of software	size of the project	project characteristics
determining the common points between different projects	similar projects	company literature
top management support is important for reusable software	top management	project administration
	reuse of software	software reuse
good planning	planning	software engineering group
motivation of project development team	team motivation	software engineering group
quality of software development team	good team	software engineering group
determining the project requirements right	requirements	project characteristics
the willingness of projects team for the project	team motivation	software engineering group
team leader in the project	leadership	software engineering group
coding the software right from the beginning to the end	good team	software engineering group
motivation of project development team	team motivation	software engineering group
coherence of project team	teamwork and coherence	software engineering group
task assignment in accordance with the personal attributes of the project team	task assignment	software engineering group
good projects planning and updating the plans when needed	planning	software engineering group
determining the project requirements right	requirements	project characteristics
on different levels of project, the project team should communicate with the client, and take approval for the job done	communication with client	project administration
project requirements should not change dramatically during the projects	requirements	project characteristics
quality management	quality management	project administration
existence of qualified decision makers	decision makers	project administration
documentation should help doing the same task without the same persons	documentation	company literature
performance measurement is difficult but good	quality management	project administration
planning the project is good but difficult	planning	software engineering group
existence of standards is necessary	standards	company literature
quality management	quality management	project administration

Codes (Open Coding)	Themes (Axial Coding)	Categories (Axial Coding)
quality of project team	good team	software engineering group
experience of project team	experienced team	software engineering group
happy and peaceful project team	teamwork and coherence	software engineering group
determining the project requirements right	requirements	project characteristics

APPENDIX D: THE MATCHING OF CODES AND CATEGORIES BY THE INTERVIEWEES

	Interviewee 1 (Program Manager)		Interviewee 2 (Software Engineering Manager)		Interviewee 3 (Senior Software Engineer)		Interviewee 4 (Junior Software Engineer)		Interviewee 5 (Software Quality Engineer)	
Codes	**Themes**	**Categories**	**Themes**	**Categories**	**Themes**	**Categories**	**Themes**	**Categories**	**Themes**	**Categories**
Capable decision makers that support the development team	Decision Mechanisms	Management	Decision Mechanisms	Management	Decision Mechanisms	Management	Decision Mechanisms	Management	Decision Mechanisms	Management
Project control in every phase	Control Mechanisms	Processes	Control Mechanisms	Management	Control Mechanisms	Management	Control Mechanisms	Processes	Control Mechanisms	Processes
Performance evaluation is difficult but effective	Control Mechanisms	Processes	Planning and Implementing	Team	Control Mechanisms	Management	Planning and Implementing	Management	Planning and Implementing	Processes
Team members should have clear objectives, task and responsibility assignment	Team Motivation	Team	Teamwork and Harmony	Management	Team Motivation	Team	Teamwork and Harmony	Team	Team Motivation	Team
Technical infrastructure should be appropriate	Infrastructure and Tools	Processes	Infrastructure and Tools	Management	Infrastructure and Tools	Processes	Infrastructure and Tools	Processes	Infrastructure and Tools	Management
Technical experience of team members is important	Team Technical Skills	Team	Team Technical Skills	Team	Team Technical Skills	Team	Team Technical Skills	Team	Team Technical Skills	Team
A living plan is necessary	Planning and Implementing	Management	Planning and Implementing	Team	Planning and Implementing	Management	Planning and Implementing	Management	Planning and Implementing	Processes
Successful decision makers foreseeing what is to come	Decision Mechanisms	Management	Decision Mechanisms	Management	Decision Mechanisms	Management	Decision Mechanisms	Management	Decision Mechanisms	Management
Standards, useful but difficult to sustain	Standard Processes	Processes	Standard Processes	Processes	Standard Processes	Processes	Standard Processes	Processes	Standard Processes	Processes
Good, standardized, easy-to-understand documentation	Standard Processes	Processes	Development Model	Processes	Standard Processes	Processes	Standard Processes	Processes	Standard Processes	Processes
Educational background of development team members	Team Technical Skills	Team	Team Technical Skills	Team	Team Technical Skills	Team	Team Technical Skills	Team	Team Technical Skills	Team
Quality management in every phase of the project	Standard Processes	Processes	Control Mechanisms	Management	Standard Processes	Processes	Standard Processes	Processes	Standard Processes	Processes
Control/reviews by an objective third eye	Communication and Feedback	User/Client	Communication and Feedback	User/Client	Control Mechanisms	Management	Communication and Feedback	Team	Communication and Feedback	Team
The development model should be appropriate	Development Model	Processes	Development Model	Processes	Development Model	Processes	Development Model	Processes	Development Model	Team

	Interviewee 1 (Program Manager)		Interviewee 2 (Software Engineering Manager)		Interviewee 3 (Senior Software Engineer)		Interviewee 4 (Junior Software Engineer)		Interviewee 5 (Software Quality Engineer)	
Codes	**Themes**	**Categories**	**Themes**	**Categories**	**Themes**	**Themes**	**Categories**	**Themes**	**Themes**	**Categories**
Clearly defined responsibilities	Teamwork and Harmony	Team	Teamwork and Harmony	Management	Decision Mechanisms	Management	Decision Mechanisms	Management	Decision Mechanisms	Management
Strong relationship between team members	Teamwork and Harmony	Team	Teamwork and Harmony	Management	Teamwork and Harmony	Team	Teamwork and Harmony	Team	Teamwork and Harmony	Team
Experience and know-how of team members	Team Technical Skills	Team	Team Technical Skills	Team	Team Technical Skills	Team	Team Technical Skills	Team	Team Technical Skills	Team
Team members should be team players	Teamwork and Harmony	Team	Teamwork and Harmony	Management	Teamwork and Harmony	Team	Teamwork and Harmony	Team	Team Motivation	Team
One leader in one team	Teamwork and Harmony	Team	Teamwork and Harmony	Management	Teamwork and Harmony	Team	Teamwork and Harmony	Team	Teamwork and Harmony	Team
Personal attributes of team members	Team Motivation	Team	Teamwork and Harmony	Management	Teamwork and Harmony	Team	Teamwork and Harmony	Team	Teamwork and Harmony	Team
Different team members like different works	Teamwork and Harmony	Team	Teamwork and Harmony	Management	Teamwork and Harmony	Team	Teamwork and Harmony	Team	Teamwork and Harmony	Team
Social activities for team members	Team Motivation	Team	Team Motivation	Team	Team Motivation	Team	Team Motivation	Management	Team Motivation	Team
Team members' motivation and job satisfaction	Team Motivation	Team	Team Motivation	Team	Team Motivation	Team	Team Motivation	Management	Team Motivation	Team
Customer needs and wants should be manageable	Common Goals of Shareholders	User/Client	Project Requirements	Team	Project Requirements	Team	Project Requirements	User/Client	Project Requirements	User/Client
Good communication with the customer	Communication and Feedback	User/Client	Communication and Feedback	User/Client	Communication and Feedback	User/Client	Communication and Feedback	Team	Communication and Feedback	Team
Understanding customer needs	Communication and Feedback	User/Client	Communication and Feedback	User/Client	Communication and Feedback	User/Client	Project Requirements	User/Client	Project Requirements	User/Client
Customer confirmation and feedback at predefined steps	Communication and Feedback	User/Client	Communication and Feedback	User/Client	Communication and Feedback	User/Client	Communication and Feedback	Team	Project Requirements	User/Client
Experts to create common understanding between customer and the project team	Communication and Feedback	User/Client	Communication and Feedback	User/Client	Communication and Feedback	User/Client	Communication and Feedback	Team	Communication and Feedback	Team

	Interviewee 1 (Program Manager)		Interviewee 2 (Software Engineering Manager)		Interviewee 3 (Senior Software Engineer)		Interviewee 4 (Junior Software Engineer)		Interviewee 5 (Software Quality Engineer)	
Codes	**Themes**	**Categories**	**Themes**	**Categories**	**Themes**	**Themes**	**Categories**	**Themes**	**Themes**	**Categories**
Common goals of stakeholders affects team motivation	Team Motivation	Team	Team Motivation	Team	Team Motivation	Team	Team Motivation	Management	Common Goals of Shareholders	Management
Project complexity and challenges may work as both positive and negative triggers affects motivation	Complexity and Challenges	Nature of Project	Size and Scope	Nature of Project	Complexity and Challenges	Nature of Project	Complexity and Challenges	Nature of Project	Size and Scope	Nature of Project
Clear definition of the work	Project Requirements	User/Client	Project Requirements	Team	Planning and Implementing	Management	Project Requirements	User/Client	Planning and Implementing	Processes
Capable team members	Team Technical Skills	Team	Team Technical Skills	Team	Team Technical Skills	Team	Team Technical Skills	Team	Team Technical Skills	Team
Clearly defined requirements.	Project Requirements	User/Client	Project Requirements	Team	Planning and Implementing	Management	Project Requirements	User/Client	Planning and Implementing	Processes
Capable decision makers	Decision Mechanisms	Management	Decision Mechanisms	Management	Decision Mechanisms	Management	Decision Mechanisms	Management	Decision Mechanisms	Management
Motivated team members	Team Motivation	Team	Team Motivation	Team	Team Motivation	Team	Team Motivation	Management	Team Motivation	Team
Customer involvement should be under control	Communication and Feedback	User/Client	Communication and Feedback	User/Client	Communication and Feedback	User/Client	Communication and Feedback	Team	Communication and Feedback	Team
Team members' relationships	Teamwork and Harmony	Team	Teamwork and Harmony	Management	Teamwork and Harmony	Team	Teamwork and Harmony	Team	Teamwork and Harmony	Team
Standards and tools for software development	Standard Processes	Processes	Standard Processes	Processes	Standard Processes	Processes	Standard Processes	Processes	Standard Processes	Processes
Removing human effect is not necessary, in fact maybe useful and may add value to work	Standard Processes	Processes	Standard Processes	Processes	Standard Processes	Processes	Standard Processes	Processes	Standard Processes	Processes
A living plan, updated at all phases	Planning and Implementing	Management	Planning and Implementing	Team	Planning and Implementing	Management	Planning and Implementing	Management	Planning and Implementing	Processes
Monitoring the plan and the happenings, control mechanisms	Planning and Implementing	Management	Control Mechanisms	Management	Control Mechanisms	Management	Control Mechanisms	Processes	Control Mechanisms	Processes
Requirements should be as clear as possible	Project Requirements	User/Client	Project Requirements	Team	Planning and Implementing	Management	Project Requirements	User/Client	Planning and Implementing	Processes

	Interviewee 1 (Program Manager)		Interviewee 2 (Software Engineering Manager)		Interviewee 3 (Senior Software Engineer)		Interviewee 4 (Junior Software Engineer)		Interviewee 5 (Software Quality Engineer)	
Codes	**Themes**	**Categories**	**Themes**	**Categories**	**Themes**	**Themes**	**Categories**	**Themes**	**Themes**	**Categories**
Customer feedback	Communication and Feedback	User/Client	Communication and Feedback	User/Client	Communication and Feedback	User/Client	Communication and Feedback	Team	Communication and Feedback	Team
Customer feedback, but there are boundaries to which extent the extra demands will be met	Communication and Feedback	User/Client	Communication and Feedback	User/Client	Communication and Feedback	User/Client	Communication and Feedback	Team	Communication and Feedback	Team
Training of team members affects know-how and experience	Team Technical Skills	Team	Team Technical Skills	Team	Team Technical Skills	Team	Team Technical Skills	Team	Team Technical Skills	Team
Experienced and skilled managers	Decision Mechanisms	Management	Decision Mechanisms	Management	Decision Mechanisms	Management	Decision Mechanisms	Management	Decision Mechanisms	Management
Should eliminate start-overs	Planning and Implementing	Management	Planning and Implementing	Team	Planning and Implementing	Management	Planning and Implementing	Management	Planning and Implementing	Processes
Team members motivation	Team Motivation	Team	Team Motivation	Team	Team Motivation	Team	Team Motivation	Management	Team Motivation	Team
Good documentation, common understanding, independent from people	Standard Processes	Processes	Standard Processes	Processes	Standard Processes	Processes	Standard Processes	Processes	Standard Processes	Processes
Planning all phases by a development model	Planning and Implementing	Management	Planning and Implementing	Team	Planning and Implementing	Management	Planning and Implementing	Management	Planning and Implementing	Processes
Standards should be followed	Standard Processes	Processes	Standard Processes	Processes	Standard Processes	Processes	Standard Processes	Processes	Standard Processes	Processes
Documentation is necessary but affected by standards, without standards and processes you can't make developers create documents	Standard Processes	Processes	Standard Processes	Processes	Standard Processes	Processes	Standard Processes	Processes	Standard Processes	Processes
Software development methods are affected by the nature of projects	Size and Scope	Nature of Project	Size and Scope	Nature of Project	Size and Scope	Nature of Project	Size and Scope	Nature of Project	Size and Scope	Nature of Project
Quality management	Control Mechanisms	Processes	Control Mechanisms	Management	Control Mechanisms	Management	Control Mechanisms	Processes	Control Mechanisms	Processes
Team members' technical capabilities	Team Technical Skills	Team	Team Technical Skills	Team	Team Technical Skills	Team	Team Technical Skills	Team	Team Technical Skills	Team

	Interviewee 1 (Program Manager)		Interviewee 2 (Software Engineering Manager)		Interviewee 3 (Senior Software Engineer)		Interviewee 4 (Junior Software Engineer)		Interviewee 5 (Software Quality Engineer)	
Codes	**Themes**	**Categories**	**Themes**	**Categories**	**Themes**	**Themes**	**Categories**	**Themes**	**Themes**	**Categories**
Training adds value to project team members	Team Technical Skills	Team	Team Technical Skills	Team	Team Technical Skills	Team	Team Technical Skills	Team	Team Technical Skills	Team
Happy and motivated team members	Team Motivation	Team	Team Motivation	Team	Team Motivation	Team	Team Motivation	Management	Team Motivation	Team
Balanced user/client involvement, not too much, not too little	Communication and Feedback	User/Client	Communication and Feedback	User/Client	Communication and Feedback	User/Client	Communication and Feedback	Team	Communication and Feedback	Team
Clearly defined requirements	Project Requirements	User/Client	Project Requirements	Team	Planning and Implementing	Management	Project Requirements	User/Client	Planning and Implementing	Processes
Size and scope of the project is important	Size and Scope	Nature of Project	Size and Scope	Nature of Project	Size and Scope	Nature of Project	Size and Scope	Nature of Project	Size and Scope	Nature of Project
Similar goals of stakeholders affect motivation	Team Motivation	Team	Team Motivation	Team	Team Motivation	Team	Team Motivation	Management	Team Motivation	Team
Job satisfaction, affects motivation	Team Motivation	Team	Team Motivation	Team	Team Motivation	Team	Team Motivation	Management	Team Motivation	Team

APPENDIX E: TEZ FOTOKOPİSİ İZİN FORMU

ODTÜ

ENFORMATİK ENSTİTÜSÜ

YAZARIN

Soyadı : DİLBER

Adı : BAŞAK

Bölümü : BİLİŞİM SİSTEMLERİ

TEZİN ADI (İngilizce) : SOFTWARE DEVELOPMENT IN PRACTICE: A QUALITATIVE STUDY IN TURKISH DEFENSE SECTOR

TEZİN TÜRÜ : Yüksek Lisans ☐ Doktora ☐

1) Tezimden fotokopi yapılmasına izin vermiyorum ☐

2) Tezimden dipnot gösterilmek şartıyla bir bölümünün fotokopisi alınabilir ☐

3) Kaynak gösterilmak şartıyla tezimin tamamının fotokopisi alınabilir ☐

Yazarın imzası Tarih