CHALMERS

On Process Tailoring - An Agile Example

Master of Science Thesis in Software Engineering

Abdallah Salameh

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, April 2011

On Process Tailoring - An Agile Example
Master of Science Thesis in Software Engineering

Abdallah Salameh

Examiner: Per Zaring

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden April 2011

Abstract

Over the past few years, numerous organizations have realized the benefits of successfully adoption of the Agile Practices. Therefore, many organizations are now seeking to adopt the agile way of working, but how they can proceed with this adoption? Unfortunately, there is no existence of a structured approach for agile adoption, which is widely deployed and proved by the industry, that suites different organizational structures and can constitute a silver bullet that can maintain a focus on rapid delivery of business values, produces a qualitative software, satisfies the customers' needs, and can help on solving the humanity issues and other factors in different Organizational Cultures. The main problem addressed by this paper is the absence of research that addressing the validity of Agile Software Development as a silver bullet which comes from the absence of guidance to adopt the Agile Practices in different organizational cultures and structures. Therefore, the organizations have problems of selecting and adopting the suitable Agile Practices. In this paper we are introducing a simple framework (**Three-Step**) for agile adoption in different Organizational Cultures that needs a repeatable assessment of the used practices in each project in order to be able to tailor the possible Agile Practices.

Table of contents

1. Introduction

Over the past few years, numerous organizations have realized the benefits of successfully adoption of the Agile Practices. Therefore, many organizations are now seeking to adopt the agile way of working, but how they can proceed with this adoption? Unfortunately, there is no existence of a structured approach for agile adoption, which is widely deployed and proved by the industry, that suites different organizational structures and can constitute a silver bullet that can maintain a focus on rapid delivery of business values, produces a qualitative software, satisfies the customers' needs, and can help on solving the humanity issues and other factors in different Organizational Cultures. The main problem addressed by this paper is the absence of research that addressing the validity of Agile Software Development as a silver bullet which comes from the absence of guidance to adopt the Agile Practices in different organizational cultures and structures. Therefore, the organizations have problems of selecting and adopting the suitable Agile Practices. In this paper we are introducing a simple framework (**Three-Step**) for agile adoption in different Organizational Cultures that needs a repeatable assessment of the used practices in each project in order to be able to tailor the possible Agile Practices.

1.1. Problem background

In order to be able to conduct this research, we needed to have an extensive review and study of the literature. Therefore, we focused on process improvement, Agile Software Development methodologies, and the existence of different Organizational Cultures and Structures. Thereby, we paid some attention to the current implications of Software Engineering and its potentials since it was hard to find a research that addressing the validity of Agile Software Development as a silver bullet that would kill all problems.

1.2. Problem statement

A substantial economic activity is comprised by software development. Yet, software projects have high failure rates. According to Ralph and Wand (Ralph and Wand 2008), "*It is estimated that in 2004, 18% of projects failed outright and 53% of projects were "challenged," i.e., were delivered over budget, late, or with a reduced feature set*". Since that the process, which is used to develop software, contributes to project success. Therefore, some attention should be paid to the employed process of developing software.

The current state in Software Engineering and Development have some issues that need to be fixed and improved in a better way, such as having a mixture of practices and detailed processes. Organizations employ numerous processes to perform the needed tasks in projects development in different ways according to their organizational culture and structure. In practice, there are a numerous of methodical choices that software developers can use, they can choose to follow one of them completely or partially, or combine the needed aspects of different methods. Therefore, Paul Ralph and Yair Wand

noticed that "*neither existing methods nor existing process models explain the full spectrum of software development phenomena*" (Ralph and Wand 2008).

Since many aspects of the Software Engineering are human-centric and people thinking in a different ways, besides having a lack of systematic quality controls. The produced quality and productivity of their software production is differentiated from one person to another according to their creativity. The complexity of the applications is considered also as one of the most important factors for our business limitation that may causes a high level of uncertainty, communication problems, cost overruns, schedule delays and unreliability. Therefore, the practitioners should examine evolutionary and incremental improvements to software development rather than to wait for revolutionary ones.

It was hard to find a research addressing the validity of Agile Software Development as a silver bullet that would solve most issues in Software Engineering to improve it in a better way. Therefore, this study involves a theoretical and practical approach to the activities needed in software development. We addressed some Agile Software Development methodologies in order to have some insight about the deployed practices and to examine their principles and values. On the other hand, we focused on providing more information and the different Organizational Cultures and Structures that an organization might have. By considering the humanity issues and other factors in different Organizational Cultures and relating them to the Agile Software Development, besides addressing the implications of Software Engineering and its potentials, we could introduce a lightweight framework that would enable us to tailor some Agile Practices in different Organizational Cultures or Structures.

1.3. The purpose

The main purpose of this paper is to tailor some Agile Practices in different Organizational Cultures and to investigate whether Agile Software Development would constitute a silver bullet which can maintain a focus on rapid delivery of business values and can help on solving the humanity issues. The paper will in a detailed manner describe how we can do so by providing a framework that can help project managers on tailoring the required practices.
The case study initially had the following set of research questions that were to be answered:

- Would Agile Software Development constitute a Silver bullet?
- Would software development methods (mechanisms) constitute a successful way of developing systems? Would these mechanisms contradict with the human behaviors?
- Would Agile Software Development processes enable the construction of qualitative software in different organizational cultures?
- Would humanity issues in different Organizational cultures affect the construction of software development? By considering the agile way of software development, do think it would solve the humanity issues, and thereafter would increase the productivity of the employees and would produce qualitative software?

- Should we have Multi-Cultural Organizations of software development? Would using lightweight processes and by considering the humanity issues we can constitute successful software?

The continuation of this paper aims to answer these questions through providing some hypotheses, as well as discussion around these research questions.

2. Theoretical Background

2.1. Agile Software Development

2.1.1. Overview about agile methods

Since that the Waterfall model can lead to a large quality improvement when developing software, because of its way of working; where the results of one process phase falling down as input to the next one, it has an essential flaw; that is the requirements changes over time, so with fairly small projects over short time periods this process worked (Steinberg and Palmer 2004). If we followed the waterfall model in iterative way to make the required changes, this will delay the process of the development and as a consequence, the cost will be increased and the delivery time will be delayed as well (Highsmith and Cockburn 2001).

The people behind the idea of Agile Software Development (ASD) suggested to build first a prototype, and by discussing it with the customer we can gain more insight to build another one to discuss it again and so on till they have the needed one that not only fulfills the specification, but also satisfies and fulfills the customer's needs (Highsmith and Cockburn 2001; Steinberg and Palmer 2004). The changes that are introduced from discussing each version are considered as the driving force that provide them with more insight that enable us to develop a software which really satisfies customer's needs (Steinberg and Palmer 2004).

Hence ASD embrace change and promotes evolving the product. Therefore, Alistair Cockburn and Jim Highsmith in (Highsmith and Cockburn 2001) realized that ASD teams considered as a quick responsiveness to change iff it can reduce the cost of moving the information between people and iff it can reduce the elapsed time between making a decision to see the consequences of that decision. Reducing the cost of moving the information between people can be don by placing people physically closer, by using face-to-face communication with using whiteboards, instead of documentation and by improving the team's amicability. Reducing the elapsed time between making a decision to see the consequences of that decision can be done by having the experts available to the team and by producing the work incrementally. On the other hand, the iterative waterfall model tolerates change. But at the same time, its underling considers the changes as problem (Steinberg and Palmer 2004).

So, instead of trying to have a predictable process to achieve a good product quality at the end by using waterfall model processes, Agile Software Development would focus directly on maintaining a high quality "prototype" through the entire software development cycle (Steinberg and Palmer 2004). Therefore, agile processes apply best practices for product quality directly on the prototype.

Pekka Abrahamsson, Juhani Warsta, Mikko T. Siponen and Jussi Ronkainen said "*Agile proponents claim that the focal aspects of light and agile methods are simplicity and speed*" (Abrahamsson, Warsta et al. 2003). They realized that the development method can be agile one, if the development done incrementally by starting from small releases

with rapid cycles, if communication between the customer and developers was in a cooperative way, if the method was straightforward where the practitioner can learn and modify it easily, and if it was able to support the last moment changes (Abrahamsson, Salo et al. 2002).

On February 2001, Agile Software Development Alliance realized that there were some underlining similarities in what they were doing. So they had met in order to see if they could find some common ground. They agreed on Agile Manifesto and a number of principles for agile software development. Martin Fowler and Jim Highsmith had analyzed this manifesto and it's principles at August 2001 (Fowler and Highsmith 2001).

2.1.2 The Agile Manifesto

The Manifesto for Agile Software Development according to the Agile Alliance (Cockburn 2002):
"*We are uncovering better ways of developing software by doing it and helping others do it. We value:*
(i) Individuals and interactions over processes and tools.
(ii) Working software over comprehensive documentation.
(iii) Customer collaboration over contract negotiation.
(iv) Responding to change over following a plan.
That is, while we value the items on the right, we value the items on the left more."

2.1.3. The Agile Principles

The Agile Software Development principles according to the Agile Alliance (Beck, Martin et al. 2001):

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity--the art of maximizing the amount of work not done--is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

2.1.4. Agile Methods or Approaches:

There exist a numerous of different agile software development methods that can be considered as a part of the agile family. But these are the most well-known methods:

- Scrum
- eXtreme Programming (XP)
- Crystal Methods
- Agile Modeling
- Adaptive Software Development (ASD)
- Dynamic Systems Development Method (DSDM)
- Feature-Driven Development (FDD)
- Lean Development
- Agile Unified process (AUP)
- Microsoft Solutions Framework (MSF4ASD)
- Essential Unified Process (EssUP)
- Open Unified Process (OpenUP)
- Pragmatic Programming

In below sections, the current state of agile software development methods is reviewed. They are providing an overview of agile processes, roles, practices and scope of use and lastly a comparison between them. The following methods are included in this review: eXtreme Programming (XP), Scrum and Rational Unified Process (RUP).

2.1.4.1. eXtreme Programming (XP)

XP has been introduced by Ward Cunningham, Kent Beck, and Ron Jeffries during developing a payroll system at Chrysler Corporation where these guys were working on. The philosophy behind it is based on promoting the good ideas that we find in order to deliver working software that fulfils the customer needs as quickly as possible, then we should do these all the time. On the other hand, if something causes problems that contradict that agility of the development of the project, then we must not do it at all (Steinberg and Palmer 2004).

2.1.4.1.1. Process

XP's life cycle has five phases: Exploration, Planning, Iterations to Release, Productionizing, Maintenance and Death. These phases have been described by Beck as below (Abrahamsson, Salo et al. 2002), and the Figure 2.1 illustrates them.

In the **Exploration phase**, the customers describe each feature they wishing to have in the first release of the system by writing it out into a story card. At the same time, the development team arrange for the architecture possibilities by building a prototype of the

system. This phase takes from few weeks to a few months; depending on how largely the development team is familiar with this system.

The **Planning phase** takes a couple of days, as a first step the programmers estimates the required effort that is needed for each story, then the stories are prioritized and the content of the first release is specified where the time span of the schedule of this release does not exceed two months.

In the **Iterations to release phase**, the schedule set in the planning phase is broken down into several iterations before the first release where the implementation of each one takes one to four weeks, then the system become ready for production at the end of last iteration. In the first iteration, the stories that enforce building the architecture of the system are selected as a first step. The customer usually selects the stories according to the features that he wishes to have at each iteration.

Checking the performance and doing the extra testing of the system are required by the **Productionizing phase** before releasing the system to the customer. Thus, some changes may arise and decisions have to be made to include these changes or to document and postponed some of them for later implementation.

The **Maintenance phase** started after delivering the first release of the system to the customer. As a consequence, the development velocity may decelerate because of the extra effort that is required for customer support tasks and the need of incorporating new people into the team might change the team structure.

The **Death phase** started when the customer has no longer any stories to be implemented, thus no changes will be done on the architecture of the system and the necessarily documentation of the system is written as a final step.

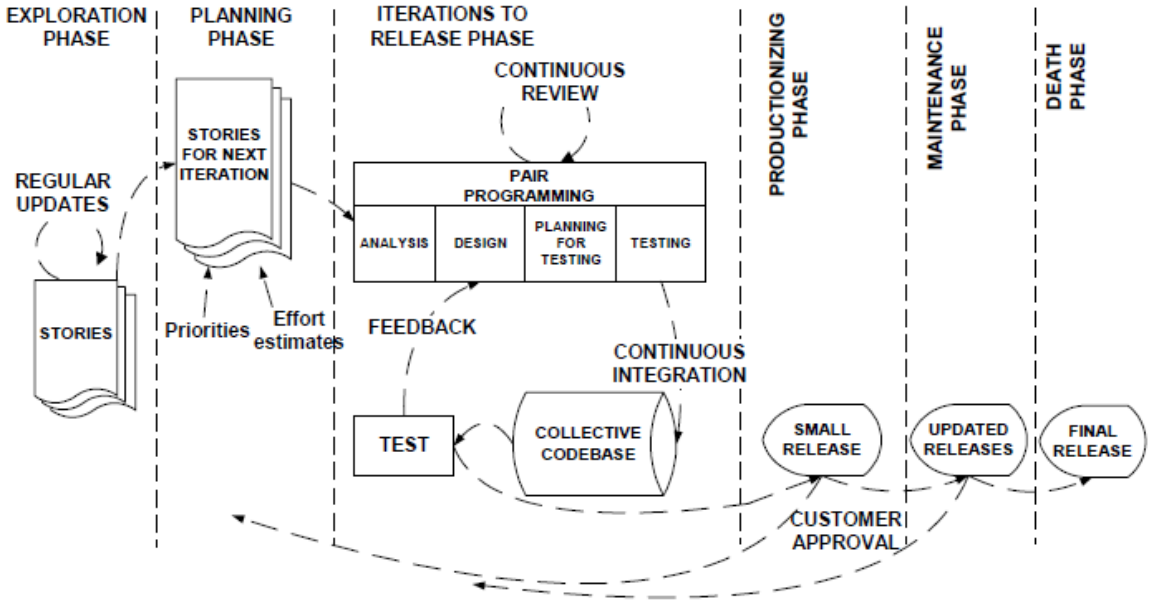


Figure 2.1. The life cycle of XP process (Abrahamsson, Salo et al. 2002).

2.1.4.1.2. Roles

There are different roles for different tasks in XP, these roles are presented in Table 2.1 (Abrahamsson, Salo et al. 2002; Steinberg and Palmer 2004).

Role Name	Description
Programmer	Responsible on writing the tests and the program code. Takes in consideration keeping the code as simple as possible.
Customer	Responsible on writing the stories and the functional tests, specify the priority of implementing the stories, and decides whether the requirements are satisfied or not.
Tester	Help the customer to write the functional tests in order to be able to run them regularly and broadcast their results.
Tracker	Responsible on tracking the estimates that have been done and gives the feedback in order to improve future estimations, also responsible on tracing the progress of each iteration in order evaluate whether the selected stories could be implemented within the given time and resource constraints or if it is needed to make some changes in the process.
Coach	Responsible for the whole process and guides the team members in following the XP's process.
Consultant	Consultant is an external member who guides the team in solving specific problems.
Manager	Manager communicates with the team to determine the current situation in order to be able to make the required decisions and to distinguish any difficulties in the process.

Table 2.1. XP's roles and responsibilities.

2.1.4.1.3. Practices

The practices of XP are presented in the following according to Daniel Steinberg and Daniel Palmer (Steinberg and Palmer 2004) and according to Pekka Abrahamsson, Juhani Warsta, Mikko T. Siponen and Jussi Ronkainen (Abrahamsson, Salo et al. 2002). Some practices have different names in different books and references; therefore some alternative names are listed between the parentheses.

• Whole Team (Customer Team Member, on-site customer)
Everybody involved in the project works together as one team in the same place (open workspace), and one team member is the customer or the customer representative.

• Small releases (Short cycles)
The system is frequently released and deployed to the customer, at least once every two to three months. Each release is planned ahead and not allowed to slip, and the release can be changed right up to the end in case that some features can not be finished at the planned time.

• Continuous Integration
As soon as a piece of code is ready regarding functionality, it is integrated into the code-base. Thus a new version of the complete system is build. All tests have to be passed in order to accept the new piece of code in our new version.

• Test-Driven development (Testing)
No single line of code is written without first writing a unit test that tests it. This means that all tests should be written in a test framework like JUnit so they become fully automated.

• Customer tests (Acceptance Tests, Testing)
The Customer or the person who represents the customer writes the functional tests which verify that the system satisfies and fulfills customer's needs.

• Pair Programming
The code should be written by two people at one computer. Thus positive effects can be gained by working in this manner, i.e. spreading the information, skills spreading and better code quality.

• Collective Code Ownership
Anyone has the rights to change any part of the code at anytime. Thus no one should take pride and responsibility for the quality of code that he/she wrote, but rather on for the complete program.

• Coding standards
The coding style should be the same for the whole system in order to have a code-base that is readable and understandable by whole the team. Thus coding rules must be exist and followed by the programmers.

• Sustainable Pace (40-hour week)
The work pace should be constant throughout the project so that people do not drain their energy reserves. The overtime is allowed, but not two weeks in a row.

• The Planning Game
As soon as the programmers estimate the required effort to implement the customer's stories, then the customer decides about the time and the scope releases.

• Simple Design
Prepare the simplest design that is needed for the current state of the implementation. Do not make any up front design decisions, just consider the current state.

• Design Improvement (Refactoring)
Restructure the system by finding the duplication and removing it, by finding ways to improve the design to make the communication easier.

• Metaphor
Try to find a few metaphors for the system. These metaphors should aid in communicating design decisions. This “shared story” guides all the development of the system by describing the way that the system working on.

2.1.4.1.4. Scope of use

XP is an effective development process and has shown to produce very high quality code. It is aimed for small and medium teams where the team size is limited between three to twenty project members (Abrahamsson, Salo et al. 2002; Steinberg and Palmer 2004).

XP’s applicability is restricted by two things. First, it is based on face-to-face communication that required from everybody involved in the project to be available in the same room. Therefore, it does not work if more than twenty people are involved in. Second, XP can not be used for projects that build critical systems because it has sacrificed control for agility purposes (Abrahamsson, Salo et al. 2002; Steinberg and Palmer 2004).

2.1.4.2. Scrum

Scrum is an approach that has been developed for managing and controlling the systems development process in an agile way. It promotes the ideas of flexibility, adaptability, productivity and self-organizing product development process (Abrahamsson, Salo et al. 2002). Scrum was designed to increase speed of development, provide consistent communication of performance, enhance individual development and align individual and organization objectives (Sutherland, Viktorov et al. 2007). Scrum can be adapted easily to different kind of projects and easily combined with other processes like Rational Unified Processes (RUP).

2.1.4.1.1. Process

Scrum life cycle includes three phases; Pre-game, Development and Post-game. These phases have been described according to Schwaber and Beedle as below (Abrahamsson, Salo et al. 2002), and the Figure 2.2 illustrates them.

Pre-game phase includes two sub-phases: Planning and Architecture. **Planning phase** includes the Product Backlog list, tools, verification management approval, and risk assessment and controlling issues. All the gathered requirements of the system are included in the created Product Backlog list where the work effort to implement these requirements is estimated and prioritized. The product Backlog is updated and reviewed continuously in each iteration by the Scrum team. The high level design of the system is planned according to the current state of the Product Backlog in the **Architecture phase**, a design review meeting is held to check the suggestions for the implementation and decisions are made accordingly.

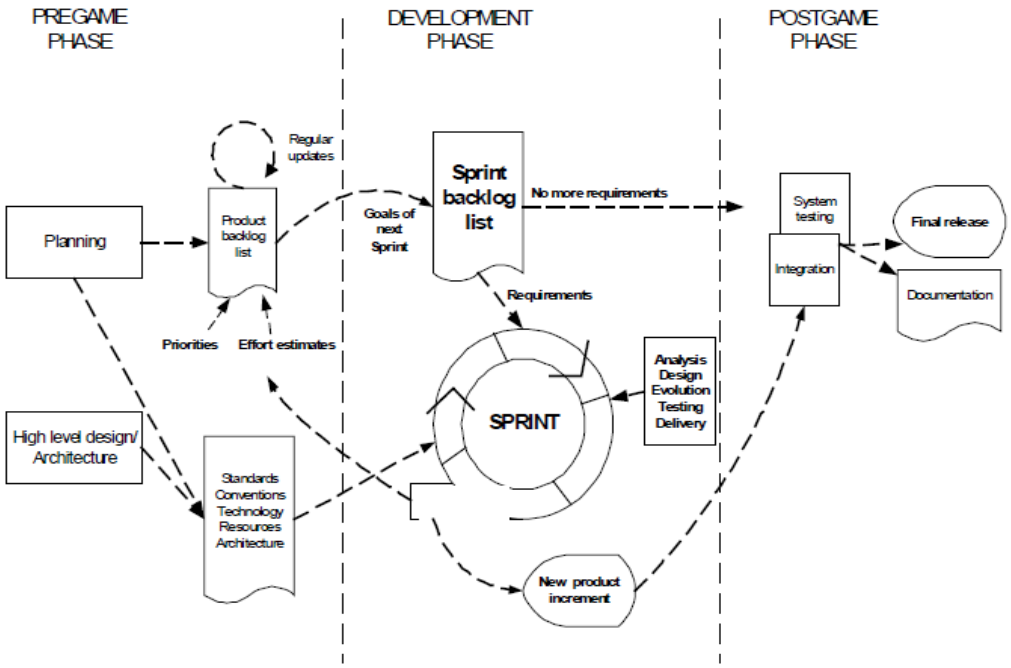


Figure 2.2. Scrum life cycle (Abrahamsson, Salo et al. 2002).

The **Development phase** is the agile part. The different environmental and technical variables that are identified in Scrum are observed and controlled during the Sprints through various practices. The environmental and technical variables (such as time frame requirements tools, resources) may change during the process. Each Sprint includes the traditional phases of software development, the functionalities are developed to produce new increments trough iterative cycles (Sprints). The duration of each Sprint is from one week to one month.

The release always ends by the **Post-game phase** when the environmental variables are completed, such as the requirements. Thus the system is ready to be released.

2.1.4.1.2. Roles

There are different roles for different tasks in Scrum; these roles are presented in Table 2.2 according to Schwaber and Beedle as below (Abrahamsson, Salo et al. 2002).

Role name	Description
Scrum Master	Responsible to ensure that the project is progressing as planned and it has considered the rules, values and practices of Scrum. Also responsible on removing obstacles that might face the team in order to have as much productivity as possible.
Product owner	Responsible of managing, making visible Product Backlog list, makes the final decisions concerning the tasks that are related to Product Backlog list, estimates the development effort.
Scrum team	Scrum team is a self organized project team that has the authority to make the required actions in order to achieve the goals of each Sprint.

Customer	Participates to help on preparing the items of the Product Backlog list in order to enhance or develop the system.
Management	Participates in setting the goals and requirements and Responsible on setting on making the final decision.

Table 2.2. Roles in Scrum

2.1.4.1.3. Practices

In order to avoid the chaos that is caused by complexity and unpredictability, Scrum requires certain management practices in the various phases. Therefore it does not require any specific practices (Abrahamsson, Salo et al. 2002). The description of the following practices is given based on Schwaber and Beedle (Abrahamsson, Salo et al. 2002).

• Product Backlog
Product Backlog is maintained by the product owner and contains a prioritized list of items that describes the changes and the additions that should be made to the system so that the final system can be produced. The Backlog items can include functions, features, defects, enhancements requests and technology upgrades, where these items prepared by the participation of the customer, project team, management, marketing and sales.

• Effort estimation
The Backlog items are estimated in iterative process which more accurate as long as more information is available in the certain Product Backlog item. This estimation is done by both of the Product Owner and the Scrum Team.

• Sprint
A Sprint is an iteration that lasts thirty calendar days. The features are not allowed to be changed by the customer during the Sprint. It is a way that is adapted to change the environmental variable (such as the requirements, time technology etc.). The working tools for the Scrum team are the Sprint Planning Meeting, Sprint Backlog and Daily Scrum meeting.

1. Sprint Planning Meeting
A Sprint Planning Meeting is organized by the Scrum Master into two meetings. In the first meeting, the customer, Scrum Team, Product Owner, and management participate to decide upon the goals for the next Sprint. While in the second meeting, the Scrum Team and the Scrum Master participate in order to focus on how to implement the product increment during the Sprint.

2. Sprint Backlog
Sprint Backlog is a list of product Backlog items selected as a starting point for each Sprint to be implemented in the next Sprint. Its items are selected based on its prioritization and based on the goals set for the Sprint. These Items are selected by the Scrum Owner, Scrum Master and the Scrum Team in the Sprint Planning meeting.

3. Daily Scrum meeting
The purpose of the Daily Scrum meeting is to keep track of the progress of the Scrum Team, also used as a planning meeting. During this meeting, the problems are discussed and controlled. The Scrum Master and the Scrum team participate in the Daily Scrum meeting.

Spring Review Meeting

The work results are presented by the Scrum Master and Scrum Team to the customer, management and Product Owner at the last day of the Sprint in an informal meeting. An assessment and decisions taken by the participants concerning the presented results may led to add or change some items in the Backlog list and even may change the direction the system.

2.1.4.1.4. Scope of use

Scrum is suitable for small teams of less than ten members. If there are more than ten members, then multiple teams should be formed.

2.1.4.3. The Rational Unified Process (RUP)

RUP was developed by Rational Corporation to complement UML. RUP considered as an iterative approach that is used for object-oriented systems.

2.1.4.1.1. Process

RUP's life cycle has four phases: Inception, Elaboration, Construction and Transition. Each phase is split into iterations; the purpose of each iteration is to provide a piece of working software. The duration of each iteration may vary from two weeks till six months. These phases are described as below (Abrahamsson, Salo et al. 2002), and the Figure 2.3 illustrates them.

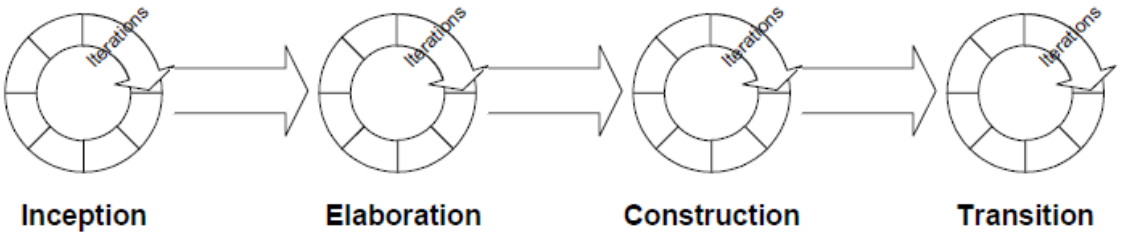


Figure 2.3. RUP process (Abrahamsson, Salo et al. 2002).

In the **Inception phase**, the objectives of the life cycle are by considering the needs of every stakeholder. Thus the scope boundary is specified and an estimation of the cost and schedule are prepared.

Analyzing the problem domain, specifying the architecture and the project plan of the system are prepared in the **Elaboration phase**. At this phase, the development environment and the process are described in detail. By the end of this phase, most use cases and actors should be described beside the architecture, and an executable prototype of the architecture created.

All the system features are developed and integrated into the product and tested in the **Construction phase**. This phase emphasizes on controlling the costs, schedules and the quality. During this phase, one or several releases are produced.

When the maturity of the system become enough to be released to the customer, the **Transition phase** starts. Subsequent releases are produced according the customer's feedback. This phase also consists of beta testing, training the end-users and the support team, and distribute the system to sales and marketing teams.

2.1.4.1.2. Roles

There are different roles based on the activities of the RUP, these roles are categorized into Business Modeling and Environment Workflow; these roles are presented in Table 2.3 as below (Abrahamsson, Salo et al. 2002).

Category	Role Name	Description
Business Modeling	Business-Process Analyst	Coordinates defining the business use cases by cooperating with team members. Also Responsible on defining the business object model, which describes the interaction between the actors and the processes.
	Business Designer	Responsible on elaborating the split parts on the business use-case model into business use case. Also responsible on documenting the entities and roles.
	Business-Model Reviewer	Responsible on reviewing all the produced artifacts by both of the Business Designer and Business-Process Analyst.
Environment Workflow	Course Developer	Responsible on producing the course materials for the end-users of the system that is under the development.
	Toolsmith	Developing the needed tools to support the development in-house, the purpose is to enhance the automation and to improve the integration between tools.

Table 2.3. Roles in RUP.

2.1.4.1.3. Practices

There are six practices that represents the cornerstones of RUP. These Practices had been described by detail by Kruchten and summarized in Table 2.4 by Pekka Abrahamsson, Juhani Warsta, Mikko T. Siponen and Jussi Ronkainen (Abrahamsson, Salo et al. 2002).

Practice	Description
Develop software iteratively	Developing software in short iterations increases the possibilities of identifying the risks in early stages of the development, which allows an adequate reaction.
Manage requirements	A continuous process used to identify, prioritize, filter and trace the requirements that have a great impact on the project goals, and that change continuously.
Use component-based architecture	By dividing the software architecture into components, a high flexibility can be gained, especially when separate the most likely to change parts.
Visually model software	Models are built by using a visualized method that describes the architecture and the design of the system (such as UML). This model can be understand easily by the development team and used as a communication tunnel between all parties.
Verify software quality	The verification of the software quality reduces the cost of fixing the defects that may arise at late stages of the software development life cycle. Thus testing is applied for each iteration in order to find as much defects as possible in the earlier stages.
Control changes to software	Managing the requirements change, and the ability to trace their effect on the software. The types of the made changes and their frequency, used as a baseline for measuring the maturity of the software.

Table 2.4. Practices in RUP.

2.1.4.1.4. Scope of use

RUP contains extensive guidelines for process phases which do not provide any clear implementation guidelines. Therefore, RUP does not for instance specify the required documentation and project size (Abrahamsson, Salo et al. 2002).

2.1.5. Summery about Agile methods

There are numerous different agile methods that have been emerged in the last decades. Because of that, the practitioners and researchers are not aware of these methods (Abrahamsson, Warsta et al. 2003).

Scrum, FDD, Crystal, and ASD still remains in a "building up" phase (Abrahamsson, Salo et al. 2002). On the other hand, XP, RUP, and DSDM are methods that have been well documented and experienced in the industry and active (Abrahamsson, Salo et al. 2002). Pekka Abrahamsson, Juhani Warsta, Mikko T. Siponen and Jussi Ronkainen have summarized the features and the shortcomings of most of the well-known agile methods in (Abrahamsson, Salo et al. 2002); Table 2.5 presents their summery for XP, Scrum and RUP.

Method name	Key points	Special features	Identified shortcomings
XP	Customer driven development, small teams, continuous builds and daily builds.	Refactoring	Less attention for managing the practices and for getting an overall view of the whole process.
Scrum	Small and self-organizing development teams. An iteration (Sprint) takes thirty calendar days, ends by a release.	Emphasis on Scrum view of development.	The integration and acceptance tests are not interpreted in detail.
RUP	Include tool that support the complete software development model.	Business modeling.	It does not provide a description of how to tailor changing needs. On the other hand, it has no limitation in its scope of use.

Table 2.5. Summary of the general features and shortcomings of agile methods (Abrahamsson, Salo et al. 2002).

Agile thinking is a people-centric view; therefore, it depends on people to choose the right method for the suitable at the right time, "*Agile methods claim to place more emphasis on people, interaction, working software, customer collaboration, and change, rather than on processes, tools, contracts and plans*" (Abrahamsson, Salo et al. 2002).

2.2. Organizations

The performance of different organizations is differentiated because of the different structures that they have. That is why some organizations achieve the success while others suffer and fail. The organizations that could success are those that could review and redesign their structures according to the internal and external factors on an ongoing basis.

According to Henry Mintzberg, an organization's structure emerges from the interplay of the organization's strategy, the environmental forces it experiences, and the organizational structure itself (Mintzberg 1983; Mintzberg 1983).

2.2.1. Organizational structure

Organization Structure refers to internal pattern of authority, relationships and communication inside the organization (Fredrickson 1986). According Henry Mintzberg (Mintzberg 1983), an organization's structure emerges from the interplay of its strategy, the environmental forces it experiences, and the structure that it has.

There are four building blocks making up company's structure according to Talya Bauer and Berrin Erdogan (Bauer and Erdogan 2009). These building blocks are: Centralization, Formalization, Hierarchical levels and Departmentalization.

Centralization refers to the degree to which the authority of making decisions, solving problems and the evaluation of the activities are concentrated at higher levels in an organization, whereas in decentralized organizations, solving the problems and taking the required decisions are made at lower levels in an organization by the employees who are closer to problems (Fredrickson 1986; Bauer and Erdogan 2009). In order to be able to take decisions, a significant cognitive is required from those managers who work in the organizations that have a high level of centralization (Fredrickson 1986). Many companies noticed the inefficiencies in their decision making as long as they use the centralization (Fredrickson 1986).

Formalization is the extent to which an organization uses rules, policies, procedures and job descriptions to prescribe behavior. It makes the behavior of employees predictable and consistent by checking the procedure guidelines to solve any problem. Therefore, employees respond in the same way for the same problems (Fredrickson 1986; Bauer and Erdogan 2009).

Hierarchical level is the number of levels that the organization have in the hierarchy (Fredrickson 1986; Bauer and Erdogan 2009). An organization that has several layers of management between the frontline employees and the top levels is considered an organization that has a ***Tall structure***, whereas the organization that have a few layers and often has a large numbers of employees that reports to a single or few top managers considered an organization that has a ***Flat structure*** (Mintzberg 1983; Bauer and Erdogan 2009). The concept *Span of control* means the number of employees that are reporting to a single manager. The span of control is small for Tall structure, which implies to greater opportunities for the managers to control and monitor employees' activities, while the span of control in Flat structures is wider, which implies a greater levels of freedom for the employees and managers unable to have a close supervision (Bauer and Erdogan 2009).

The organizational structures are differentiated in terms of **Departmentalization** point of view into five types according to Talya Bauer and Berrin Erdogan (Bauer and Erdogan 2009); Functional structures, Divisional structures, Mechanistic structure (Bureaucracy structure), Organic structure and Matrix structure.

On the other hand, Henry Mintzberg suggested a typology of **five basic configurations** for the organizational structures: Simple Structure, Machine Bureaucracy, Professional Bureaucracy, Divisionalized Form, and Adhocracy (Innovative, Organic or Developmental) (Mintzberg 1983; Mintzberg 1983). According to Mintzberg, each organization can consist of **five basic parts**, figure 2.4 illustrates the dominate part of each organization and shows the franchising of each organization, these parts are: Strategic apex (top management), Middle line (middle management), Operating core (operational processes), Technostructure (analysts that design systems and processes),

and Support staff (support outside of operating workflow) (Mintzberg 1983; Mintzberg 1983).

There are **five valid coordinating mechanisms** in organizations according to Mintzberg. Each one of the five configurations relies on one of the five coordinating mechanism and tends to support one of the five parts (Mintzberg 1983; Mintzberg 1983). These coordinating mechanisms are;

1. **Direct supervision**. The key part in Entrepreneurial structure is the strategic apex, which coordinates by direct supervision.
2. **Standardization of work**. The key part in the Machine Bureaucracy structure is the technostructure, which coordinates primarily by the imposition of work standards and houses the analysts who do the standardizing.
3. **Standardization of skills**. Professional Bureaucracy structure relies on the standardization of skills in its operating core for coordination.
4. **Standardization of outputs**. A good deal of power in the middle line management of Divisionalized structure is delegated to market based units.
5. **Mutual adjustment.** The primarily way of coordination in Adhocracy organizations is the mutual adjustment among all parts, especially by supporting the collaboration between the support staff.

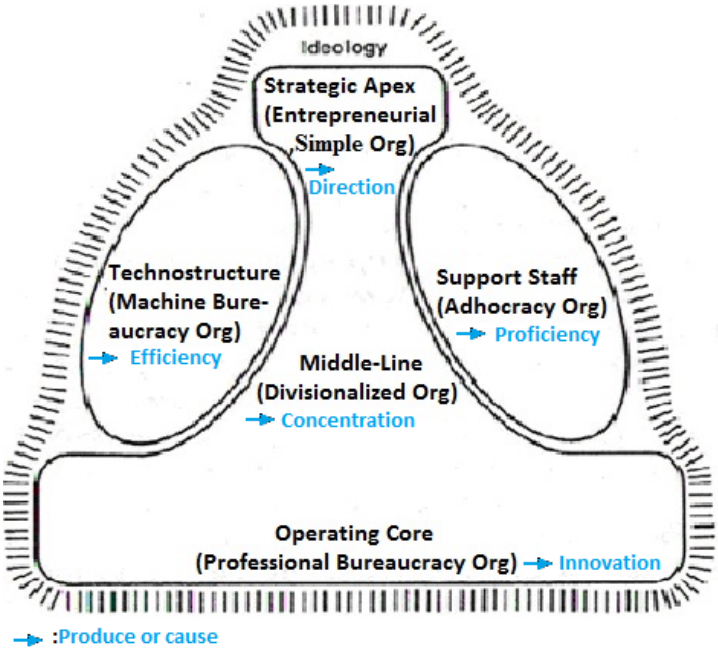


Figure 2.4. Basic parts of each organization according to Hery Mintzberg (adapted and reproduced from (Mintzberg 1983) page 11).

Below sections describe the five basic configurations for the organizational structures according to Mintzberg. And describe the Functional and Matrix Organization that have been proposed by Talya Bauer and Berrin Erdogan.

2.2.1.1. Simple Organization (Entrepreneurial Organization)

This type of organization has a flat structure (as described before). The organization is relatively unstructured as it is minimally elaborated and highly centralized by having strong leaders (usually their owners). This kind of structure, which tends to be found in new and small organizations that have lack of standardized systems, is flexible.

The structure of this kind of organizations has a few layers and often has a large numbers of employees that reports to a single or few top managers, thus it becomes inadequate as the organization grow because of the bad decisions that are overwhelmed by the top managers. Therefore, their top managers need start sharing the decision making. Also, they need to share the decision making when they notice that the organization could face a significant risk because of its dependency on the individuals to achieve the success.

2.2.1.2. Machine Bureaucracy Organization

According to Talya Bauer and Berrin Erdogan (Bauer and Erdogan 2009),"*Mechanistic structures are similar to bureaucracies, as they are highly formalized and centralized*". The communication inside the organization tends to follow formal channels, and the employees have a specific roles and job description. The decision making is centralized vertically at the strategic apex with limited horizontal decentralization to the technostructure, and tasks are grouped by functional departments (Mintzberg 1983; Mintzberg 1983).

Using standardized methods for performing the work, beside a high degree of control in order to ensure a standardized performance, are considered the essential elements of bureaucratic organization (Asopa and Beye 1997). Machine Bureaucratic basic structure is based on efficient performance of standardized routine work, formalized procedures in the operating core and formalized communication throughout the organization, large sized units at the operating level, and elaborates the administrative structure highly with a sharp distinction between line and staff, because of the tight regulation of the operating core (Mintzberg 1983).

The operating tasks are considered simple and repetitive, performing the tasks require a minimum of skill and little training. Hence, the labor is divided sharply in the operating core, it emphasizes division of labor strongly in all forms (vertical, horizontal, functional hierarchical and status), in order to emphasis on the standardization of work processes for coordination (Mintzberg 1983). As mentioned before, standardization of work is the coordination mechanism in Machine Bureaucracy structure. The key part in such structure is the technostructure, which coordinates primarily by the imposition of work standards.

According to Mintzberg (Mintzberg 1983), "*the Machine Bureaucracy is a structure with an obsession-namely, control. A control mentality pervades it from top to bottom*", the obsession with control reflects two central facts; First, the attempts of eliminating all possible uncertainties in order to run smoothly. Second, containing all the conflicts that Machine Bureaucracies structure can be ridden from by virtue of their design.

Figure 2.5 shows the Machine Bureaucracy symbolically, with a full elaboration of administrative and support structure. The staff parts of the organization are focused on the operating core, and large operating units but narrower ones in the middle line to reflect the tall hierarchy of authority.

Mechanistic organization resists the change, which makes it unsuitable for innovation. Despite that, it has many advantages when it used for stable environments (Bauer and Erdogan 2009). It is often associated with older and larger organizations that have a stable environments, and sometimes externally controlled. Also, it relies heavily on economies of scale for its success (Mintzberg 1983; Mintzberg 1983).

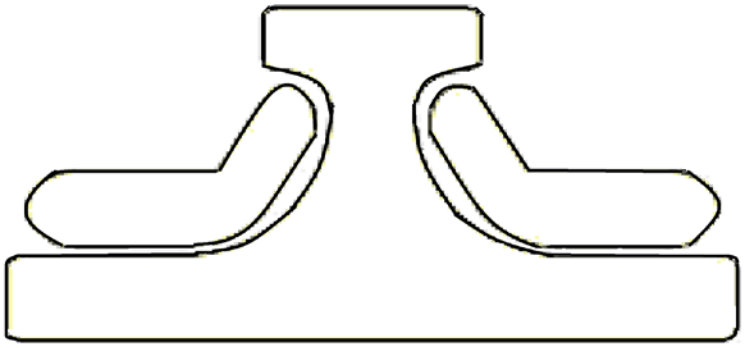


Figure 2.5. The Machine Bureaucracy (Mintzberg 1983).

2.2.1.3. Professional Bureaucracy Organization

Professional organization is essentially bureaucratic (Mintzberg 1983; Mintzberg 1983; Bauer and Erdogan 2009), its coordination achieved by standards that defines what is to be done in advance (Mintzberg 1983). It requires specialists with high skills, and depends upon efficient performance of standardized but complex work (Asopa and Beye 1997). Professional Bureaucracy relies on the standardization of skills and its associated design parameter "*training of operators, their vertically enlarged jobs, the little use made of behavior formalization or planning and control systems*" in its operating core for coordination (Mintzberg 1983; Mintzberg 1983), the standardized skills that professionals have are applied to predetermined situations (called contingencies or standardized). Hence, the professionals are working autonomously from their colleagues and closely to the clients they serve, and they have two basic tasks; First, categorizing the client's need in terms of contingency (find out which skills are needed to use), then apply the needed skills to the situation (Mintzberg 1983).

The difference between professional and machine organization is its relying on specialists and large number of knowledge workers with high skills who demand control of their own work. Therefore, the decision making is decentralized (Mintzberg 1983; Mintzberg 1983).

Professional organization enjoys the efficiency benefits of the machine structure because of its complexity and the large amount of rules that it has. The lack of control, that specialists can exercise, is considered as a disadvantage in such organizations, because of

spreading down the power and decision making through the hierarchy (Mintzberg 1983; Mintzberg 1983).

Seeking collective control of the administrative decisions, (such as; hiring colleague, promote them, and distribute resources) that affect professionals, complements the main responsibility of the professionals on controlling their own work. On the other hand, the supporting staff follows the machine structure; it is focusing on serving the key part of the Professional Bureaucracy which is the operating core (Mintzberg 1983). Therefore, a parallelism of administrative hierarchies are emerged in the Professional Bureaucracy, as you can see in Figure 2.6, one is machine bureaucratic (top-down) for the support stuff where the power and status reside in administrative office by practicing the administration to attain status, and a second democratic (bottom-up) for professionals where the power resides in expertise and the influence is introduced by professionals' knowledge and skills (Mintzberg 1983). Mintzberg realized that both of the parallel hierarchies should be kept quite independent of each other, see Figure xx, since "*research indicates that a professional orientation toward service and bureaucratic orientation toward disciplined compliance with procedures are opposite approaches toward work and often create conflict in organizations*" (Mintzberg 1983).

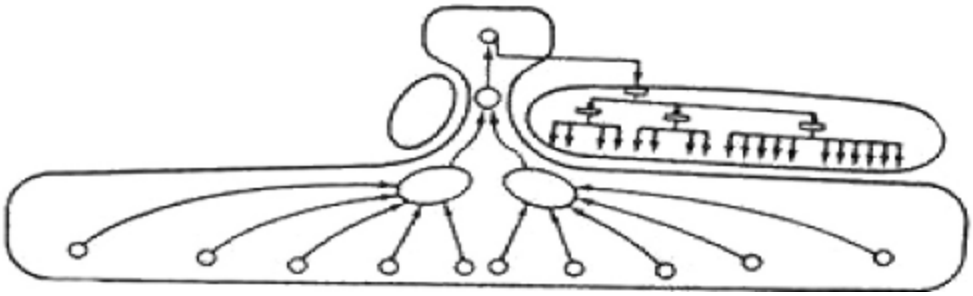


Figure 2.6. Parallel Hierarchies in the Professional Bureaucracy (Mintzberg 1983).

2.2.1.4. Divisional Organization

In such a structure, each unique product, service, or customer that the organization is serving, has its own department that it represents to facilitate management planning and control (Asopa and Beye 1997; Bauer and Erdogan 2009). Divisional organizations can be found in large and mature organizations that have product or market diversification where each department is considered as an autonomous division, that makes its own decisions and has its own unique structure, which is supported by a central headquarters (Mintzberg 1983; Mintzberg 1983).The organizations that have such Divisional structure are considered more agile and can perform better in turbulent environments (Bauer and Erdogan 2009). The employees are responsible on performing different kinds of tasks in the service of the product (Bauer and Erdogan 2009).

As mentioned before, the prime coordinating mechanism in such structure is the standardization of outputs. A good deal of power in the middle line management of Divisionalized structure is delegated to market based units, and a key design parameter is the performance control system (Mintzberg 1983).

The Divisionalized Form is represented symbolically in terms of Mintzberg's logo as you can see in Figure 2.7. Headquarters is illustrated in three parts; a small technostructure to the left, a small strategic apex of top managers, and a slightly larger staff support to the right. The divisions (which represented as Machine Bureaucracies) are presented below the headquarters (Mintzberg 1983).

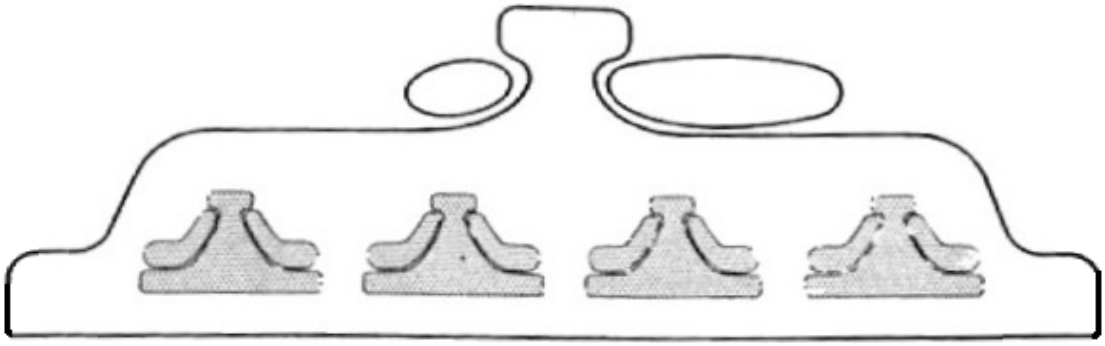


Figure 2.7. The Divisionalized Form (Mintzberg 1983).

Allowing the line mangers to maintain more control and accountability is considered a key benefit of such structure over the machine structure where the decision making is decentralized. Thus, the central team has the ability to focus on the strategic plans that guaranty the success (Mintzberg 1983; Mintzberg 1983).

The organization that has such a structure has a significant weakness. Those are, the duplication of resources and activities, the tendency of conflict between their divisions because of their need to get the required resources which is limited from the headquarters, and these organizations can be inflexible. Therefore, this kind of structures works best in industries that are stable and not too complex (Mintzberg 1983; Mintzberg 1983).

Many organizations' structures in industry are a hybrid of functional and divisional structures, depending upon their size and activities (Asopa and Beye 1997; Bauer and Erdogan 2009). The organization that have such a structure works well with Machine Bureaucracy structures in its divisions and drives them toward the Machine Bureaucracy form (Mintzberg 1983).

2.2.1.5. Adhocracy Organization (Innovative, Organic or Developmental)

According to Talya Bauer and Berrin Erdogan (Bauer and Erdogan 2009), "*Organic structures are flexible, decentralized structures with low levels of formalization* ". It has a flexible communication lines where the primarily way of coordination in such organization is the mutual adjustment among all parts, especially by supporting the collaboration between the support staff (Mintzberg 1983; Mintzberg 1983). The job description of the employees is broader, and the responsibilities are specified according the organization's needs based on experience level of each employee (Bauer and Erdogan 2009). This kind of organization is very difficult to control since that the power is decentralized and delegated to wherever it is needed (Mintzberg 1983; Mintzberg 1983).

The organization brings in experts from different areas to form a creative team. These experts can be drawn to solve problems at any time and to work in a highly flexible way. It can respond quickly to change, by providing the experts with the ability to meet new challenges, because of the continuous movement of its workers from one team to another as projects are completed (Mintzberg 1983; Mintzberg 1983).

The autonomy of employees and managers is considered highly in Adhocracy organization. The relational barriers between superiors and subordinates are minimized. Therefore, they have opportunity to improve the workspace functions and to be more innovative by providing the ideas and the suggestions.

Since that the organization which has Adhocracy structure considered an innovative one, and the innovative organization has challenges, lots of conflicts may be introduced because of the ambiguity of the authority and power. Also the high level of autonomy may allow some employees to do unethical behaviors. On the other hand, since the organization can respond quickly to the change, it is considered as a stressful environment for the workers which make it difficult to find and keep talent.

2.2.1.6. Functional Organization

The jobs are organized and grouped according to their functional similarities, such as having departments for marketing, finance, information technology and etc. In such organization, each person handles large volumes of transactions and serves a specialized role (Bauer and Erdogan 2009).

As long as the organization does not have a large number of products that requires a special attention and it has a stable environment that is slower to change, it tends to be more effective (Bauer and Erdogan 2009).

2.2.1.7. Matrix Organization

A cross between Functional structures with Divisional structures (Product structure). In Matrix organizations, the employees report to department managers as well as the project manager (Bauer and Erdogan 2009). The project manager is responsible on controlling the overall matters that are related to the project (Bauer and Erdogan 2009). By having the Matrix structure, the organization can utilize the balance of the benefits of both Divisional structure and Functional structures.

According to Davis and Lawrence (Davis and Lawrence 1977), Matrix structure is a preferable structure when three basic conditions exist simultaneously. These conditions are

1. Outside pressure for dual focus.
   It is the focusing of undivided human effort on two or more critical and essential tasks, functions, products or services at the same time.

2. Pressures for high information–processing capacity.

There is a special combination of circumstances that can lead to high level of information processing capacity, these circumstances are:

- Uncertainty**:** the demands are changing and unpredictable.
- Complexity: if the organization was doing a simple job, these uncertainties would be manageable. As long as the complexity of the tasks led to a major adaptation of management, the centralized and decentralized models maintained the traditional singular chain of command (Davis and Lawrence 1977).
- Interdependence: the information processing load increases when the interdependence increased among the people who work on a certain issue.

3. Pressure for shared resources.

It is the pressure that the organization face in order to achieve economies of scale in both of human terms and high performance in terms of both costs and benefits.

2.2.2. Organizational Change

Organizational change is triggered routinely as a response to the failure of people to create continuously adaptive organizations (Dunphy 1996; Weick and Quinn 1999). The change is vary from one organization to another, depends on the level of analysis, how an organization functions, its structure and culture, its members and the style of leadership (Dunphy 1996; Weick and Quinn 1999).

From the perspective of organizational development, organizational change is the planned change of the organizational setting through a set of behavioral science-based theories (such as values, strategies and techniques) for the purpose of improving its performance and enhancing individual development (Porras and Robertson 1992; Weick and Quinn 1999).

Organizational change has been investigated through various categories, e.g. revolutionary vs. evolutionary, transactional vs. transformational, and incremental vs. transformative**.** Revolutionary changes is considered a radical changes where the connections with the past is broken, whereas the evolutionary changes are small changes that lakes longer time to be done and do not cause overthrow of the current system (de Wit and Meyer 2005; Guler 2010).

Usually, the evolutionary change is more common than revolutionary. It is associated with transactional change and incremental change. Whereas, the revolutionary change is associated with transformational change and transformative change. Sometimes the radical changes are needed to stimulate the organization because of the rigidity within the organization where the small and incremental changes do not help (de Wit and Meyer 2005; Guler 2010).

2.2.2.1. Change Management

Change management is an organizational process for shifting individuals, teams, and organizations from a current state to a desired state which serves the business

environment. A successfully deployed method for change management in past does not mean that it will do so in the future, especially in the dynamic environments and where the organizations faces increasing of its complexity (Zeffane 1996; Guler 2010).

2.2.2.2. Change Management Process Models

There are numerous models for change management processes that have been developed by different researchers. These models have some common aspects between them, even though each one of them has its own way and benefits. These models started by investigating and analyzing the current status in order to ensure the readiness and need for change, then they proceed with some actions than end by process review (Pugh 2007; Guler 2010). Human factors such as people's behaviors and emotions should be compatible with technology and organizational formation and strategy in order to have systematic change processes (Beer and Eisenstat 1996; Guler 2010).

Pugh cited Lewin's change management process model which has three stages (Pugh 2007; Guler 2010), these stages are:

1. Unfreezing: Being aware of the need for change to create a healthy organization, involve the top management, building confidence and communication, identifying stakeholders and managing their expectations.
2. Moving: Following a comprehensive strategy which helps to develop the required knowledge while moving to a new level.
3. Refreezing: Making sure that change is a part of organizational culture while enhancing and changing the organizational structure, attitudes, and spreading the information.

Based on Lewin's model, Carter has developed a new one which has seven stages by addressing three factors; **structure** of the organization, **skills** and **strategy** that should be followed while carrying out changes. These stages have been illustrated in Figure 2.8 by Sevim Guler (Guler 2010).

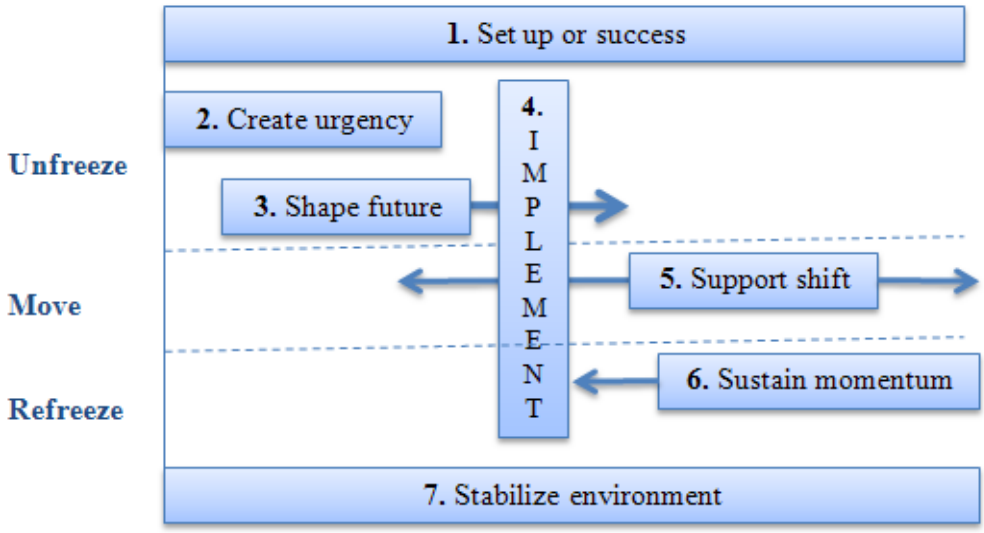


Figure 2.8. Carter's seven-stage model for change management (Guler 2010).

2.2.2.3. People Issues in Change Management

In order to have a successful change, the attention should be paid and balanced to both of the business change and the people issues. Therefore, a positive relationship between the phases of change for employees and the phases of change project should be continuously considered during the change, as illustrated in Figure 2.9 **(Hiatt and Creasey 2003).**

Hiatt and Creasey asset that the organization may face a loosing of valued staff, a delay of the projects and a reducing of the quality, if the emphasis was to the business change higher than the employees issues. On the other hand, it may not achieve the business goals if the emphasis was on the employees' issues higher than the business change **(Hiatt and Creasey 2003)**.

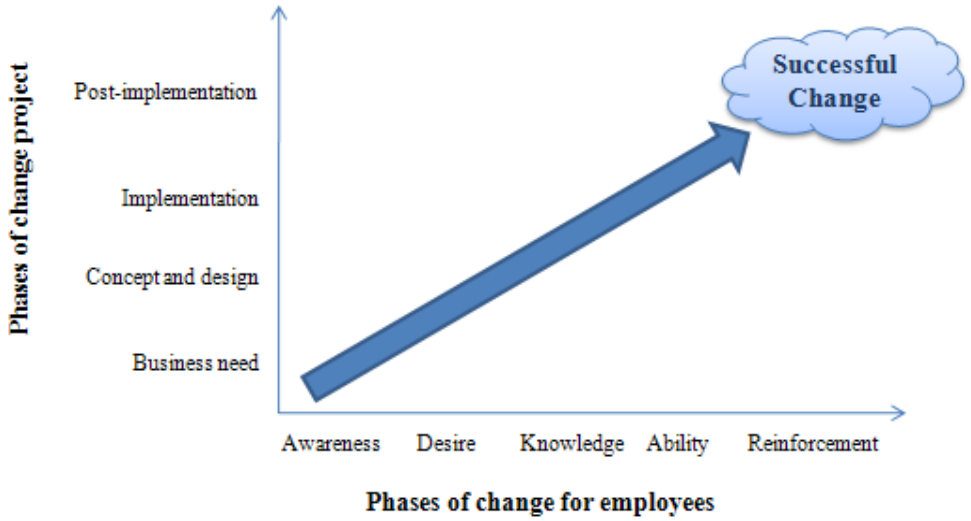


Figure 2.9. The relationship between the phases of change for both of the business and employees (Hiatt and Creasey 2003).

2.2.3. Organizational Culture

Organizational culture has several interpretations; it can be based on everything in an organization such as beliefs, values, models of behavior, practices, symbols, artifacts, and technology (Schein 1985; Gagliardi 1986). Values and beliefs represent the highest priorities and deeply held driving forces for an organization. Therefore, we will consider the Competing Value Model (CVM), which is based on the values of an organization, as a theoretical model that describes the organizational culture (Quinn and Rohrbaugh 1983; Quinn and Kimberly 1984).

2.2.3.1. Competing Values Model (CVM)

The CVM is based on two distinctions: change versus stability and internal focus versus external focus. Figure 5 illustrates the opposite ends of each dimension which represents the conflicting demands on the organizations. Change emphasizes on flexible processes, whereas stability emphasizes on controlled and continuous oriented processes. Internal

focus emphasizes integration and maintenance of the socio-technical system, whereas external focus emphasizes competition and interaction with the organizational environment (Quinn and Rohrbaugh 1983; Iivari and Huisman 2007). The opposite ends of these dimensions impose competing and conflicting demands on the organization.

Based on the two dimensions, Quinn and Rohrbaugh proposed four types of culture, Group culture, Developmental culture, Rational culture and Hierarchical culture (Quinn and Rohrbaugh 1983; Iivari and Huisman 2007). These cultures described as below:

1. The group culture, which emphasizes flexible processes (change) and internal focus, is primarily concerned with human relations and flexibility. Sense of family, cohesiveness, belonging, trust, and participation are its core values. The strategy of the organization that has group culture emphasize toward developing human resources and commitment.

2. The Adhocracy or developmental culture, which is characterized by flexible processes (change) and external focus, is future-oriented, considering what might be. The strategy of the organization that has such culture emphasize toward growth, resource acquisition, innovation, and adaptation to the external environment. Therefore, it supports the enterprise agility.

3. The rational culture, which emphasizes control-oriented processes (stability) and external focus, is achievement-oriented. Therefore, the organizations that have such culture emphasize toward goal achievement, productivity, competitive advantage, market superiority and efficiency.

4. The hierarchical culture, which is characterized by control-oriented processes (stability) and internal focus, is oriented toward security and routinization. It emphasizes toward control, stability, predictability, smooth operations and efficiency through following specific rules and regulations.
As the opposite ends of each dimension in Figure 2.10 represents a conflicting demands on the organizations, a group organizational culture is contrasted with a rational organizational culture, and a developmental or Adocracy organizational culture is opposed by a hierarchical organizational culture (Quinn and Rohrbaugh 1983; Iivari and Huisman 2007). According to Iivari and Huisman (Iivari and Huisman 2007), "*CVM stresses a reasonable balance between the opposite orientations, although some cultural types may be more dominant than others. This imposes paradoxical requirements for effective organizations*".

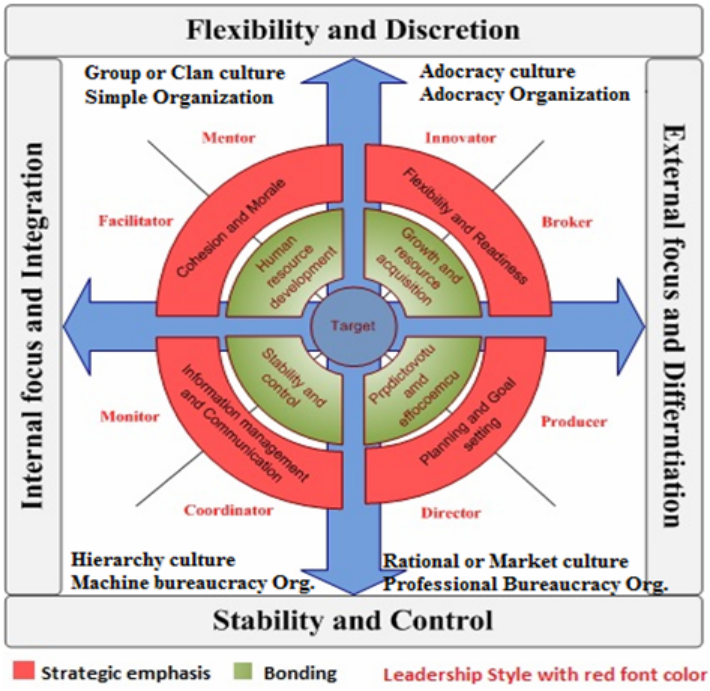


Figure 5a. Competing Valued Model for organizational cultures. (adapted and reproduced from Figure 5b).

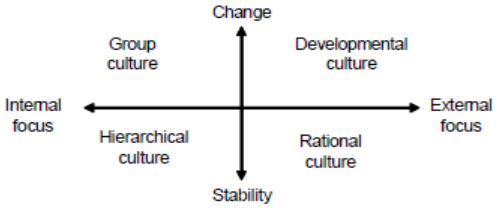


Figure 5b. Competing Valued Model for organizational cultures [21] p.37.

Figure 2.10. Competing Valued Model for organizational cultures.

2.3. Implications for Software Engineering

Frederick Brooks asserted and argued that within ten years from the paper's publication in 1986 "*There is no single development, in either technology or management technique, which by itself promises even one order of magnitude improvement in productivity, in reliability, in simplicity*" (Brooks 1995). Therefore, he tried to see why, by examining the accidents (its implementation process) that could happen in the software development and by checking the properties of his own proposed bullets which address the essential parts of software tasks (essence is the mental crafting of conceptual construct).

2.3.1. Irreducible essence of modern software systems

He considered the complexity, conformity, changeability and invisibility as inherent properties of irreducible essence of modern software system (Brooks 1995). Since that the software entity does not scaling-up in repetition manner of the same elements in a linear fashion, the **complexity**, which is an essential property and not an accidental one (accidental does not mean occurring by chance, but more nearly incidental or appurtenant), increases nonlinearly and thereafter causes a "*difficulty of communication among team members, which leads to product flaws, cost overruns, schedule delays… much less understanding… unreliability… comes the difficulty of extending programs to new functions without creating side effects…*" (Brooks 1995).

The **conformation** to other interfaces is the source of much complexity which can not be simplified out by redesigning the software. Since that the software product is affected by many factors (such as users, laws and other applications) that change continually, these **changes** enforce changing the software product.

Software is **invisible** and unvisualizable; the attempting to diagram the software structure produce a nested general graphs that superimposed one upon another, according to Brooks, "*The several graphs may represent the flow of control, the flow of data, patterns of dependency, time sequence, name-space relationships*. These are usually not even planar, much less hierarchical". In spite of that, the software product remains inherently unvisualizable (Brooks 1995).

2.3.2. The fruitful steps in software technology

After his examination of the three most fruitful steps in software technology in the past (High-level languages, Time-sharing and Unified programming environments), he found that each one of them attacks a different major difficulty in building software, these difficulties have been the accidental not the essential ones (Brooks 1995).

High-level languages are powerful techniques for increasing software productivity, reliability, and simplicity that frees the software from accidental complexity. It enables the construction of an abstract program where the level of sophistication "*about data structures, data types, and operations is steadily rising, but at an ever-decreasing rate*" (Brooks 1995).

Time-sharing has been credited by most observers with a major improvement in the productivity of developers and quality of the product, but less than the high-level languages. According to Brooks, "Time-*sharing preserves immediacy, and hence enables us to maintain an overview of complexity*".

Unified programming environments; since that UNIX and Interlisp provide integrated libraries, unified files formats and pipes and filters, they improved the productivity by integral factors and they attack the accidental difficulties. Hence, these environments stimulated the development of whole tool benches by using a standard formats (Brooks 1995).

2.3.3. Potential silver bullets.

Frederick Brooks has discussed the technical developments (such as; high level language advances, Object-oriented programming, Artificial intelligence, Expert system, Automatic programming, Graphical programming, Programming verification, Environments and tools, and Workstations), as potential silver bullets, by addressing their problems, by checking if these problems are essence or accidental difficulties, and if they offer revolutionary advances or incremental ones. As a result of that, he came to address the essential parts of the software task that have great complexity. Thus, he suggested:

1. **Exploiting the mass market to avoid constructing what can be bought**.
As long as buying off-the-shelf package or software is cheaper than to build a new one in house, it is recommended to exploiting the mass market by using the available products that satisfies the customer's needs, especially when the requirements are not too specialized, the delivery is immediate where the product is well documented and better to maintain than the homegrown one.

2. **Using rapid prototyping as part of planned iteration in establishing software requirements.**
Since establishing the detailed technical requirement, that is used to decide precisely what to build, is the hardest part of building software, the most important way to develop software is the iterative extraction and refinement of the product requirements. On the other hand, since complex software system is hard to imagine, it is necessary using extensive iterations in order to allow continuous planning and to arrange between the client and the designer to define the required system. Thus, the client can have a continuous prototype of the system that enables the testing for consistency and usability purposes.

3. **Growing software organically, adding more and more function to systems as they are run, used and tested.**
If the software is too complicated to specify its details accurately in advance, then we must take different approach of development. Harlan Mills (as it has been cited by Brooks in (Brooks 1995)) suggested building the software incrementally by adding more and more function to systems as they are run.

4. **Identifying and developing the great conceptual designers of the rising generation.**
According to Frederick Brooks (Brooks 1995), "*We can get good designs by following good practices instead of poor ones. Good design practices can be taught*". Building software is a creative process that requires a great design that comes from great designers to get success. According him (Brooks 1995), "*the very best designers produce structures that are faster, smaller, simpler, cleaner, and produced with less effort*". He proposed some steps to how to grow great designers in (Brooks 1995).

2.4. Potentials in Software Engineering

After nine years from the assertion and argument that there is no single software engineering development will produce an order-of-magnitude improvement in programming productivity, Brooks had argued against that at 1995. He had cited that Herzberg, Mausner, and Syderman in 1957 found that the motivational factors can increase the productivity, while environmental and accidental factors can not, but they will decrease the productivity if they were negative (Brooks 1995).

Brooks recognized that indeed crafting the conceptual construct of software have inherent difficulties from complexity, conformity, changeability, and invisibility. He noticed that the most inherent difficulty is the complexity that immerges from the arbitrary

complexity of the system that contains myriad details; those details are hard to be specified exactly. Thus, the complexity can be reduced by adding the necessary complexity to the software by: **Hierarchically**, by layered modules or objects. **Incrementally,** so that the system always works (Brooks 1995).

2.4.1. Productivity

It has been argued by Coqui (cited by Brooks (Brooks 1995)) that since there is a lack of systematic quality controls, that could lead to disasters such as schedule delay, rather than productivity concerns, the systematic software development disciplines were developed. Kindly Coqui said; "*The driving force to use Software Engineering principles in software production was the fear of major accidents that might be caused by having uncontrollable artists responsible for the development of ever more complex systems*".

Defining and finding the productivity numbers are very hard, but using workstations and software tools could get the improvement fold for many times (Brooks 1995).

Quality, product performance, and support cost are the most important issues of selling packages of the mass-market software, rather than the development cost. On the other hand, the development cost is so crucial for custom system which is developed whether in-house or out-house. Therefore it is recommended to buy and not to build (Brooks 1995).

Buying off-the-shelf instead of building it is considered a most dramatic way to improve the productivity of management information systems (Brooks 1995). Thus, immense flexibility can be given by integration off-the-shelf application into a compatible one that suites the needs. Such tools are recommended to be used by end users, not the professional, not the software people who will lose the essential features that could be very important (Brooks 1995).

2.4.2. Object-Oriented Programming (O-O Programming)

Object-Oriented programming is a discipline that enforces modularity, encapsulation, inheritance, hierarchical structure of classes, and strong abstract data-typing. According to Brooks (Brooks 1995), "*The attractiveness of object-oriented approach is that of a multivitamin pill: in one fell swoop (that is, programmer retraining), one gets them all. It is a very promising concept*".

Object-Oriented programming grown slowly because of its tied to a variety of complex languages. Unfortunately, people learn it as a use of a particular tool instead of learning its design principles. As a result of that, people produce bad designs by using these languages with a little value (Brooks 1995). Therefore, the benefits of retraining the programmers to think in a quite new way are vast and worthy, even though costs might be large (Brooks 1995).

2.4.3. Reusability

The program reuse is considered a one of the strongest attractions of O-O programming that attacks the essence of building software. According to Jones as it has been cited by Brooks (Brooks 1995), usually programmers reusing their own handiwork with about 30 percent reused code by volume, but the reusability at the corporate level aims fro 75 percent, which requires a special libraries and administrative support.

The reusable modules are generic functions that would be used in different modules or components of the product (Brooks 1995). Therefore, they should have a very good design and documentation in order to use it frequently and easily. It has been confirmed that building reusable modules take a big expense, it takes threefold the effort of 'one-shot' component according the estimation of the effort ratio of Brooks (Brooks 1995).

2.4.4. Unchanged Position

As a summarization of Brooks views in 1995, the complexity is the most important factor for our business limitation. Therefore, the practitioners should examine evolutionary and incremental improvements to software productivity rather than to wait for revolutionary ones (Brooks 1995).

3. Theories Applied.

A substantial economic activity is comprised by software development. Yet, software projects have high failure rates. According to Ralph and Wand (Ralph and Wand 2008), "*It is estimated that in 2004, 18% of projects failed outright and 53% of projects were "challenged," i.e., were delivered over budget, late, or with a reduced feature set*". Therefore, some attention should be paid to the employed process of developing software.

Since that the process, which is used to develop software, contributes to project success. Organizations employ numerous processes to perform the needed tasks in projects development. The employment of these processes depends on the organizational structure, organizational culture, and on the managerial strategies that are used to perform the tasks. These processes may support other processes and helps the organization to develop the needed products.

After an extensive research; we provided below hypothesis which are considered as a starting point to guide the organization in the software development, and there should be a continuous attention paid to assess, improve and tailor the needed processes according to different circumstances in the organization.

3.1. Hypotheses

3.1.0. Hypothesis H0: Using standards of software development contradicts human behavior (creativity, improvisation and ability to adapt), thus standards does not guarantee qualitative software.

In practice, there are a numerous of methodical choices that software developers can use, they can choose to follow one of them completely or partially, or combine the needed aspects of different methods. Therefore, Paul Ralph and Yair Wand noticed that "*neither existing methods nor existing process models explain the full spectrum of software development phenomena*" (Ralph and Wand 2008).

According to Cockburn (Cockburn 2002), "*Core to agile software development is the use of light-but-sufficient rules of project behavior and the use of human- and communication-oriented rules*". Kent Beck includes humanity as a principle of XP, since that Agile puts a great focus on the humanity of the team members where each person should be treated fairly as a human being. He points out (Beck 2005): *"People develop software. This simple, inescapable fact invalidates most of the available methodological advice. Often, software development doesn't meet human needs, acknowledge human frailty, and leverage human strength. Acting like software isn't written by people exacts a high cost on participants, their humanity ground away by an inhumane process that doesn't acknowledge their needs. This isn't good for business either, with the cost and*

disruption of high turnover and missed opportunities for creative action". His definition to the developer needs in somehow is like Maslow's hierarchy of needs.

Abraham Maslow had created a well known hierarchy of needs that is illustrated in Figure 3.1 (Poston and CST 2009). These needs are listed in a hierarchical order in the form of a pyramid, the basic needs, which are at the bottom of the pyramid, must be met before the higher order needs. In order to be a good developer, people need to have:

1. Physiological: the basic needs, such as; food, water, shelter and etc.
2. Safety: the basic safety from immediate danger, such as; hunger, threats to loved ones, and fear of job losses.
3. Belongingness and love: the ability to identify with and be part of a group, to have close friends to confide with, and contribute to shared goals.
4. Esteem: The need for recognition, accomplish some meaning work to other people, and feeling of moving up in the world.
5. Self-actualization: the ability to know exactly who you are, where you are going, and what you want to accomplish. Thus, the employee can provide creative activities.

By encouraging developers and raising the satisfaction level of the Maslow's hierarchy of needs, a positive effect on the quality of the software can be produced by developers. Beck mentioned that "*Everybody wants to do a good job, and they work much better if they feel they are doing good work. If you deliberately downgrade quality, your team might go faster at first, but soon the demoralization of producing crap will overwhelm any gains you temporarily made from not testing, or not reviewing, or not sticking to standards*" (Beck 1999). Therefore, encouragement plays positively on producing qualitative software by allowing creative activities to be introduced by the developers. On the other hand, having a set of values that support the long-term social goals such as encouragement beside the good communication and continuous feedback is not enough to have a high quality. Therefore, comes the need of backed up these values by myths, rituals, punishments, and rewards (Beck 1999). If we did not do so, the employees will focus on the short-term goals.

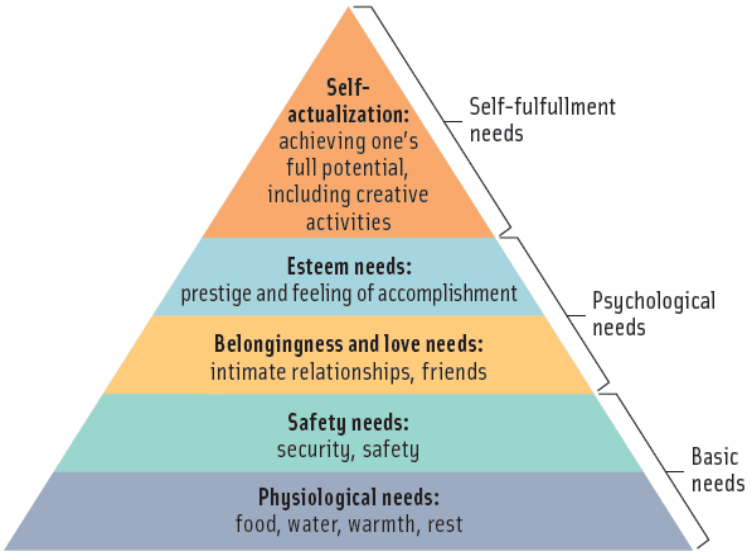


Figure 3.1. Maslow's Hierarchy of Needs (Poston and CST 2009).

Certain moral values should be embraced in order to have moral or ethical standards that keep all stakeholders working in an optimum way. Thus, people can autonomously emphasis on the quality of the software by working creatively.

3.1.1. Hypothesis H1: Larger and distributed teams may decrease the communication and the information transformation by having regulations and rules that emphasize on an enhanced software processes and on using infrastructures as a supplementary strategies, and thereby by using a stricter coding and testing standards that supports the development teams.

Usually team members get immediate feedbacks from their peers when all members are exist in the same location or by having a pair programming where the code is written by two people at the same time by using one computer (XP) (Steinberg, 2004;Abrahamsson, 2002 ;Sharp, 2007). On the other hand, having a larger team, which requires dividing it into smaller or distributed teams, slows down getting the feedback and slows down re-factoring cycle, because of many challenges that may arise, such as the lack of physical presence (Young and Terashima 2008).

As mentioned before in chapter three, one of the benefits of agile software development is to respond quickly to changes and provide a quick feedback to all parties (agile principle 2 in chapter 3.3) (Highsmith and Cockburn 2001). In order to provide this benefit for larger or distributed teams beside the need for enhancing the software-engineering activities which reduces the delay and provide a good way of communication that does not affect the productivity and the quality of their products, they should pay attention to the software processes and focus on infrastructure as supplementary strategy (Aaen, Bøttcher et al. 1997).

3.1.1.1. Hypothesis H1a: The Professional Bureaucracy organizations may decrease the need for communication and the information transformation by using optimal software processes based on experience and by using infrastructures as supplementary strategy.

The Professional Bureaucracy organizations emphasize on decentralized decision making and require specialists with high skills, and depend upon efficient performance of standardized but complex work. The strategy of these organizations, that have rational culture, emphasize toward competitive advantage and market superiority (Quinn and Rohrbaugh 1981).

By focusing on developing optimal software processes based on experience, the organization can have efficient and effective processes with less rework, more reuse and with self-improving (Aaen, Bøttcher et al. 1997).The approaches that support these objectives are the Experience-based Component Factory and the Mature Software Organization (Aaen, Bøttcher et al. 1997).These two approaches are suitable for the organizations that have the Rational culture and thereby to the Professional Bureaucracy organization.

As mentioned before in chapter three, the lack of control, which specialists can exercise in professional bureaucracy organizations, is considered as a disadvantage in such organizations, because of spreading down the power and decision making through the hierarchy. Therefore, using infrastructures as supplementary strategies that can help on developing environments in which processes and professionals are better supported (Aaen, Bøttcher et al. 1997).

3.1.1.2. Hypothesis H1b: **The Machine Bureaucracy organizations may decrease the need for communication and the information transformation by using infrastructures based on automated tools for conveying the information quickly and for providing a high degree of control in order to ensure a standardized performance.**

The Machine Bureaucracy organizations emphasize on standardized processes, centralized decision making at the strategic apex, and on having a formal channels for the communications. The strategy of these organizations, that have Hierarchical culture, emphasize toward stability and predictability by taking on consideration the rules and procedures to reach the defined expectations (Quinn and Rohrbaugh 1981).

By focusing on the creation of an infrastructure to support the software process, the organization can increase the quality and the productivity of the development and it can produce a framework for Integrated Software Development Environments (ISDEs) as basic building blocks which support environments tailored to the tasks that should be performed by users in specific projects (Aaen, Bøttcher et al. 1997). The approaches that support these objectives are the Industrialized Software Organization and the Generic Software Factory (Aaen, Bøttcher et al. 1997). These two approaches are suitable for the organizations that have the Hierarchical culture and thereby to the Machine Bureaucracy organization.

3.1.2. Hypothesis H2: Developers should be able to communicate continuously but meetings should be held when it is needed. Pair programming should only be used for training and for investigation when it is needed. Senior developers should have the authority to contact the customer regarding their tasks without having a specific customer representative (Customer representative is a developer that has the role contacting the customer directly), **unless there were a situation that prevents all developers from being on direct contact with the customer.**

As mentioned in chapter three, Cockburn and Jim Highsmith realized that agile software development teams considered as a quick responsiveness to change by reducing the cost of moving the information between people which can be don by placing people physically closer, by using face-to-face communication with using whiteboards instead of documentation and by improving the team's amicability (Highsmith and Cockburn 2001). Figure 3.2. illustrated the effectiveness of different modes of communication

Face-to-face communication is the most efficient way of conveying information to and within a development team (agile principle 6 in chapter 3.3). According to Steve McConnell in (McConnell 1998), "*The most difficult part of requirements gathering is*

not documenting what the users 'want'; it is the effort of helping users figure out what they 'need' that can be successfully provided within the cost and schedule parameters available to the development team.". Therefore the stakeholders and software developers should have a healthy conversation about the product features and customers requirements, so they will not be in chaotic situation about the product they are going to develop. In-case the stakeholders are not available for face to face meeting, they should use electronic or other suitable media for communication (Ambler 2002).

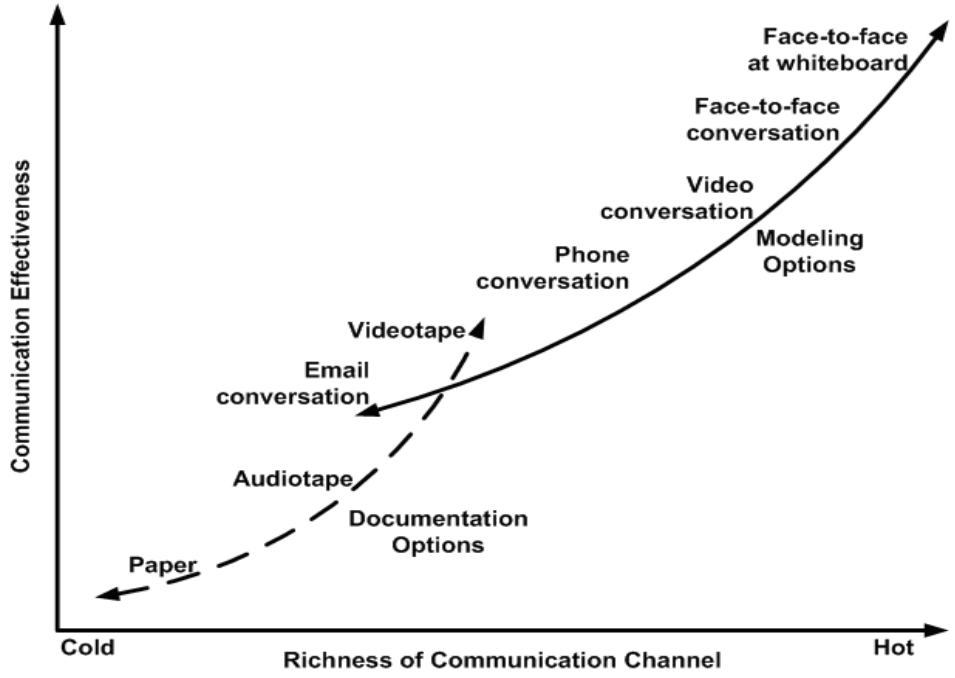


Figure 3.2. Effectiveness of different modes of communication (Cockburn 2002).

Based on the Competing Values Model CVM that has been described in chapter three, the Adhocracy organizations that have a Developmental culture, which emphasize toward growth, flexible processes, resource acquisition, innovation, and adaptation to the external environment, promotes the agility, therefore it is recommended for senior developers to be on direct contact with the customers. In contrast, the organizations that have a Hierarchical cultures, which emphasize on the integration of the socio-technical systems and on stability (control oriented processes), degenerate the agility because of their structure where the communication inside them tends to follow formal channels and the employees have a specific roles, therefore they prefer to have a customer representative.

Pair programming has positive effects that can be gained, such as spreading the information, skills spreading and better code quality. On the other hand, it has some negative effects on both developers and organizations, since that most programmers prefer to work by themselves; they have some negative feeling about pair programming (the sense of autonomy), there maybe a personality conflicts between pairs such as a bad communication between them, and pair programming consumes resources; which implies

to take longer time to complete projects. This impose the needs of having more programmers in order finish the project at the deadline, thereby the organization will be concerned about the costs that are associated with the pair programming. Therefore, Pair programming should only be used for training and investigation issues when it is needed.

3.1.3. Hypothesis H3: Specifying the software type and the characteristics of that software, besides specifying the suitable development model, should be used in order to help the organization to choose the most suitable processes and practices that help to improve the performance, productivity and the quality of the products.

The management team can specify the type of software which is ordered by the customer (i.e. new software, reuse of existing software, a stand alone software, or customized software) by considering the products they have, thus the software can be considered as a new software that needs to be developed from scratch, a reuse software which means using an existing components from the products that the organization have and reuse them to develop the needed one, a customized software by modifying an existing software according to the customer's need, or a combination of different kinds (Singh 1995).

By specifying the project characteristics (such as: product size, criticality, complexity, and etc) beside the software type, the appropriate development model can be specified. There are different software development models, such as the Waterfall, incremental evolutionary and the spiral model. Sometimes, a project needs a combination of these models for some phases or for the whole development life cycle (Singh 1995). The software lifecycle models mentioned here are identified as a starting point. More information about these models and about safety-critical lifecycle models can be found in (Boyd, 2009;USDA, 2007 Singh, 1995;DeGrace, 1990;May, 1996;ISO, 1998).

Iivari and Huisman conducted a survey at 2007 on the relationship between organizational culture based on the CVM and the development of systems development methodologies (Iivari and Huisman 2007). As a response for their survey, they received completed IT manager questionnaires from 73 organizations and completed IS developers from 234 developers from 71 organizations. As a result of their survey, they noticed one striking finding, which is the positive relationship between the hierarchical culture orientation and the deployment of software development methods in the case of IS developers. They noticed also positive relationship between the developmental culture and the deployment of software development methods, but not systematically. On the other hand, they found that the IT managers realized that the degree of supporting software development methods and their impact become more critical when cultural orientation become more and more rational, while the rational cultural orientation was negatively associated with the deployment of software development methods from the perspective of IS developers.

The use of the **Waterfall model** should be limited to the projects that have a very well understanding of the requirements and the implementation on advanced. According to **Blum (Bruce 1992), "***if a company has experience in building accounting systems, I/O*

*controllers, or compilers, then building another such product based on the existing designs is best managed with the waterfall model***”.** Figure 3.3 illustrates its life-cycle.

Figure 3.3. Waterfall model’s life-cycle (May and Zimmer 1996).

When the organization can not take the risk of developing the whole system at one, the **Incremental model** should be used. According to Whitgift (Whitgift 1991), "*If it is too risky to develop the whole system at once, then the incremental development should be considered*". It combines elements of the Waterfall model with iterative philosophy of prototyping. The first increment is often represents the core product, while the subsequent iterations used to produce the supporting functionalities that a customer would like to see. More details about the Incremental model can be found in section 4.3.3. (Hypothesis H4). As mentioned in chapter three, Cockburn and Jim Highsmith realized that agile software development teams considered as a quick responsiveness to change by reducing the elapsed time between making a decision to see the consequences of that decision can be done by having the experts available to the team and by producing the work incrementally (Highsmith and Cockburn 2001).

Evolutionary models is an iterative and incremental approach to software development that enables the increasingly development of more complex versions of the software. Figure 3.4 illustrates its life-cycle. It is applied to the systems for which the requirements are not well understood even though the need for the system is understood and approved. As a first step, the requirements for the system are partially defined, then refined iteratively with each successive version. A significant reduction in risk for software development can be recognized by using this model; according to Elaine and Barbara (May and Zimmer 1996), this risk might be associated with “*missing scheduled deadlines, unusable products, wrong feature sets, or poor quality. By breaking the project into smaller, more manageable pieces and by increasing the visibility of the management team in the project, these risks can be addressed and managed*”.

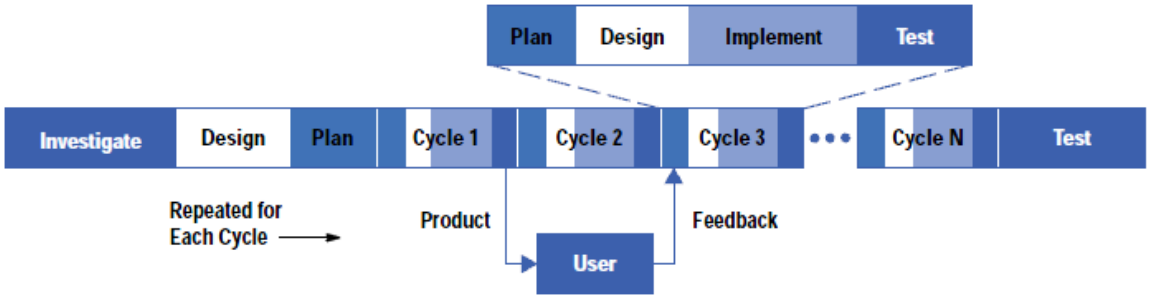


Figure 3.4. Evolutionary model’s life-cycle (May and Zimmer 1996).

The development of large systems in-house that are risky in nature and those that require much computation (i.e. Aerospace, Defense and Engineering) the ultimate waterfall model which is the **Spiral model** is recommended to be used. It is an Evolutionary model that combines the iterative way of prototyping with the systematic aspects of the

Waterfall model. DeGrace in (DeGrace and Stahl 1990) believes that "*the spiral model actually is applicable to many business applications, especially those for which success is not guaranteed or the applications require much computation, such as in decision support systems*". The first iteration in the Spiral model could produce a prototype or artifact model for the system (unlike the Incremental model, where the first iteration used to produces the core product), and the subsequent iterations used to produce the supporting functionalities that a customer would like to have. The spiral model is a special case of the evolutionary model when the full requirements are not known at the start for development portion of each version. Figure 3.5 illustrates the life-cycle of the spiral model.

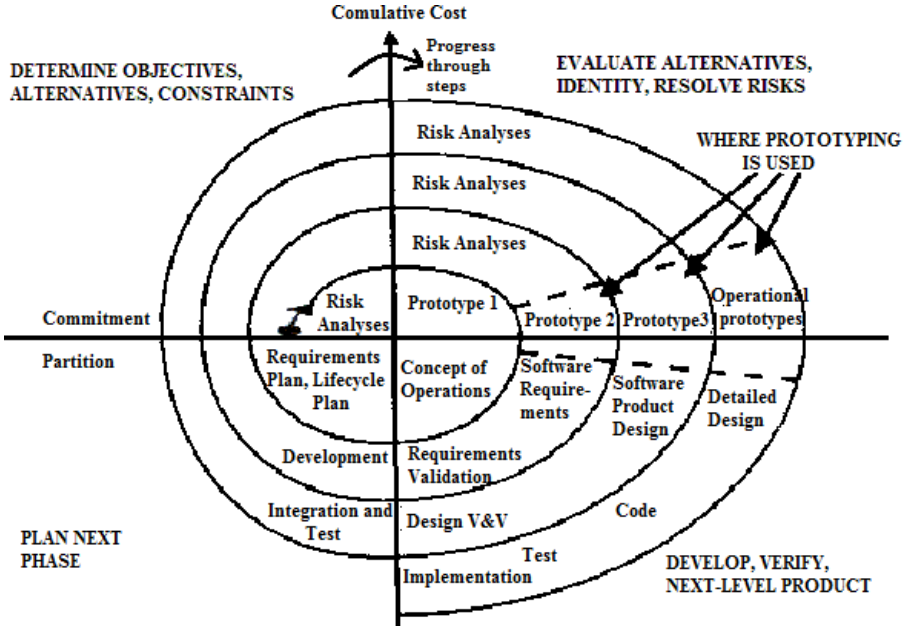


Figure 3.5. Spiral model's life-cycle (DeGrace and Stahl 1990).

3.1.4. Hypothesis H4: Iterations of software development should be flexible and early releases of usefully working software provide all parties a good indication of progress and can provide the customer with the ability to gain the competitive advantage where each release can take a place in the market.

Agile methodologies recommended different length of iterations, these lengths varies from one week to several months. I.e. XP's iteration takes from one to three weeks, Scrum takes from one to four weeks, DSDM takes from two to six weeks, Crystal takes from one to three months and etc. According to the Agile Alliance (Cockburn 2002), "*We are uncovering better ways of developing software by doing it and helping others do it. We value….Working software over comprehensive documentation*". Therefore, working system can be considered as the only thing that shows what has been built and describes the progress. Running code is ruthlessly honest (Cockburn 2002).

From the project manager's view, According to Markula (Markula 2002), "*agile projects resemble more likely chaos instead of control within the iterations. This is mostly because*

the team members have to be empowered for making many, also critical, decisions". Therefore, a constant change and many corrective actions, which mostly are small and can be done quickly, can be taken in fast-paced agile software projects. The way of doing corrective actions in agile methodologies is managed in different ways. In XP, the critical decisions should be taken by the customer, since it has been recommended to have on-site customer (Markula 2002; Steinberg and Palmer 2004). On the other hand, Scrum, Crystal Clear and ASD recommend having regular reviews to improve the product and the methodology (Highsmith and Cockburn 2001).

Incremental Model is a software process model that has a structured set of activities required to develop a software system and is considered as a Delivery Strategy software lifecycle mode (Boyd 2009). It performs the waterfall model in overlapping builds attempting to compensate for the length of projects by providing earlier releases which have usable functionalities (Whitgift 1991; USDA 2007), see Figure 8. The operation and support of each release is done in parallel with the development, operation and support of successive releases (the several parallel implementations of the system can be seen in Figure 3.6).

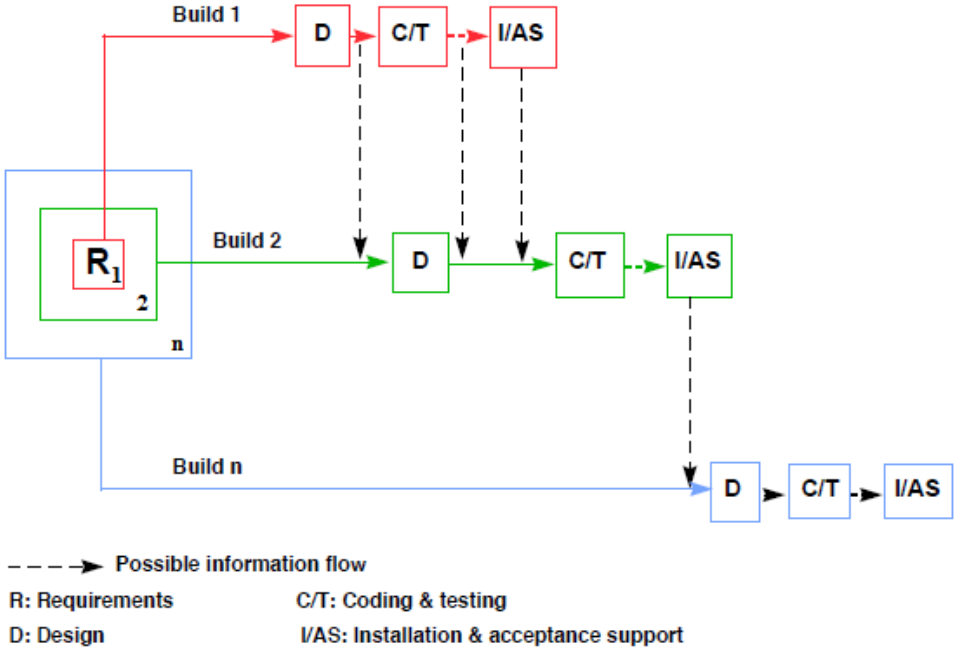


Figure 3.6. Incremental Model (ISO 1998).

This model applied to enterprise projects that introduce new releases of a product at *regular or pre-planned intervals*. A complete upfront set of requirements may be involved to be implemented in a series of small projects, *or* some portion of the complete defined objectives of the project may be implemented as a first release and the rest of them can be implemented in different releases (Whitgift 1991; USDA 2007). Thus, companies can satisfy customer needs by providing early capabilities and can provide the customer with the ability to gain the competitive advantage where each release can take a place in the market. Also, the incremental model can enable the exploration of the possible market place, even though the early releases have limited capabilities.

Since that the incremental model provides the releases or versions of the system at *regular or pre-planned intervals* according to the management team, the iterations of software development can be flexible and the customer can have a version of the system that contains the functionalities that have been agreed on before.

Since that the incremental model implements a portion of the requirements as a first version, the customer may be unsatisfied with the limited set of capabilities in the initial releases or versions of the system, thus the customer loses the interest of buying the next version. The management team may be uncomfortable with the use of general or some objectives rather than the complete requirements, because of selecting some modules to developed before others and there can be a tendency to push difficult problems or issues to be solved in future (Whitgift 1991; USDA 2007). It is not recommended to have too short intervals of new releases unless the customer needed that, because of the possibility of customer's dissatisfaction with the cost of upgrade and retraining.

3.1.5. Hypothesis H5: The documentation content should be comprehensive and should be minimized to an extent that satisfies the current and future need of operational and support personnel to be useful and helpful. The documentation process should be synchronized with the development process.

According to the Agile Manifesto (Cockburn 2002), "*We are uncovering better ways of developing software by doing it and helping others do it. We value….Working software over comprehensive documentation*" where the value of the documentation is less than the working software. The documentation usually contains the requirements, analysis, design, object interaction sequences charts, and etc. The team uses them as aids that help on guessing or planning for the future or to maintain and change the system according the customer's request in future. According to Cockburn (Cockburn 2002), "Documents can be very useful, as we have seen, but they should be used along with the words "just enough" and "barely sufficient.". On the other hand, from Markula's planning perspective (Markula 2002), she thinks that in most cases this does not hold true for project planning, since Cockburn's purpose is to emphasize on communication, not to write documents and since changing project manager in the middle of the project will lead to a catastrophe.

The existed knowledge inside the documents is a small part of what the team (who participated in the project) has. According to Cockburn (Cockburn 2002), What Highsmith means by saying, "*Don't confuse documentation with understanding*," is that much of the knowledge that people needs and that is bound to the project is a tacit knowledge that people have inside them, not inside the documentation. He noticed that even with a best documentation, the new team can not pick up from where the previous one left off. Therefore the new team needs to build the necessary tacit knowledge base in order to be productive and to start making progress (Cockburn 2002).

When writing the documentation, all stakeholders who are involved in should determine the needs of that documentation in order to produce a comprehensive one that will help all parties in future and present. Its extent should considered upon the project size, the needs of the stakeholders, and how important to have such details in the documentation.

It is important to proceed in preparing the required documentation during the development process. This synchronization of process can help to put in the needed knowledge that can in future and even the new employees, but it should be "just enough" and "as sufficient as needed" for the different kinds of parties. This synchronization of process can save cost and time, since it may take longer time to produce the documentation at the end of the project, some important issues can be forgotten, and by this way we can avoid unexpected risks.

3.1.6. Hypothesis H6: Top level management should pay attention to the continuous assessment and improvement of these guidelines (hypotheses) according to the circumstances.

The management team needs to establish the required infrastructure according to the organization's processes and structure. The infrastructure and processes need to be assessed continuously in order to find out the applicability of these processes with the organization's environment and to evaluate the performance. Thus, the management team can make the required enhancements or changes on the organizational level in order to improve their processes. Figure 3.7 adapted from ISO/IEC 12207 stander (Singh 1995), it depicts the activities of improving processes.

According to Clarke and Manton (Guler 2010), "*Many companies tend to focus on the change process rather than the key factors of success behind it. However, it is not just what you do but how well you do it* ". Some of the key success factors that need to be considered in order to produce a high performance are: communication, participation, motivation, commitment, leadership and training, more information and description of these key factors can be found in (Guler 2010).

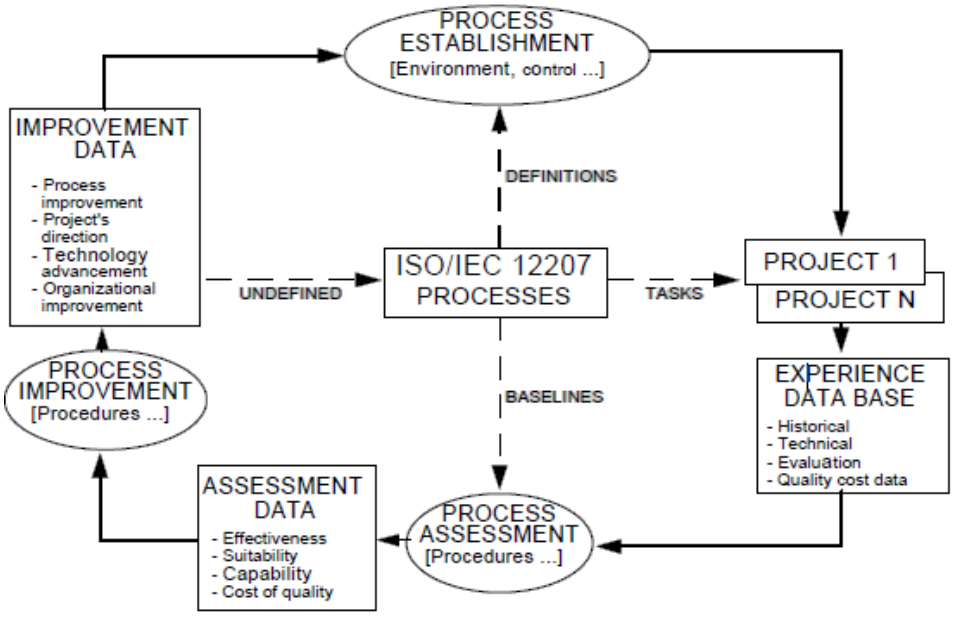


Figure 3.7. Process improvement (ISO/IEC 12207 stander (Singh 1995)).

Significant management activities should be paid to larger and critical systems, while less management activities should be paid to smaller and non-critical systems in order to reduce the cost of employing the management activities on both time and money. More information about safety-critical systems can be found in (Boyd 2009).

3.2. The relationship between the hypotheses

Since that many aspects of the Software Engineering are human-centric and people thinking in a different ways to solve and to manage issues, we are considering the first hypothesis (**H0**) as the main one and the core of all of the others. It takes the human behavior and issues as the essential factor for software development instead of using standards of software development which do not guarantee qualitative software.

On the other hand, since the communication and continuous feedback are considered from the main values of supporting long-term social goals, and the supporting of long-term social goals is needed besides backing up its values by myths, rituals, punishments, and rewards in order to have qualitative software (as mentioned at **H0**). Therefore, we had proposed the **H1** by considering the communication in larger and distributed teams since they are facing more communication issues and problems than the smaller once. Thereby, in **H2** we assumed that developers do not need for regular meetings since they can communicate continuously, but they can do so when it is needed. On the other hand, in H2 we argued that having paired programming has some negative effects on both developers and organizations; despite this we need it for training and investigation issues to improve the quality of the software.

In practice, there are a numerous of methodical choices that software developers can use, they can choose to follow one of them completely or partially, or combine the needed aspects of different methods. Therefore, we had proposed hypothesis **H3**, since specifying the software type (i.e. new software, reuse of existing software, a stand alone software, or customized software) beside the characteristics of that software(such as: product size, criticality, complexity, and etc) would help the organization on specifying the suitable development model (such as: Waterfall, incremental evolutionary, the spiral model, and etc) in order to choose the most suitable processes and practices that help to improve the performance, productivity and the quality of the products. As mentioned before, sometimes a project needs a combination of these models for some phases or for the whole development life cycle. Thereby, using an incremental way of development (Incremental Model), which has been suggested in hypothesis **H4**, should emphasize on using a flexible iterations instead of having a regular one, thus we can provide the customer with a useful prototype that can be used to get the required feedback from the customer and to be deployed or used in customer's market to gain the competitive advantages.

As soon as we specified the required processes that are needed to be used during the development of a specific project, we should synchronize writing the comprehensive documentations with the development processes (hypothesis **H5**). A comprehensive

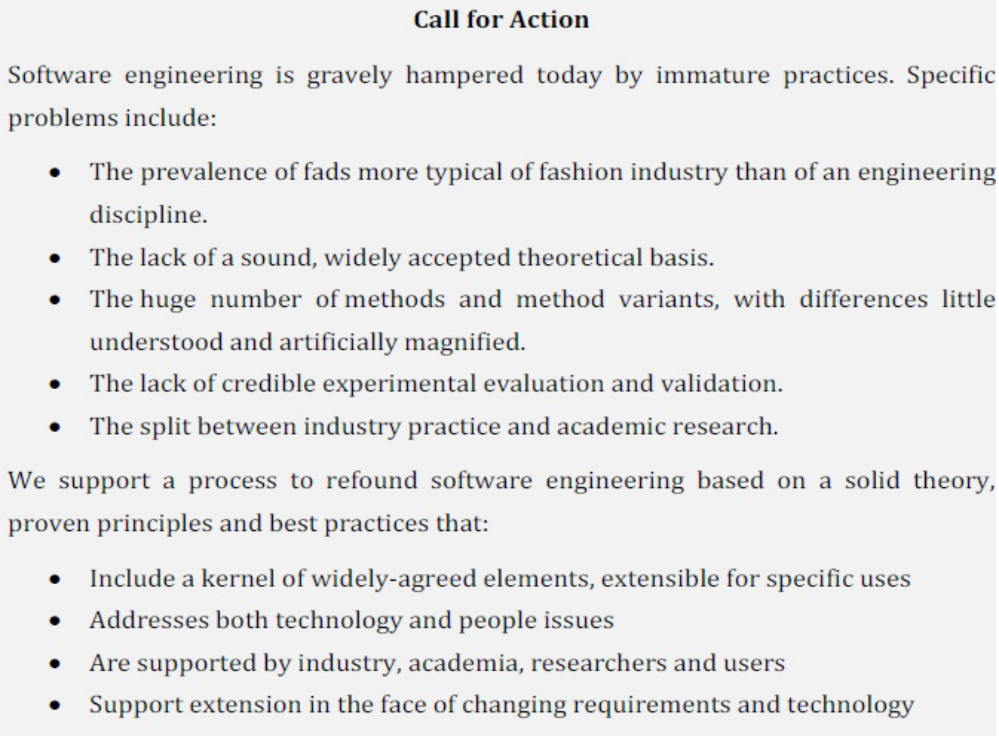
documentation would be used as an instrument that would help all parties to maintain, reuse, upgrade, change and improve the software in future.

As a support strategy to all of the software development processes and to all of the hypotheses, we had suggested the hypothesis H6 which would emphasize on paying the required and the continuous attention to assess and improve all the software development processes and hypotheses according to the circumstances that the management team may face during the software development.

4. Discussion
4.1 Introduction

The current state in Software Engineering and Development have some issues that need to be fixed and improved in a better way, such as having a mixture of practices, detailed processes have been described into books that people do not like to read, and difficulties of extending with process improvement approaches; such as the Capability Maturity Model Integration (CMMI) which is used to improve their performance which is very hard to adapt. Nevertheless, we have nowadays many proven practices such as; use-cases, components, architecture, Iterations, and UML that replaces too many modeling languages.

Jacobson, Bertrand and Richard mentioned that the software engineering is seriously obstructed by immature practices, mentioned the specific problems, and their initiative to solve them as you can see in Figure 4.1 (Jacobson, Meyer et al. 2010).



Call for Action

Software engineering is gravely hampered today by immature practices. Specific problems include:

- The prevalence of fads more typical of fashion industry than of an engineering discipline.
- The lack of a sound, widely accepted theoretical basis.
- The huge number of methods and method variants, with differences little understood and artificially magnified.
- The lack of credible experimental evaluation and validation.
- The split between industry practice and academic research.

We support a process to refound software engineering based on a solid theory, proven principles and best practices that:

- Include a kernel of widely-agreed elements, extensible for specific uses
- Addresses both technology and people issues
- Are supported by industry, academia, researchers and users
- Support extension in the face of changing requirements and technology

Figure 4.1. SEMAT's Call for Action (Jacobson, Meyer et al. 2010).

This idea must be encouraged and improved by most of greybeards of the software engineering community in the world and should be approved by the industry to take it seriously. Cooperation is required from all stakeholders (from industry and academia) to formalize the proposed Kernel in a useful and reliable way that is widely approved so that all stakeholders can benefit from.

On the other hand, since that some aspects of the Software Engineering are human-centric (i.e. requirement engineering) and people thinking in a different ways to solve and

to manage issues, it might be hard to use their solution widely in the industry or it might take longer time to have matured and implemented. The industry might face some problems in evolving their old system to SEMAT's solution or in adapting the suitable practices for them according where they have many unique conditions that require using or formulizing new practices and much upgrading and training, thus end up by a complex and old fashion. SEMAT might take longer time until their solution become mature, within this time it might be that a new solution or ideas formalized in other way. People do not like to read the detailed processes' books but they really need them to understand how and what they should do in their situation, therefore if we adapted SEMAT's solution, it might be hard to recognize that, or their solution will lead us to the old fashion by having many detailed books that describes how, when, where, and what should be used in each situation.

I think that SEMAT's call for action is a revolutionary way to solve the current issues in Software Engineering and Development (as mentioned before). Therefore, it would be better to have an evolutionary one instead of waiting to have that revolutionary solution. Thereby, we propose below stages of adapting Agile Practices to different Organizational cultures as an evolutionary solution that would help different kinds of organizations in software development.

4.2. Three-Stages to adapt agile practices to different Organizational cultures.

Agile Practices would improve the software processes. In order to apply Agile practices to different organizational cultures, we need to assess both of the current status of the organizational culture and the project that the organization will develop in order to obtain the agile practices (according to the Agile Principles) that are applicable to both of them. As depicted in Figure 4.2, it first assesses the current status of the organization. Secondly, evaluates and assesses the project which will be developed. Then, based on the both assessments, we provide the possible Agile Practices and guidelines that can be. Below sections describes these stages in detail.

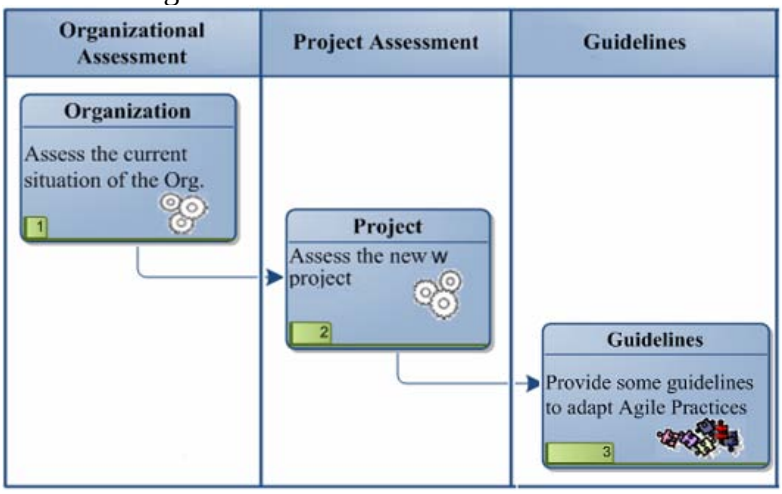


Figure 4.2. Three-Stages to provide the required Agile Practices according to the Organizational culture.

4.2.1. Organizational Culture Assessment

Since that the organization must be assessed to determine whether it is ready to adopt agile practices or not. We described the current situation of the different organizational cultures based on two categories (the current situation was based on Mintzberg). The first one was by considering the Agile Principles groups (by having five Agile Principles groups instead of the twelve) that has been prepared by Sidky and Arthur (Fujita and Zualkernan 2008) as you can see in Table 5.1. The second one was based on four dimensions (4-D) as you can see in Table 5.2 (these dimensions are; team size, proficiency of the team, complexity of the product, and structure of the organization).

Figure 4.3 depicts the current situation of different organizational cultures based on the two categories (by considering the Agile Principles and the 4-D).

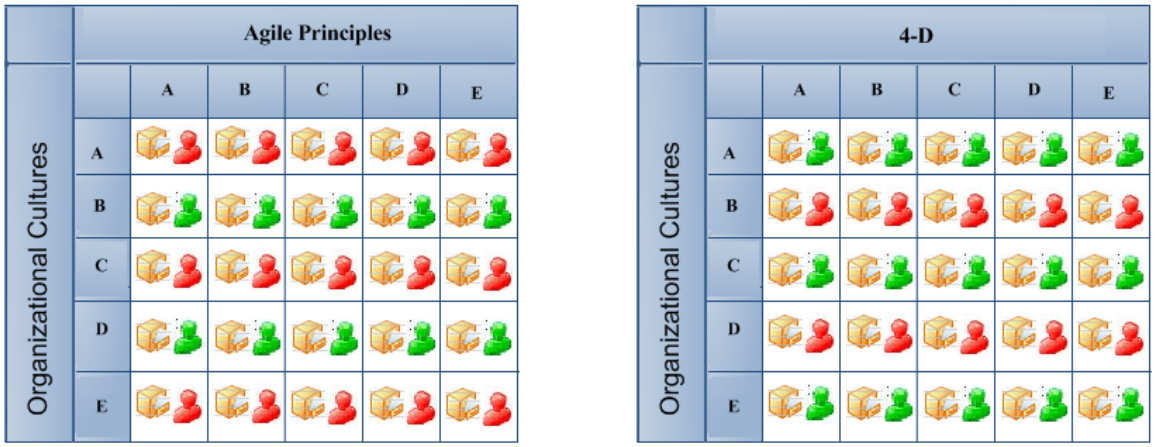


Figure 4.3. Current situation of the Organizational Cultures according to the Agile Principles and 4-D.

4.2.1.1. Organizational Cultures and Agile Principles

The agile principles grouped into five principles, after careful studying about agile software development principles and the agile manifesto, which capture the essence of the twelve. These five principles are:

1. **Technical Excellence:** produce the code with a high quality is essential in agile development environments (Highsmith and Cockburn 2001; Koch 2005).

2. Communication within the development team and with the customer: the communication and interaction between all the stakeholders of the project should be frequently in order to make sure that the development team will produce a product that satisfies the customer needs (Beck, Martin et al. 2001; Scott 2002).

3. Plan and deliver working software early and frequently: providing the customer with working software frequently and early is important to get the feedback that would

help on producing a plan for the next iterations (Beck, Martin et al. 2001; Scott 2002; Cohn 2005). An early release of incomplete software is a better indication of progress.

4. Support requirements change to produce valuable software: the purpose of developing software is to provide the customer with a valuable system that satisfies the customer's needs. By continuous attention to deliver working software at each iteration and by getting the feedback from the customer, the development team can gain more knowledge about the requirements that may arise because of the customer realization of additional values that emerged (Steinberg, 2004; Abrahamsson, 2002; Highsmith, 2001).

5. Human-centric: The reliance on how self-organizing do people intend and the interaction between them is the core of the agile processes (Cockburn 2002; Scott 2002; Fujita and Zualkernan 2008).

The relationship between these five principles and the organizational cultures is used to describe the current situation of the organization in order to help us on providing the guidelines later on. Table 4.1 illustrates the details.

4.2.1.2. Organizational Cultures and 4-D

The four dimensions are considered the essence factors for each organizational culture. By considering these dimensions we can gain more insight about the current situation of the organization, this insight would help us later on to provide some guidelines. These four dimensions are:

1. Team size: it describes which kind of grouping the team has and their focus area. Also it pinpoints the preferred size of the teams inside the organization.

2. Proficiency of the team: it represents the degree proficiency which is required to have in both of the individuals and for the whole team. On the other hand, it describes to which extend there is a specialization of jobs in each organization.

3. Complexity of the product: it provides us with insight about the systems that the organizations would usually develop; some organizational cultures are not suitable for developing sophisticated systems, wile others do. Also some organizational cultures are not suitable for developing complex and dynamic state systems, wile others do. And some organizational cultures are suitable to develop innovative products, but others are more suitable for producing standard of outputs.

4. Structure of the organization: It describes the coordination mechanism, the power if it is centralized or decentralized, the formalization of behavior, and usually how many layers are exists inside the organization.

The relationship between these 4-D and the organizational cultures is used to describe the current situation of the organization in order to help us on providing the guidelines later on. Table 4.2 illustrates the details.

	Agile principles				
	Technical Excellence (P7+P8+P9 + P10+12)	**Communication within the team & with the customer (P4+P6)**	**Plan & Deliver working software early and frequently (P1+P3)**	**Support req. change to produce a valuable software (P2)**	**Human centric (P5+ P11)**
Group culture (Simple Org.)	- Technical systems are not sophisticated and the power is delegated to technical staff. - Little specialization of jobs.	-Direct supervision. - Usually has functional grouping. - Its structure has a few layers and often has a large numbers of employees that reports to a single or few top managers.	- Little planning and control.		- Top managers exhibit bad decisions because of being overloaded by taking many decisions and by being on direct contact with all of the employees. - The organization is highly dependent on the individuals.
Developmental culture. (Adhocracy Org.)	- Has experts from different areas. -Much horizontal specialization of jobs. - High autonomy. - The managers should use the conflicts and aggressiveness as instruments to increase the productivity and not to loose these talent workers.	- Mutual adjustment. - Usually has functional and market grouping. - The power is delegated to wherever it is needed; thereby it has difficulty of controlling.	- Limited action planning.	- It can respond quickly to change, by providing the experts with the ability to meet new challenges.	- Decentralization of power makes it suitable for creative people. - Supports the enterprise agility. - People may exhibit some ambiguity of the authority which causes some conflicts -Stressful environment for the workers. - Spending more time on talking enables them to develop innovative ideas (cost of communication is high).

	Agile principles				
	Technical Excellence (P7+P8+P9 + P10+12)	**Communication within the team & with the customer (P4+P6)**	**Plan & Deliver working software early and frequently (P1+P3)**	**Support req. change to produce a valuable software (P2)**	**Human centric (P5+ P11)**
Rational culture. (Professional Bureaucracy Org.)	- Has specialists with high skills and knowledge. - Much horizontal specialization of jobs - Professionals resist the rationalization of their skills.	- Standardization of skills. - Usually has functional and market grouping. - Administrators and supporting staff follow machine structure, while professionals follow democratic way of work.	- Little planning and control.	- It can respond quickly to change, by providing the professionals with the ability to meet new challenges.	- Coordination and discretion problems maybe introduced because of its relying on standardization of skills where the decision making is decentralized. - Communication difficulties maybe emerged because of having incompetent professionals inside the team. - High autonomy of the professionals allows them to ignore the needs of both of the organization and the client.
Hierarchical culture. (Machine Bureaucracy Org.)	- It requires a minimum of skill and little training. - Much horizontal and vertical specialization of jobs. - The strategic apex is responsible on eliminating all possible uncertainties and conflicts. - Unsuitable for innovation (resists the change).	- Standardization of work. - Usually has functional grouping.	- Strong planning.	- Mechanistic organization resists the change, which makes it unsuitable for innovation.	- Some employees found it a boring way of work (routine work). - Prevent employees from being creative (does not provide the required human satisfaction) - Consider people as machines, but they aim for self-actualizing by being creative.

P1: Indicates Principle number 1 and so on for the rest. This numbering is according the agile principles that have been mentioned in chapter 3.

Table 4.1. The current situation of Organizational Cultures based on the Agile Principles categories.

	4-D			
	Team size	**Proficiency of the team**	**Complexity of the product**	**Structure of the organization**
Group culture (Simple Org.)	- Since this kind of structure tends to be found in new and small organizations that have lack of standardized systems, it has small team. - It is usually has a functional grouping. - Wide unit size.	- Since the organization is dependent on the individuals to achieve the required success, it requires employees with good skills because of developing technical systems which are not sophisticated and does not call for bureaucratization of the operating core. - It requires little specialization of jobs.	- Non-sophisticated systems.	- The organization has a flat structure and relatively unstructured - Highly centralized by having strong leaders. - Coordinates by direct supervision. - Its structure is flexible. - Its structure has a few layers and often has a large numbers of employees that reports to a single or few top managers.
Developmental culture. (Adhocracy Org.)	- It is usually has a functional and market grouping. - Narrow unit size. - The project teams must be small to encourage mutual adjustment among their members and to focus on a particular function or market.	- Brings in experts from different areas to form a creative team. Thus it supports innovation. (Much horizontal specialization of jobs). - Experts solve problems at anytime; they work in a highly flexible way, and can respond quickly to change. - High autonomy. - It uses exiting knowledge and skills to build new ones. - The managers should use the conflicts and aggressiveness as instruments to increase the productivity and not to loose these talent workers.	- Complexity is far too limiting. - This kind of organizations suites most operating environments that have complex and dynamic state.	-Its structure is decentralized with low levels of formalization, which implies to difficulty of controlling where the power is delegated to wherever it is needed. -It has a flexible communication lines (mutual adjustment) - Low level of formalization; The job description of the employees is broader and the responsibilities are specified according the organization's needs based on experience level of each employee.

	4-D			
	Team size	**Proficiency of the team**	**Complexity of the product**	**Structure of the organization**
Rational culture. (Professional Bureaucracy Org.)	- It is usually has a functional and market grouping. - Wide unit size at the bottom. (The labor is divided sharply in the operating core).	- It requires specialists with high skills (much horizontal specialization of jobs). - Professionals are working autonomously. - It relies on specialists and large number of knowledge workers with high skills to control their own work. - Professionals resist the rationalization of their skills.	- It is suitable for producing its standard outputs but not suitable for adapting to the production of new products.	- Its structure relies on the standardization of skills in its operating core for coordination. - The decision making is decentralized. - It enjoys the efficiency benefits of a machine structure. - It has inflexible structure. - Administrators and supporting staff follow machine structure, while professionals follow democratic way of work.
Hierarchical culture. (Machine Bureaucracy Org.)	- It is usually has a functional grouping. - Wide unit size at the bottom. (The labor is divided sharply in the operating core).	- The operating tasks are considered simple and repetitive, performing the tasks require a minimum of skill and little training. - The strategic apex is responsible on eliminating all possible uncertainties in order to run smoothly and containing all the conflicts. - Unsuitable for innovation (resists the change). - It requires much horizontal and vertical specialization of jobs.	- Too complex.	- It coordinates primarily by the imposition of work by following formal channels (Standardization of work). - It is highly formalized. - It is based on efficient performance of standardized routine work. - The decision making is centralized vertically at the strategic apex with limited horizontal decentralization to the technostructure.

Table 4.2. The current situation of Organizational Cultures based on the 4-D.

4.2.2. Project Assessment

The management team can specify the type of software which is ordered by the customer (i.e. new software, reuse of existing software, stand alone software, or customized software) by considering the products they have. By specifying the project characteristics (such as: product size, criticality, complexity, and etc) beside the software type, the appropriate development model can be specified.

By exploiting the experiences that have been gained from previous projects, the can specify the project type, thereby they can reduce the costs of the development and can optimize the software development processes by applying the highest level of agile practices that suites this project. On the other hand, there are other circumstances that may affect the adoption of agile practices and might constraint it, such as the impossibility of having face-to-face communication with the customer or even it might be hard to a frequent meetings with the customer.

It has been recognized as mentioned before that the IS developers in the organizations that have a Hierarchical Culture orientation would prefer to have a software development methods in a systematic way, and the same thing for the Developmental Culture orientation but not systematically since they prefer to work on more agile and creative way. On the other hand, the IS developers in the organizations that have a Rational Cultural orientation does not like to have software development methods because of having professionals with high skills.

Some of the suggested Agile Practices might not be suitable in any kind of the organizational cultures because of outside circumstances that are related to each project. For instance, the face-to-face communication may not be suitable for a specific project, since the customer can not provide an on-site customer. Moreover, it might be not possible to frequent meetings that guarantee having the required feedback from the customer which makes the customer inaccessible immediately. Therefore, it would be better to have a customer contract that guarantees his commitment and contribution is such processes. On the other hands, having a certain moral values may be embraced in order to have ethical standards that keep all stakeholders working in an optimum way (as discussed before).

Since an organization may have an experience in different projects which they have already developed before, it might have an opportunity to reuse of existing software or to customize it according to the customer’s needs. Therefore, even in Machine Bureaucratic organizations we can use more Agile Practices instead of following the same routine way of work. As it has been mentioned at chapter 3, Brooks has been suggested to have reusable modules that would be used in different modules or components of the product (Brooks 1995). Therefore, the organization should have a very good design and documentation in order to use it frequently and easily. Thereby, we can reduce the process of software development.

4.2.3. Guidelines to adapt Agile Practices

Based on the evaluation of the current situation of the organizational culture that which is prevalent (according to Mintzberg), and by considering the assessment of the project which we would like to develop, beside what have been argued by Brooks for the potential of silver bullet, we conducted below guidelines to adapt some Agile Practices for each organizational culture. Table 4.3 summarizes some generic Agile Practices that can be used in different Organizational Structures.

4.2.3.1. Group culture (Simple organization)

Since the strategic apex coordinates by direct supervision in such organizations, and the decision making is highly centralized by having strong leaders (usually their owners). People in such culture may exhibit some issues, as below.

1. When top managers exhibit bad decisions because of being overloaded by taking many decisions and by being on direct contact with all of the employees. Then the top manager should know that he/she should spread down some power to the people in the operating core in order to have adequate time to make the required decisions that would improve the organization toward growing.

2. Since the organization is highly dependent on the individual inside the operating core, it might face some risks if they leaved the organization or if they were having unethical behaviors such as the monopolization of knowledge which prevent sharing the knowledge to the others. Therefore, the top manager should use his/here power to guarantee the sharing of knowledge.

Some suggested Agile Practices that suites such kind of organizations:

1. Frequent face-to-face communication by having a customer contract that guarantees a commitment of collaboration. If not possible the customer should be immediately accessible.
2. Daily progress tracking meetings.
3. Collaborative planning and Deliver Software Frequently.
4. No big design up front.
5. Tracking iteration progress.
6. Continuous integration.
7. Continuous improvement (refactoring).
8. Agile documentation that contains the most important things that would be needed in future and it should be synchronized with the software development.
9 Reflect and tune the processes continuously and motivating the team members.
10. Task volunteering. Instead of asking employees to do a specific task, it would be great if we asked them to volunteer for doing a specific task that they maybe interested in to do.

4.2.3.2. Developmental culture (Adhocracy organization)

Since the Adhocracy organization is the only configuration for those who believe in more democracy which implies to support the enterprise agility as mentioned before, it would suites the adapting of many Agile Practices, such as; smaller and more frequent releases, getting continuous feedback from the customer, face-to-face communication with the parties, self-organization teams, daily progress tracking of the team, continuous improvement of the system (refactoring), and many more practices.

But people in such culture may exhibit some issues, as below.

1. People may exhibit some confusion of the authority and power that allows lots of conflicts. Therefore, the management team needs to provide a little bit of job formalization that does not introduce a formal standardization of job description for each member, but it would reduce the confusion and the ambiguity that might happened.

2. It is considered as a stressful environment for the workers, since this culture responds quickly to the changes by providing frequent releases to the customer in order to get a quick feedback that enables them to make the required changes. Therefore, the organization may loose the talent workers.

For this reason, the managers should make sure that the employees do not burn their selves by continuous hard work that would decrease their interest in work. Therefore, the managers should have some rules, such as; the employees should not have to work over-time for two weeks in sequentially, and the managers should have more constraints on the over-time by a way that suites both of the employees and the company. On the other hand, the organization should provide some kinds of the incentives to the employees.

3. Since this kind of organizations supports the enterprise agility, people may exhibit some conflicts and aggressiveness because some employees are trying to monopolize the experience they have. Therefore, the managers should try in a political way to minimize the relational barriers between the employees to work in one spirit to remove the conflicts and to spread the knowledge.

4. The cost of the flexible communication (people spend more time on talking) is considered high compared to the bureaucratic organizations which have a standardization way of communication. Therefore, the managers should have a certain moral values that should be embraced in order ethical standards that keep all stakeholders to communicate in an optimum way that reduces the talking about personal issues instead of the work it self.

4.2.3.3. Rational culture (Professional Bureaucracy organization)

Since this kind of organizations is relying on specialists with high skills to control their own work, the decision making and power is decentralized. People in such culture may exhibit some issues, as below.

1. A lack of control in such culture emerges from spreading down the power to the professionals. Thus, the discretion and the autonomy of the specialists or professionals allow them to ignore the needs of both of the organization and the client. Therefore, an external control is required to a specific degree that does not bother both of the professionals and the client and this control should not affect professionals' way of innovative in order to get the required benefits of having them.

2. A loose of coordination can be emerged from having a standardization of skills where the professionals working autonomously from their and a communication difficulties would be emerged also from dealing professionals once with the incompetent professionals. Therefore, we should emphasize on having a well trained professionals before providing them with the required power to do their jobs. Even though, after providing the incompetent professionals with the required training, we should gradually provide them with power, not at once, to make sure that they are progressing well and working on a standardized way of skills.

Some suggested Agile Practices that suites such kind of organizations:

1. Self organizing teams.
2. Frequent face-to-face communication by having a customer contract that guarantees a commitment of collaboration. If not possible the customer should be immediately accessible.
3. Daily progress tracking meetings.
4. Agile project estimation.
5. Plan and Deliver Software Frequently. The team should maintain a list of all features and their status (like having a backlog)
6. No big design up front.
7. Tracking iteration progress.
8. Test driven development.
9. Continuous integration.
10. Continuous improvement (refactoring).
11. Software configuration management.
12. Agile documentation that contains the most important things that would be needed in future and it should be synchronized with the software development.

4.2.3.4. Hierarchical culture (Machine Bureaucracy organization)

Since this kind of organizations is based on efficient performance of standardized routine work, the decision making is highly centralized and people in the operating core follow standardized routine work. People in such culture may exhibit some issues, as below.

1. Some employees found it a boring way of working, since it is based on efficient performance of standardized routine work. And since that the training does not take that much time and resources, it would be better to switch the kind of tasks between the employees in a systematic way that guarantee the same efficiency, thereby we reduce that boring way of working and we improve employees' skills.

2. It prevents employees from being creative because it has formalized procedures in the operating core and formalized communication throughout the organization. Therefore, if we followed what we mentioned in the previous point we can improve the creativity of the individuals and we can guarantee the efficiency at the same time.

3. This kind of organization considers people as machines that follow procedures in a formalized way, but people are not machines, they are human beings who some times do not like to be told what to do and do not like to follow procedures, they aim for self-actualizing by being creative in order to have the required security that they can imagine. Therefore, the employees at the operating core do not have the required human satisfaction from this kind of organizations.

4. Carrying the required information through this chain of authority would reduce the tangible details that are needed and must be carried out to the strategic apex in order to make the required decisions and adaptation that reduces non-routine problems. This implies to produce a communication and coordination problems. Therefore, the top managers at the strategic apex should turn over temporally into direct management in order to get the required information to make the correct decisions and changes.

Some suggested Agile Practices that suites such kind of organizations:

1. Coding standards.
2. Knowledge sharing tools.
3. Task volunteering.
   As we suggested before in the first point of this section (regarding the inhabitance of some issues that people may face). Therefore, it would be great if we asked the employees to volunteer for doing a specific task that they maybe interested in to do.
4. Plan and Deliver Software Frequently (No big design up front unless the organization is developing a critical systems).
   Since most of the projects that are developed in such organization is considered too complex, it would be better to not to plan to deliver only one release of working system. Planning to develop a big design up front system and developing it would take much more time than building it incrementally. Thus we can provide an incremental way of development that enable getting a frequent feedback from the customer and can guarantee the customer's satisfaction by having a frequent releases which they can benefit from (as we discussed in the hypotheses H3 and H4).
6. Tracking iteration progress.
7. Continuous integration.
8. Software configuration management.

Agile Practices	Machine Bureaucracy	Professional Bureaucracy	Adhocracy organization	Simple organization
Self organizing teams		**Yes**	**Yes**	
Face-to-Face communication		**Yes**	**Yes**	**Yes**
Daily progress tracking meetings		**Yes**	**Yes**	**Yes**
Agile project estimation		**Yes**	**Yes**	
Plan and Deliver Software Frequently	**Yes**	**Yes**	**Yes**	**Yes**
No big design up front		**Yes**	**Yes**	**Yes**
Tracking iteration progress	**Yes**	**Yes**	**Yes**	**Yes**
Test driven development		**Yes**	**Yes**	
Continuous integration	**Yes**	**Yes**	**Yes**	**Yes**
Continuous improvement (refactoring)		**Yes**	**Yes**	**Yes**
Software configuration management	**Yes**	**Yes**	**Yes**	
Agile documentation		**Yes**	**Yes**	**Yes**
User stories		**Yes**	**Yes**	
Client driven iterations		**Yes**	**Yes**	
Adaptive planning		**Yes**	**Yes**	
Unit tests		**Yes**	**Yes**	
Customer contract for continues delivering of software and for communication		**Yes**	**Yes**	
Coding standards	**Yes**			
Knowledge sharing tools	**Yes**	**Yes**	**Yes**	
Task volunteering		**Yes**	**Yes**	**Yes**

Table 4.3. Some generic Agile in different Organizational Structures.

5. Results: Would Agile Software Development constitute a Silver bullet?

Numerous authors have realized a limited effect on software productivity by the technology (Brooks 1995; Beck, Martin et al. 2001; Cockburn 2002). There is a huge difference of productivity between individuals, according to DeMarco and Lister as it has been cited by Nierstrasz (Nierstrasz 2002), "*The major problems of our work are not so much technological as sociological in nature*". Since that many aspects of the Software Engineering are human-centric and people thinking in a different ways to solve and to manage issues, the produced quality and productivity of their software production is differentiated from one person to another according to their creativity. And since there is a lack of systematic quality controls that could lead to disasters such as schedule delay, rather than productivity concerns, as mentioned before, a systematic software development disciplines were developed which tried to control the software development. But sociology is not the only factor for this differentiation of productivity. Important factors, which shown to far outweigh any technological factor in the success of projects, are emerged from the communication and motivation (Cockburn 2002; Nierstrasz 2002). Therefore, we need to emphasize also on improving communication and motivation skills of the employees. According to Nierstrasz (Nierstrasz 2002), Alistair Cockburn noticed from his 20 years experience that: "*Almost any methodology can be made to work on some project, any methodology can manage to fail on some project, and heavy processes can be successful. Light processes are more often successful, and more importantly, the people on those projects credit the success to the lightness of the methodology*".

By accepting the idea of no existence of a purely technological silver bullet (Nierstrasz 2002), we need to emphasize and pay more attention to the sociological issues in software process as it has been mentioned in the Agile Software Development Manifesto (Beck, 2001; Cockburn, 2002; Nierstrasz, 2002). According to Nierstrasz (Nierstrasz 2002), "*Although there appears to be no claim in this manifesto that agile processes will improve productivity in the long run, it is clear that, as a metaphor, agile software development gives change a central role, and downplays purely technological tactics. This, we feel, is an important step in the right direction*".

Since agile thinking is a people-centric view which depends on people to choose the right method which is suitable at the right time and depends on the communication, collaboration and change, rather than on processes, tools and plans. Agile Software Development would directly focus on maintaining a high qualitative prototype through continuous iterations of the software development cycle that enable a continuous communication between all parties to get the required feedbacks that increase the level of quality and guarantee developing the required system that satisfies the customer needs.

The continuous verification of the software quality, by verifying each prototype through each iteration of the software development, reduces the cost of fixing the defects that may arise at late stages of the software development life cycle. Thus, by having the required testing in each iteration to find as much defects as possible in the earlier stages, we can

reduces the costs and we can get the required feedback from the customer to make sure that we are developing the required system with the required quality.

In the systems that contain myriad details that are hard to be specified exactly (especially at advance), the complexity of these system become an essential issue and difficulty that needs to be considered. In order to reduce the unnecessarily costs that may lead to disasters, such as schedule delay that may emerge from requirements change which is followed by many other required processes, the development team needs to harness an iterative way of software development that enables getting the details of each module, so that the complexity of the whole system can be decreased and the development team can get the required feedback of each presented prototype at the end of each iteration.

As the uncertainty level is high and the scope is not well defined, we need to adapt agile processes to the project. By combining Agile methods such as Scrum (which provides guidance for efficient management in a high flexibility and allows adaptability) and CMMI (which provides insight into what processes are needed to maintain a disciplined mature organization in order to predict and improve the performance of organization and the projects), we can gain a higher quality and performance software that more effectively meets the customers' needs in a shorter time (May and Zimmer 1996). Both of them provide a powerful combination of adaptability and predictability than using either one alone. We can recognize the benefits of using agile methods especially when we work on large projects, while in small projects the productivity is insignificantly changed (May and Zimmer 1996). Thus we reduce the number of coding defects in the final test and the project finished early. By Supplying a working versions of the system early (small iterations), we get early feedbacks that reduces the uncertainty and prevents the risks of getting late changes that might needs a major changes in the system with highly costs (May and Zimmer 1996).

Usually people focus on learning the programming languages as a tool that helps them to implement their system. But if we trained individuals and taught them the design principles, they can produce strong designs that reduce the costs of unexpected flaws which may immerge as a response of requirement changes during software development, and we can have a common view between the individuals to communicate through it and to improve they way of thinking. Thereby, we provide them with an opportunity to be more creative. By paying more attention to the design issues during the iterations of software development we can increase the quality of our software.

By paying attention to Mintzberg's organizational structures and by considering Brooks' way of thinking, which have been discussed before, we considered and presented the humanity issues in the most dominant organizational cultures which would affect their way of working in below sections.

5.1. Humanity in Machine Bureaucracy Organization

Since that Mintzberg's Machine Bureaucratic basic structure is based on efficient performance of standardized routine work, some employees found it a boring way of

work because of doing the same thing every time. Having formalized procedures in the operating core and formalized communication throughout the organization prevent employees from being creative and from doing new things where some employees would like to do. This kind of organization consider people as machines that follow procedures in a formalized way, but people are not machines, they are human beings who some times do not like to be told what to do and do not like to follow procedures, they aim for self-actualizing by being creative in order to have the required security that they can imagine. Therefore, the employees at the operating core do not have the required human satisfaction from this kind of organizations.

As a result of that, a communication and coordination problems may arise since handling these issues should be done by the administrative structure and not at the operating core where the decision making is highly centralized through a chain of authority (direct supervision). Carrying the required information through this chain of authority would reduce the tangible details that are needed and must be carried out to the strategic apex in order to make the required decisions and adaptation that reduces non-routine problems. Mintzberg concluded that this kind of structures are non-adaptive structure and ill-suited to changing their strategies, but "*the top managers succeed in changing the Machine Bureaucracy only by reverting temporarily to the leaner, more flexible Simple structure*" (Mintzberg 1983).

5.2. Humanity in Professional Bureaucracy Organization

Since Professional Bureaucracy organizations relying on specialists with high skills to control their own work, the decision making is decentralized and these organizations allow professionals to perfect their skills which enable an effective coordination in their operating core. Spreading down the power through the hierarchy causes a lack of control which is considered as a disadvantage which results in a coordination and discretion problems.

A loose of coordination can be emerged from having a standardization of skills where the professionals working autonomously from their colleagues and closely to the clients they serve. Communication difficulties would be emerged also from dealing professionals once with incompetent professionals where such organization relies on highly competent professionals, therefore we should emphasize on having a well trained professionals before providing them with the required power to do their jobs. Even though, after providing the incompetent professionals with the required training, we should gradually provide them with power, not at once, to make sure they are progressing and working on a standardized way of skills.

On the other hand, the discretion and the autonomy of the specialists or professionals allow them to ignore the needs of both of the organization and the client; this unconscientiously may happened because of the lack of control and the high demands on professionals to get work done. Therefore, an external control is required to a specific degree that does not bother both of the professionals and the client and this control should

not affect professionals' way of innovative in order to get the required benefits of having them.

5.3. Humanity in Adhocracy Organization

Since those Adhocracy organizations are organic and have a decentralized power, creative people found these organizations are the healthiest environments for them. According to Mintzberg (Mintzberg 1983), "*Adhocracy is the only configuration for those who believe in more democracy with less bureaucracy*". As mentioned before; the strategy of these organizations emphasize toward growth, resource acquisition, innovation, and adaptation to the external environment. Therefore, it supports the enterprise agility.

Although these kinds of organizations are considered the healthiest environments for creative and innovative people, people may exhibit some confusion and ambiguity of the authority and power that allows lots of conflicts. Also since the organization can respond quickly to the change, it is considered as a stressful environment for the workers which make it difficult to find and keep talent. But since Adhocracy organizations are considered the most politicized one, the managers should use these conflicts and aggressiveness as instruments to increase the productivity and not to loose these talent workers by minimizing the relational barrier.

Adhocracy organization has a flexible communication lines where the primarily way of coordination in such organization is the mutual adjustment among all parts, people spend more time on talking which enable them to spread out the knowledge and to develop new innovative ideas that provide the opportunity to improve the workspace functions. On the other hand, this way of communication may decrease the efficiency where the cost of this communication is high as compared to the bureaucratic organizations that have a standardization way of communication.

5.4. The answer of whether Agile Software Development would constitute a silver bullet or not

Since many aspects of the Software Engineering are human-centric and people thinking in a different ways, besides having a lack of systematic quality controls. The produced quality and productivity of their software production is differentiated from one person to another according to their creativity. Thereby, the sociological factor beside the communication and the motivations play and important role in the software development. Therefore, we need a lightweight framework that would help teams to improve the quality, the productivity of the software construction and significantly would reduce the overall risk associated with software development. By paying more attention to the sociological issues in software development, as it has been mentioned in the Agile Software Development Manifesto, beside the communication, collaboration and change, we can harness Agile Software Development way of working as an instrument that would directly focus on maintaining a high qualitative prototype through continuous iterations of the software development cycle that enable a continuous communication between all

parties to get the required feedbacks that increase the level of quality and guarantee developing the required system that satisfies the customer needs.

Certain moral values should be embraced in order to have ethical standards that keep all stakeholders working in an optimum way. Thus, people can autonomously emphasis on the quality of the software by working creatively. On the other hand, having a set of values that support the long-term social goals such as encouragement beside the good communication and continuous feedback is not enough to have a high quality. Therefore, comes the need of backed up these values by myths, rituals, punishments, and rewards.

For these reasons and as it has been discussed before in Chapter 4, we think Agile Software Development would constitute a silver bullet that can maintain a focus on rapid delivery of business values and can help on solving the humanity issues. As a result of that, organizations are capable of significantly reducing the overall **risk** associated with software development and they are capable to **adapt** the requirements change easily throughout the process. On the other hands, the **visibility** of the actual progress is obvious for all parties during the software development.

6. Conclusion.

The current state in Software Engineering and Development have some issues that need to be fixed and improved in a better way, such as having a mixture of practices and detailed processes. Organizations employ numerous processes to perform the needed tasks in projects development in different ways according to their organizational culture and structure. Brooks realized that the complexity of projects is the most important factor for our business limitation (Brooks 1995). By accepting the idea of no existence of a purely technological silver bullet (Nierstrasz 2002), and since numerous authors have realized a limited effect on software quality and productivity by the technology and there is a huge difference of quality and productivity between individuals (Beck, 2001; Cockburn, 2002; Brooks, 199;Nierstrasz, 2002), we need to pay more attention to the sociological issues in software processes as it has been mentioned in the Agile Software Development Manifesto (Beck, 2001; Cockburn, 2002; Nierstrasz, 2002). Therefore, we introduced a 3-Step framework that evaluates the current organizational culture and the projects that the organization would like to develop in order to provide some guidelines. By this way, we are addressing the humanity factors in different organizational cultures beside the technology in order to provide some Agile Practices that are needed to be assed and improved continuously according to the different circumstances that an organization may face (the details of the 3-Step framework can be found in Chapter 5).

6.1. Critical discussion

A continuous attention to both of the business change and to the employees' issues would lead to a successful change of the adopted processes. Thereby, a steadily rising of satisfying both of the business needs and employees' needs would lead to achieve business goals and to have qualitative software.

Certain moral values should be embraced in order to have ethical standards that keep all stakeholders working in an optimum way. Thus, people can autonomously emphasis on the quality of the software by working creatively. On the other hand, having a set of values that support the long-term social goals such as encouragement beside the good communication and continuous feedback is not enough to have a high quality. Therefore, comes the need of backed up these values by myths, rituals, punishments, and rewards.

Humanity issues in different Organizational cultures would affect the construction of software development. By considering the agile way of software development, we think it would solve the humanity issues, and thereafter would have lightweight processes that increase the productivity of the employees and would produce qualitative software. By using the lightweight processes "Agile Practices", managers should pay a continuous attention to assess and improve these practices to cope with different situations and with changes, see figure 6.1. This way, we can have Agile Practices that suited different kinds of projects and which are suitable for the organizational culture.

We can harness Agile Software Development way of working as an instrument that would directly focus on maintaining a high qualitative prototype through continuous

iterations of the software development cycle that enable a continuous communication between all parties to get the required feedbacks that increase the level of quality and guarantee developing the required system that satisfies the customer needs. As a result of that, organizations are capable of significantly reducing the overall **risk** associated with software development and they are capable to **adapt** the requirements change easily throughout the process. On the other hands, the **visibility** of the actual progress is obvious for all parties during the software development.

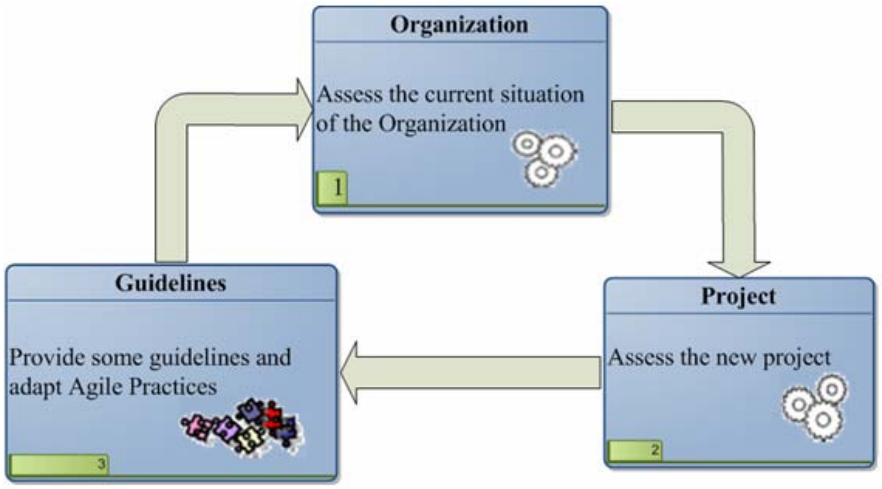


Figure 6.1. Top managers should pay a continuous attention to assess and improve the Agile Practices.

I think that most organizations should aim for having the Professional Bureaucracy Culture, by having a parallelism of administrative hierarchies, one is machine bureaucratic (top-down) for the support stuff where the power and status reside in administrative office by practicing the administration to attain status, and a second democratic (bottom-up) for professionals where the power resides in expertise and the influence is introduced by professionals' knowledge and skills. This parallelism of hierarchies may create conflict in the organization. Therefore, the top-managers should cooperate with the professionals at the operating core in order to remove any conflicts that may arise, and they should work on minimizing the relational barriers between them in order to work in more effective way.

6.2. Limitations

The applied research strategy was a straightforward way by reviewing literatures. Therefore, we spent an extensive effort on research, and it was hard to find a research addressing the validity of Agile Software Development as a silver bullet.

Some difficulties aroused while collecting data due to time mismatch, and potential lack of such research that connects the Software Engineering with the Construction Management. Therefore we did not have enough time to conduct a survey or even to deploy this lightweight framework in the industry due to the time limitations.

6.3. Future research

Further research is needed to provide factors that prevent the adoption of different Agile Practices in different organizational cultures under specific circumstances that might happened during the software development.

In addition to that, it would be useful to propose different agile levels that categorize the different Agile Practices that could be adapted in different organizational cultures. This way we can provide the project manager to measure the current agility level of the organization in order to be able to work on improving the maturity level of the Agile Practices inside the organization. By this way, we need also to provide some kind of indicators for each level of the agility in each organizational culture to be able to measure the agility level of the organization.

Further research might be needed to connect the CMMI (which provides insight into what processes are needed to maintain a disciplined mature organization in order to predict and improve the performance of organization and the projects) to the previous suggested agile levels for different organizational cultures (which provides guidance for efficient management in a high flexibility and allows adaptability). Thus, we can gain a higher quality and performance software that more effectively meets the customers' needs in a shorter time

Then for sure we need to conduct a survey and to deploy this framework in the industry to get the feedback that would enable us to prove it.

References:

Aaen, I., P. Bøttcher, et al. (1997). The Software Factory: Contributions and Illusions. In: Proceedings of the Twentieth Information Systems Research Seminar in Scandinavia, Oslo.

Abrahamsson, P., O. Salo, et al. (2002). Agile Software Development Methods Review and Analysis. VTT Publications, Espoo, Finland, University of Oulu**:** 107

Abrahamsson, P., J. Warsta, et al. (2003). "New Directions on Agile Methods: A Comparative Analysis." Proceedings of ICSE: 244-254.

Ambler, S. (2002). Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process. New York, John Wiley & Sons, Inc.

Asopa, V. and G. Beye (1997). Management of agricultural research: A training manual, Food and agriculture Organization of the United Nations. Rome, Italy.

Bauer, T. N. and B. Erdogan (2009). "Organizational Behavior." New York: Flat World Knowledge.

Beck, K. (1999). Extreme Programming Explained. Boston, Addison-Wesley.

Beck, K. (2005). Extreme Programming Explained: Embrace Change Boston, Addison-Wesley.

Beck, K., R. C. Martin, et al. (2001). Manifesto for Agile Software Devleopment.

Beer, M. and R. Eisenstat (1996). "Developing an organization capable of implementing strategy and learning. Human Relations." **49**(5): 597-620.

Boyd, R. W. (2009) Software Lifecycle Model Selection Criteria for Safety-critical Software.

Brooks, F. P. (1995). The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition. Reading, MA, Addison-Wesley.

Bruce, B. (1992). Software Engineering : A Holistic View. Oxford University Press. New York, Oxford University Press.

Cockburn, A. (2002). Agile Software Development. Boston, MA, USA, Addison-Wesley.

Cohn, M. (2005). Agile Estimating and Planning, Prentice Hall PTR.

Davis, S. M. and P. R. Lawrence (1977). "The Matrix organization-who need it? In Matrix, Reading." MA: Addison-Wesley.

de Wit, B. and R. Meyer (2005). *Strategy Synthesis: Resolving Strategy Paradoxes to Create Competitive Advantages – Text and Readings*. 2nd edition, Thomson Learning, London. pp. 81-83.

DeGrace, P. and L. H. Stahl (1990) Wicked problems, righteous solutions: A catalogue of modern software engineering problems. 116- 117.

Dunphy, D. (1996). "Organizational change in corporate setting." Hum. Relat **49**(5): 541–552.

Fowler, M. and J. Highsmith (2001, August). "The Agile Manifesto. Software Development." from http://andrey.hristov.com/fht-stuttgart/The_Agile_Manifesto_SDMagazine.pdf.

Fredrickson, J. W. (1986). "The strategic decision process and organizational structure." Academy of Management Journal: 280-297
211.

Fujita, H. and I. Zualkernan (2008). New Trends in Software Methodologies, Tools and Techniques. Proceedings of the seventh SoMeT_08, IOS Press.

Gagliardi, P. (1986). "The Creation and Change of Organizational Cultures: A Conceptual Framework." Organization Studies **7**(2): 117-134.

Guler, S. (2010). Change Management: A Case Study of SAP Implementation in a Major Company. Civil and Environmental Engineering. Gothenburg, Chalmers University. **Master:** 74.

Hiatt, J. and T. Creasey (2003) Change Management: The People Side of Change. 38-40.

Highsmith, J. and A. Cockburn (2001). "Agile Development: The Business of Innovation." IEEE Computer **34**(9): 120-122.

Highsmith, J. and A. Cockburn (2001). "Agile Software Development: The People Factor." IEEE Computer **34**(11): 131-133
184.

Huber, P., K. Sutcliffe, et al. (1993). Understanding and predicting organizational change and redesign. In: P. Huber and W.H. Glick (eds.) Organization Change and Redesign: Ideas and Insights for Improving Performance. Oxford: Oxford University Press, pp. 215–265.

Iivari, J. and M. Huisman (2007). "The relationship between organizational culture and the development of systems development methodologies." MIS Quarterly **31**(1): 35-58.

ISO (1998). "Information technology – Guide for ISO/IEC 12207 (Software Life Cycle Processes)." ISO/IEC TR

Jacobson, I., B. Meyer, et al. (2010) Software Engineering Method and Theory - A vision Statement. 20.

Koch, A. S. (2005). Agile software development: evaluating the methods for your organization. Boston, MA, Artech House.

Markula, J. (2002). Agile Software Development – Control or Chaos. T-76.650 Software Engineering Seminar, Helsinki University of Technology.

May, E. and B. Zimmer (1996). "The Evolutionary Development Model for Software." Hewlett-Packard Journal **47**(2): 39-45.

McConnell, S. (1998). "Software Project Survival Guide." Redmond, Wash: Microsoft Press.

Mintzberg, H. (1983). Structures in fives: Designing effective organizations. Englewood Cliffs, New Jersey, Prentice-Hall, Inc.

Mintzberg, H. (1983). Structures in fives: Designing effective organizations. New Jersey, Prentice-Hall, Inc.

Nierstrasz, O. (2002). Software Evolution as the Key to Productivity. In Proceedings Radical Innovations of Software and Systems Engineering in the Future, 9th International Workshop, RISSEF 2002, Venice, Italy, October 7-11, 2002, Revised Papers, (2004), Springer-Verlag Berlin Heidelberg 2004, 274-282.

Porras, J. I. and P. J. Robertson (1992). Organizational development: Theory, practice, and research. In M. D. Dunnette & L. M. Hough (Eds.), *Handbook of industrial and organizational psychology* (2nd ed., Vol. 3, pp. 719–822). Palo Alto, CA: Consulting Psychologists Press.

Poston, B. and CST (2009) An Exercise in Personal Exploration: Maslow's Hierarchy of Needs.

Pugh, L. (2007). Change Management in Information Services. Ashgate Publishing Limited, Aldershot, Gower Pub Co.

Quinn, R. E. and J. R. Kimberly (1984). Paradox, Planning, and Perseverance: Guidelines for Managerial Practice. in *New Futures: The Challenge of Managing Organizational*

Transitions, J. R. Kimberly and R. E. Quinn (eds.), Dow Jones Irwin, Homewood, IL, pp. 295-313.

Quinn, R. E. and J. Rohrbaugh (1981). "A competing values approach to organizational effectiveness." **5**(2).

Quinn, R. E. and J. Rohrbaugh (1983). "A Spatial Model of Effectiveness Criteria: Towards a Competing Values Approach to Organizational Analysis." Management Science **29**(3): 363-377.

Ralph, P. and Y. Wand (2008). A Teleological process theory of software development. Journal of the Association of Information Systems Theory Development Workshop, International Conference on Information Systems. Paris, France.

Schein, E. H. (1985). Organizational Culture and Leadership. San Francisco, Jossey-Bass.

Scott, W. A. (2002). "in Agility from Internet-Based Development." IEEE Software **19**(2): 66-73.

Sharp, H. and H. Robinson (2007). "Collaboration and co-ordination in mature extreme programming teams." International Journal of Human-Computer Studies **66**(7): 506-518.

Singh, R. (1995). "An introduction to ISO/IEC 12207 (Tutorial)." ISO/IEC.

Steinberg, D. H. and D. W. Palmer (2004). Extreme Software. Engineering–A Hands on Approach, Pearson-Prentice Hall.

Sutherland, J., C. Jacobson, et al. (2007) Scrum and CMMI Level 5: A Magic Potion for Code Warriors.

Sutherland, J., A. Viktorov, et al. (2007). Distributed Scrum: Agile Project Management with Outsourced Development Teams. Hawaii International Conference on Software Systems, in HICSS'40. Big Island, Hawaii, IEEE.

USDA (2007). "Information Technology System Development Life Cycle Guide." Version 1.0. United States Department of Agreculture. Retrieved February 23, 2011, from http://prd_prvw.ocio.usda.gov/e_arch/doc/USDA_SDLC_GUIDEv1.0_011507.pdf.

Weick, K. E. and R. E. Quinn (1999) Organizational Change and Development. **361**, 26.

Whitgift, D. (1991) Methods and Tools for Software Configuration Management.

Young, C. and H. Terashima (2008). How did we adapt agile processes to our distributed development. Agile IEEE Conference**:** 304-309.

Zeffane, R. (1996). "Dynamics of strategic change: critical issues in fostering positive organizational change." *Leadership & Organization Development Journal* **17**(7): 36-43.