UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

ENHANCING TEAMWORK IN SOFTWARE PROJECTS DEVELOPED IN THE ACADEMIA

TESIS PARA OPTAR AL GRADO DE
MAGISTER EN CIENCIAS MENCIÓN COMPUTACIÓN

MAÍRA REJANE MARQUES SAMARY

PROFESOR GUÍA:
SERGIO OCHOA DE LORENZI

MIEMBROS DE LA COMISIÓN
ALEXANDRE BERGEL
NELSON BALOIAN TATARYAN
YADRAN ETEROVIC SOLANO

SANTIAGO DE CHILE
NOVIEMBRE 2011

Abstract

Software engineering is an important area within industry and academia. Normally there is a high demand for well-trained software engineers, since chips and code are embedded in almost all consumer products. Consequently young professionals who finish their studies in Computer Science or Informatics have many job opportunities, and the majority of them will work on software development, a human centered process.

As a human centered process, human factors have a great impact on the process and its performance. Although human factors have been proven to have an impact on the software development process, they are still overlooked by researchers. One of the most important human centered processes involved is the one that deals with the coordination of the activities and the ability to combine people skills and teamwork.

In Computer Science, particularly in software engineering, effective teamwork can mean the difference between a positive or negative outcome of a development project. Educational institutions offering Computer Science programs must accept the responsibility to prepare their graduate students not only in technical issues, but also in soft skills that allow them to work efficiently in their professional careers.

Trying to address this problem is complicated; I state there exists a short list of variables that systematically influence teamwork in software projects conducted by small and novice development teams. We also stated that ThinkLets (activity or process that produces predictable results to deal with recurring collaboration problems) could be used to mitigate recurrent situations that affect teamwork.

To do so, a Software Engineer Project Course was observed during two semesters. After a literature review on the subject three variables were chosen to be evaluated: Communication, Coordination and Motivation. With bases on these we concluded that these variables were the most important ones. The most recurrent team problems were found in the literature and so they're possible solution. The teams observed generated a list of problems and so a list of ThinkLets was created and the practices were tested.

An analysis of the data observed showed that the three variables found were the most important ones and that the ThinkLets created were able to effectively mitigate the negative situations affecting teamwork.

Resumen

La ingeniería de software es un área relevante en la comunidad científica y también en la industria. Normalmente existe una importante demanda por ingenieros de software bien entrenados, dado que las líneas de código en los productos de consumo masivo, se duplican cada dos años aproximadamente. Los profesionales que terminan los estudios de las Ciencias de la Computación o Informática tienen muchas oportunidades de trabajo, porque existe una demanda no satisfecha en el mercado laboral. La mayoría de estos profesionales trabaja en desarrollo de software; un proceso centrado en las personas.

En todo proceso centrado en las personas, los factores humanos tienen un gran impacto en el esfuerzo de ejecución del mismo y en los resultados que se obtienen. A pesar de ello, recién ahora la ingeniería de software le está dando la importancia que esto se merece. Uno de los procesos humanos más importantes en el desarrollo de software es el trabajo en equipo. Un trabajo en equipo eficaz puede hacer la diferencia entre un buen y un mal resultado en un proyecto de desarrollo. Las instituciones de educación superior deben asumir su responsabilidad de enseñar sus alumnos no solamente temas técnicos, sino también las habilidades blandas, que les permitan llevar a cabo sus actividades profesionales como miembros de un equipo de trabajo.

Este trabajo de tesis ha definido dos hipótesis al respecto: (H1) hay un pequeño número de variables que sistemáticamente influencia el trabajo en equipo en proyectos de software ejecutados por equipos de desarrollo pequeños e inmaduros, y (H2) el uso de ThinkLets podría ser útil para ayudar a mitigar las situaciones negativas que afectan al trabajo en equipo.

En base a una extensa revisión bibliográfica y a la observación directa de varios equipos de desarrollo del curso CC51A: Ingeniería de Software, se identificaron preliminarmente tres variables que influyen de manera sistemática en el trabajo en equipo: comunicación, coordinación y motivación. Estas variables generan problemas típicos, tanto al interior del equipo de desarrollo, como entre éste y los clientes y usuarios. Para paliar estos problemas se definió un conjunto de ThinkLets. Estos ThinkLets son actividades o procesos que producen resultados predecibles, para hacer frente a problemas recurrentes de colaboración entre los miembros de un equipo de trabajo.

El uso de algunos de estos ThinkLets fue validado a través de la observación directa de siete equipos de desarrollo del curso CC61A: Proyecto de Software. A través de dichas observaciones, que involucraron dos semestres, se pudo constatar que las variables identificadas efectivamente fueron las

que generaron mayor cantidad de inconvenientes para el trabajo en equipo. Por otra parte el uso de los ThinkLets para paliar dichos problemas tuvo un impacto positivo. Si bien los resultados obtenidos aún son escasos para sacar conclusiones sólidas, estos están alineados con las hipótesis definidas.

Acknowledgement

I am truly indebted and thankful to my husband who believed in me and helped me to achieve this thesis. I would like to show all my gratitude to my daughter and my mom who were with me in this challenge since the beginning.

I am sincerely and heartily grateful to my advisor, Sergio Ochoa, for the support and guidance he showed me throughout my dissertation writing. I am sure it would have not been possible without his help.

This thesis work has been partially supported by the Fondef Project N°: D09I1171.

Table of Contents

Table Index

Figure Index

1 Introduction

Software engineering is a highly relevant area in academia and also within industry. Typically, there is an important demand for well-trained software engineers, since the code in consumer products is doubling approximately every two years (Bagert, et al. 1999) (Simmons 2006). Professionals who have completed their studies in Computer Science or Informatics have many job opportunities, as there is an increasingly high demand for these professionals.

The professional skills required for today's software industry are on the increase. New trends in software development such as offshore and distributed software development require professionals with new skills (Hawthorne and Dewayne 2005). One of these skills is "teamwork".

The importance of the word "teamwork" began in sports along with the creation of various types of collective sports. In the twentieth century "teamwork" became the keyword for all companies in general; and it is defined by Wikipedia "as the capability to comprehend and recognize the diverse strengths and abilities in a group setting and then applying them to one final solution" (Wikipedia 2011).

In a 1992 article, Peter Denning (Denning 1992) reported a study that showed recently graduated engineers did not know how to communicate with others and had insufficient experience and preparation to work as part of a team. Denning states that the responsibility to provide such skill belongs to the university where these people were educated. Trying to solve this problem, ABET (ABET 2010) emphasized the teaching of communication and teamwork skills as a requirement for accrediting engineering programs.

In computer science, more particularly in software engineering, teamwork can mean the difference in the success or failure of a project in several development scenarios. This is the major reason why companies around the world consider teamwork a norm to which employees must attend. It is assumed that universities must educate software engineers not only in the scientific and technical aspects of the discipline, but also in the social capabilities that allow them to be effective in teamwork (Bagert, et al. 1999). Therefore, educational institutions must prepare their undergraduate students to work in a more interconnected manner and in social software development scenarios (Bareisa, et al. 2007). The research community has recognized the complexity of developing specific skills in the students (Simmons 2006). Clearly, teamwork is considered a soft skill, hard to teach and lean in practice. Giraldo and Jazayeri (Giraldo, et al. 2010) (Jazayeri 2004) stated that the majority of the projects done in

software engineering courses are short projects and normally they do not have a real customer, it is usually the course professor that plays the role of the client.

1.1. Problem to Address

Concerning the teamwork skills of recently graduated software engineers, there are several reports indicating the gap between software engineering education and the industry needs (Denning 1992) (Hilburn and Bagert 1999) (Gorla and Lam 2004) (Wellington, Briggs and Girard 2005). Clearly this is a major challenge that seems difficult to address. This problem can be confronted from two perspectives: (1) human behaviour and (2) practices involved in software processes. The human behaviour approach must follow a more psychological perspective through which the students must be trained to have attitudes that contribute to teamwork and avoid those that jeopardize teamwork. Typically this perspective will involve changes in the engineering curricula; any changes in the curricula are extremely difficult and take time; therefore only through an institutional decision it can be done.

The second perspective seems to be more feasible to address in a computer science master thesis. Such a perspective should change the practices embedded in a software process, which leads to a closer definition of teamwork. Of course, designing software processes that promote teamwork require one to know which are the most influential variables and also how these variables affect teamwork in a software project. The resulting solution will depend on each project context.

Identifying these variables, the relationships among them and also the project context, will allow us to design small process solutions that can be used by software engineering students during their education. Thus these students could learn through their own experience, some of the teamwork skills required by the industry.

Provided that every project context represents a potential study scenario, this thesis studies only projects that develop Web information systems involving teams with 5 to 7 novice developers. The projects lasted between 3 and 5 months. The reasons to select such context for the study are several:

1. This type of projects represent an important percentage of the developments conducted by the Chilean industry (A.G., GECHS Softwares y Servicios Chile 2010). This is the scenario in which this work could have some impact.
2. Recently graduated software engineers are initially involved in short and low risks projects.

3. The author of this proposal had access to a couple of software engineering undergraduate courses in which these kind of projects are developed. These courses were used as study scenarios for this thesis.

1.2. Work Hypotheses

Trying to help solve this problem in this specific context, this thesis work defines the following hypotheses:

H1: There is a short list of variables that systematically influence the teamwork in software projects conducted by small and novice development teams (5-7 developers).

H2: Thinklets can be used to help mitigate the recurrent situations negatively affecting teamwork.

In this thesis we define a thinkLet as an activity or process that produces predictable results to deal with recurring collaboration problems in software development teams. A thinkLet can be seen as a kind of process pattern to address collaboration problems. This definition is based on the one stated by Noor et al (Noor, Grunbacher and Briggs 2007): "a thinkLet is a named, scripted, and well-tested activity that produces a known pattern of collaboration among people working together towards a goal". ThinkLets can also be seen as building block for collaborative processes (Briggs, et al. 2001). In Chapter 5, is possible to see the Thinklets created.

1.3. Objectives

This thesis proposal sets out to improve teamwork among Computer Science undergraduate students when they participate in a software development team. In order to do that, this work starts by formalizing the context in which the research will be conducted. We will identify the variables that can favourably or negatively impact teamwork effort. Based on the results, a set of thinkLets will be proposed to generate positive impacts on the teamwork and mitigate the negative ones. As a consequence of using these thinkLets in the software project, the students should improve or enhance their teamworking skills. Summarizing, the specific goals derived from the general one are the following:

1. Identify the variables favourably or negatively affecting teamwork.

2. Define a set of thinkLets that can be used by teammates to promote/enhance teamwork and also mitigate possible negative effects produced by particular variables. The research in this area will produce a set of thinkLets that help increase and/or enhance teamwork inside a group.

3. Propose a set of guidelines indicating how to address particular communication and coordination problems using the proposed thinkLets.

1.4. Methodology

This thesis will involve the qualitative research approach (ethnographical) and the interpretivism as paradigm of the research. First, to do so, a literature review was carried out (historical study) on the subject along with an analysis of the historical course information.

Course CC61A (Software Project) is a course that allows formative evaluation of student's performance regarding their technical skills and in their teamwork capabilities. During the course duration (one semester), the students have to work in their client's facilities (real clients) at least 20 hours per week, and have to attend a 1.5-hour meeting with their software engineering instructors once a week. The students have to formally present their project three times and they are graded by the client, software engineering instructors and by their peers (their own team only). The course CC51A (Software Engineering) also allows formative evaluation of the students, but the students do not have to attend formal meetings or work in the client's facilities; they can work from home or the university in their own time.

At the same time a Focus Group (ethnographical study) was conducted with software engineering instructors and also with the students from CC61A – Software Project course. The main goal was to identify variables that affect teamwork.

In this research, courses CC51A and CC61A were used as a laboratory. According to Wohlin (Wohlin, et al. 2000) software engineering is mainly a social process; therefore empirical studies, even with computer science students, is a valid research methodology in this context. Based on an extensive literature review and direct observation of several development teams of the Course CC51A: Software Engineering, the variables that systematically influence teamwork were preliminarily identified. Then I observed the Software Project Course (CC61A) for two semesters (Spring 2010 and Autumn 2011) and I performed a total of 45 hours of team meetings and a 50-hours focus group with team members alone. During the semester Spring 2010, the data gathered was related to the literature and patterns were identified and a design of ThinkLets was done. The designs of ThinkLets were based on the literature

proposals and experiences from instructors of courses involving academic software development projects. During the semester Autumn 2011, the design of the thinkLets and their adherence to software engineering education were evaluated along with their outcomes.

The author is aware that a qualitative empirical study is not concerned only with collecting verifiable data, and sometimes not repeatable in other contexts (other cultures as an example). However it helps understand a social environment that is present in the teaching-learning process.

1.5. Structure of the Thesis Document

This thesis proposes a set of practices to be used to improve teamwork on software projects developed in Academia. Most of the practices proposed come from the Computer Science area and from different areas of knowledge that have great influence on the subject of this thesis: Psychology, and Management Theory.

In Chapter 2, I summarize the relevant work done in the subject of teamwork - variables which affect teamwork and thinkLets. Chapter 3 focuses on the preliminary identification of the variables affecting teamwork. Chapter 4, describes the practices that I found in the literature that can be used to enhance teamwork, and the literature review that support it.

Chapter 5 discusses real problems that a team doing software projects can face and what can be done to mitigate them. Chapter 6 discusses the Experimental Results found in our observations of CC61A during two semesters and the grades of the last nine semesters. Chapter 7 talks about the expected contributions. The final Chapter 8, presents the conclusions of this thesis and speaks about future work.

2 Related Work

The IEEE (Institute of Electrical and Electronic Engineers) defines Software Engineering as "the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software" (Tripp 1994). For the industry it is crucial that users work within a team-based framework while developing a software product.

2.1 Team and Teamwork

A team is a group of people working collaboratively to reach a common goal. Teams are more than collections of individuals and teamwork is more than the aggregate of their individual's behaviours (Paris, Salas and Cannon-Bowes 2000), e.g. a team of experts is not necessarily an expert team. This means that it is not possible to label a group of individuals as a team, and to expect them to behave like one (Bass 1980).

Team members use processes to coordinate their activities and also to combine their skills appropriately in order to be more effective (Koslowski and Ilgen 2006). As Lingard states (Lingard 2010) Teamwork is a well-researched topic and skills are supposedly taught and refined through university courses. Several software engineering researchers have emphasized the importance of teamwork in the software industry (Aranda, Easterbrook and Wilson 2007) (Demirors, Sarmasik and Demirors 1997) (Wellington, Briggs and Girard 2005). However, most of the scientific work in this area comes from psychology researchers (Gorla and Lam 2004) (McDonough III and Cedrone 2000) (Safizadeh 1991) (Hernandez 2010) (Zika-Viktorsson and Ingelgard 2006) (Gladstein 1984) (Shenhar and Dvir 1996).

Research on the topic in the 1970s focused on orientation, resource distribution, timing, response coordination, motivation and morale. In the 1980s it was on collective self-efficacy, coordination activities, tasks and motivational reinforcement. In the 1990s it was mutual performance, monitoring, collective orientation, flexibility, potency, cohesion. Today the main topics talks about attitudes, collective efficacy, shared vision, team cohesion, mutual trust, collective orientation and importance of teamwork and for most all of these topics, researched software engineering helped create software to measure, control and help teamwork development (Paris, Salas and Cannon-Bowes 2000).

According to Morgenson (Morgenson, Aiman-Smith and Campion 1997) and Meister (Meister 1985) the variables that influence team performance are presented in Table 1.

Table 1. Variables affecting teamwork

Variable	Description	Examples	Possible Interventions
Contextual	Pertaining to the environment, in which the team activity is embedded, differences between members of the team (e.g. culture or education), or rules of the team (e.g. rewards). They are often amenable to change by the organization and they typically influence team performance by creating a work environment conducive to teamwork effectiveness.	Culture, working climate, educational level, or reward systems.	1. Team selection 2. Training 3. Team Design
Structural	External sources affecting the team, such as the physical environment where the team will work, organizational arrangements (hours and days of work), technologies to be used for developing and supporting the project and the team. They are not so often amenable to change and could represent potential barriers to effective teamwork performance.	Physical environment, organizational arrangements, technological systems	1. Task design 2. Training
Process	Inherent to the team itself and the way in which it behaves. The general rules of the game: communication (who will talk with the client, what kind of documentation will be used, which level of detail will be used in communication, what are the norms of the project/team, mandatory meetings of project status.	Boundary management, task cohesion, performance norms, communication, team interactions	1. Team selection 2. Task design 3. Training
Contingency	Internal and external situations affecting the team; for example the lack of an important resource (e.g. knowledge, man power) for the team would cause poor teamwork effectiveness regardless of the teams standing on other effectiveness factors (e.g. cohesion, potency, efficacy) Trying to avoid future problems, the team has to have a clear mission and operation rules in terms of team members and technology.	Resources availability, procedural requirements, rules of operation	1. Task design 2. Training

Hoegel and Gemuenden (Hoegl and Gemuenden 2001) stated that some of the variables that directly affect teamwork in software development are: *communication* (it should be frequent, informal, direct and open), *coordination* (individual efforts should be structured and synchronized within the team), *balance of members' contributions* (all team members should share their expertise as much as possible), *mutual support* (team members should help and support each other, while they perform the assigned tasks), *effort* (team members should exert all efforts to the teams tasks) and *team cohesion* (team members should be motivated to maintain the team and the team spirit).

In short there are many variables that can influence teamwork (without a context analysis) and sometimes these variables are more like a category of influences than variables in themselves. Some are:

communication, coordination, balance of team member's contributions, mutual support, effort, team cohesion, contextual, structural, process and contingency.

2.2 Teamwork and Computer Science Education

Educational institutions delivering computer science programs must accept the responsibility of preparing their graduate students not only in technical issues, but also in soft skills that allow them to work properly in their professional career (Bareisa, et al. 2007). In order to address this challenge, practice-based approaches for software engineering education have been recognized as a best practice (Carver, et al. 2003).

Students of computer science programs have to address not only the technical aspects of software engineering, but also the social and behavioural aspects of this discipline (Giraldo, et al. 2010). It represents a great challenge for students enrolled in software engineering courses. However, teaching software engineering is not a trivial problem to address in universities scenarios, since a good software engineer must combine formal knowledge, good judgment and taste, experience and ability to interact with and understand the needs of clients. Moreover, these skills must be transmitted to the students through two or three courses, not enough, which are typically focused on the phases of the development process (requirements, analysis, specification, design, implementation and testing) (Jazayeri 2004).

In order to provide some level of experience, other than what is taught in textbooks, many courses include a project where students have to develop a software application, which aims to show the student the "real world" of software development. Adhering to such an idea Gehrke et al. (Gehrke, et al. 2002) designed a software engineer course focused on the creation of an "industrial strength" student skill. Therefore the instructor worked in one semester long courses simulating the tough reality of the industry. The course was taught over four years at the University of Paderborn and one year in the University of Braunschweig, both in Germany. The projects developed in the course involved making extensions to an existing product and not developing something from the scratch.

These projects take into consideration hard deadlines, well-defined deliverables, requirement changes, and also the fact that some team members come into and, leave the team during the project execution. At the end of the semesters they achieved their goal of showing students the reality of the industry, while expressing the importance of teamwork. They came to two major conclusions for this project: (1) the students wanted and required weekly meetings with the client (teacher) and themselves more often as the project was being developed; and (2) the more they pushed the deadlines the more the teams stuck together (improving team cohesion).

Another experience was reported by Tvedt et al. (Tvedt, Tesoriero and Gary 2001) about the creation of a software factory. They made major changes in the curricula and software engineering courses were taught in all semesters (8 semesters courses). In these new software engineering courses each student in each course had a specific role to play within a project. All the selected projects involved developing software, which dealt with problems from real companies. The objectives were to meet the needs of industry, attract and retain quality students, conduct empirical software engineering research, encourage teamwork and multidisciplinary collaboration.

In the Computer Science Department at the Universidad de Chile, there are two advanced courses that should have promoted teamwork skills to undergraduate students. These courses are CC51A: Software Engineering and CC61A: Software Project. In such courses undergraduate students are grouped to form a development team. Each team is in charge of developing a software product that solves a problem for a real client. Development teams used in each course are different in terms of structure, responsibilities and the methodology they have to follow to obtain a final product. The team size, the macro-activities to perform and the final goal are similar. However In Software Project (CC61A) the students have to work a regular amount of time with the client. In Software Engineering (CC51A) the students work on their own time. In the beginning of this work Software Engineering (CC51A) was considered, though after some data analysis it became clear that obliging students to follow a schedule created a big difference between the two courses.

These courses do not include particular mechanisms to promote teamwork, or strategies to guide uncoordinated teams toward coordination of team members' activities. Teaching students how to keep and promote teamwork in their teams is an important skill for their professional life. For that reason this thesis hypothesizes that a set of thinkLets can be used to address this challenge. A thinkLet is a codified and encapsulated facilitation technique (i.e. a process or activity) that creates a predictable pattern of collaboration (Kolfschoten, et al. 2006). These thinkLets produce a predictable pattern of interactions among people working together toward a goal. And can be used as snap-together building blocks for team process designs.

2.3 ThinkLets

According to Briggs (Briggs, et al. 2001) in one of the first definition of a thinkLet, he stated that, is the smallest unit of intellectual capital required to create one repeatable, predictable pattern of thinking among people working toward a goal. A thinkLet is a named, packaged, thinking activity that creates predictable and repeatable pattern of collaboration among people working towards a goal. A

thinkLet has three components: Tool – the specific version of the specific hardware and software technology used to create a pattern of thinking. Configuration – The specifics of how the hardware and software were configured to create a pattern of interaction. Script – the sequence of events and instructions given to the group to create the pattern of thinking.

ThinkLets thoughts of this way have huge limitations, as they tend to be technologically dependent. Another problem is with the ThinkLet definition itself. Any change on the script, tool or configuration generate a completely new thinklet. So Kolfschoten et al (Kolfschoten, et al. 2006) worked on a re-conceptualization of thinkLets. This new thinkLet conceptualization describes the requirements to create a certain pattern of collaboration independent of technology and its configuration. They re-defined thinkLets in terms of its principle: tools, configuration and script, and in terms of transitions and modifiers. It gave thinkLets the capacity to grow and change without the concern of technology and configuration.

The definition of thinkLet used in this thesis is an activity or process that produces predictable results to deal with recurring collaboration problems in software development teams. A thinkLet can be seen as a kind of process pattern to address collaboration problems or challenges.

We envision that a set of thinkLets, containing processes which can be used in particular situations and work scenarios, can help promote and/or enhance teamwork. The thinkLets should help neutralize the negative effect produced by some variables affecting communication, coordination and motivation within the team. Based on the use of specific solutions to deal with particular communication and coordination problems in practice, the hope is that students can improve their teamworking skills. These solutions are the recommended practices in the following section.

3 Preliminary Identification of Influencing Variables

This section describes the initial steps that were followed to identify the variables that influence teamwork. Such processes involve three steps: a literature review, the observation of the CC51A course, and a preliminary validation. Next, these steps are briefly explained.

3.1 Literature Review

During the development of this thesis a lot of bibliographic material on the subject was found. Much of it spoke of teamwork in different contexts, where teamwork is essential in life or death situations (e.g. in hospitals and fire response processes). Other texts dealt with management styles within companies. Compared with other fields, those that spoke of computer science teamwork focused on distributed teams.

Koslowski and Bell (Kozlowski and Bell 2003) identified coordination, cooperation and communication as the key team behavioural processes. Much of the research is centered on effort coordination of as the critical behavioural process in teamwork (Salas, Stagl and Burke 2004). Communication is normally used as a means to prompt and maintain coordination in teams.

Others researchers studied the impact of motivation on teamwork. For example, Ryan and Deci (Ryan and Deci 2000) say that "to be motivated means to be moved to do something; a person who feels no impetus or inspiration to act is thus characterized as unmotivated, whereas someone who is energized or activated towards an end is considered motivated". Specifically in the computer science field we did not find research works that measure the impact of team members´ motivation. However, Humphrey (W. S. Humphrey 1996) reported a lot of his practical experience in the field: "in technology, there are many failures for every success, and it is easy to become discouraged, unmotivated…. people need to be charged up and reminded that the goal is important and achievable."

Thus we found that many variables could influence teamwork, but there are three that stand out in many of the papers. Sometimes the authors used different names to describe the same problem from another perspective. In the end the assigned meaning to these variables is the same. These variables are: *communication*, *coordination* and *motivation*.

Coincidently the most prominent variables of the literature were the variables that we have found empirically to be the most relevant ones during the teamwork observations in the course CC51A. The

course CC51A has been taught for at least 5 years, and since 2005 the course has a self-evaluation, where students grade themselves and the other team members. In this self-evaluation, they are asked to evaluate each of the team member's performance and to write down the strengths and weaknesses of each other. And during the last presentation the teams are always asked what were the worse challenges that they had. Looking over this past data, we see that the major problems were: communication, coordination and motivation.

3.2 Observation of the CC51A Course

At the beginning of this thesis, we observed teamwork in two undergraduate courses (CC51A – Software Engineering and CC61A – Software Project). The student teams worked together to develop a software product in 12 weeks. In the first observations performed in CC51A we identified that the three influencing variables (i.e., communication, coordination and motivation) were present in most cases. Moreover, looking at the co-evaluation grades (i.e., the scores assigned by each team member to their teammates' job) of these courses during the last four to five years, it was also possible to see the problems of communication, coordination and motivation that these teams had.

Given that CC51A did not have a pre assigned time for the team to work in the project, the students have problems organizing their time. This and other particular features make these courses difficult to compare between them.

3.3 Preliminary Validation

Trying to determine the importance of these three variables influencing the teamwork, I talked with experienced people (e.g. instructors of the two courses mentioned here). We discussed the nature of the teamwork problems and how repeatable they can be. This discussion was also opened to some coaches of the CC61A – Software Project course. These people use their experience to determine if the identified variables were the most influencing ones for teamwork. After various discussions we all agreed in the relevance of communication, coordination and motivation as important variables that affect teamwork.

The next chapter presents the practices that could be eventually useful to address problems of communication, coordination and motivation in a software development team in the academia. Several of these practices have been already proposed by researchers of software engineering.

4 Recommended Practices

This section presents a list of practices that can be used to resolve some problems a team may face during a project. I will borrow a definition of practices used by Aranda (Aranda, A Theory of Shared Understanding for Software Organizations 2010), that says that they "are contained, repeatable, and transferable techniques used to improve some aspect of the performance of a software organization that is pertinent to the creation of its products. They are mechanisms to attack a known software development problem, or to gain some generally useful benefit."

The majority of the practices mentioned here are used in various contexts of software development methodologies: traditional development and the three major approaches of agile development (XP, SCRUM and Crystal Family). Some of the practices are normally used in different contexts, others than software engineering, such as business, psychology and management. I selected only the practices that were proved (all of them have references) to be effective solving teamwork problems. Few of them were created by me, grouping concepts and ideas from different areas, such as Peer Activities, Decision Making and Public Profile. It is important to remark that this thesis does not intend to bring together everything on the matter. In this sense, I have to agree with Aranda (Aranda 2010) "there is an overwhelming variety of academic disciplines that tackle these issues in different ways, and achieving mastery over any of them appears to hinder one's efforts for achieving mastery in one's domain."

The practices were classified in five categories: Thinking Practices, Collaborating Practices, Releasing Practices, Planning Practices and Developing Practices. In Thinking Practices are grouped all the practices in which team thinking and analysis are needed. In Collaborating Practices, we find the practices used to improve collaboration and communication among team members and between the team and the client. With Releasing Practices the goal is to avoid problems and conflicts within the team or with the client that can be related to the process of software engineering. The Planning Practices are practices that are used to avoid problems of miscommunication between the team and the client, and the Developing Practices are the ones used to solve or avoid technical problems and bugs. Figure 1 shows how many practices there are in which one of the practices categories created. The next

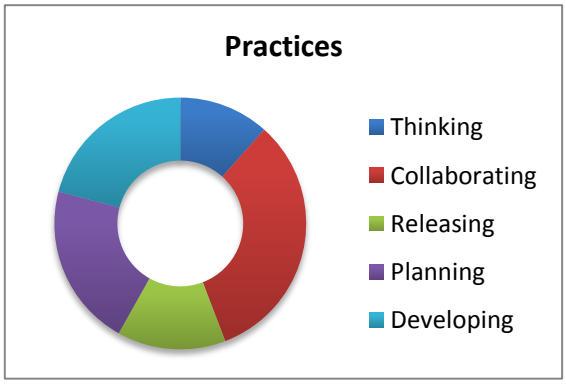

Figure 1 - Recommended Practices

section presents examples of practices belonging to these categories, considering the software development scenario.

The table below classifies the Practices listed in this chapter, and their relation with the three influencing variables: Communication, Coordination and Motivation. Here we see that most practices try to deal with coordination problems (42), and those addressing motivational issues are few (13).

Table 2. Practices Classification According to the Influencing Variables

	Practices	Communication	Coordination	Motivation
Thinking	Peer Activities	X	X	
	Energized Work	X	X	X
	Informative Workspace	X	X	
	Root Cause Analysis	X	X	
	Retrospectives	X	X	X
	Trust	X	X	X
Collaborating	Sit Together	X	X	X
	Real Customer Involvement	X	X	X
	Ubiquitous Language	X	X	
	Stand Up Meetings	X	X	X
	Coding Standards	X	X	
	Iteration Demo		X	X
	Reporting	X		
	Team-Building Workshop	X	X	X
	Peer Review	X	X	
	Coaching	X	X	X
	Kanban	X	X	
	Decision Making		X	X
	Public Profile	X	X	
	Feedback		X	
Releasing	Done Done	X		
	No Bugs		X	
	Version Control		X	
	Ten-Minute Build		X	
	Continuous Integration		X	

	Practices	Communication	Coordination	Motivation
	Collective Code Ownership	X	X	
Planning	Vision	X	X	X
	Stories	X	X	X
	Estimating	X	X	
	Planning Game	X	X	X
	Release Planning		X	
	Iteration Planning	X	X	
	Slack	X	X	
	Risk Management		X	
	Meeting Minutes	X	X	
Developing	Spike Solution	X	X	
	Test Driven Development		X	
	Refactoring		X	
	Simple Design		X	
	Incremental Design and Architecture		X	
	Performance Optimization		X	
	Customer Testing		X	
	Customer Reviews		X	
	Exploratory Testing		X	

4.1 Thinking Practices

Figure 2 - Thinking Practices

Peer Activities. This is everything that can be done in pairs during a software project and that has already be proved in the literature:

Pair Programming. It involves two people working together on one keyboard. One person writes the code – the driver, and the other one (the navigator) has to think. Sometimes the navigator's work is to think of what the driver is writing. Other times, he/she has to think about what to do next and how their work will fit into the general design of the software. It is normally used to increase the power of thinking and problem resolution. It also reinforces the use of development methodologies and decreases the number of mistakes in the code. Chong and Hurlbutt (Chong and Hurlbutt 2007) pointed out important insights into collaboration with the use of pair programming, and concluded that it can improve the mental model of team members and consequently produce a better product. There is a variation of this practice called Side-by-Side Programming, when two people sit close enough together to see each others screen easily, but work on their own assignments (Cockburn 2004).

Pair Designing. This involves two designers who work on the same design document, on the same machine and at the same time: the first designer is the designated driver and writes the document, while the designated observer reviews it. The two roles can be switched which usually happens when the driver does not know how to proceed, and when the observer has already elaborated a candidate solution for the problem. The observer can also accomplish other activities apart from reviewing, which might help to reach the goal of the current task. Pair design brings confidence to the team in that the design will have fewer mistakes and will be more suitable to the clients´ needs (Canfora, et al. 2007). According to Al-Kilidar et al (Al-Kilidar, et al. 2005) and Canfora et al (Canfora, et al. 2007) pair design empirically has higher quality than solo design, regarding: functionality, usability, portability and maintainability.

Pair Analysis. According to Williams et al (Williams, et al. 2000) it is important for the pair to collectively agree on the development direction and strategy outlined during these stages. Additionally, it is doubtlessly true that "two brains are better than one" when performing analysis and design. Together, pairs have been found to consider many more possible solutions to a problem and more quickly converge on which is best to implement. Their constant feedback,

debate, and idea exchange significantly decreases the probability of proceeding with a bad design, improving the team efficacy.

Peer Code Review. Here one-team member revises the code written by another team member. The idea of this task is to help team members to read code and to learn how to extract useful alternate methods to solve a problem, improving the capacity and knowledge of the team. According to Trytten (Trytten 2005): "writing code and reading code are very different activities, and both have value".

Energized Work*.* Here, team members need to maintain their personal mental health. They go home on time everyday so they can spend time with family and friends and take part in activities that take their mind off work. It also allows one to stay home when one is sick. It is used to guarantee that each member of the team can accomplish his/her best and be more productive, to maintain high levels of motivation and assure that the progress of the work being done is constant. Shore and Warden (Shore and Warden 2008) state, "When your team is energized, there is a sense of excitement and camaraderie. As a group, you pay attention to detail and look for opportunities to improve your work habits."

Informative Workspace. The workspace provides simple and direct information about the project in the environment (workspace) that everyone share, allowing everyone to know what each other is doing, and where the team is going. Big white boards and visible hand-made graphs are typically used for this workspace. The purpose is to keep all those interested in the project updated, without having to interrupt someone's job to ask. Sharp, Robbinson and Petre (Sharp, Robinson and Petre 2009) concluded that the Wall of the Informative Workspace is an artefact of significance and meaning to developers. The Wall shape mediates and manages the life of developers. It is the symbolic means by which work is managed, by which code is created, judged and accepted. The Wall comes with a litany and a liturgy that those present accept, understand and respond to."

Root Cause Analysis*.* It is a routine of self-evaluation that can be done by a group of team members or the team as a whole, where they ask: what, where and why at least five times; sometimes more, depending on how deep the problem is (Shore and Warden 2008). It is used to solve problems encountered during development without blaming anyone. As the code belongs to everyone, no one is to blame. This type of self-evaluation helps team members maintain focus on the project and avoid relationship conflicts among team members.

Retrospectives*.* This practice performs an analysis made by all team members of the work done; what was good and what was bad. Normally it finishes with a short list of attention points that can be used as new stories for the next iteration. It is used to help keep the team from making the same

mistakes again; to improve the development process, to make the team more cohesive and to solve problems within the team. Retrospectives are an opportunity for the team to learn from what worked and what did not. According to Rising and Derby (Rising and Derby 2003) “retrospectives provide a wonderful opportunity for capturing knowledge as patterns to improve team knowledge and team cohesion.” It is also know as Reflection Workshop (Cockburn 2004).

4.2 Collaborating Practices

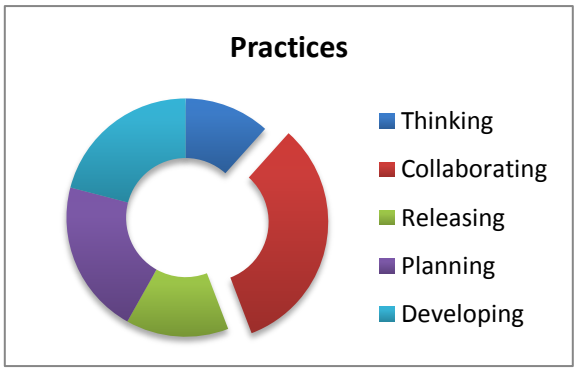

Figure 3 - Collaborating Practices

Trust. It helps a team to work efficiently, with confidence that each member is doing his/her best. There are many strategies that can be applied to a team to accomplish that goal, according to Shore and Warden (Shore and Warden 2008), there are two major types of strategies to use; team strategies and the organizational. This practice is normally used to improve team cohesion and to improve the relationship between the client and the team. Normally the performance of the team increases, as does the final product.

The team strategies are (Shore and Warden 2008):

Customer-Programmer Empathy – A recurrent problem of customers feeling that programmers do not care enough about their needs and deadlines and programmers often feel forced into commitments they cannot meet. Sitting together is the most efficient way to build empathy according to Shore and Warden.

Programmer-Tester Empathy – When this kind of problem occurs programmers tend not to show respect for the tester abilities, resulting in testers responding to programmers by becoming excessively critical of programmers code. To avoid this, the authors suggest that the team show empathy in all areas and occasions and to remind the team that a mistake is not one person’s problem; it is the team’s problem.

Eat Together – There is anecdotal knowledge among project managers says that team members that spend some time outside work, can break down barriers between them, and according to Shore ”Something about having meals breaks down barriers and fosters team cohesiveness”.

Team Continuity. – After a project ends, the team typically breaks up and all the wonderful trust and cohesiveness that the team has formed is lost. The idea is to keep productive teams together. It is a good idea to take advantage of this effective team as a training ground for other teams.

The organizational strategies are (Shore and Warden 2008):

Show Some Hustle – The team has to show that they are doing productive work, as the author said: "a fair day's work for a fair day's pay".

Deliver on Commitments – Normally stakeholders have worked with software teams before, they probably have plenty of war wounds from slipped schedules, unfixed defects and wasted money. The teams have to create a plan that can be achieved, and demonstrate that they can deliver on commitments.

Manage Problems – "When the team identifies a problem, let the stakeholders know about it. They will appreciate your professionalism even if they did not like the problem."

Respect Customer Goals – No matter how impossible or different a customer goal or requirement is, the team always has to manage this important issue. The team must have the customer at their side to show the customer alternatives, estimates and ask for priorities.

Promote the Team – Let everyone know what the team is doing. Invite everyone to the Iteration Demo. Being open and clear about what you are doing also helps people appreciate the team more.

Be Honest – Concentrate on looking good only to customers and stakeholders is a common mistake. Do not do it, be honest; only count stories that are completely finished and tested. Does not extend iterations for a few days in order to finish something. The cost of not being honest is much bigger than the gain a team can achieve with these kinds of "white lies".

***Sit Together*.** It consists of putting the team members and the client together. It requires wide and open spaces, with no barriers of access between the team members. It is used to increase the effectiveness of communication between the members of the team and also the client. The goal is to let team members eavesdrop and get involved in other team members' conversations so that they can contribute with their ideas and opinions, participating directly in the conversation. Teasley et al (Teasley, et al. 2002)explored the Sit Together, or radical co-location as he/she calls it in software development of automobile companies – we believe that co-location of the project team and customers in a war room

can be effective in reducing such communication breakdowns and facilitating speedy resolutions of conflicts. By improving communication, productivity and timeliness of the projects will also improve."

Real Customer Involvement. This task is involves the client. The client must stay together with the development team, and he/she should be a real team member, with full commitment. The purpose of this practice is to guarantee that the team will have a better understanding of the project's goal, and the clients' problem. The knowledge needed for the project is transmitted in a direct and clear way. The team can quickly access the client for any questions or problem; if not the software development could have to wait for the client response. Korkala et al (Korkala, Abrahamsson and Kyllonen 2006) conducted empirical research on the communication in software development and they concluded that the rate of defects fixing in the cases where there were no Real Customer Involvement practice being used was two times greater than the worse case that utilized Real Customer Involvement.

Ubiquitous Language. This refers to the use of a common language. The language of the development team and the client should be unique, clear and concise. Despite the fact that developers tend to use their own language, the language used should be the one used by the expert in the problem, normally the client. It is important to decrease the communication failures and to let the client get more comfortable with the communication between team members. Hayes and Andrews (Hayes and Andrews 2006) stated that the practice of using Ubiquitous Language helps ensure everyone is working on the same concept.

Stand Up Meetings. Here, the team has a daily meeting, scheduled at the start of the project, in a set place and time. All team members should attend and the team members have to stand up in a circle. Members of the team have to talk about what they did yesterday, what they will do today and report if they have something keeping them from doing their work. It is a brief meeting where everyone should be concise and only talk about what really matters. It is very useful to make team members aware of each other's work, problems and challenges. According to Larman (Larman 2007) "It supports openness and allows resolution of dependencies and conflicts in real time to maximize throughput."

Coding Standard. Team members should follow development guidelines during the project. These guidelines should include: development practices, tools, files and archive layouts, build conventions, error handling, assertions, design conventions. It is used to increase maintainability and reading of the code written by others. Sfetsos et al (Sfetsos, Angelis and Stamelos 2006) found out that "the use of rules in writing code emphasizes communication through the code and ensures readability of the system."

Iteration Demo. The team presents to themselves, the client and anyone that is interested in what the team produced on each iteration. It is useful to mitigate errors of communication, because it brings confidence to the client and to team about what work is being done. Hibbs et al stated that (Hibbs, Jewett and Sullivan 2009) "Ending the iteration with a demo gives the development team a chance to show off what it has been doing over the course of the iteration. … It indicates to the customer that the team is dedicated to reaching milestones and delivering promises."

Reporting. This helps keep anyone who is interested in the project informed. The idea is to publish various reports in the common area of the team. Some of the reports suggested by Shore y Warden (Shore and Warden 2008) are: Vision (a general description of what the team is doing and why), Release and Iteration Plans, Burn Up Chart (how much work is done and how much work needs to be done). They are very useful in gaining and increasing the trust of the client and of any project stakeholder.

Team-building Workshop. It aims to improve the cohesion of the team. Though there are lots of techniques written and well researched on that topic, the general idea is to place the team outside their work place, motivating them to participate in games that show them the importance of knowing and trusting in each other. Kapp (Kapp 2009) conducted research on the improvement of team cohesion by a "team building intervention", a short workshop where students have to play a game to get to know each other. He compared the performance and grades from the team who had the - Team-Building Workshop with the ones that did not have the workshop. The improvement was conclusive. The games that can be used in this kind of event are numerous. For example: Twister, Faster Drawn, Amoeba, Obstacles, Group Story Telling Chunks and many more… A short explanation of the game Chunks:

> "Chunks" (Parker and Hoffman 2006) - Prepare for the exercise by printing out a sentence and then cutting it into eight to ten pieces. Divide your team into subgroups, as many as the number of pieces you have. Describe the exercise to the participants like this: The sentence describes an important team principle. The sentence is cut into pieces, and the challenge is to reassemble the sentence without knowing in advance how it should read. The chunk that contains a period is the last chunk. Any chunk that begins with a space is the beginning of a new word. A chunk that looks like the beginning of a word but does not have a space in front could be the first word in the sentence. Some of the sentences suggested by Parker and Hoffman are: *There is no "I" in team. Nothing of importance was ever done without a plan. If the going gets easy, you may be going downhill*. The authors suggest picking the sentence according to the team challenges at stake in the moment.

Peer Review. The origin of Peer Review was first written by Naur, in his research he discussed the value of "students mutual evaluation, to supplement the normal grading of project work." In his work he remarked that the process of Peer Review should be a motivational technique and not a punishing methodology. The idea of the practice of Peer Review is to let team members evaluate the performance of other members of the team according to some criteria. To avoid retaliation the feedback of the review should be done anonymously, in other words the team member being evaluated does not need to know who gave him which evaluation. This practice helps the team to improve and shows the team how they are behaving as a team so far. According to Patit and Wilemon (Patit and Wilemon 2005) "In fostering a software development culture, Peer Reviews help foster healthy intra-group dynamics.

Coaching. This practice intends to bring out the best of everyone in the team, trying to maximize each one's performance as well as that of the team. According to Hackman and Wageman (Hackman and Wageman, A Theory of Team Coaching 2005) proposed a model of team coaching, "Team coaching being a direct interaction with a team intended to help members make coordinated and task appropriate use of their collective resources in accomplishing team's work". In this work Hackman and Wageman empirically proved the anecdote that coaching really makes a difference in an academic environment.

Kanban. Has the goal of limiting the work that will be done by the team. The idea is to timebox each iteration. After clients and the team create the user stories, estimate and prioritize them, the team or the project manager has to agree to the length of the iteration, and the team according to the iteration will get the user stories that will be done. Normally a Computer Science Kanban is made of a white board with at least four parts. Each part shows the status of the stories being done: To Do List, Work in Progress, Test, Release and Done. According to the teams work the story cards will be moved doing the iterations. It is normal to state a limit of two or three story cards per status, depending on the project. The general idea of the Kanban is to keep everyone informed of what is going on in the team so no one has to stop working to ask questions about what each one is doing. According to Kniberg and Skarin (Kniberg and Skarin 2010) Kanban "is an approach for introducing change into an existing software development lifecycle or project management methodology.". They state that Kanban can be used no matter what "flavour of agile methodology or traditional methodology you are using, it will help to see through the process and have a clear vision of what is happening in our project.

Decision Making. This is about helping the team make a decision. There are four different strategies for a team to make a decision: Dialectical Inquiry, Devil's Advocacy, Consensus and Voting. Schweiger and Sandberg (Schweiger and Sandberg 1989) did extensive research on the first three strategies:

- Dialectical Inquiry – uses a structural debate among two sets of group members who represent diametrically opposed recommendations and assumptions, whereas
- Devils Advocacy uses a structured critique by one set of group members of recommendations and assumptions developed by a second set as bases for critical examination.
- Consensus encourages open discussion among group members but does not formally structure or encourage conflict
- Voting is a useful tool, for example, to rapidly have a compact summary of what a large group thinks about a particular issue or anonymous voting can reduce bias of dominant individuals. Voting was studied by Hietala et al (Hietala, Koivunen and Ropo 2004) "one way to structure decision-making".

According to Schweiger and Sandberg (Schweiger and Sandberg 1989), Dialectical Inquiry should lead to higher quality solutions than Devils Advocacy because it seeks to identify alternatives from the original set of diametric recommendations and assumptions, whereas Devils Advocacy focuses only on what is wrong with recommendations and assumptions, rather than on identifying suitable alternatives. Consensus could be hard to come to and only used when there is no dominant member. Moreover, voting should be used not only to end the Decision Making process but also to reveal the lack of consensus, and to enable the group to explore the issue at a deeper level.

Public Profile. It consists of knowing in advance, before the project really starts, who each team member is and also knowing their strengths, abilities, knowledge, likes, dislikes and more. It would be ideal to have an MBTI (Myers-Brigg Type Indicator) of every member of the team, so everybody could know how to deal with difficult situations between one another, avoiding conflict and enhancing team efficacy. Amato and Amato (Amato and Amato 2005) states that knowing each other can give a new team the ability to know what to expect from the others, so cohesion and confidence increases.

Feedback. This consists of the team constantly receiving feedback from the client and from each team member receiving opinions of their work. It is essential that all participants stay informed of changes in order to provide feedback regarding the results and implications of these changes and to be certain that each change is acceptable to all project stakeholders (Patit and Wilemon 2005). They also reported that continuous and rapid feedback from customers not only leads to earlier problem identification, but also improves software quality. According to Mathieu et al (Mathieu, et al. 2008) Feedback has a positive impact on motivation, interpersonal trust and ultimately performance.

4.3 Releasing Practices

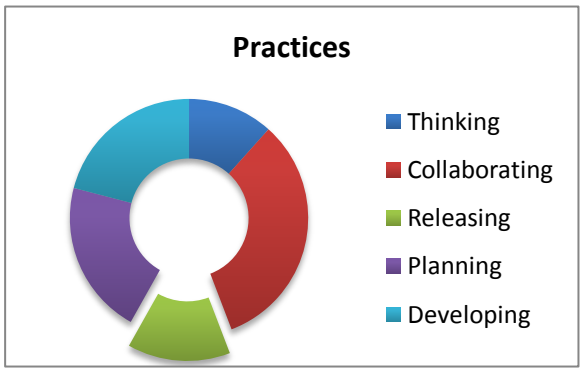

Figure 4 - Releasing Practices

Done Done. In this practice the team has to define at the beginning of the project what is considered "done work. It is very important because work done is not only code developed; it is work developed, tested, integrated, installed, revised (by the client) and accepted by the client. It is used to avoid cascade errors in integrations. It also guarantees that the team will have functional code for the demos at the end of each iteration, and it can result in the avoiding of misunderstandings with the client. An example: the client asks about the status of some functionality. A developer can say that it is done; and the client could ask to test it, but it is not integrated and not tested, and the client will have it to start testing days later, leading misunderstanding between the team and the client. According to Warden and Shore "when your stories are Done Done, you avoid unexpected batches of work and spread wrap-up and polish work throughout the iteration."

No Bugs. This practice is about writing code without errors. In order to accomplish a goal such as this, almost all the previously mentioned practices here are needed. It is used to increase the quality of the developed software and consequently raise the trust of the client in the team. Shore and Warden (Shore and Warden 2008) points out that "The agile approach is to generate fewer defects. This is not a matter of finding defects earlier; it is a question of not generating them at all."

Version Control. Concern it with project artefacts that should be all in one place - normally this decision is authority. By artefacts I mean every, and any file, archive or document that was used in the project and all its versions. It is used to maintain a security copy of everything that was used in the project. In addition the newer versions control systems allow team members to concurrently develop code. Abrahamsson et al (Abrahamsson, et al. 2002) points out that the role of version control is that it must be orchestrated and run continuously, day and night, and the developers themselves have highly varying skill levels and backgrounds.

Ten-Minute Build. This is about automating the compilation, construction and tests of the developed software. It is very useful to make the release phase easy and fast, and it can be done at any given time. The automated build let the team spend their time doing what really matters, not having to update servers, tests or any other routine that does not bring value to the project. According to Shore and Warden (Shore and Warden 2008) "When your build is fast and well-automated, you build and test

the whole system more frequently. You catch bugs earlier and, as a result, spend less time debugging. You integrate your software frequently without relying on complex background build systems, which reduces integration problems."

Continuous Integration. It consists of integrating all the code done in the last couple of hours and maintaining up-to date test, infrastructure and code. This allows you to avoid problems with integration, therefore not leaving you unable to deliver to the client. When the team does, the deliveries to the client are painless. This provides immediate feedback on newly created code, and also "reduces time that people spend on searching for bugs and allows detection of compatibility problems early" says Huo et al (Huo, et al. 2004).

Collective Code Ownership. It is a principle where each member is responsible for doing and maintaining a code of high quality, no matter where it is. It is used to avoid breakdowns in the team when a team member is absent for whatever reason. It also helps to increase maintainability and knowledge spread. Nordberg (Nordberg III 2003) points out "Collective Code Ownership is a lofty goal embodying altruism, positive team dynamics, good communication, and individual accountability." He also states "collective ownership is infeasible without several other XP practices related to source code quality during system implementation."

4.4 Planning Practices

Figure 5 - Planning Practices

Vision. It is the disclosure of the project, its goals, reasons, projected impact of the project and the criteria to measure the project success. It is used to maintain the focus of the project, making prioritization an easy task in planning meetings. Larman (Larman 2007) puts the importance of this practice "Establishing and reiterating a common vision is frequent advice from agile leaders. It may be seen as absurd to highlight such an obvious idea, but in over 10 years of post-project reviews with hundreds of project members, Standish Group in 2002 did not find even two people who stated the same purpose or vision for their project."

Stories. This is about the creation of one or two lines of description of what the development team has to produce. It should be written in the "language" of the users and not in any development or

complex language. Normally they are written on index cards or post-its. The idea is to make clients and developers understand each other and the tasks at hand. Each story should be as short as possible and as independent from each other as possible. It is used to help clients and developers understand one another regarding tasks to be done during the project. Mike Cohn (Cohn 2004) wrote a book that only talks about user stories. He points out that they are verbal communications of the client wishes, are comprehensible, are the right size for planning, work well for iterative development, encourage deferring detail, support opportunistic development, encourage participatory design and build tacit knowledge.

Estimating. It is the work of estimating the time that the developers will need to code each user story. The estimations are normally done in work days or work hours. This is an iterative process where each developer must state his/her own estimate. If there is not a consensus on the estimation made; the developers have to talk to each other, trying to explain the estimation they did. They have to do this until they come to consensus in each user story. It is helpful to everyone, since the velocity and predictability of the team will be based on that estimate. There are different techniques that can be used to do the estimations, but in all of them it is important that the team makes the estimation. Everyone on the team will be fully committed to what they estimated. Goodpasture (Goodpasture 2009) wrote "next to requirements, estimates are probably the most influential factor on the predictability of the project outcomes. In the agile methodologies, estimating is a team activity. The team both comes up with the estimate and lives with the estimate."

Planning Game. This consists of a meeting where the client has to explain what problem he wants solved through the stories that he wants. During the meeting any member of the team can create a story. Each story need to be explained and well understood by everyone attending the meeting. The client has to prioritize all the stories. This spreads the knowledge of what the client wants and what the team has to do. Normally the first step is a brainstorming where the client shows a picture of the whole idea, and then the team begins to cut this whole idea up into short user stories. Sfetos et al. (Sfetsos, Angelis and Stamelos 2006) found out empirically that, the Planning Game was considered a good process oriented practice with technical and social impact on programmers and managers. It is also known as Blitz Planning.

Release Planning. At this point the team has to plan which stories created at the Planning Game will be done in each iteration. The client, the product owner and the whole team must attend this meeting. Normally the team does a release planning for the whole project, but as time passes the stories can also change. The product owner can reprioritize the release plan at any moment. At this phase of the

project the stories do not need to be well defined, but they should be listed and estimated, though not necessarily with a lot of details and tasks as needed in the Iteration Planning. This process makes the idea of the final product clear in the mind of all the team at the time they will need to deliver the final product. According to Ruhe (Ruhe and Saliu 2005), release planning is an important and integral part of any type of incremental product development.

Iteration Planning. It is a meeting planning of what the team will do in the next cycle or iteration. Normally in the beginning of the project the time for the iteration is defined, but there is no right amount of time for iteration. Each team has to adjust themselves to their own pace. In this meeting, at the beginning a retrospective of the last iteration (the good and the bad things) is carried out. It is a meeting where the developers choose which stories they will do according to their velocity and estimations previously done. They must always take the client priorities into account. It is very helpful to guarantee a commitment by the developers to the stories they will do. It helps the team to create a predictable and constant development cadence. Shore and Warden (Shore and Warden 2008) points out that if you use iterations well, your team can have consistent and predictable velocity so stakeholders will know what to expect and will trust that it will deliver on its commitments.

Slack. This practice consists of adding time (slack) in the project to unforeseen events. This extra time can be used to find solutions and new features. It is used to help the team guarantee that they will accomplish the desired velocity and finish all the stories they committed to in the iteration time. This can be considered a management practice that can badly influence the development of the product. The team or the project manager has to agree on how much slack will be "safe" to consider, to guarantee that the product will be delivered as promised and to avoid overtime of the team. Shore and Warden (Shore and Warden 2008) suggests that slack must be incorporated so the team can consistently meet their commitments and maintain high morale. He also suggests spending the slack time paying technical debt (time spent correcting know bugs) when the team has finished everything on time.

Risk Management. This focuses on management and knowledge of project risks. There are generic risks in all projects such as new requirements, interruptions, and team members abandoning the team, among other things. It is used to inform the client, the stakeholders and team members about the risks the project faces and what can be done to mitigate them. According to Shore and Warden (Shore and Warden 2008) the risk management in agile software belongs to the whole team, the team must guarantee delivery on their commitments even in the face of disruptions.

Meeting Minutes. This practice is simple. After all meetings, someone that was previously assigned makes a summary of the major points of the meeting, and what was agreed upon among all the

those involved. Everyone involved in the project should receive the Meeting Minutes and people that participated in the meeting should sign off on the Meeting Minutes in some way (written or electronically). According to Lutz (Lutz 2009) "Minutes can be used as explicit means of documentation to ensure that common ground is reached for comprehension, interpretation and controlling." Meeting Minutes can avoid breakdowns between the team and the client and within the team because it is an explicit documentation of the commitments made by everyone in the project.

4.5 Developing Practices

Figure 6 - Developing Practices

Spike Solutions. It is the practice of small and isolated experiments and research, which tries to make better development decisions. Normally this experiment has a maximum time deadline. It is important to clarify technical problems and to evaluate possible solutions without spending too much time and resources on the matter. Mitigating the risks of something that is not within the knowledge of the team, and helping the team to make proper decisions is also important. Shore and Warden (Shore and Warden 2008) states, "when you clarify technical questions with well directed, isolated experiments, you spend less time speculating about how your program will work. "

Test Driven Development (TDD). In this technique the team member has to first develop a test and then develop the code that passes the test. TDD has five principles: Think (think about the test), Red Bar (to make code that fails the test), Green Bar (to make code that passes the test), Refactor (refactor the code done to make it more clear, clean and concise), Repeat (start this process again with another functionality). It decreases mistakes and consequently the time spent in tests and debugs, helping the team to maintain cohesion and to achieve their goal in the time committed. George and Williams (George and Williams 2003) found out in an empirical study that developers using TDD techniques produced higher quality code.

Refactoring. For this, the team has to look at the developed code and try optimizing it in some way, avoiding repetitions and ambiguities. It refers to a change in the structure of the code but not in the behavior of the code. It is used to improve the code constantly, increasing the maintainability and the integration. It reinforces the commitment to the team to achieve the goal and the best quality, helping the team to develop shared understanding.

Simple Design. It is a principle, to make code as simple as possible. One example can be, trying to isolate foreign code (code that was done for people outside the project), trying to find the mistakes and failures in design as fast as possible, trying to write code only once. It is helpful to avoid component support problems and update problems. Shore and Warden point out: "when you create a simple design, you avoid adding support for any features other than the ones you are working in the current iteration." It helps the team to maintain the focus in the stories at hand and to achieve a quality product.

Incremental Design and Architecture. The team has to try and develop exactly what they will need in the functionality they are working with, nothing more. They should not try to anticipate anything, and do everything step by step regardless if the team member already had seen something that could be useful in the future or not. It helps to maintain a constant development rate and in the improvement of software quality. "Only if a product is developed and delivered incrementally, frequent feedback can be given." Concluded Carbon et al (Carbon, et al. 2006).

Performance Optimization. This is about optimizing only when there is a real customer issue, when something needs to be done such as finding out which performance a client wants or at what value is expected, which has to somehow be measured. It is very helpful to team members to direct their efforts to what is important to the client. This kind of practice improves the relationship between client and the team. Shore and Warden (Shore and Warden 2008) said: "when you optimize code as necessary, you invest in activities that customers have identified as valuable over perceived benefit. …Your code is more maintainable, and you favor simple and straightforward code over highly optimized code."

Customer Testing. The client tells the developers which test he/she would do in each story to be accepted. With this in hands, the developers have the entire acceptance test, and they can be included in the development of the TDD technique. It is used to mitigate the number of logical mistakes, rules, ambiguities and special cases of the software. According to Shore and Warden (Shore and Warden 2008) "reduce the number of mistakes in your domain logic. You discuss rules in concrete, unambiguous terms and often discover special cases you hadn't considered. "

Customer Reviews. The client runs an extensive analysis of the software as a trial to establish how the software will be used and how it will behave in a real environment. It also helps in uncovering errors and logic failures of the software.

Exploratory Testing. Here, tester designed tests are conducted with the help of one of the developers. This test does not have a pre-defined script; the idea is to follow the flow of the information. Some methodologies used are: None, Some, All, Too Big Too Small, Just Right, Beginning, Middle, End,

Create, Read, Update, Delete, Command Injection and Data Type Attacks. It is used to reveal mistakes that will be discovered easily by the users. It may help the team improve their own process and reduce the number of problems in future iterations. Shore and Warden (Shore and Warden 2008) states: ”When you use exploratory testing, you discover information about both the software and the process used to create that software.”

5 Influence Model

After a short period of time during the research observations we saw that the three influencing variables (Communication, Coordination and Motivation) stood out among the variables mentioned in the Related Work. It was also possible to envision that the problems belonging to these three variables could be categorized according to two different points of view: Internal (within the team) and Client (outside the team). Figure 7 shows the influence model: variables influencing the teamwork and the different points of view they have.

Aranda (Aranda 2010) points out in his thesis that "Achieving effective coordination and communication is the central problem of software development". He recognizes that in the research literature this could be controversial, but states that coordination and communication are two linked aspects showing the essential complexity of the nature of any software development project.

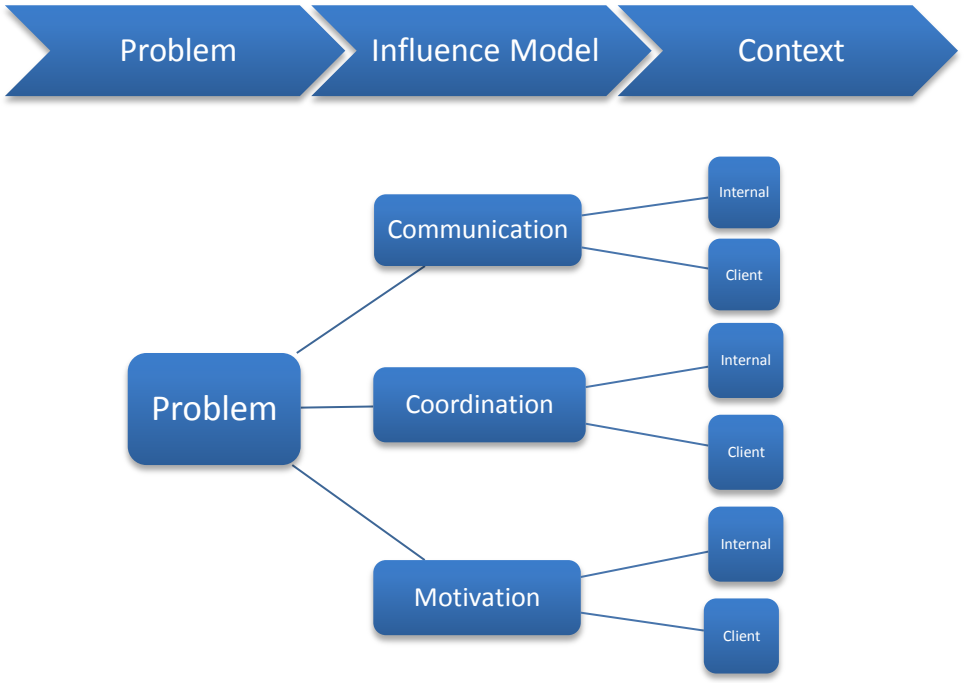

Figure 7 - Influence Model

Ryan and Deci (Ryan and Deci 2000) wrote about motivation: "to be motivated means to be moved to do something; a person who feels no impetus or inspiration to act is thus characterized as unmotivated, whereas someone who is energized or activated towards an end is considered motivated."

Humphrey (W. S. Humphrey 1996) says that "in technology, there are many failures for every success, and it is easy to become discouraged, unmotivated…. people need to be charged up and reminded that the goal is important and achievable."

For software development, in general, communication, coordination and motivation need to occur among all individuals involved in the project: the team members and the clients and stakeholders. In this thesis I refer to a client as anyone who is involved in the project that has any interactions with team members that can affect the project; e.g. they could also be users of the product under development.

5.1 Communication

This section presents a list of thinklets that deal with recurring communication problems. Subsection 5.1.1 is focused on internal communication problems and subsection 5.1.2 is focused on external communication (i.e. with the user/client).

5.1.1 Internal Communication

Internal communication deals with the challenge of each team member to conduct effective communication with his peers. This subsection talks about internal communication, the major recurrent problems that were caused by it and the ThinkLets that can be used to address them.

ThinkLets	Recurring Problem	Corrective Actions	Useful Practices
Stage fright	There are members that are not willing to express their opinion in public. This problem reduces the capacity of the team to generate ideas, to identify challenges or the capacity to generate warnings during the project (Hackman 1990).	○ Weekly meetings, where all team members must report their work status (Round-Robin Meeting). According to Humphrey (W. S. Humphrey 1996) meetings directed by a team member, where this member exerts some kind of pressure to get everybody to speak, efficiently mitigates the problem. ○ Try to generate confidence between the team members. This can be done with Team-Building activities (games) or with some social extra project	○ Stand Up Meeting ○ Trust ○ Sit Together ○ Team-Building Workshop ○ Coaching

ThinkLets	Recurring Problem	Corrective Actions	Useful Practices
		activity. Kapp (Kapp 2009) research found out that the Team-Building Workshops at the beginning of the project could guarantee a higher level of trust and confidence between team members.	
Playing dumb	In the team there are members who do not listen or take the ideas of their peers seriously. This goes against the process of trust generation and the team cohesion (Pfaff and Huddleston 2003).	○ Meetings with the only purpose of conciliation or confrontation, whatever the need of the team members involved may be. This kind of meeting could also involve the whole team if needed (Whitten 1995). ○ According to Page y Donelan (Page and Donelan 2003), one solution can be to begin a project with Team-Building Workshop. These techniques should have the ability to analyze the capacity and skills of team members prior to beginning the real work. With this kind of strategy the team knows in advance how each one works best and how to avoid this kind of conflict. Amato and Amato (Amato and Amato 2005) suggest conducting personality tests, such as the MBTI (Myers-Brigg Type Indicator) prior to team formation, to enhance the team effectiveness.	○ Trust ○ Sit Together ○ Peer Activities ○ "Done, Done" ○ Collective Code Ownership ○ Public Profile ○ Team-Building Workshop
Team hijacking	The team relies on the knowledge and capacity of one or a few team members only. In a project there are some points in which the	○ In any scope, knowledge has to be transferred; the activities should be done at least in pairs (ex. analysis, design or development). To make this really useful and efficient, one of the	○ Peer Activities ○ Collective Code Ownership

ThinkLets	Recurring Problem	Corrective Actions	Useful Practices
	discussion/validation/ generation of ideas cannot be done because there is not enough knowledge in just one person (or a few). Pfaff y Huddleston (Pfaff and Huddleston 2003) defined this kind of behaviour as a "team hijacking".	team members must be an expert on the subject. According to Karhatsu et al (Karhatsu, et al. 2010) the redundancy of the agile methodologies, where one member has the ability to do the tasks of any other team member, and should be assured by the whole team through self-management. o Peer Activities sessions to share the knowledge. It is important to clarify that the responsibility of acquiring the transmitted knowledge is a team member responsibility. To Page and Donelan (Page and Donelan 2003) team members have to know and take responsibility in the development of their own team, always trying to increase their capacity for accomplishment. o Perform research or a spike solution lead by an expert. Pfaff y Huddleston (Pfaff and Huddleston 2003) talk about constantly controlled interventions as compelling ways to ensure effective practices.	o Vision o Planning Game o Ubiquitous Language o Spike Solution
Why bother to answer	Team members do not answer, or not answer in time other team member's requests. This goes against the dynamics of teamwork and the execution of the project itself.	o Make the team behave towards written communication in order to maintain good levels of persistence and traceability. o On a daily basis team members must access the official communication system of the team. This policy has to	o Peer Activities o Collective Code Ownership o Trust o Coaching

ThinkLets	Recurring Problem	Corrective Actions	Useful Practices
		be stated and the tool selected in the beginning of the project. ○ Define an answer protocol to emails or other persistent means of communication, where the urgency is stated in the subject of the message (Cohn 2009). For example: ○ FYI (No need to answer). ○ Low Importance (Answer needed within the week). ○ Important (Answer needed within 48 hours). ○ Very Important (Answer needed within 24 hours). ○ URGENT (Answer needed within 3 hours). ○ According to Page y Donelan (Page and Donelan 2003) the roles of action that each team member has should be known to the whole team (not only formal roles). The team should also develop a "psychological contract" between its members at the beginning of the project, where the parameters of what each member believes are acceptable, well stated and understood by the whole team.	
Ego	Members of the team with strong personalities (ego) engage in conflict for some specific subject of the project.	○ When the scope of the conflict is of technical nature and relevant; the client is the only person who can decide which solution is better and	○ Trust ○ Sit Together ○ Planning Game

ThinkLets	Recurring Problem	Corrective Actions	Useful Practices
	This kind of problem can create a supremacy contest within the team, hampering the process of generating ideas and trust within the team. According to Eisenhardt and Schoonhoven (Eisenhardt and Schoonhoven 1990) some task conflicts could bring benefits to the project development. Jehn et al (Jehn, Nothcraft and Neale 1999) found out that these benefits have a limit. High levels of conflict within the team can distract and affect the final product, leading to unachieved goals.	give him real value. When the subject is not relevant the team has to choose the solution itself, seeing which one fits better with the project in terms of permanent solution. (Cohn 2009) o Define a work protocol where the tasks are assigned to each team member with the technical or logical solution pre-defined. This kind of decision should be made in a design meeting where all the team members address challenges and what to do in the project. o Active client participation in all the stages of the project, using techniques such as Planning Game, which are shown to be empirically effective according to Fraser (Fraser and Wotawa 2007).	o Peer Activities o Coding Standards o Ubiquitous Language o Collective Code Ownership
I do not belong	Some team members do not feel they belong to the team. This kind of problem directly affects the final result of the team, because without team cohesion, all the advantages of working in a team are lost. Synergy only happens when the resulting work done by the team is greater than the sum of individual's work. If the team members do not feel they belong to the team this synergy ceases to exist.	o Team meetings with the purpose of promoting team integration and trust. One known technique is to promote meetings where there are activities that are different from the usual ones performed by a team (Barr, Dixon and Gassenheimer 2005) o More frequent team meetings; short ones, with the intention of familiarizing the team members with each other, so everyone involved knows the skills that each possess. o Organize social activities or a shared meal (Shore and Warden 2008).	o Trust o Sit Together o Peer Activities o Energized Work o Team-Building Workshop

ThinkLets	Recurring Problem	Corrective Actions	Useful Practices
Free Riders	Some team members do not want to fully participate on the project, so they try to emulate work. They are used to pretend to be busy, never commit themselves to tasks and they easily let others do their work. Humphrey (Humphrey and Tomas 2010) says ”nothing can be more disruptive than to have some people in a group openly getting away with something.” Dommeyer (Dommeyer 2007) defines free riders “Group members who shirk their obligation in the hopes of benefiting from the work of others are often referred to as social loafers or free riders”.	○ At the beginning of the project, a Team-Building Workshop can help team members bond.. ○ A team meeting with the purpose of letting this team member know that his action brings consequence to the whole team. The team can threaten to kick the free rider from the team. Humphrey (Humphrey and Tomas 2010) recognizes that this kind of problem is a major threat to academic projects, because in industries people can be fired, and the consequence for this kind of behaviour in the academia is not so severe. ○ Dixon and Gassenheimer (Dixon and Gassenheimer 2003) suggests providing students opportunities to familiarize themselves with their group members in a social context, away from the high pressure of the academic setting.	○ Peer Review ○ Team-Building Workshop ○ Root Cause Analysis

5.1.2 Client Communication

Client communication involves the communication between the development team and the client. This subsection introduces the major recurrent problems that can be present during a project and the ThinkLets we can use to address them.

ThinkLets	Recurring Problem	Corrective Actions	Useful Practices
Client availability	A client having real availability to communicate with the team is a difficult goal to reach. This	○ To establish a fixed day and time of a weekly meeting with the client, prior to the beginning of the project. It ensures	○ Trust ○ Retrospectives ○ Real

ThinkLets	Recurring Problem	Corrective Actions	Useful Practices
	problem is a threat to the project, the client is an important part of the project; he/she is the expert. It is his /her responsibility to define and clarify key points to the project. If he/she is not available he/she can become a bottleneck for the project, and lockout the decision-making process and validation of the project.	that such time period will be available for the team; therefore, the need for negotiation does not exist, trying to guarantee at least some participation of the client in the project. ○ Define a decision relevance protocol that should be used in every meeting. The decisions to be made and their relevance, have to be reported to the client at least a day prior to the meeting. With this kind of information at hand, the client can summon other people needed to discuss the decisions that need to be acted upon. For example: ○ Low Importance (Project Status and simple decisions that the technical counterpart can easily decide). ○ Important (Project Status and complex decisions where decisions would be better made by someone else than the counterpart). ○ Critical (Project Status and critical decisions where it must be someone else other then the counterpart to do).	Customer Involvement ○ Sit Together ○ Informative Workspace ○ Team-Building Workshop ○ Kanban
Client communication	The client is not able to effectively communicate with the team, so he/she is not able to transmit his real needs to	○ Call a meeting that all people involved in the project must attend, with the purpose of defining the scope of the project and mitigate it. The team has	○ Vision ○ Planning Game ○ Stories ○ Estimating

ThinkLets	Recurring Problem	Corrective Actions	Useful Practices
	the team. This kind of problem can generate anxiety among the team members because they do not understand what the client really wants. It can cause the team to start working on something different from what the client really needs or wants.	to gather together after this scope meeting and debate all the points discussed. Afterwards then get back to the client with something (a document, a user story, user case), so the client can confirm that there are no misunderstandings among them. ○ Holding meetings with all the people involved in the project using agile techniques such as: Planning Game and Divide and Conquer.	○ Real Customer Involvement ○ Sit Together ○ Ubiquitous Language
You did what I asked, but is not what I need	At the end of the project the client has a product that addresses all the requisites he/she asked for, but he/she is not satisfied with the product obtained. This can generate frustration and low morale among team members severing the relationship between the client and the team.	○ At the beginning of the project the acceptance standards must be stated by the client. ○ Hold at least a weekly meeting with the client, where the team presents the project status and the product itself.	○ Planning Game ○ Iteration Planning ○ Reporting ○ Slack ○ Stories ○ Estimating ○ Real Customer Involvement ○ Sit Together ○ Exploratory Testing
I know what I want, but I do not know why	The client knows what he/she wants to have at the end of the project, but he/she does not know the problems trying to solve.	○ To focus the meeting on finding the problem in which to solve and the context in which it belongs. Maintain this approach until the problem and the context can be identified.	○ Vision ○ Planning Game ○ Stories ○ Real Customer Involvement
I do not know what	Unjustified and constant request changes frustrate the	○ At the beginning of the project the Vision of the project should be well	○ Vision ○ Real

ThinkLets	Recurring Problem	Corrective Actions	Useful Practices
I want	team since it can change the project goal and / or the scope of the project.	stated and the whole team should participate in Planning Games and Estimates to guarantee that the knowledge is spread among everyone,; team members and client. ○ Team members have to let the client know that changes can be done. Though it is healthy for the project, all changes have a consequence. With each change, a new Iteration Planning is done and probably a new Release Planning.	Customer Involvement ○ Iteration Planning ○ Estimating
I did not say that	The client and / or the team do not recognize the agreement reached in the meeting.	○ During the meeting the team has to guarantee that a Ubiquitous Language is being used, so everybody is talking about the same subject. ○ At every meeting, minutes should be taken and everybody involved has to agree to what has been recorded. ○ Team-Building Workshops can help to break the conflict between client and the team and help them see that they are all together in the project, seeking the same goal.	○ Meeting Minutes ○ Ubiquitous Language ○ Team-Building Workshop.

5.2 Coordination

This section shows a list of thinklets that deal with recurring coordination problems. Subsection 5.2.1. is focused on internal coordination problems and subsection 5.2.2 is focused on external coordination problems.

5.2.1 Internal Coordination

Internal coordination is the process through which the team members coordinate their own activities to reach a common goal. Limitations in the coordination activities produce major recurrent problems. The following table presents such problems as well as the ThinkLets that can be used to deal with them.

ThinkLets	Recurring Problem	Corrective Actions	Useful Practices
Lone wolf	There is one team member who monopolizes project tasks. This attitude can generate a knowledge concentration resulting in the project becoming dependent on one team member. This can turn into a bottleneck at any moment. Lone wolf is considered a "psychological state" in which one prefers to work alone when making decisions and accomplishing goals (Barr, Dixon and Gassenheimer 2005). They also define lone wolves as people highly committed, who devote a lot of energy to complete tasks.	○ Be assure that the workload between team members is equally distributed. ○ Have a weekly meeting of the team to evaluate what was done, where tasks can be reassigned between team members according to each one capacity or knowledge, taking always the workload balance in account. ○ According to (Barr, Dixon and Gassenheimer 2005), "the lone wolf with some guidance could be an ideal candidate for mentoring others who share interest in their work, and so contribute to the team effort.".	○ Trust ○ Sit Together ○ Ubiquitous Language ○ Stand Up Meetings ○ Iteration Planning ○ Public Profile ○ Coaching

ThinkLets	Recurring Problem	Corrective Actions	Useful Practices
Knowledge of a few	The team relies on the knowledge and skill of one or a few members of the team. The team capacity to generate ideas and make decisions fall on just one or a few people.	○ Define and assign responsibilities and tasks in a clear and explicit way. ○ Define a work protocol that is based on written communications (as a wiki for instance).	○ Peer Activities ○ Sit Together ○ Stand Up Meetings ○ Coding Standards
Meetings absence	Team members do not attend the meetings. This goes against the knowledge transfer that all projects need. Besides the team fells that this member is not doing their share and the commitment of the team deteriorates. As stated by Lee Iacoca (Iacoca and Novak 1984): "If everybody is suffering equally, you can move a mountain. But the first time you find someone goofing off or not carrying his share of the load, the whole thing can be unravelled."	○ Have a conciliation meeting with the problematic team member sometimes can be the whole team conciliation. ○ Social events and games (Game Theory) with the purpose of bringing cohesion and unity to the team.	○ Trust ○ Energized Work ○ Stand Up Meetings ○ Team-Building Workshop ○ Feedback
I only know my own belly button	Team members do not know what others are doing. This cause feelings of uncertainty among team members, and may cause some tasks to be done in duplicity. According to	○ Structured weekly team meetings, with a specific agenda to address, such as: project status, each member's task status, task analysis and task reassignments (if needed). ○ Define a work protocol of documentation where each team	○ Stand up Meeting ○ Informative Workspace ○ Kanban ○ Public Profile

ThinkLets	Recurring Problem	Corrective Actions	Useful Practices
	Humphrey (Humphrey and Tomas 2010) "Without timely and complete communication, the team members do not know what their teammates are doing, they cannot support each other, and they do not feel a sense of progress."	member has to write about what he/she did during the week (in a Wiki for example) besides the existence and usage of any version control system.	
I do what I think is needed	Team members perform changes in the project that were not assigned to him/her, or worse, not even stated by the client. Without any repercussion analysis the final product and the work other team members are doing can be affected by this change. During design and development phases, developers always see ways to improve their work. These well-intentioned changes are hard to control and hard to avoid without a defined process and plan. (Robillard 1996)	○ Define a protocol of work to the team, where each member of the team has to follow and to work just on tasks previously assigned to him. ○ Weekly meetings of the team to evaluate what was done and the status of each task, and task reassignment when needed.	○ Informative Workspace ○ Stand-up Meeting ○ No Bugs ○ Collective Code Ownership ○ Continuous Integration ○ Test driven development ○ Refactoring ○ Performance optimization ○ Kanban
Last minute delivery	Team members do their work but they deliver their work just before the deadline. This is a problem	○ When the project starts, the estimation technique that will be used has to be defined. The estimation technique chosen has to	○ Planning Game ○ Iteration Planning ○ Estimating

ThinkLets	Recurring Problem	Corrective Actions	Useful Practices
	to the final quality of the problem, because a task ended just before the deadline probably will not have all the validation and tests needed to assure the product quality.	be in the mind of the team members when the estimation process begins. When needed (scope change as an example) the estimation must be re-evaluated. All team members must participate and agree on the estimates to make sure that they all commit with the project. ○ Conduct weekly team meetings to evaluate what was done and the status of each task, besides task reassignment when needed. To help the team keep track of the tasks it is suggested by many authors and methodologies (PSP, CMM, CMMI, TSP) to use a task management tool to maintain control of the project in a simple and real manner.	○ Stand up Meeting ○ Slack
I do in my own time	Team members are not respecting the due dates of their tasks. This increases the probability that the project will fail to be delivered to the client on the agreed date.	○ Prior to the beginning of the project the estimation technique that will be used must be defined. The chosen estimation technique has to be in the mind of the team members when the estimation process begins. When needed (scope change as an example) the estimation must be re-evaluated. All team members must participate and agree on the estimates, to make sure that they all commit to the project. ○ Weekly meetings with team members who have the	○ Planning Game ○ Iteration Planning ○ Estimating ○ Stand up Meeting

ThinkLets	Recurring Problem	Corrective Actions	Useful Practices
		responsibility of taking care of the tasks, their estimation and their real time. It is suggested to use a management tool to help in that control.	
No notes	Team members do not use any methodology or tracking device during clients meeting. They do not take notes of the requirements, changes or requests that the clients have. This kind of problem can affect the relationship between client and team, because team members will start to ask the client the same questions, over and over.	○ Define a documentation protocol of the project since the beginning. The documentation should be written and updated constantly. It is suggested by various methodologies to keep all project documentation in a single repository where the whole team can have access. ○ Choose a team member to be responsible for the documentation (if there is no project manager). This person is responsible for taking written meetings minutes of all the clients meetings with the team. All the minutes should be available to the whole team for consultation.	○ Planning Game ○ Stories ○ Real Customer Involvement ○ Meeting Minutes
Where are we	Team members do not have visibility of the status of the project; they do everything that the client asks. This kind of problem can shake up the relationship within the team. Motivation and morale decreases and the team looses commitment to their members, resulting in lost in productivity and quality.	○ Define a protocol that states that the team will only do what the client asked formally and according to the priorities defined by him. One typical problem occurs with performance, the team does not have to improve performance if not formally asked and prioritized by the client. ○ At the beginning of the project partial deliveries must be defined between client and team. ○ The client must state at the start of	○ Planning Game ○ Iteration Planning ○ Release Planning ○ Informative Workspace ○ Kanban

ThinkLets	Recurring Problem	Corrective Actions	Useful Practices
		the project the acceptance criteria of the project, and as soon as the requisites are defined, the client must prioritize them.	
Paralysis analysis	The team is frozen, waiting for possible solutions to a problem, or trying to analyze and learn new technologies (paralysis by analysis). This halt of the team can affect the time agreements.	○ Define in advance the amount of time that can be spent in analysis. For example: ○ 1st. Day Web search, tutorials If the search is positive, one day of Spike Solution / Spin Off. ○ 2nd. Day: Try to make contact with someone that is an expert on the subject. ○ 3rd. Day: Evaluate the impact of change in the project strategy. ○ Perform an evaluation phase of the new technology together with any initial phases of the project. This is helpful when the team knows in advance that the project will use new technologies that no one in the team knows.	○ Test Driven Development ○ Refactoring ○ Spike Solutions ○ Decision Making ○ Coaching
Why to decide	Team members or the whole team do not have enough confidence to make the decisions needed. This kind of problem can deeply affect the delivery date of	○ Define a work protocol where the task assigned to each team member contains previously defined technology or logic. This kind of definition can be developed in a design meeting at the beginning of	○ Peer Activities ○ Root Cause Analysis ○ Retrospectives ○ Stand Up Meetings ○ Decision Making

ThinkLets	Recurring Problem	Corrective Actions	Useful Practices
	the project.	the project. ○ Define a protocol for making decisions within the team. If the decision is a minor decision the team member has to decide what is the best thing to do within the day. If he/she is not capable of making this decision, he/she has to call their peers to help him decide what to do the following day.	○ Coaching
I decide	The team has members who make important decisions alone, without properly analysing or at least thinking it through beforehand. This goes against the dynamics of the team and can put the quality of the final product at risk. There also exist the potential possibility for having to rework the product details.	○ Define a work protocol where the tasks assigned to each team member already have the technology or logic needed previously defined. This kind of definition can be done in a design meeting usually done in the beginning of the project. ○ Define a decision protocol, this way the team members can have a clear understanding of what they can decide on their own and what should involve other team members. For example: ○ Low Importance (the decision affects only the team member tasks and do not have any integration with other tasks) – the decision can be made by any team member alone. ○ Important (the decision to be made affects other team	○ Incremental Design and Architecture ○ Peer Activities ○ Retrospectives ○ Decision Making ○ Simple Design

ThinkLets	Recurring Problem	Corrective Actions	Useful Practices
		members' tasks) – the team has to make the decision. ○ Critical (the decision transversally affects the whole project) – the decision must be made by the team together with the client.	
No sell	The team did not prepare themselves for delivering the product to the client; so the product was sub evaluated. This problem can affect the motivation and the morale of the team, reflecting badly in their future endeavours.	○ Define a delivery protocol and presentation of the product (partial or final). For example: ○ Demo testing. ○ Presentation must have all the major milestones of the project history ○ Presentation must answer or show all the clients questions / use cases/ stories. ○ All presenters must rehearse, memorize and recite the presentation without reading anything.	○ Real Customer Involvement ○ Customer Reviews ○ Customer Testing ○ Iteration Planning ○ Iteration Demo ○ Ten minutes build ○ Feedback ○ Version Control
"I" not "us"	Team members are still acting selfish. They do not take into account that they now have to work on a team, and act as team members. This kind of problem reduces the team ability to generate ideas and synergy.	○ Meetings with the purpose that team members get to know each other and each other's skill. ○ Team meetings with the purpose of promoting team integration and trust. One known technique is to promote meetings with activities different from the usual ones they perform as a team (Barr, Dixon and Gassenheimer 2005) ○ Extra project meetings, social meetings, sharing a meal. The idea is	○ Energized Work ○ Peer Activities ○ Retrospectives ○ Sit Together ○ Team-Building Workshop ○ Peer Review

ThinkLets	Recurring Problem	Corrective Actions	Useful Practices
		to let team members bond freely. According to (Shore and Warden 2008): "Something about sharing a meal breaks down barriers and fosters team cohesiveness."	
Nobody responsible	There is no coordination, since nobody is responsible for anything.	○ Hold a Team-Building Workshop to demonstrate to team members the importance of trust and commitment. ○ Rotating the coordination of the team between team members can help the team to understand the importance of coordination and responsibilities	○ Coaching ○ Team-Building Workshop

5.2.2 Client Coordination

Client coordination involves a set of activities that allows the development team to coordinate their effort with the client. This subsection presents the main recurrent problems generated by client coordination activities and the ThinkLets we can use to address them.

ThinkLets	Recurring Problem	Corrective Actions	Useful Practices
I did not asked that	The client or the team does not remember some requirements or changes asked for during the project. This can be a nasty source of conflicts between the client and the team.	○ Define a documentation or process protocol at the beginning of the project. The documentation or the processes have to be written and the documentation has to be constantly updated. It is suggested that everything be maintained in a shared directory where the whole team has access. ○ The team has to choose one	○ Planning Game ○ Stories ○ Customer Review ○ Real Customer Involvement ○ Meeting Minutes

ThinkLets	Recurring Problem	Corrective Actions	Useful Practices
		responsible member for all the documentation (if there is no project manager). This person is responsible for taking meetings minutes, especially for those where the client is present. All the minutes should be available to the whole team.	
What problem	The client has no clear idea of the problem he/she wants to solve. This kind of problem clashes with project implementation because the client will probably change his/her mind and the project scope many times, until they have an idea of what the problem is. In the meantime the relationship between client and the team will erode.	○ Have a scope and definition meeting with the client to help the client think about what he/she wants, freethinking techniques such as Brainstorming or LeafHooper. ○ Define an early prototype protocol so the team can get an early feedback of the client and change the project scope early in the project.	○ Vision ○ Planning Game ○ Stories ○ Iteration Demo
Client decision	The client is not available to make the decision the project needs. The delays in decision-making can affect the delivery schedule of the final product.	○ Maintain all communication in written format, so it can be traced and stored. ○ Hold weekly meetings between the client and the team. These meetings should be scheduled at a set day and time at the start of the project to try and guarantee the client attendance. ○ Define an answer protocol to mails or other persistent means of communication, where the urgency is stated in the subject of the message	○ Real Customer Involvement ○ Informative Workspace ○ Sit Together ○ Planning Game ○ Stories ○ Kanban

ThinkLets	Recurring Problem	Corrective Actions	Useful Practices
		(Cohn 2009). For example: ○ FYI (No need to answer). ○ Low Importance (Answer needed within the week). ○ Important (Answer needed within 48 hours). ○ Very Important (Answer needed within 24 hours). ○ Urgent (Answer needed within 3 hours).	
Change again	The client changes the requisites of the project. This can have a huge impact on the project schedule and in the quality of the final product.	○ Define an analysis process of the changes requested by the client ("Change Management"). This analysis has to estimate the impact of the changes in the project and have to be done by all the team members. The results of this analysis have to be reported to the client. The team has to negotiate with the client, a new deadline or a decrease in the number of the requirements if necessary. ○ Define a tool to implement a formal workflow to manage the change management of the project.	○ Planning Game ○ Stories ○ Estimating ○ Risk Management
Client meeting	It is difficult to establish a meeting with clients.	○ Hold weekly meetings between the client and the team. These meetings should be scheduled on fixed days and times at the start of the project to try to guarantee client attendance.	○ Real Customer Involvement ○ Sit Together ○ Planning

ThinkLets	Recurring Problem	Corrective Actions	Useful Practices
		○ Have a Team-Building Workshop with the client to show them the importance of their active participation on the project.	Game ○ Team-Building Workshop

5.3 Motivation

This section presents a list of thinklets that deal with recurring motivation problems. Subsection 5.3.1. is focused on internal motivation problems and subsection 5.3.2 is focused on external motivation (i.e. with the user/client).

5.3.1 Internal Motivation

Internal motivation allows a team to maintain high morale and to try guaranteeing positive development and a strong relationship among team members. The lack of internal motivation generates major recurrent problems within the team. This subsection presents these problems and the ThinkLets that can be used to deal with them.

ThinkLets	Recurring Problem	Corrective Actions	Useful Practices
I do what I'm told	Some team members stopped contributing to the project, choosing to only take on project tasks when assigned, and then only if the tasks are simple, not requiring further evaluation and analysis. This kind of apathy can be easily spread among team members, gravely affecting the advantages of teamwork.	○ Motivational activities with the team members bring back cohesion to them and show that the commitment to the project is relevant to everyone. ○ Remind the team of the vision and the relevance of the project. According to Humphrey (Humphrey and Tomas 2010) it is periodically necessary to reinforce team commitment. ○ "Commitment is based on four requirements: should be voluntary, must be visible, must be credible and must be owned by the people who will do the required work." (Humphrey and Tomas 2010). Based on that, a meeting has to be made to evaluate why requirements are not being met and the project manager should directly address this problem.	○ Trust ○ Stand Up Meetings ○ Sit Together ○ Energized Work ○ Retrospectives ○ Coaching

ThinkLets	Recurring Problem	Corrective Actions	Useful Practices
Bad decisions	The project manager (or the person responsible for the team) is not making the best decisions for the team. This problem can rupture team cohesion and consequently negatively influencing the results that the team could achieve.	○ Plan an open team meeting with the purpose of dealing with the project manager problem (or the person in charge). All members have to express their views and concerns. The project manager has to listen and to address all the problems directed to him. It is a good idea to have someone a human resources staff running conciliation process. ○ In some cases when the project is self-directed the team may freely choose another project manager or choose an alternate project manager.	○ Trust ○ Planning Game ○ Stand Up Meetings ○ Retrospectives ○ Decision Making
Us vs. Them	Team members are unmotivated because of the constant friction between them and the client or within the team.	○ Remind the team and the client of the vision and relevance of the project. According to Humphrey (Humphrey and Tomas 2010) it is periodically necessary to reinforce team commitment. ○ Team-Building Workshop to break the barrier between the team and the client ○ Define a documentation or process protocol at the beginning of the project. The documentation or the processes has to be written and constantly updated. It is suggested that everything be maintained in a shared directory where the whole team has access.	○ Vision ○ Real Customer Involvement ○ Team-Building Workshop ○ Coaching

5.3.2 Client Motivation

Client motivation is the ability that a team has to keep the client involved and satisfied with the project in some way. The lack of this motivation generates a set of recurrent problems, which are presented in the next table. The table also presents the ThinkLets able to deal with them.

ThinkLets	Recurring Problem	Corrective Actions	Useful Practices
Client commitment	The client is not committed to the project, as he/she should be. This complicates the whole project, because the project depends on the client to define requisites and solve ambiguity issues during the project development.	○ Make the client aware of the importance of his commitment to the project. Involve the client in the strategic decisions of the project. ○ Define an answer protocol to emails or other persistent means of communication, where the urgency is stated in the subject of the message (Cohn 2009). For example: ○ FYI (No need to answer). ○ Low Importance (Answer needed within the week). ○ Important (Answer needed within 48 hours). ○ Very Important (Answer needed within 24 hours). ○ Urgent (Answer needed within 3 hours). ○ Hold weekly meetings between the client and the team. These meetings should be scheduled at set a day and time at the start of the project to try and guarantee client attendance.	○ Real Customer Involvement ○ Sit Together ○ Planning Game ○ Stories ○ Iteration Demo ○ Team-Building Workshop
Them vs. Us	Clients / Users unmotivated because of the constant	○ Define a documentation or process protocol in the early stages of the project. The documentation or the	○ Real Customer Involvement

ThinkLets	Recurring Problem	Corrective Actions	Useful Practices
	bickering with the developers.	processes should be written and the documentation has to be constantly updated. Everything should be maintained in a shared directory where the whole team has access. ○ Conduct Team-Building Workshops to break the barriers between the team and the client. ○ Remind the team and the client of the vision and the relevance of the project. According to Humphrey (Humphrey and Tomas 2010) it is periodically necessary to reinforce team commitment.	○ Vision ○ Team-Building Workshop

5.4 Correspondence Matrix

This section presents a correspondence matrix that links all the problems listed with the practices that can be used to mitigate them.

Table 3. Correspondence Matrix

Practice / ThinkLet	Peer Activities	Energized Work	Informative Workspace	Root Cause Analysis	Retrospectives	Trust	Sit Together	Real Customer Involvement	Ubiquitous Language	Stand Up Meetings	Coding Standards	Iteration Demo	Reporting	Team Building Workshop	Peer Review	Coaching	Kanban	Decision Making	Public Profile	Feedback	Done Done	No Bugs	Version Control	Ten-Minute Build	Continuous Integration	Collective Code Ownership	Vision	Stories	Estimating	Planning Game	Release Planning	Iteration Planning	Slack	Risk Management	Meeting Minutes	Spike Solution	Test Driven Development	Refactoring	Simple Desing	Incremental Design and Architecture	Performance Optimization	Customer Testing	Customer Review	Exploratory Testing
Stage fright						X	X			X				X		X																											
Playing dumb	X					X	X							X					X		X					X																	
Team hijacking	X								X																	X	X			X						X							
Why bother to answer	X					X										X										X																	
Ego	X					X	X		X		X															X				X													
I do not belong	X	X				X	X							X																												
Free Riders				X										X	X																											
Client availabity			X		X	X	X	X						X		X																											
Client communication							X	X	X																		X	X	X	X														

Practice / ThinkLet	You did what I asked, but is not what I need	I know what I want, but I do not know why	I do not know what I want	I did not say that	Lone wolf	Knowledge of a few	Meetings absence	I only know my own belly button	I do what I think is needed
Peer Activities						X			
Energized Work							X		
Informative Workspace								X	X
Root Cause Analysis									
Retrospectives									
Trust					X		X		
Sit Together	X				X	X			
Real Customer Involvement	X	X	X						
Ubiquitous Language				X	X				
Stand Up Meetings					X	X	X	X	X
Coding Standards						X			
Iteration Demo									
Reporting	X								
Team Building Workshop				X			X		
Peer Review									
Coaching					X				
Kanban								X	X
Decision Making									
Public Profile					X			X	
Feedback							X		
Done Done									
No Bugs									X
Version Control									
Ten-Minute Build									
Continuous Integration									X
Collective Code Ownership									X
Vision		X	X						
Stories	X	X							
Estimating	X		X						
Planning Game	X	X							
Release Planning	X								
Iteration Planning	X		X		X				
Slack	X								
Risk Management									
Meeting Minutes				X					
Spike Solution									
Test Driven Development									X
Refactoring									X
Simple Desing									
Incremental Design and Architecture									
Performance Optimization									X
Customer Testing									
Customer Review									
Exploratory Testing	X								

I did not asked that	Nobody responsible	"I" not "us"	No sell	I decide	Why to decide	Paralysis analysis	Where are we	No notes	I do in my own time	Last minute delivery	Practice / ThinkLet
		X		X	X						Peer Activities
		X									Energized Work
							X				Informative Workspace
					X						Root Cause Analysis
		X		X	X						Retrospectives
											Trust
		X									Sit Together
X			X					X			Real Customer Involvement
											Ubiquitous Language
					X				X	X	Stand Up Meetings
											Coding Standards
			X								Iteration Demo
											Reporting
	X	X									Team Building Workshop
		X									Peer Review
	X				X	X					Coaching
							X				Kanban
				X	X	X					Decision Making
											Public Profile
			X								Feedback
											Done Done
											No Bugs
			X								Version Control
			X								Ten-Minute Build
											Continuous Integration
											Collective Code Ownership
											Vision
X								X			Stories
									X	X	Estimating
X							X	X	X	X	Planning Game
							X			X	Release Planning
			X				X		X	X	Iteration Planning
										X	Slack
											Risk Management
X								X			Meeting Minutes
						X					Spike Solution
						X					Test Driven Development
						X					Refactoring
				X							Simple Desing
				X							Incremental Design and Architecture
											Performance Optimization
			X								Customer Testing
X			X								Customer Review
											Exploratory Testing

Practice / ThinkLet	Peer Activities	Energized Work	Informative Workspace	Root Cause Analysis	Retrospectives	Trust	Sit Together	Real Customer Involvement	Ubiquitous Language	Stand Up Meetings	Coding Standards	Iteration Demo	Reporting	Team Building Workshop	Peer Review	Coaching	Kanban	Decision Making	Public Profile	Feedback	Done Done	No Bugs	Version Control	Ten-Minute Build	Continuous Integration	Collective Code Ownership	Vision	Stories	Estimating	Planning Game	Release Planning	Iteration Planning	Slack	Risk Management	Meeting Minutes	Spike Solution	Test Driven Development	Refactoring	Simple Desing	Incremental Design and Architecture	Performance Optimization	Customer Testing	Customer Review	Exploratory Testing
What problem												X															X	X		X														
Client decision			X				X	X									X											X		X														
Change again																												X	X	X				X										
Client meeting							X	X						X																X														
I do what I'm told		X			X	X	X			X						X																												
Bad decisions					X	X				X								X												X														
Us vs. Them								X						X		X											X																	
Client Commitment							X	X				X		X															X		X														
Them vs. Us								X						X													X																	

6 Experimental Results

This chapter presents the experimental results of this thesis work. It shows the variables of the influence model observed during two semesters in the course CC61A – Software Project, and also the outcomes produced by the ThinkLets used.

CC61A – Software Project is a mandatory course that computer science students from University of Chile have to complete before they graduate. At the beginning of the semester the students are assigned to teams (composed of 4-7 students) and each team is assigned to a client. They have to plan and run a project that usually involves 16 weeks. These teams work 12 weeks (20 hours per week) in the client office. Each team has a coach (an experienced engineer) that meets with the students weekly in a fixed day and time to review the project's advance. The teams are normally self-managed and they have to make three presentations and release three product versions during the semester. The final goal is to put the software developed by each team into production.

6.1 Experimentation Scenario

The experimentation scenario used in this thesis was mentioned earlier in the Methodology section. The course CC61A was used as a laboratory for the analysis and experimentation. Two semesters were observed: Spring 2010 where 13 students were enrolled in three different teams (T1, T2, and T3), Autumn 2011 where 23 students were enrolled in four different teams (T4, T5, T6 and T7).

During the two semesters I accompanied the teams in all their 90-minute long meetings with their software development coach, which occurred every Thursday during the semester. I also attended their three presentations to clients, software engineering instructors and the other students of the course. Meeting minutes were taken, and once a week I talked more informally with the students to understand their perceptions of the team and of the problems that they were facing at that moment. In the Appendix there is more information on how these meetings were monitored.

6.2 Obtained Results

This section presents the results of the observations. In table 4 we see the observations made and the variables involved according to the influence model. During the observations the teams had only one recurring problem that did not fit in these variables. Such instances were classified as "Others" and the two problems observed were with the Clients Infrastructure.

Fifty-eight observations were made in total during the research: from 7 to 13 observations per team, and an average of 8 observations per team.

Analysing the results in terms of the influence model variables, it was found that the variable that had the most impact on a team was Coordination (46%), followed by Communication (30%) and Motivation (22%). Looking at the numbers in detail we found: 36% Internal Coordination, 18% Internal Communication, 12% Client Communication, 10% Client Coordination, 16% Internal Motivation, 7% Client Motivation and 2% were classified as others. The most interesting finding was that 71% of the observations were of the Internal problems of the team and just 29% were Clients problem.

Table 4 shows all the observations made by team and by influence model variables.

Table 4. Variables Observed

Team	Observation	Internal Communication	Client Communication	Internal Coordination	Client Coordination	Internal Motivation	Client Motivation	Others
T1	1	X		X				
T1	2		X		X			
T1	3			X				
T1	4	X		X				
T1	5	X				X		
T1	6				X		X	
T1	7			X				
T1	8				X		X	
T1	9					X		
T1	10	X		X		X		
T1	11					X		
T1	12							Client Infra-structure
T2	13			X				
T2	14		X			X		
T2	15	X		X				
T2	16			X				
T2	17		X					
T2	18			X				
T2	19			X				
T3	20		X					
T3	21			X		X		

Team	Observation	Internal Communication	Client Communication	Internal Coordination	Client Coordination	Internal Motivation	Client Motivation	Others
T3	22			X				
T3	23	X		X		X		
T3	24			X				
T3	25				X			
T3	26			X				
T4	27	X		X				
T4	28	X						
T4	29		X			X		
T4	30		X					
T4	31	X		X				
T4	32			X		X		
T4	33			X				
T4	34				X			
T4	35					X		
T5	36	X		X				
T5	37		X				X	
T5	38			X				
T5	39	X		X				
T5	40	X		X				
T5	41			X				
T5	42		X		X			
T5	43					X		
T5	44				X		X	
T6	45	X		X				
T6	46		X	X				
T6	47			X				
T6	48	X		X				
T6	49				X			
T6	50			X		X	X	
T6	51						X	
T7	52	X		X				
T7	53			X		X		
T7	54		X					

Team	Observation	Internal Communication	Client Communication	Internal Coordination	Client Coordination	Internal Motivation	Client Motivation	Others
T7	55	X		X				
T7	56		X		X			
T7	57					X		
T7	58							Client infra structure
Total	58	16	11	33	9	14	6	2

Table 5 shows the observation, their severity level, the ThinkLets used on each observation and the outcome after applying the thinklet. Each observation as well as the ThinkLets, can involve one or more variables of the influence model according to the nature of the problem. The severity levels were classified as Low (when it has a low impact on the team performance), Medium (if it has some impact on the team performance) and High (it has a high impact on the team performance). Finally, the outcomes were classified as Positive (the use of the ThinkLet changed the outcome of the team in a positive way), Negative (the use of the ThinkLet changed the outcome of the team in a negative way) and Neutral (the use of the ThinkLet did not change the outcome in any way).

Table 5. ThinkLets vs. Outcomes

Team	Observation	Influence Model Variables	Severity Level	Thinklets	Outcome
T1	1	Internal Communication	Medium	Playing Dumb	Positive
		Internal Coordination	Medium	Playing Dumb	Positive
T1	2	Client Communication	High	I do not know what I want	Positive
		Client Coordination	High	Client Meeting	Positive
T1	3	Internal Coordination	High	No notes	Positive
T1	4	Internal Communication	Medium	Lone wolf	Positive
		Internal Coordination	Medium	Team hijacking	Positive
T1	5	Internal Communication	Medium	I do what I'm told	Positive
		Internal Motivation	Medium	I do what I'm told	Positive
T1	6	Client Coordination	Low	Change Again	Positive
		Client Motivation	Low	Change Again	Positive
T1	7	Internal Coordination	Low	Paralysis Analysis	Positive
T1	8	Client Coordination	High	Client Availability	Negative
		Client Motivation	High	Client Availability	Negative
T1	9	Internal Motivation	Low	Nobody responsible	Neutral
T1	10	Internal Communication	High	Ego	Positive
		Internal Coordination	High	Why to Decide	Positive
		Internal Motivation	High	Team hijacking	Positive
T1	11	Internal Motivation	Low	Bad Decisions	Negative
T1	12	Others	Low		Neutral
T2	13	Internal Coordination	High	"I" not "us"	Positive
T2	14	Client Communication	High	What problem	Positive
		Internal Motivation	High	What problem	Positive
T2	15	Internal Communication	Medium	No sell	Positive

Team	Observation	Influence Model Variables	Severity Level	Thinklets	Outcome
		Internal Coordination	Medium	No sell	Positive
T2	16	Internal Coordination	Medium	Why bother to answer	Positive
T2	17	Client Communication	High	Us vs. Them	Positive
T2	18	Internal Coordination	Medium	Client Decision	Positive
		Internal Coordination	Medium	I decide	Positive
T2	19	Internal Coordination	Medium	I do in my own time	Positive
T3	20	Client Communication	High	You did what I asked but it is not what I need	Positive
T3	21	Internal Coordination	High	Ego	Positive
		Internal Motivation	High	Bad Decisions	Positive
T3	22	Internal Coordination	Medium	I do in my own time	Positive
T3	23	Internal Communication	Medium	No sell	Neutral
		Internal Coordination	Medium	No notes	Neutral
		Internal Motivation	Medium	No sell	Neutral
T3	24	Internal Coordination	Medium	Where are we	Positive
T3	25	Client Coordination	Medium	Change Again	Positive
T3	26	Internal Coordination	Low	Stage Fright	Positive
T4	27	Internal Communication	High	I do what I think is needed	Positive
		Internal Coordination	High	I do what I think is needed	Positive
T4	28	Internal Communication	High	Stage Fright	Positive
T4	29	Client Communication	Medium	I know what I want but I do not know why	Positive
		Internal Motivation	Medium	I know what I want but I do not know why	Positive
T4	30	Client Communication	Medium	Client Communication	Positive
T4	31	Internal Communication	High	Where are we	Positive
		Internal Coordination	High	Where are we	Positive
T4	32	Internal Coordination	Medium	I do not belong	Negative

Team	Observation	Influence Model Variables	Severity Level	Thinklets	Outcome
		Internal Motivation	Medium	I do not belong	Negative
T4	33	Internal Coordination	Medium	I only know my belly button	Positive
T4	34	Client Coordination	Low	What problem	Positive
T4	35	Internal Motivation	Medium	I do what I am told nothing else	Neutral
T5	36	Internal Communication	Medium	I only know my belly button	Positive
		Internal Coordination	Medium	I only know my belly button	Positive
T5	37	Client Communication	Low	Client Availability	Positive
		Client Motivation	Low	Client Availability	Positive
T5	38	Internal Coordination	Medium	Where are we	Positive
T5	39	Internal Communication	High	Knowledge of a few	Positive
		Internal Coordination	High	Knowledge of a few	Positive
T5	40	Internal Communication	Medium	Last Minute Delivery	Negative
		Internal Coordination	Medium	Last Minute Delivery	Negative
T5	41	Internal Coordination	Low	I do not belong	Neutral
T5	42	Client Communication	Medium	You did what I asked but it is not what I need	Positive
		Client Coordination	Medium	You did what I asked but it is not what I need	Positive
T5	43	Internal Motivation	Low	Nobody responsible	Positive
T5	44	Client Motivation	Medium	Client Commitment	Positive
		Client Coordination	Medium	Client Commitment	Positive
T6	45	Internal Communication	Medium	Where are we	Positive
		Internal Coordination	Medium	Where are we	Positive
T6	46	Client Communication	High	I did not say that	Positive
		Internal Coordination	High	I did not say that	Positive
T6	47	Internal Coordination	Medium	Stage Fright	Neutral
T6	48	Internal Communication	High	I only know my belly button	Positive

Team	Observation	Influence Model Variables	Severity Level	Thinklets	Outcome
		Internal Coordination	High	I only know my belly button	Positive
T6	49	Client Coordination	High	I do what I think is needed	Positive
T6	50	Internal Coordination	High	Us vs. Them	Neutral
		Internal Motivation	High	Us vs. Them	Neutral
		Client Motivation	High	Us vs. Them	Neutral
T6	51	Client Motivation	Medium	I do not belong	Neutral
T7	52	Internal Communication	Medium	Meetings Absence	Positive
		Internal Coordination	Medium	Playing Dumb	Positive
T7	53	Internal Coordination	High	Team hijacking	Positive
		Internal Motivation	High	Team hijacking	Positive
T7	54	Client Communication	Medium	Them vs. Us	Positive
T7	55	Internal Communication	Medium	I only know my belly button	Positive
		Internal Coordination	Medium	I only know my belly button	Positive
T7	56	Client Communication	Medium	I do what I think is needed	Positive
		Client Coordination	Medium	I do what I think is needed	Positive
T7	57	Internal Motivation	High	Ego	Positive
				I decide	Positive
T7	58	Others	High		Positive

Looking at the outcomes it can be seen that from the 58 observations made, 46 of them had a positive outcomes (79%), 4 of them a negative outcome (7%) and 8 of them had a neutral outcome (14%). Negative outcomes were found in three different teams, neutral outcome in 5 teams, and in 2 teams we only found positives outcomes. Table 6 summarizes this.

Table 6. Obtained Results

Outcomes	Number of Observations	% of Total	Details
Positive Interventions	46	79%	2 teams had only positive outcomes
Negative Interventions	4	7%	3 teams
Neutral Interventions	8	14%	5 teams
	58	100%	

Upon analysis of the ThinkLets we see that ThinkLets were used 90 times during the research; the team that used the most ThinkLets, used 19, the team that used less ThinkLets used 10, and the overall average use of ThinkLets was almost 13. The ThinkLet most used "I only know my belly button", 7 times; "Where are we", 6 times; "I do what I think is needed", 5 times; and "Client availability", "I do not belong", "No sell", "Team hijacking" and "Us vs. Them", 4 times each.

Performing a more detailed analysis of the Negative Outcome show us that sometimes no matter what the team does or what ThinkLet is used, there are some problems that concern teams but cannot be changed unless the team is changed; for example:

- *Observation 8, Team 1* – The problem was that the client was not willing to commit to the project as they should be. The technical counterpart of the team changed and the new one did not have a clue of what the team was doing. The team asked the client for a possible users list of the software. The team used the Client Availability" ThinkLet, which consists of the practices: Trust, Retrospectives, Real Customer Involvement, Sit Together, Informative Workspace. The team carried out the tasks and insisted on it by the end of the project the client still had not delivered the possible users list and the software was delivered without a final users test.
- *Observation 11, Team 1* – One team member started to act as leader and started to make decisions for the team, but the team did not agree on the decisions made. The team used the "Bad Decisions" ThinkLet, which consists of: Trust, Planning Game, Stand Up Meetings,

Retrospectives and Decision Making. The team started to use the ThinkLet and in the beginning it helped, but as the time passed and they started to have problems with the client, the team started to become unmotivated and allowed this team member to take charge of the team again.

- *Observation 32, Team 4* – The Problem was that the team was not a team; they have two people working completely separate from the others. So they used the ThinkLet "I do not belong" which consisted of practices: Trust, Sit Together, Peer Activities and Energized Work. As the team started using this practices they were able to slightly improve the participation of one of the team members with problems. The more they pushed, the further away went the other member. Anything that the team did to include him in the team made him feels less integrated and more alone. This team member has a strong introspective and shy personality, so most efforts only served to place him at a distance.

- *Observation 40, Team 5* – The team failed to conduct an early risk evaluation before transferring from developing to production servers. Trying to correct the problem they used the ThinkLet "Last Minute Delivery", which consisted of: Planning Game, Iteration Planning, Estimating, Stand Up Meeting and Slack. The outcome was negative at the end, because the team started to use the ThinkLet too late and there was not enough time to change everything.

An analysis of a few Neutral Outcomes:

- *Observation 41, Team 5* – There is a team member that did not communicate with the team and did not express his opinion. The ThinkLet "I do not belong was used, which consisted of: Trust, Sit Together, Peer Activities, Energized Work, Team Building. The outcome was Neutral because this team member has a particular personality that does not let him express himself and be an effective team member.

- *Observation 50, Team 6* – The client hit verbally the team pretty hard, which sparked a phase of being unmotivated within the team. The team started to protect itself from further client attacks. To address this issue the team used the ThinkLet "Us vs. Them", which consisted of the practices: Vision, Real Customer Involvement and Team Building Workshops. The outcome was neutral because the team was not capable of overcoming the fear of a possible new client outburst.

7 Discussion and Expected Contributions

In this thesis two hypotheses were stated. The Hypothesis 1 states that there is a *short list of variables that systematically influences teamwork in software projects conducted by small and novice development teams (5-7 developers)*. In the proposed Influence Model (Chapter 4), three variables were considered: *Communication*, *Coordination* and *Motivation* from two different points of view: *Internal* and *with the Client*. In the experimental observations we found that the majority of issues affecting teamwork were related to these variables.

The Hypothesis 2 states that *thinklets can be used to help mitigate the recurrent situations negatively affecting the teamwork*. During the experimental observations a catalogue of the issues was developed. At the same time extensive research on them was done to try to look for patterns that had been addressed before, regardless of the context that had been previously addressed. Therefore a list of thinkLets was created along with the practices that can be used to mitigate the recurrent problem. Analysing our results we saw to see that the majority of the problems were mitigated with the use of the ThinkLets.

The results obtained up to now are well aligned with the two hypotheses raised in this thesis. It is possible to conclude that the idea of creating a framework of thinkLets is feasible to help improve teamwork in computer science teams in Academia.

However, this thesis still has some limitations, mainly regarding the context and the number of people being researched. The observations performed covered just one course of a University, and today it is well known that people from different cultures, ages, and professional fields work differently. Also the University context of the observations is different from the context of a real company. In a real company someone can be fired which is a pressure point that can be used to make people behave as they are supposed to, but in the University this pressure point does not work so smoothly.

We expect that this work will help four groups of people: students, course instructors, software industry people and the software engineering scientific community. Students often complain about the relevance that a course will have in their future working life. With that in mind, a course designed with ThinkLets, with one of its main goals being to teach teamwork through experimentation, will expose students directly to the reality of the companies and will help them learn how to deal with it.

The instructor will have the chance to teach in a less conventional way, challenging his teaching capacities and establishing a more direct and closer communication channel between students and instructor. There will be more motivated students and consequently better results among them by the end of the course.

The software industry will find professionals better prepared for teamwork. On the other hand the industry will be able to apply the thinkLets designed in this work, in order to improve teamwork inside software organizations. Finally, the software engineering scientific community will have a new tool to help train and motivate students to conduct teamwork. To the best of my knowledge, there are no reported solutions that use thinkLets to promote or enhance teamwork; therefore this thesis work proposes an innovative idea to deal with the stated problem (Jehn, Nothcraft and Neale 1999).

81

8 Conclusions and Future Work

Human factors have shown to have a great impact on most process conducted by people, an also in software development. However they are still overlooked by researchers of this area. One of the most important human centred activities involved in the software process is the teamwork.

In Computer Science, particularly in software engineering, effective teamwork can mean the difference in the outcome of a development project. Educational institutions offering Computer Science programs must accept the responsibility to prepare their graduate students not only in technical issues, but also in soft skills that allow them to work efficiently in their professional career.

Trying to address this problem we have stated in this thesis that there exists a short list of variables that systematically influence teamwork in software projects conducted by small and novice development teams. We have also stated that ThinkLets (activity or process that produces predictable results to deal with recurring collaboration problems) could be used to mitigate recurrent situations that affect the teamwork.

To do so, first we performed an extensive literature review and direct observation of several development teams of the Course CC51A: Software Engineering. Based on the results of such activities we identified a preliminary list of three variables that systematically influence teamwork: communication, coordination and motivation. It included the internal work and also with the client.

Afterward the course CC61A -Software Project was observed over two semesters in order to check how suitable the preliminary influencing variables were. Based on these observations it was possible to conclude that these variables were the most important ones, at least in the observed scenario. The most recurrent team problems were found in the literature and consequently their possible solution. The teams observed generated a list of problems and also a list of ThinkLets was created and the practices were tested. An analysis of the data observed showed that the three variables found were the most important ones and that the ThinkLets created were able to effectively mitigate the negative situations affecting teamwork.

In the author´s opinion, this thesis is just the tip of the iceberg. The challenges in the human area of software engineering are tremendous and the research being done is just beginning. This thesis did not have the intention to cover every aspect of how to enhance teamwork in software projects in the universities; it only shows a recurring fraction. However it is a beginning.

82

This work has to be extended to other Universities (possible to other cultural contexts) to try to evaluate the adherence of the ThinkLets to other instructional scenarios. The work could then be extended to evaluate the adherence in a real software company.

I hope to see this emerging area in Computer Science grow, because the major problems found in software projects today are about people. Now we have the challenge of helping people to work better as team members, so they can fully contribute in their work make better software and also to improve their work environment.

83

9 References

A.G., GECHS Softwares y Servicios Chile. www.gechs.cl. GECHS. 2010. http://bligoo.com/media/users/0/32814/files/GECHS-6o_informe_diagnostico.pdf (accessed 5 18, 2011).

ABET. ABET. 7 1, 2010. http://www.abet.org (accessed 5 17, 2011).

Abrahamsson, P., O. Salo, J. Ronkainen, and J. Warsta. Agile Software Development Methods. Technical Research Centre of Finland, Technical Report Review and Analysis, Espoo, Finland: VTT Publications, 2002, 478.

Al-Kilidar, H., P. Parkin, A. Aurum, and R. Jeffery. Evaluation of Effects of Pair Work on Quality of Designs. Software Engineering Conference. IEEE, 2005. 78-87.

Amato, C. H., and L. H. Amato. Enhancing Student Team Effectiveness: Application of Myers Briggs Personality Assessment in Business Courses. Journal of Marketing Education 21 (2005): 41-51.

Aranda, J. A Theory of Shared Understanding for Software Organizations. Doctorate Thesis. University of Toronto, 2010.

Aranda, J., S. Easterbrook, and G. Wilson. Requirements in the Wild: How Small Companies Do It. 15th IEEE Requirements Engineering Conference. IEEE, 2007. 39-48.

Bagert, D. J., T. B. Hilburn, M. Lutz, M. McCracken, and S. Mengel. Guidelines for Software Engineering Education - Version 1.0. Technical Report CMU/SEI-99-TR-032 (Citeseer), 1999.

Bareisa, E., E. Karciauskas, E. Macikenas, and K. Motiejunas. Research and Development of Teaching Software Engineering Process. International Conference in Computer Systems and Technologies. ACM, 2007. 75.

Barr, T. F., A. L. Dixon, and J. B. Gassenheimer. Exploring the Lone Wolf Phenomenon in Student Teams. Journal of Marketing Education (SAGE) 27, no. 1 (2005): 81.

Bass, B. M. Individual Capability, Team Performance and Team Productivity. Human Performance and Productivity Journal, 1980: 179-232.

84

Briggs, R. O., G. J. De Vreede, J. F. Nunamaker Jr, and D. Tobey. ThinkLets: Achieving Predictable, Repeatable Patterns of Group Interaction with Group Support Systems. 34 Annual Hawaii International Conference on System Sciences. Hawaii: IEEE, 2001. 9.

Canfora, G., A. Cimitile, F. Garcia, M. Piattini, and C. A. Visaggio. Evaluating Performances of Pair Designing in Industry. Journal of Systems and Software (Elsevier) 80, no. 8 (2007): 1317-1327.

Carbon, R., M. Lindvall, D. Muthing, and R. Costa. Integrating Product Line Engineering and Agile Methods: Flexible Design Up-front vs. Incremental Design. International Workshop on APLE. IEEE, 2006.

Carver, J., L. Jaccheri, S. Morasca, and F. Shull. Issues in Using Students in Empirical Studies in Software Engineering Education. 9th International Software Metrics Symposium (METRICS'03). JSTOR, 2003. 239-249.

Cockburn, A. A Human-Powered Methodology for Small Teams. Addison Wesley, 2004.

Chong, J., and T. Hurlbutt. The Social Dynamics of Pair Programming. ICSE'07 - International Conference of Software Engineering. IEEE, 2007.

Cohn, M. Succeeding with Agile: Software Development Using Scrum. Boston: Addison Wesley, 2009.

Cohn, M. User Stories Applied: for Agile Software Development. Addison Wesley, 2004.

Demirors, E., G. Sarmasik, and O. Demirors. The Role of Teamwork in Software Development Microsoft Case Study. 23rd Euromicro Conference: New Frontiers of Information Technology. IEEE, 1997. 129-133.

Denning , P. J. Educating a New Engineer. Communications of the ACM, 12 1992: 82-97.

Dixon, A. L., and J. B. Gassenheimer. Identifying the Lone Wolf: A Team Perspective. Journal of Personal Selling and Sales Management, no. 23 (2003): 205-219.

Dommeyer, C. J. Using the Diary Method to Deal with Social Loafers on the Group Projects: Its Effects on Peer Evaluations. Group Behaviour and Attitudes. Journal of Marketing Education, no. 29 (2007): 175-188.

Eisenhardt, K. M., and C. Schoonhoven. Organizational Growth: Linking Founding Team, Strategy, Environment and Growth among US. Administrative Science Quarterly 35, no. 3 (1990): 504-529.

Fraser, G., and F. Wotawa. Test-Case Prioritization with Model-Checkers. International Multi Conference: Software Engineering. ACTA, 2007. 267--272.

Gehrke, M., et al. Reporting About Industrial Strength Software Engineering Courses for Undergraduates. 24th International Conference on Software Engineering. ACM, 2002. 395-405.

George, B., and L. Williams. An Initial Investigation of Test Driven Development in Industry. ACM Symposium on Applied Computing. ACM, 2003. 1135-1139.

Giraldo, F. D., C. Z. Collazos, S. F. Ochoa, and S. Zapata. Teaching Software Engineering from a Collaborative Perspective: Some Latin-American Experiences. 2010 Workshops on Database and Expert Systems Applications. IEEE, 2010. 97-110.

Gladstein, D. L. Groups in Context: a Model of Task Group Effectiveness. Administrative Science Quarterly (JSTOR) 29 (1984): 499-517.

Goodpasture, J. C. Project Management the Agile Way: Making It Work in the Enterprise. J. Ross Publishing, 2009.

Gorla, N., and Y. W. Lam. Who Should Work with Whom? Building Effective Software Project Teams. Communications of the ACM, 2004: 79-82.

Hackman, J. R. Groups that Work (and those that Don't): Creating conditions for effective teamwork. San Francisco: Jossey-Bass, 1990.

Hackman, J. R., and R. Wageman. A Theory of Team Coaching. Academy of Management Review (JSTOR) 30 (2005): 269-287.

Hawthorne, M., and E. Dewayne. Software Engineering Education in the Era of Outsourcing, Distributed Development and Open Source Software: Challenges and Opportunities. 27th International Conference on Software Engineering (ICSE). Saint Louis: ACM, 2005. 633-644.

Hayes, S., and M. Andrews. An Introduction to Agile Methods. Pace University. 2006. http://csis.pace.edu/~marchese/CS616/Agile/IntroToAgileMethods.pdf (accessed 09 06, 2011).

Hernandez, J. Universidad Adolfo Ibañez. Escuela de Psicologia. Work Under Pressure as Labor Mobilizing (In Spanish). 2010. http://www.uai.cl/mundo (accessed 10 17, 2010).

Hibbs, C., S. Jewett, and M. Sullivan. The Art of Lean Software Development: A Practical and Incremental Approach. Boston: O'Reilly, 2009.

Hietala, P., K. Koivunen, and E. Ropo. Analysis of Student Decision-Making in Online Collaboration . Journal of Information Technology Impact (Loyola) 4, no. 2 (2004): 99-120.

Hilburn, T. B., and D. J. Bagert. A Software Engineering Curriculum Model. ASEE Annual Conference. IEEE, 1999. 12A4-6.

Hoegl, M., and H. G. Gemuenden. Teamwork Quality and the Success of Innovative Projects: A Theoretical Concept and Empirical Evidence. Organization Science (INFORMS) Vol 12, no. 4 (2001): 435-449.

Humphrey, W. S. Managing Technical People: Innovation, Teamwork and the Software Process. Boston: Addison-Wesley Longman Publishing Co., 1996.

87

Humphrey, W., and W. Tomas. Reflection on Management: How to Manage your Software Projects, your Teams, your Boss and Yourself. Boston: Pearson Education, 2010.

Huo, M., J. Verner, L. Zhu, and M. A. Babar. Software Quality and Agile Methods. Computer Software and Applications Conference. IEEE, 2004. 520-525.

Iacoca, L., and W. Novak. Iacoca: An autobiography. New York: Bantam Books, 1984.

Jazayeri, M. The Education of a Software Engineer. 19th Automated Software Engineering Conference. IEEE, 2004. 16-27.

Jehn, K. A., G. B. Nothcraft, and M. A. Neale. Why Differences Make a Difference: A Field Study of Diversity, Conflict and Performance in Workgroups. Administrative Science Quarterly (JSTOR) 44, no. 4 (1999): 741-763.

Kapp, E. Improving Student Teamwork in a Collaborative Project Based Course. (Heldref Publications) 57, no. 3 (2009): 139-143.

Karhatsu, H., M. Ikonen, P. Kettunen, F. Fagerholm, and P. Abrahamsson. Building Blocks for Self-Organizing Software Development Teams a Framework Model and Empirical Pilor Study. 2nd International Conference on Software Technology and Engineering (ICSTE). IEEE, 2010. 290-297.

Kniberg, H., and M. Skarin. Kanban and Scrum: Making the Most of Both. InfoQ, 2010.

Kolfschoten, G. L., R. O. Briggs, G. J. De Vreede, P. Jacobs, and J. H. Appelman. A Conceptual Foundation of the Thinklet Concept for Collaborating Engineering. International Journal of Human-Computer Studies (Elsevier) 64, no. 7 (2006): 611-621.

Korkala, M., P. Abrahamsson, and P. Kyllonen. A Case Study of the Impact of Customer Communication on Defects in Agile Software Development. Agile Conference. IEEE, 2006.

Koslowski, S. W., and D. R. Ilgen. Enhancing the Effectiveness of Work Groups and Teams. Psychological Science in the Public Interest (SAGE) 7, no. 3 (2006): 77-124.

Kozlowski, S. W. J., and B. S. Bell. Work Groups and Teams in Organizations. Handbook of Psychology (Wiley) 12 (2003): 333-375.

Larman, C. Agile & Iterative Development - A Manager's Guide. Boston: Addison Wesley, 2007.

Lingard, R. W. Teaching and Assessing Teamwork Skills in Engineering and Computer Science. Journal of Systemics, Cybernetics and Informatics 18, no. 1 (2010): 34-37.

Lutz, B. Linguistic Challenges in Global Software Development: Lessons Learned in an International SW Development Division. Global Software Engineering. IEEE, 2009. 249-253.

Mathieu, J., M. T. Maynard, T. Rapp, and L. Gilson. Effectiveness Effectiveness 1997-2007: A Review of Recent Advancements and a Glimpse into the Future. Journal of Management (SAGE) 34, no. 3 (2008): 410.

McDonough III, E. F., and D. Cedrone. Meeting the Challenge of Dispersed Team Management. Research & Technology Management 43 (July-August 2000): 12-17.

Meister, D. Behavioural Foundations of System Development. 2nd Edition. J. Wiley, 1985.

Morgenson, F. P., L. D. Aiman-Smith, and M. A. Campion. Implementing Work Teams: Recommendations From Organizational Behaviour and Development Theories. Advances in Interdisciplinary Studies if Work Teams. 4 (1997): 1-44.

Noor, M. A., P. Grunbacher, and R. O. Briggs. A Collaborative Approach for Product Line Scoping: A Case Study in Collaborative Engineering. 25th International Multi-Conference Software Engineering. Innsbruck: Springer, 2007. 69-83.

Nordberg III, M. E. Managing Code Ownership. IEEE Software (IEEE), 2003: 26-33.

Page, D., and J. G. Donelan. Team-Building Tools for Students. The Journal of Education for Business 78, no. 3 (2003): 125-128.

Paris, C. R., E. Salas, and J. A. Cannon-Bowes. A Teamwork in Multiperson Systems: a Review and Analysis. Ergonomics (Taylor & Francis) 43, no. 8 (2000): 1052-1075.

Parker, G., and R. Hoffman. Meeting Excellence: 33 Tools to Lead Meetings that Get Results. Jossey-Bass, 2006.

Patit, J. M., and D. Wilemon. Creating High-Performing Software Development Teams. R&D Journal (Wiley Online Library) 35, no. 4 (2005): 375-393.

Pfaff, E., and P. Huddleston. Does it Matter if I Hate Teamwork? What Impacts Student Attitudes Toward Teamwork. Journal of Marketing Education 25, no. 1 (2003): 37.

Rising, L., and E. Derby. Singing the Songs of Project Experience: Patterns and Retrospectives. The Journal of Information Technology Management 16, no. 9 (2003).

Robillard, P. N. Teaching Software Engineering Through a Project-Oriented Course. 10th CSEE. Virginia Beach: IEEE, 1996. 85.

Ruhe, G., and M. O. Saliu. The Art and Science of Software Release Planning. IEEE Software (IEEE), 2005: 47-53.

Ryan, R. M., and E. L. Deci. Intrinsic and Extrinsic Motivations: Classic Definitions and New Directions. Contemporary Educational Psychology (Elsevier) 25, no. 1 (2000): 54-67.

Safizadeh, M. H. The Case of Workgroups in Manufacturing Operations. California Management Review 33 (1991): 61-82.

Salas, E., K. C. Stagl, and C. S. Burke. 25 Years of Team Effectiveness in Organizations: Research Themes and Emerging Needs. International Review of Industrial and Organizational Psychology 19 (2004): 47-91.

Schweiger, D. M., and W. R. Sandberg. The Utilization of Individual Capabilities in Group Approaches to Strategic Decision Making. Strategic Management Journal (Wiley Online) 10 (1989): 31-43.

Schweiger, D., and W. Sandberg. The Utilization of Individual Capabilities in Group Approaches to Strategic Decision Making. Strategic Management Journal (Willey Online) 10, no. 1 (1989): 31-43.

Sfetsos, P., L. Angelis, and I. Stamelos. Investigating the Extreme Programming System - An Empirical Study. Empirical Software Engineering (Springer), 2006: 269-301.

Sharp, H., H. Robinson, and M. Petre. The Role of Physical Artefacts in Agile Software Development: Two Complimentary Perspectives. Interacting with Computers (Elsevier) 21, no. 1 (2009): 108-116.

Shenhar, A. J., and D. Dvir. Long and Short Term Success Dimensions in Technology-Based Organizations. Handbook of Technology Management (McGraw Hill), 1996: 32.1-32.15.

Shore, J., and S. Warden. The Art of Agile Development. 2nd. Edition. O'Reilly, 2008.

Simmons, D. B. Software Engineering Education in the New Millennium. 30th Annual International Computer Software Applications Conference - COMPSAC'06. IEEE PRESS, 2006. 46-47.

Teasley, S. D., L. A. Covi, M. S. Krishman, and J. S. Olson. Rapid Software Development Through Team Collocation. Transactions on Software Engineering (IEEE) 28 (2002): 671-683.

Tripp, L. IEEE Standards Collection: Software Engineering Standard 610. Standard, Institute of Electrical and Electronic Engineers, IEEE, 1994.

Trytten, D. A. A Design for Team Peer Code Review. SIGCSE. ACM, 2005. 455-459.

Tvedt, J. D., R. Tesoriero, and K. A. Gary. The Software Factory: Combining Undergraduate Computer Science and Software Engineering Education. 23rd International Conference on Software Engineering. IEEE, 2001. 633-642.

Van Eerd, W. Procrastination: Self-Regulation in Initiating Aversive Goals. Applied Psychology (Wiley Publisher) 49, no. 3 (2000): 372-389.

Wellington, C. A., T. Briggs, and C. D. Girard. Examining Team Cohesion as an Effect of Software Engineering Methodology. 2005 Workshop on Human and Social Factors of Software Engineering. ACM, 2005. 1-5.

Wellington, C. A., T. Briggs, and C. D. Girard. Examining Team Cohesion as an Effect of Software Engineering Methodology. Workshop on Human and Social Factors of Software Engineering. ACM, 2005. 1-5.

Whitten, N. Managing Software Development Projects. New York: John Wiley & Sons, 1995.

Wikipedia. Wikipedia. 4 21, 2011. http://en.wikipedia.org/wiki/Teamwork (accessed 5 17, 2011).

Williams, L., R. R. Kessler, W. Cunningham, and R. Jeffries. Strengthening the Case for Pair-Programming. Software IEEE (IEEE) 17, no. 4 (2000): 19-25.

Wohlin, C., P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Wesslen. Experimentation in Software Engineering. Norwell: Kluwer Academic Publishers, 2000.

Zika-Viktorsson, A., and A. Ingelgard. Reflecting Activities in Product Developing Teams: Conditions for Improved Project Management Process. Research in Engineering Design (Springer) 17, no. 2 (2006): 103-111.

Appendix

In this Appendix I will describe in more detail how the Observations were done in the course CC61A – Software Project.

The teams have an assigned time and room to do the meetings, each team had a software engineering instructor and the course had a professor who supervised everything. All the meetings happened in the same room and at the same time. The software-engineering instructors are professors or professionals with large experience in software engineering projects; the majority of them are used to use the agile approach for developing software. Their experience in projects is used to help the teams to reach the goal of finishing successfully their project.

A colleague and I did all the observations, we divided the teams according to their position, and normally each one of us had two different teams to observe. We always sited in a certain distance of the teams and refrained ourselves of participating in any of the meetings, our position was always of listening and taking notes, since any intervention from us could affect the development of the team. We took online notes from the meetings recording their behavior towards a topic, the problems they found, the observations made by the instructors and their response to that; in short we took note of everything they did in this 1.5-hour meetings they had every week.

During these meetings the software engineering instructor always started asking them how was the week, what were the problems they found and what happened with the problems they discussed the previous meetings. The instructor sometimes asked the team what they thought about the impact that a particular problem could have on their project, so this information became our severity level. When he asked about past problems, they always asked if it was solved, if they found a solution or if they used the solution he proposed, so this information was our outcome. And when a problem was identified by the instructor or by the team, they were considered a thinkLet.

92