Using Thinklets to Support the Team Work in Novice Software Teams

Maíra Marques, Sergio Ochoa
Computer Science Department, University of Chile
mmarques@dcc.uchile.cl, sochoa@dcc.uchile.cl

Abstract—This technical report presents in detail a set of ThinkLets that can be used to improve teamwork in computer science students teams. The ThinkLets reported here are focused on dealing with the three major problems that a student team can face: communication, coordination and motivation. The main idea is to find an easy "recipe" (i.e. a ThinkLet) that can be used to address recurrent students problems when facing a team project. The ThinkLets were used and validated in an academic scenario, and the obtained results are being published in a scientific article.

1. Introduction

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.

After monitoring the software development in the academia for many years, the authors think that thinkLets could be used to mitigate recurrent situations that affect teamwork. These thinkLets are activities or processes that produce predictable results to deal with recurring collaboration problems [Kolf06]. The empirical observation indicates that the team work is affected by recurrent communication, coordination and motivation problems. After identifying these problems we developed and evaluated a list of thinkLets, which was designed to deal with these problems.

2. ThinkLets

According to Briggs [Brig01] 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 generates a completely new thinklet. Moreover Kolfschoten et al. [Kolf06] 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 that 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.

In the section below the thinkLets are presented in a schematic that shows the thinkLet name, the recurring problem that the thinkLet deals with, corrective actions that can be used and useful practices to mitigate the recurring problems. 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. A few of them were created by the authors, grouping concepts and ideas from different areas, such as Peer Activities, Decision Making and Public Profile. These practices are described in detail in the Annex A.

3. ThinkLets for Communication

This section presents a list of thinklets that deal with recurring communication problems. Subsection 3.1 is focused on internal communication problems and subsection 3.2 is focused on external communication (i.e. with the user/client).

3.1. 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.

<table>
<tr><td>ThinkLet:
Stage fright</td><td colspan="2">Recurring Problem: 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 [Hack90].</td></tr>
<tr><td colspan="2">Corrective Actions: Weekly meetings, where all team members must report their work status (Round-Robin Meeting). According to Humphrey [Hump96] 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 activity. Kapp [Kapp09] 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.</td><td>Useful Practices:

o Stand Up Meeting
o Trust
o Sit Together
o Team-Building Workshop
o Coaching</td></tr>
</table>

ThinkLet: Playing dumb	**Recurring Problem:** 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 [Pfaf3].

Corrective Actions: 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 [Whit95].

According to Page and Donelan [Page03], one solution can be to begin a project with Team-Building Workshop. These techniques should have the ability to analyse 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 [Amat05] suggest conducting personality tests, such as the MBTI (Myers-Brigg Type Indicator) prior to team formation, to enhance the team effectiveness.

Useful Practices:

- Trust
- Sit Together
- Peer Activities
- "Done, Done"
- Collective Code Ownership
- Public Profile
- Team-Building Workshop

ThinkLet: Team hijacking	**Recurring Problem:** 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 discussion/validation/ generation of ideas cannot be done because there is not enough knowledge in just one person (or a few). Pfaff y Huddleston [Pfaf03]) defined this kind of behaviour as a "team hijacking"

Corrective Actions: 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 team members must be an expert on the subject. According to Karhatsu et al. [Karh10] 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.

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 [Page03] team members have to know and take responsibility in the development of their own team, always trying to increase their capacity for accomplishment.

Perform research or a spike solution lead by an expert. Pfaff y Huddleston [Pfaf03] talk about constantly controlled interventions as compelling ways to ensure effective practices.

According to Page y Donelan [Page03] 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.

Useful Practices:

- Peer Activities
- Collective Code Ownership
- Vision
- Planning Game
- Ubiquitous Language
- Spike Solution

ThinkLet: Why bother to answer	**Recurring Problem:** 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.
Corrective Actions: Make the team behave towards written communication in order to maintain good levels of persistence and traceability. On a daily basis team members must access the official communication system of the team. This policy has to 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 [Cohn09]. For example: o FYI (No need to answer). o Low Importance (Answer needed within the week). o Important (Answer needed within 48 hours). o Very Important (Answer needed within 24 hours). o URGENT (Answer needed within 3 hours).	**Useful Practices:** o Peer Activities o Collective Code Ownership o Trust o Coaching

ThinkLet: Ego	**Recurring Problem:** Members of the team with strong personalities (ego) engage in conflict for some specific subject of the project. 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 [Eise90] some task conflicts could bring benefits to the project development. Jehn et al. [Jehn99] 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.
Corrective Actions: 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 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. [Cohn09] 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. 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 [Fras07].	**Useful Practices:** o Trust o Sit Together o Planning Game o Peer Activities o Coding Standards o Ubiquitous Language o Collective Code Ownership

ThinkLet: I do not belong	**Recurring Problem:** 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.

Corrective Actions: 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 [Barr05]. 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. Organize social activities or a shared meal [Shor08].	**Useful Practices:** o Trust o Sit Together o Peer Activities o Energized Work o Team-Building Workshop

ThinkLet: Free Riders	**Recurring Problem:** 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 [Hump10] says "*nothing can be more disruptive than to have some people in a group openly getting away with something*". Dommeyer [Domm07] 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*".

Corrective Actions: 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 knows that his action brings consequence to the whole team. The team can threaten to kick the free rider from the team. Humphrey [Hump10] 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 behavior in the academia is not so severe. Dixon and Gassenheimer [Dixo03] suggests providing students opportunities to familiarize themselves with their group members in a social context, away from the high pressure of the academic setting.	**Useful Practices:** o Peer Review o Team-Building Workshop o Root Cause Analysis

3.2. 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.

ThinkLet: Client availability	**Recurring Problem:** A client having real availability to communicate with the team is a difficult goal to reach. This 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.

Corrective Actions: To establish a fixed day and time of a weekly meeting with the client, prior to the beginning of the project. It ensures 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: o Low Importance (Project Status and simple decisions that the technical counterpart can easily decide). o Important (Project Status and complex decisions where decisions would be better made by someone else than the counterpart). o Critical (Project Status and critical decisions where it must be someone else other than the counterpart to do).	**Useful Practices:** o Trust o Retrospectives o Real Customer Involvement o Sit Together o Informative Workspace o Team-Building Workshop o Kanban

ThinkLet: Client communication	**Recurring Problem:** The client is not able to effectively communicate with the team, so he/she is not able to transmit his real needs to 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.

Corrective Actions: 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 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.	**Useful Practices:** o Vision o Planning Game o Stories o Estimating o Real Customer Involvement o Sit Together o Ubiquitous Language

ThinkLet: You did what I asked, but is not what I need	**Recurring Problem:** 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.

Corrective Actions: 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.	**Useful Practices:** o Planning Game o Iteration Planning o Reporting o Slack o Stories o Estimating o Real Customer Involvement o Sit Together o Exploratory Testing

ThinkLet: I know what I want, but I do not know why	**Recurring Problem:** 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.
Corrective Actions: 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.	**Useful Practices:** o Vision o Planning Game o Stories o Real Customer Involvement

ThinkLet: I do not know what I want	**Recurring Problem:** Unjustified and constant request changes frustrate the team since it can change the project goal and / or the scope of the project
Corrective Actions: At the beginning of the project the Vision of the project should be well 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.	**Useful Practices:** o Vision o Real Customer Involvement o Iteration Planning o Estimating

ThinkLet: I did not say that	**Recurring Problem:** The client and / or the team do not recognize the agreement reached in the meeting.

Corrective Actions: 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.	**Useful Practices:** o Meeting Minutes o Ubiquitous Language o Team-Building Workshop.

4. Coordination

This section shows a list of thinkLets that deal with recurring coordination problems. Subsection 4.1 is focused on internal coordination problems and subsection 4.2 is focused on external coordination problems.

4.1. 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.

ThinkLet: Lone wolf	**Recurring Problem:** 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 [Barr05]. They also define lone wolves as people highly committed, who devote a lot of energy to complete tasks
Corrective Actions: Be assured 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 [Barr05], "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".	**Useful Practices:** o Trust o Sit Together o Ubiquitous Language o Stand Up Meetings o Iteration Planning o Public Profile o Coaching

ThinkLet: Knowledge of a few	**Recurring Problem:** 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.

Corrective Actions: 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).	Useful Practices: o Peer Activities o Sit Together o Stand Up Meetings o Coding Standards

ThinkLet: Meetings absence	**Recurring Problem:** 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 [Iaco84]: "*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 unrevealed*".
Corrective Actions: 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.	**Useful Practices:** o Trust o Energized Work o Stand Up Meetings o Team-Building Workshop o Feedback

ThinkLet: I only know my own belly button	**Recurring Problem:** 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 Humphrey [Hump10] "*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*".
Corrective Actions: 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 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.	**Useful Practices:** o Stand up Meeting o Informative Workspace o Kanban o Public Profile

ThinkLet: I do what I think is needed	**Recurring Problem:** 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 [Robi96].

Corrective Actions: 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.	**Useful Practices:** o Informative Workspace o Stand-up Meeting o No Bugs o Collective Code Ownership o Continuous Integration o Test driven development o Refactoring o Performance optimization o Kanban

ThinkLet: Last minute delivery	**Recurring Problem:** Team members do their work but they deliver their work just before the deadline. This is a problem 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.
Corrective Actions: When the project starts, the estimation technique that will be used has to be defined. The estimation technique chosen 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 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.	**Useful Practices:** o Planning Game o Iteration Planning o Estimating o Stand up Meeting o Slack

ThinkLet: I do in my own time	**Recurring Problem:** 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.
Corrective Actions: 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 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.	**Useful Practices:** o Planning Game o Iteration Planning o Estimating o Stand up Meeting

ThinkLet: No notes	**Recurring Problem:** 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.
Corrective Actions: 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.	**Useful Practices:** o Planning Game o Stories o Real Customer Involvement o Meeting Minutes

ThinkLet: Where are we	**Recurring Problem:** 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 loose commitment to their members, resulting in lost in productivity and quality.
Corrective Actions: 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 the project the acceptance criteria of the project, and as soon as the requisites are defined, the client must prioritize them.	**Useful Practices:** o Planning Game o Iteration Planning o Release Planning o Informative Workspace o Kanban

ThinkLet: Paralysis analysis	**Recurring Problem:** The team is frozen, waiting for possible solutions to a problem, or trying to analyse and learn new technologies (paralysis by analysis). This halt of the team can affect the time agreements.

Corrective Actions: Define in advance the amount of time that can be spent in analysis. For example: o 1st. Day Web search, tutorials: If the search is positive, one day of Spike Solution / Spin Off. o 2nd. Day: Try to make contact with someone that is an expert on the subject. o 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.	**Useful Practices:** o Test Driven Development o Refactoring o Spike Solutions o Decision Making o Coaching

ThinkLet: Why to decide	**Recurring Problem:** 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 the project.
Corrective Actions: 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 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.	**Useful Practices:** o Peer Activities o Root Cause Analysis o Retrospectives o Stand Up Meetings o Decision Making o Coaching

ThinkLet: I decide	**Recurring Problem:** The team has members who make important decisions alone, without properly analyzing 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

Corrective Actions: 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: o 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. o Important (the decision to be made affects other team members' tasks) – the team has to make the decision. o Critical (the decision transversally affects the whole project) – the decision must be made by the team together with the client.	**Useful Practices:** o Incremental Design and Architecture o Peer Activities o Retrospectives o Decision Making o Simple Design

ThinkLet: No sell	**Recurring Problem:** 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.
Corrective Actions: 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.	**Useful Practices:** o Real Customer Involvement o Customer Reviews o Customer Testing o Iteration Planning o Iteration Demo o Ten minutes build o Feedback o Version Control

ThinkLet: "I" not "us"	**Recurring Problem:** 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.
Corrective Actions: 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 [Barr05]. Extra project meetings, social meetings, sharing a meal. The idea is to let team members bond freely. According to [Shor08]: "Something about sharing a meal breaks down barriers and fosters team cohesiveness".	**Useful Practices:** o Energized Work o Peer Activities o Retrospectives o Sit Together o Team-Building Workshop o Peer Review

ThinkLet: Nobody responsible	**Recurring Problem:** There is no coordination, since nobody is responsible for anything.
Corrective Actions: 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	**Useful Practices:** o Coaching o Team-Building Workshop

4.2. 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.

ThinkLet: I did not asked that	**Recurring Problem:** 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.
Corrective Actions: 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 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	**Useful Practices:** : o Planning Game o Stories o Customer Review o Real Customer Involvement o Meeting Minutes

ThinkLet: What problem	**Recurring 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.
Corrective Actions: 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 Leafhopper. 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.	**Useful Practices:** : o Vision o Planning Game o Stories o Iteration Demo

ThinkLet: Client decision	**Recurring Problem:** 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.
Corrective Actions: 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 [Cohn09]. 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).	**Useful Practices:** o Real Customer Involvement o Informative Workspace o Sit Together o Planning Game o Stories o Kanban

ThinkLet: Change again	**Recurring Problem:** 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.
Corrective Actions: 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.	**Useful Practices:** : o Planning Game o Stories o Estimating o Risk Management

ThinkLet: Client meeting	**Recurring Problem:** It is difficult to establish a meeting with clients.
Corrective Actions: 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. Have a Team-Building Workshop with the client to show them the importance of their active participation on the project.	**Useful Practices:** o Real Customer Involvement o Sit Together o Planning Game o Team-Building Workshop

5. Motivation

This section presents a list of thinkLets that deal with recurring motivation problems. Subsection 5.1 is focused on internal motivation problems and subsection 5.2 is focused on external motivation (i.e. with the user/client).

5.1. 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.

ThinkLet: I do what I'm told	**Recurring Problem:** 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.
Corrective Actions: 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 [Hump10] 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". [Hump10]. 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.	Useful Practices: o Trust o Stand Up Meetings o Sit Together o Energized Work o Retrospectives o Coaching

ThinkLet: Bad decisions	**Recurring Problem:** 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.
Corrective Actions: 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.	**Useful Practices:** o Trust o Planning Game o Stand Up Meetings o Retrospectives o Decision Making

ThinkLet: Us vs. Them	**Recurring Problem:** Team members are unmotivated because of the constant friction between them and the client or within the team.

Corrective Actions: Remind the team and the client of the vision and relevance of the project. According to Humphrey [Hump10] 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.	**Useful Practices:** o Vision o Real Customer Involvement o Team-Building Workshop o Coaching

5.2. 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.

ThinkLet: Client commitment	**Recurring Problem:** 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.
Corrective Actions: 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 [Cohn09]. 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.	**Useful Practices:** o Real Customer Involvement o Sit Together o Planning Game o Stories o Iteration Demo o Team-Building Workshop

6. Conclusions

In this technical report we presented a list of thinkLets that can be used to deal with recurring communication, coordination and motivation problems, that software engineering teams face during software projects. All these thinkLets have been used in practice and the results are being reported in a scientific paper.

7. References

[Abra02] 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.

[Alki05] 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.

[Amat05] 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.

[Aran10] Aranda, J. A Theory of Shared Understanding for Software Organizations. Doctorate Thesis. University of Toronto, 2010.

[Barr05] 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.

[Brig01] 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.

[Carb06] 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.

[Canf07] 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.

[Chon07] Chong, J., and T. Hurlbutt. The Social Dynamics of Pair Programming." ICSE'07 - International Conference of Software Engineering. IEEE, 2007.

[Cock04a] Cockburn, A. A Human-Powered Methodology for Small Teams. Boston: Addison Wesley, 2004.

[Cohn04b] Cohn, M. User Stories Applied: for Agile Software Development. Addison Wesley, 2004.

[Cohn09] Cohn, M. Succeeding with Agile: Software Development Using Scrum. Boston: Addison Wesley, 2009.

[Cohn09] Cohn, M. User Stories Applied: for Agile Software Development. Addison Wesley, 2004.

[Dixo03] 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.

[Donm07] 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.

[Eise90] 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.

[Frase07] Fraser, G., and F. Wotawa. Test-Case Prioritization with Model-Checkers. International Multi Conference: Software Engineering. ACTA, 2007. 267-272.

[Geor03] George, B., and L. Williams. An Initial Investigation of Test Driven Development in Industry. ACM Symposium on Applied Computing. ACM, 2003. 1135-1139.

[Good09] Goodpasture, J. C. Project Management the Agile Way: Making It Work in the Enterprise. J. Ross Publishing, 2009.

[Hack90] Hackman, J. R. Groups that Work (and those that Don't): Creating conditions for effective teamwork. San Francisco: Jossey-Bass, 1990.

[Hack05] Hackman, J. R., and R. Wageman. A Theory of Team Coaching. Academy of Management Review (JSTOR) 30, 2005, 269-287.

[Haye11] 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).

[Hibb09] Hibbs, C., S. Jewett, and M. Sullivan. The Art of Lean Software Development: A Practical and Incremental Approach. Boston: O'Reilly, 2009.

[Hiet04] 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.

[Huo04] Huo, M., J. Verner, L. Zhu, and M. A. Babar. Software Quality and Agile Methods. Computer Software and Applications Conference. IEEE, 2004, 520-525.

[Hump10] Humphrey, W., and W. Tomas. Reflection on Management: How to Manage your Software Projects, your Teams, your Boss and Yourself. Boston: Pearson Education, 2010.

[Hump96] Humphrey, W. S. Managing Technical People: Innovation, Teamwork and the Software Process. Boston: Addison-Wesley Longman Publishing Co., 1996.

[Iaco84] Iacocca, L., and W. Novak. Iacocca: An autobiography. New York: Bantam Books, 1984.

[Jehn99] 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.

[Kapp09] Kapp, E. Improving Student Teamwork in a Collaborative Project Based Course. (Heldref Publications) 57, no. 3, 2009, 139-143.

[Karh10] 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.

[Knib10] Kniberg, H., and M. Skarin. Kanban and Scrum: Making the Most of Both. InfoQ, 2010.

[Kolf06] 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.

[Kork06] 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.

[Larm07] Larman, C. Agile & Iterative Development - A Manager's Guide. Boston: Addison Wesley, 2007.

[Lutz09] Lutz, B. Linguistic Challenges in Global Software Development: Lessons Learned in an Development Division. Global Software Engineering. IEEE, 2009, 249-253.

[Math08] 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.

[Nord03] Nordberg III, M. E. Managing Code Ownership. IEEE Software (IEEE), 2003, 26-33.

[Page03] Page, D., and J. G. Donelan. Team-Building Tools for Students. The Journal of Education for Business 78, no. 3, 2003, 125-128.

[Park06] Parker, G., and R. Hoffman. Meeting Excellence: 33 Tools to Lead Meetings that Get Results. Jossey-Bass, 2006.

[Pati05] Patit, J. M., and D. Wilemon. Creating High-Performing Software Development Teams. R&D Journal (Wiley Online Library) 35, no. 4, 2005, 375-393.

[Pfaf03] 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.

[Risi03] Rising, L., and E. Derby. Singing the Songs of Project Experience: Patterns and Retrospectives. The Journal of Information Technology Management 16, no. 9, 2003.

[Robi96] Robillard, P. N. Teaching Software Engineering Through a Project-Oriented Course. 10th CSEE. Virginia Beach: IEEE, 1996. 85.

[Ruhe05] Ruhe, G., and M. O. Saliu. The Art and Science of Software Release Planning. IEEE Software (IEEE), 2005, 47-53.

[Schw89] 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.
[Sfet06] Sfetsos, P., L. Angelis, and I. Stamelos. Investigating the Extreme Programming System - An Empirical Study. Empirical Software Engineering (Springer), 2006, 269-301.
[Shar09] 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.
[Shor08] Shore, J., and S. Warden. The Art of Agile Development. 2nd. Edition. O'Reilly, 2008.
[Teas02] Teasley, S. D., L. A. Covi, M. S. Krishman, and J. S. Olson. Rapid Software Development Through Team Collocation. Transactions on Software Engineering, no. 28, 2002, 671-683.
[Tryt05] Trytten, D. A. A Design for Team Peer Code Review. SIGCSE. ACM, 2005. 455-459.
[Will00] Williams, L., R. R. Kessler, W. Cunningham, and R. Jeffries. Strengthening the Case for Pair-Programming. Software IEEE 17, no. 4, 2000, 19-25.
[Whit95] Whitten, N. Managing Software Development Projects. New York: John Wiley & Sons, 1995.

Annex A: List of Recommended Practices

This section presents a list of practices that can be used to resolve some problems a team may face during a project. We will borrow a definition of practices used by Aranda [Aran10] that says that practices "*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 distinct 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. We have 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, we have to agree with Aranda [Aran10] "*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

Figure 1 - Recommended Practices

or avoid technical problems and bugs. Figure 1 shows how many practices there are in which one of the practices categories created. The next 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. 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	
	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	

	Practices	Communication	Coordination	Motivation
	Incremental Design and Architecture		X	
	Performance Optimization		X	
	Customer Testing		X	
	Customer Reviews		X	
	Exploratory Testing		X	

1.1 Thinking Practices

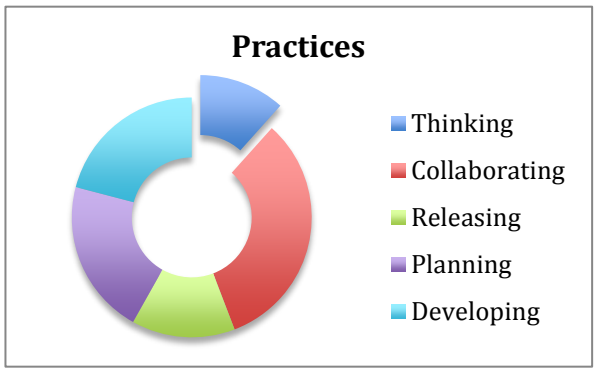

Figure 2 - Thinking Practices

Peer Activities. This is everything that can be done in pairs during a software project and that has already been 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 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 [Chon07] 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 other's screen easily, but work on their own assignments [Cock04] .

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 different 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 [Canf07]. According to Al-Kilidar et al. [Alki05] and Canfora et al. [Canf07] pair design empirically has higher quality than solo design quality, regarding: functionality, usability, portability and maintainability.

Pair Analysis. According to Williams et al. [Will00] 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 [Tryt05]: "*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 every day 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 [Shor08] 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 shares', 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 [Shar09] concluded that "*the Wall of the Informative Workspace is an artefact of significance and meaning to developers. The Wall shapes, 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 [Shor08]. 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 [Risi03] "*retrospectives provide a wonderful opportunity for capturing knowledge as patterns to improve team knowledge and team cohesion*". It is also known as Reflection Workshop [Cock04].

1.2 Collaborating Practices

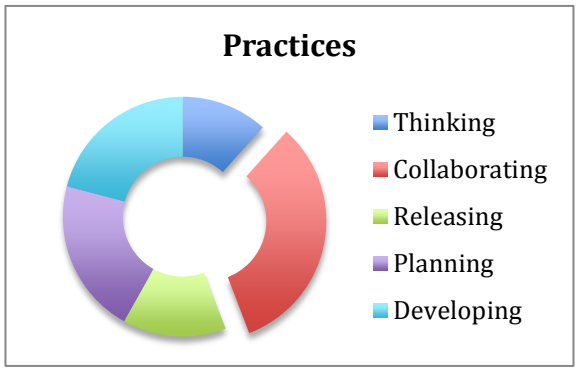

Figure 3 - Collaborating Practices

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 [Shor08]; 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 [Shor08]:

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 [Shor08].

Programmer-Tester Empathy – When this kind of problem occurs programmers tend to not 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 [Shor08]:

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 together and also the client. It requires wide and open spaces, where there are 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. [Teas02] 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. [Kork06] 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 (RCI) practice being used was two times greater than the worst case that

utilized RCI.

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 [Haye06] 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 [Larm07] "*It supports openness and allows resolution of dependencies and conflicts in real time to maximize throughput*".

Coding Standard. It is the use of development guidelines that team members should follow 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. [Sfet06] 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, 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. [Hibb09] stated that "*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 [Shor08] 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 [Kapp09] 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" [Park06] - 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 [Pati05] "*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 [Hack05] 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 [Knib10] 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 [Schw89] 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. [Hiet04] "one way to structure decision-making".

According to Schweiger and Sandberg [Schw89], 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 [Amat05] 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 [Pati05]. 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. [Math08] Feedback has a positive impact on motivation, interpersonal trust and ultimately performance.

1.3 Releasing Practices

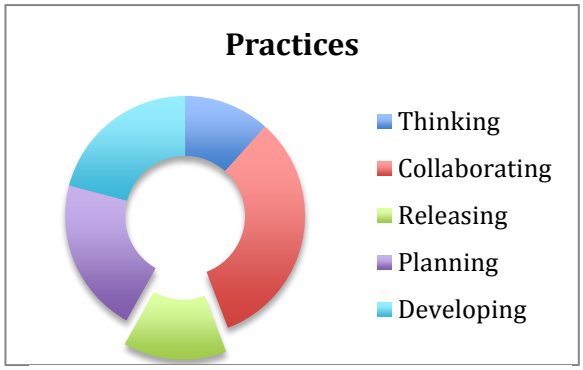

Figure 4 - Releasing Practices

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 Shore and Warden "*when your stories are Done Done, you avoid unexpected batches of work and spread wrap-up and polish work throughout the iteration*" [Shor08].

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 [Shor08] 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 we 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. [Abra02] 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 [Shor08] "*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*" say Huo et al. [Huo04].

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 [Nord03] 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*".

1.4 Planning Practices

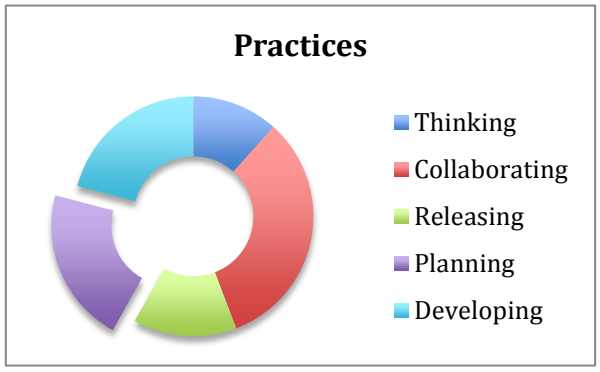

Figure 5 - Planning Practices

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 [Larm07] 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 [Cohn04] 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 [Good09] 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 who wishes 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. [Sfet06] 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 given 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 [Ruhe05], 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/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 [Shor08] point 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 research 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 [Shor08] 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 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 [Shor08] 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 [Lutz09] "*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.

1.5 Developing Practices

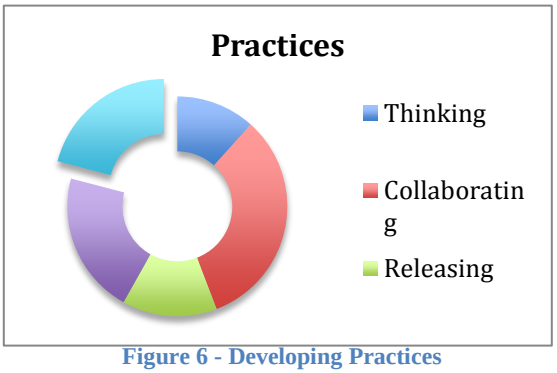

Figure 6 - Developing Practices

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 [Shor08] 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 develop a test first 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 [Geor03] 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 code done 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 behaviour 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 do 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. [Carb06].

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 [Shor08] 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 [Shor08] "*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 [Shor08] states: "*When you use exploratory testing, you discover information about both the software and the process used to create that software*".