PROJECT AND PROCESS MANAGEMENT METHODS INFLUENCING AGILE SOFTWARE DEVELOPMENT

by

Mehmet Selim DERİNDERE

A thesis submitted to

the Graduate Institute of Sciences and Engineering

of

Fatih University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Engineering

January 2013

Istanbul, Turkey

APPROVAL PAGE

This is to certify that I have read this thesis written by Mehmet Selim DERİNDERE and that in my opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science in Computer Engineering.

Prof. Dr. Sevinç GÜLSEÇEN
Thesis Co-Supervisor

Associate Professor Dr. Atakan KURT
Supervisor

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science in Electrical and Electronics Engineering.

Assistant Professor Dr. Kadir TUFAN
Head of Department

Examining Committee Members

Professor Dr. Sevinç GÜLSEÇEN	
Associate Professor Atakan KURT	
Assistant Professor Dr. Kadir TUFAN	
Associate Professor Nizamettin BAYYURT	
Assistant Professor Dr. Özgür UYSAL	

It is approved that this thesis has been written in compliance with the formatting rules laid down by the Graduate Institute of Sciences and Engineering.

Assoc. Prof. Nurullah ARSLAN
Director

January 2013

PROJECT AND PROCESS MANAGEMENT METHODS INFLUENCING AGILE SOFTWARE DEVELOPMENT

Mehmet Selim DERİNDERE

M. S. Thesis – Computer Engineering
January 2013

Thesis Supervisor: Doç. Dr. Atakan KURT

Co-Supervisor: Prof. Dr. Sevinç GÜLSEÇEN

ABSTRACT

In order to transfer the knowledge gained in highly complex industrial environments to software development projects, the paradigm of project and underlying theory of agile software project management is examined and complex adaptive systems theory is proposed as a theory for software development. The adaptation of project and process management methods used in separate industries to software development is investigated using this theoretical foundation. Adaptation and different applications of these methods under different situations to software development projects was examined.

Keywords: Software Project Management, Agile Software Development, Project Management, Complex Adaptive Systems, Lean, Kanban, Theory of Constraints

ÇEVİK YAZILIM GELİŞTİRMEYİ ETKİLEYEN PROJE VE SÜREÇ YÖNETİM METODLARI

Mehmet Selim DERİNDERE

Yüksek Lisans Tezi – Bilgisayar Mühendisliği
Ocak 2013

Tez Danışmanı: Doç. Dr. Atakan KURT

Eş Danışman: Prof. Dr. Sevinç GÜLSEÇEN

ÖZ

Yüksek karmaşıklığa sahip endüstriyel ortamlarda kazanılan bilgin ve tecrübenin yazılım geliştirme projelerine aktarılabilmesi için, proje kavramı ve çevik yazılım proje yönetimine temel teşkil eden teori incelenmiş, karmaşık adaptif sistemler teorisi yazılım geliştirme projeleri için bir teori olarak önerildi. Bu teorik altyapı kullanılarak muhtelif endüstri dallarında kullanılan proje ve süreç yönetim metotlarının yazılım geliştirme projelerine uygulanma şekilleri ve uygunlukları incelendi. Bu metotların yazılım geliştirme projelerine hangi değişik koşullarda ve nasıl uygulanabilecekleri incelendi.

Anahtar Kelimeler: Yazılım Proje Yönetimi, Çevik Yazılım Geliştirme, Proje Yönetimi, KarmaşıkAdaptif Sistemler, Yalın, Kanban, Kısıtlar Teorisi

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

CAS	:	Complex Adaptive System
RAD	:	Rapid Application Development
TOC	:	Theory of Constraints
GSD	:	Global Software Development
EVO	:	Evolutionary System Development

CHAPTER 1

INTRODUCTION

Many software development projects using traditional project management methods fail (Standish Group, 2009) for numerous reasons. Although a great number of methods are available for project managers, lack of a theoretical foundation (Koskela & Howell, 2002) for project management in general and for software development projects in particular, prevents selection and application of the proper approaches and methods thus hindering project success.

The principal goal of this thesis is to examine the process and project management methods influencing agile software development so adopting the knowledge and experience gained in numerous industries. In order to accomplish this goal first a theoretical foundation for software development projects is established. Then the paradigm of project in agile approaches is examined.

Based on the theoretical foundation and the project paradigm, the project and process methods subject to this thesis are detailed.

The choice of traversing such a path for a seemingly straightforward subject is the result of "practice without sound theory doesn't scale" understanding. When not grounded on a sound theory the practices and methods used in software projects have no means to check their validity thus scaling becomes problematic and makes transfer of knowledge and experience almost impossible.

With the increasing importance of software in business and in daily life, software project management becomes critical for the success of many organizational functions and businesses. Between the two polar opposite approaches of Waterfall and Agile software development and facing a choice of almost tens of different development methods (such as Agile Modeling, Agile Unified Process, Crystal Clear, Dynamic Systems Development Method, Extreme Programming, Feature Driven Programming, GSD, Kanban, Lean Software Development, Scrum, Velocity Tracking, EVO etc.) project managers have numbers of methods and approach alternatives at their disposal. At the same time under pressure to deliver despite all constraints and facing a great number of environmental and organizational factors, project managers have an overwhelming complexity do deal with.

In many projects managers either tries to apply methods they used previously on other project settings to the problem at hand or they try something completely new without any regard to the underlying principles, environmental settings, the problem addressed and the people in the project.

The success of a software development project depends on matching the project process and management approach with the necessities and uncertainty of the environment, the people and the nature of the requirements that the project addresses.

However with the complexity in the projects and lack of both theoretical foundation and a useful practical framework for organization and project managers to guide them in selecting, adopting and monitoring the appropriate approaches results in confusion, bad practices and failure. Leaders try to adopt some method or another without a sound understanding of requirements, people, principles of managing the complexity and environmental factors resulting in both project failure, the frustration of project people and disappointment of customers.

Although there is great amount of research on selecting and adopting different software development methods, lack of a unifying theory, applicable managerial perspective, disregarding uncertainty and not taking complexity seriously, causes project leaders to try to apply the methods used or learned in one situation to every project and every situation.

Project and process management methods examined in this thesis have been developed and applied in many industries in different settings and contexts from very stable environments to highly uncertain situations where the very survival of the organization was at risk.

Also the methods examined have a very diverse set of factors in which they are applicable. But without a sound theory to base the methods on the practices become un-scalable fast. To overcome this risk a theory is required.

In this thesis first Complexity Theory and Complex Adaptive Systems with their application to software project management is reviewed. The agile perception of project is examined to reveal the differences between the traditional and agile project management.

The project and process management methods Scrum, Lean, Kanban, Lean and TOC are examined as parts of a framework that software project managers can use in development management.

The main question of this thesis is "how project managers can use the project and process management methods influencing agile software development to increase project success"

The path of this thesis is building upon the answers to these questions:

a) What is the theoretical basis for agile software development?
b) What is the project paradigm used in agile software development?
c) What industry originated project and process management methods available for project leaders to apply in software development?
d) Using what criteria, methods are tailored to fit a particular project?
e) How the examined project and process management methods can be best utilized in software development projects.

Software is in every aspect of the daily life and plays a significant role in the business environment which faces great deal of uncertainty in its increasingly competitive environment.

Software development projects undertaken to create value for customers and organizations mostly fail (Standish Group, 2009). The main reason for this failure is that software development methods appropriate in certain environmental and organizational settings are tried to be implemented in environments and settings which they do not fit (MacCormack & Verganti, 2003).

The dominating approach to software projects has been the traditional, plan-driven method.

While many organizations were trying to implement and adopt plan-driven project management methodology in their software development efforts, Agile Manifesto is presented as an alternative by software "gurus" whom each in their own developed light-weight methods to overcome the shortcomings of the plan-driven methods.

The word agile means "a-) marked by ready ability to move with quick easy grace. b-) having a quick resourceful and adaptable character" (Webster, 1981).The core of agile software development is "the use of light-but-sufficient rules of project behavior and the use of human and communication-oriented rules" (Cockburn, 2001).

The participants of Agile Manifesto outlined what is valued is in Agile as (Beck et al., 2001):

Individuals and interactions	over	processes and tools
Working software	over	comprehensive documentation
Customer collaboration	over	contract negotiation
Responding to change	over	following a plan.

The items on the right are important but the items on the left have the priority. This is in a contrast view to the traditional project management which above else holds planning, documentation and following these plans (PMBOK, 1996).

The main reason for the emergence of agile is the shortcomings of plan-driven methods. In many earlier development methods especially plan-driven ones work begins with identifying and documenting a complete set of requirements including customer, performance, functional, structural requirements. The disruptive technologies fueled by internet, explosion of dotcom companies and the turbulent and fast change resulting in the birth and growth of e-businesses and e-commerce made revealing and fixing requirements before project start impossible (Highsmith, 2002). Ability to adopt to change became more vital to organizational survival.

In order to be able to capture value in emerging customer demands or respond to requirement changes is projects; agile methods focus on rapidly developing working software instead of documentation unless it adds value to the customer. This way time and resources are not wasted on items that may be obsolete in a very short time.

To capture value organizations needs to act very fast, so are the development teams creating value for customer through producing software. In order to produce high quality software fast and in a sustainable pace software development teams needed new processes and approaches to software projects because plan-driven methods doesn’t supply the agility required. One approach used in agile methods to achieve this, is the recognition of software development as an empiric process (Williams & Cockburn, 2003) which resembles the scientific method.

There are two main kinds of processes (Ogunnaike & Ray, 1994).

1. Deterministic Processes: In deterministic (defined) process the control system is based on the idea that processes can be started and allowed to run to completion producing the same results every time(Schwaber, 2004). Here there are inputs to the processes, the process itself and the outputs from the process. Once any two have been specified the third element may then in principle be derived. When everything is defined in enough detail the process can be automated. Example to defined processes can be given as an automobile assembly line in which the process designed by engineers with input parts, assembly order, sequence of activities, actions of workers, machines and robots(Williams & Cockburn, 2003). The basic assumption is that when the steps are followed in defined order high-quality cars are produced. Empiric experience of car manufacturers shows otherwise (Jones, 2010).
2. Empiric Processes: In empirical process modeling the process itself is treated like a “black box” and the most important characteristics of the system are “identified” entirely from its response to known inputs; no attempt is made at utilizing any mechanistic information regarding the fundamental nature of the process (Ogunnaike & Ray, 1994). With empirical processes the system must continuously be monitored in order to make adjustments. Empirical processes necessitate short “inspect-and-adapt” cycles and short feedback loops.(Schwaber, 2004)

In the deterministic approach to product development there is a definition of the product up front and then using this definition a product is created using stages of product development and decision making.

The empirical approach to product development starts with a high-level product concept and through controlled experimentation and well-defined feedback and learning loops creates an optimal interpretation of the concept. The empirical model of process control provides and exercises control through frequent inspection and adaptation for processes that are imperfectly defined and generate unpredictable and unrepeatable outputs (Hawkins, 2012).

Agile development context changes from individual agile practices (Appelo, 2011) to suitable circumstances which support agile development to scalability which is using Agile in   large scale projects with 50 developers or more to adaptability which is about blending agile practices with the existing processes in the organizations(Williams & Cockburn, 2003). Applying agile approaches has impact on larger organizations. Many organizations have job definitions with clear cut authority and decision making roles. While becoming agile (as in organic organizations) decision making is shifted from managers to developers and team members of agile development. This affects the power structure within the organization.

There is a wide range of processes available for the managers to choose from. Managers must choose "a combination of practices and integrate them into a coherent process aligned with their business context" (Cusumano et al., 2009).The law of requisite variety in cybernetics states that the managers and project managers must have a variety of capabilities at hand in order to select and apply the right approach to software development projects.

Table 1.1 Law of Requisite Variety

In cybernetics the term variety was introduced by W. Ross Ashby to denote the count of the total number of states of a system. "If a system is to be stable the number states of its control mechanism must be greater than or equal to the number of states in the system being controlled" (Ashby, 1958).

Production and process management methods originated mainly but not exclusively from Japan such as Lean and Scrum are observed to give the practicing organizations competitive advantage in their industries. What is more relevant for the purposes of this thesis is that they are reported to increase employee engagement resulting in sustainability of high performance and increase both employee and stakeholder satisfaction. These production methods with respect to the paradigms found in their foundation have been adapted to software development.

In this thesis the project management and process management methodologies which influenced agile software development will be examined with their respective software development methodologies. For example lean production will be examined with lean software development.

The significance of the selected methods is that since each one is appropriate for different circumstances, settings, teams and paradigms, project managers can use each one in most useful environments that they come along.

The main contribution of this thesis is threefold. First a theoretical foundation is laid by examining software development as a CAS. Then the paradigm of project is traditional and agile approach is reviewed. Upon these two layers by a comparative study of alternative approaches to selecting and tailoring method for software development a guide for making it possible for project managers and business leaders to select the appropriate approach to create a better fit between the project processes and the environment and requirements of the project is built.

CHAPTER 2

COMPLEX ADAPTIVE SYSTEMS

In this part of the thesis complex system paradigm or thinking will be examined as a theoretical basis which project managers in complex and uncertain situations can use for sense making and apply its principles to organize people, structures, interactions and processes.

Highsmith recognizes that agile software development requires a theory since “techniques without a theoretical base are reduced to a series of steps executed by rote” (Highsmith, 1995). The practices and tools needed for effective collaboration in software development are necessary for managing continuous change. Highsmith argues that complex adaptive systems (CAS) theory is the basis for agile software development methods. In Agile software development “agents interact to form an ecosystem”. The interaction is defined by the exchange of information and the actions of the agents are based on internal rules. The agents “self-organize to produce emergent results” and the system “exhibit characteristics of both order and chaos” (Highsmith, 2002) which forms the complex adaptive system of software development.

A complex system is a system “composed of interconnected parts that exhibit one or more properties as a whole which doesn’t exists in the properties of the individual parts” (Chu, 2011), “a highly structured system which shows structure with variations” (Goldenfeld & Kadanoff, 1999). It is a system “whose evolution is very sensitive to initial conditions or to small perturbations, one in which the number of independent interacting components is large, or one in which there are multiple pathways by which t system can evolve “(Whitesides & Ismagilov, 1999). It is a system because it has many intersecting causalities and systems of feedback which amplify or slow down inputs or change efforts.

Ant colonies, human economies, social structures, human body as a whole or in subsystems can be given as example to complex systems. Complex systems are studied in mathematics, natural sciences, social sciences and computing. Systems theory, systems ecology, cybernetics are interdisciplinary fields which study complex systems.

Complex Adaptive System (CAS) is special subset of complex systems. It is an "adaptive system which interacts with itself and its environment to achieve and end"(Diment et al., 2009).John H. Holland has defined CAS as "a dynamic network of many agents (which may represent cells, species, individuals, firms, nations) acting in parallel, constantly acting and reacting to what the other agents are doing. The control of a CAS tends to be highly dispersed and decentralized. If there is to be any coherent behavior in the system, it has to arise from competition and cooperation among the agents themselves. The overall behavior of the system is the result of a huge number of decisions made every moment by many individual agents (Holland, 2006).

George Rzevski identifies the seven criteria of complexity as (Rzevski, 2011):

1. Interdependence: A system consists of diverse components called Agents which are interdependent meaning a change in one agents behavior is likely to cause a change the behaviors of other agents
2. Autonomy: Agents are autonomy meaning they are not centrally controlled and have a degree of freedom of choice within certain laws, rules or norms.
3. Emergence: Global behavior of the system emerges from the interaction of agents and is therefore unpredictable
4. Non-equilibrium: The system is not in an equilibrium since frequent occurrences of disruptive event's do not allow the system to return to the equilibrium
5. Nonlinear: Relations of the agents and events result in small inputs may be amplified into extreme events (butterfly effect) or reduced to ineffectiveness.
6. Self-organization: A system is capable of autonomous change of its behavior and/or configuration in response to disruptive events or in response to a need.
7. Co-evolution: A system irreversibly co-evolves with its environment.

Complexity is a way of thinking (paradigm) about the world. The basic characteristics of a complex system are(Snowden & Boone, 2007):

- It has large number of interacting elements

- The interactions are nonlinear, which means minor changes can have disproportionately major consequences
- The system has properties that none of its parts has since the interaction and dynamism give arise to them. This is called emergence.
- Solutions cannot be imposed to the system, they arise from the circumstances.
- The system has a history and its present is the result of its past. The elements interact whit each other as well as the environment and the evolution is irreversible.
- Since the system continuously changes, even it appears to be ordered and predictable, hindsight doesn't lead to foresight because of the dynamism inherent.
- When the system constraints the agents it is an ordered system
- When there are no constraints it is chaotic system.
- In complex systems the agents and the system constrain one another over time. This causes the impossibility of forecasting or predicting what emerges or what happens next.

The origin of the complexity system approach is found in natural science but applying it different. Human behavior cannot be modeled with the simple rules found in nature because of human unpredictability and intellect. Some differences of human behavior that animals are

- People have multiple identities and without conscious thought can transit from one to another.
- People make decisions based of patterns of success and failure in their history rather than logic and rationality
- People can change the systems in order to create predictable outcomes.

One way to look at software development is that it is a Complex Adaptive System of Decisions (McCarthy et al., 2006). The foundation of CAS understanding is the recognition that software development is a system called and agency whose elements called agents are partially connected and have the capacity and the ability to make decisions and act socially. The agents are building blocks of a Complex Adaptive Systems and they have the following properties;

- autonomous

- interdependent
- have rules for decisions
- interact
- create outcomes

McCarthy et al. characterized and defined CAS by three mutually dependent phenomena created by the decision rules, interactions and outcomes of agents (McCarthy et al., 2006):

1. Nonlinearity of cause and effect
2. Self-organization to achieve functions and goals
3. Emergence of behavior as a result of interactions, feedbacks and rules.

Volberda and Lewin draw out three principles which they believe are the "basic higher-order principles that must underlie any theory of self-renewal and its associated enabling managerial routines and capabilities involving strategy, structures, processes and leadership" as (Volberda & Lewin, 2003)

1. Managing requisite variety by regulating internal rates of change to equal or exceed relevant external rates of change such as change in technology, customer requirements etc.
2. Optimization of self-organization
3. Synchronizing concurrent exploitation and exploration. While exploitation is generating value in the existing state, exploration means searching and generating resources and activities to ensure survival and generate value in the future.

Managing internal rates of change means organization must maintain requisite variety (Ashby, 1958). Organisms and organizations with the self-renewing capability develop behavior, capabilities and measures to monitor the environment and develop and apply internal processes to deal with the external change.

Self-organization is the way organizations fit to the environment. This doesn't mean that every unit or individual can pull to another way. Self-organization requires that the decision making power exists in the lowest possible level in the organization and every level of the organization has capabilities of comprehending the scope. The roles of managers in a self-organizing organization changes into stewardess of the evolutionary process and focus their managerial role on “devising and articulating critical values and on establishing boundary conditions that enable and guide decision making at lower levels of the organization” (Volberda & Lewin, 2003)which in other terms called “subtle management” (Takeuchi & Nonaka, 1986). This requires the management to focus on the outcomes of the processes instead of controlling the processes. Management monitors the processes and emergence of outcomes. The task of management in complex systems is facilitation of the “constructing meaning and reality, and exploring how that enacted reality provides a context for action”(Weick, 1995).

Studies of CAS are mainly interested in how certain systems are able to learn and create new rules, structures and behaviors at several interrelated levels.

2.1 CYNEFIN FRAMEWORK

In order to “broaden the leadership approaches and decision making and form a new perspective based on complexity science” Snowden and Boone developed the Cynefin framework which “allows executives to see things from new viewpoints, assimilate complex concepts and address real-world problems and opportunities” (Snowden & Boone, 2007).

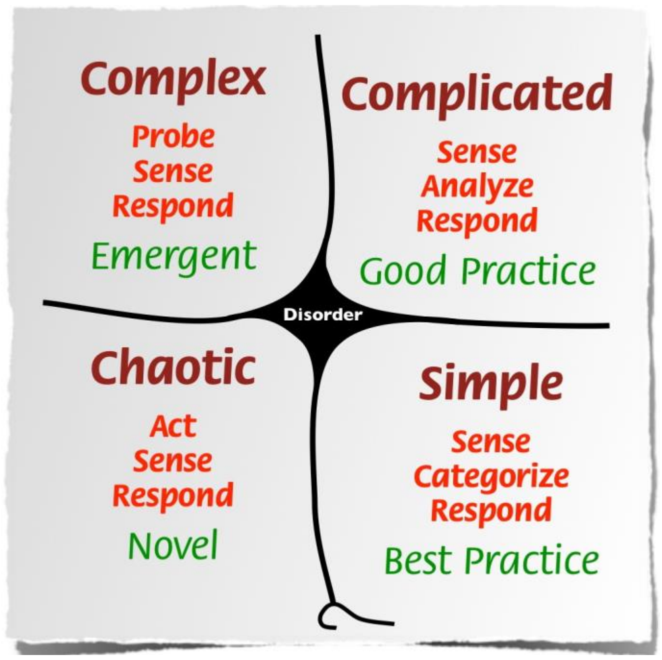

Figure 2.1 Contexts of Cynefin framework (Snowden and Boone, 2007)

They developed the Cynefin framework to be used by managers for sensing "which context they are in so that they can not only make better decisions but also avoid the problems that arise when their preferred management style causes them to make mistakes".

Cynefin framework roughly divides the sensed environment into five domains as simple, complicated, complex, chaotic and disorder, according to the cause and effect relationship of events. Each domain requires different action. Simple and complicated domains assume an ordered universe in which the cause and effect relationships are observable or perceivable. Complex and chaotic contexts are unordered meaning there is no apparent relationship between the causes and their effect and the future is the result of emerging patterns. While management in the ordered world is based on facts and experiences in the unordered world it is pattern based. Disorder which is the fifth domain is cacophony and makes impossible to make decisions.

Table 2.1 Domains of complex system understanding in Cynefin framework

	Simple	Complicated	Complex	Chaos
Context Characteristics	Repeating patterns and consistent events. Clear cause-effect relationships. Fact-based management	Expert diagnosis required. Cause-effect relationships discoverable but not apparent to everyone. Fact-based management	Flux and unpredictability. No right answers, emergent instructive patterns. Many competing ideas. Need for creative and innovative approaches. Pattern-based leadership	High Turbulence. No clear cause-effect relationships, no point in looking for right answers. Many decisions to make and no time to think. Pattern-based leadership.
Leader's Job	**Sense-categorize-respond** Use proper processes Delegation of work Communicate in clear, direct ways Extensive interactive communication may not be necessary	**Sense-analyze-respond** Create panel of experts Listen to conflicting advice	**Probe-sense-respond** create environment for experimentation and allow patterns emerge Increase interaction and communication	**Act-sense-respond** Look for what works not for right answers Immediate action Clear direct communication

While many management practices were developed for simple and complicated domains management in complex domain is an emerging practice. Snowden gives the following guidelines to cope with the ambiguities of the complex domain so leaders can be effective in managing situations within this domain.

- Open up discussion: Complex domain requires more interactive communication in order to generate innovative ideas and develop novel solutions.
- Set barriers: Containers or constraints are necessary for self-regulation so guiding principles or simple rules as strategy (Eisenhardt & Sull, 2001) must be used to delineate behavior.
- Stimulate attractors: Attractors are phenomena that arise when small stimuli and probes resonate with people. They are states or combinations of variables that system state would try to approach. For a business organization organizational values is a strong attractor (Petzinger, 1999). For example when Amazon.com experimented with the idea of letting people opening their shops in Amazon.com this resonated with buyers and sellers thus become an attractor.

- Encourage dissent and diversity: Creating an environment that people can discuss ideas and opinions, listen carefully and speak openly without taking criticism personally enables open communication.
- Manage starting conditions and monitor for emergence: Leaders cannot predict what will emerge in complex context so they focus on creating an environment where positive outcomes emerge and try to catch opportunities arise.

Viewing software development project as a complex adaptive system thus accepting software development as a socially complex activity changes the perception of the project from being a series of planned activities with a forecasted outcome to a system of intersecting feedbacks and information flows created by agents with possibly conflicting interests and diverse set of goals. In this case agility becomes vital and essential for success of the project because since outcomes and the course of the project can't be predicted the team and stakeholders must be flexible to meet the changes and monitor the emergent outcomes.

Applying CAS theory to software development reveals that achieving agility is not a pick and mix approach since all the concepts used are heavily intertwined and mutually reinforcing (Vidgen & Wang, 2006). For example focusing on time boxing and self-organization without regard to need for space and resource for exploration may result in suffering of innovation efforts.

Vidgen and Wang built a framework for "organizing the software development process to achieve agility" based on the complexity theory. Using concepts of complexity and empiric findings from their research they examined agile practices and "reflected on from the perspective of complex adaptive systems" (Vidgen & Wang, 2006).

They framed the implications of complexity concepts on agile software development as shown in the figure.

Here a comparison is made between some CAS properties and properties observed in software development.

Table 2.2 Comparison of CAS and software development

Complexity Concept	Observation on software development
Autonomy is the most important criteria for systems to be complex. If agents are in a rigid structure with command-control hierarchy or dictatorship then there is no social system.	Developers, customers and other stakeholders are not parts of machinery but participants in a problem solving effort.
Interaction of agents, feedbacks, information flows.	People in software development have a wide range of tools and options for interaction from face to face communication to document based knowledge handovers.
Emergence: Emergent Behavior is the result of rich nonlinear interactions of agents. This makes the emergent properties unpredictable.	Many times the resulting software is not as predicted but emerges as the result of constraints and requirements and innovativeness, creativity to meet them.
Edge of the Chaos defines the state in which the CAS is frequently disturbed and doesn't settle down between disturbances.	Freedom is required to cope with change but at the same time enough structure to keep the team together and functioning. Planning in cycles allows emergence and innovation in the software but also supplies barely sufficient structure by iterations and timely work cycles Planning is bottom-up through work items instead of top-down
Nonlinear behavior: a small disturbance in one part of the system may result in serious consequences in some other part or many parts.	In any software project small problems unnoticed in early phases cause big problems in later stages. Small steps that build trust between stakeholders can prevent project cancellation or failure.
Self-organization is the process by which the system and its agents respond to unpredictable events in the system or from outside the system. They change behavior or the structure therefore "adaptive to changes and resilient to attacks"	Developers and stakeholders organize around the goals and create ad-hoc networks to obtain required information and get things done whether these network and subgroups are formal or not(Mintzberg, 1992).
Information flow: Agents communicate within or with outside to unearth requirements, create learning, develop concepts, and coordinate action.	Projects include great amounts of learning. Lessons learned and knowledge generated during the process must be diffused within the project team and stakeholders fast and used for improving outcomes.
Patterns of behavior over time	Time-pacing causes stability to the project team, reduces over-work and anxiety Each team has different pacing Fixed iterations helps stability by preventing over-responding
Coevolution	Organization and project team mutually adapt to each other Coevolution requires knowledge sharing
Interdependent agents with possibly conflicting goals are the basic components of the system. There is no centralized control.	People in development team, product owners, and stakeholders form the software development ecosystem.

Wang and Conboy in their research studied the agile development from the CAS perspective manifested in autonomous but sharing teams, stability embraced with uncertainty and team learning (Wang & Conboy, 2009).

In his study of agile development and complex adaptive systems, Pelrine sampled typical activities of software development and classified them according to their complexity domain (Pelrine, 2011).

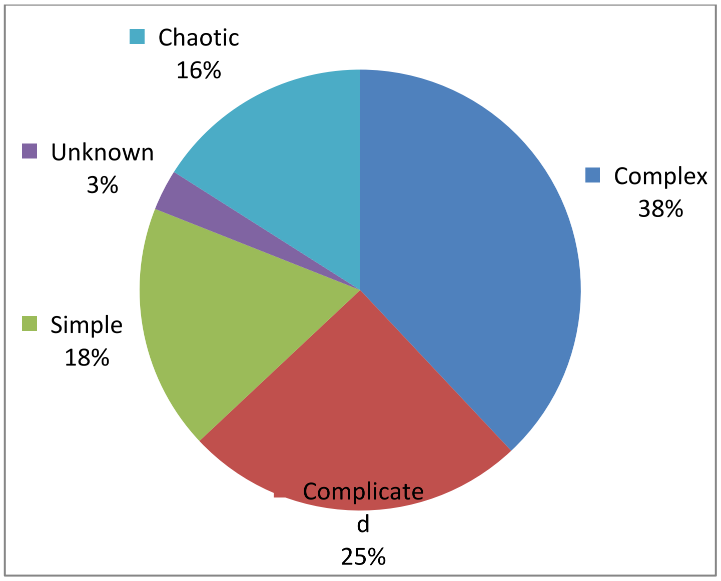

Figure 2.2 Contexts of software development by Cynefin framework (Pelrine, 2011)

The mentioned study revealed that;

- The domain of software development has aspects and activities in all domains with the interactions happening in complex space.
- Each activity of software development may have sub-activities in different domains than the activity itself.
- While technical activities tend to happen in complicated domain, project management and the tasks dealing with interactions with other people are spread over complex and chaotic domains.
-

2.2 SOFTWARE DEVELOPMENT AS A SOCIAL COMPLEX ADAPTIVE SYSTEM

People are the non-linear, first-order components in software development (Cockburn, 1999) thus they can be examined as the autonomous agents of software CAS. Since in both software projects and CAS the building blocks of the system are agents or the people in the project, designing project or methodology means “designing complex adaptive system whose active components are variable and highly non-linear components called people”(Cockburn, 1999).

The human elements of software development CAS has following properties;

- They act in unpredictable ways and one agents actions changes the context for other agents(Olson & Eoyang, 2001)
- People are communication beings requiring face-to-face, in person, real-time question and answer(Cockburn, 1999)
- People behavior change over time
- Act variably from day-to-day and place-to-place
- Have needs to take ownership, responsibility, and initiative about what needs to be done (Herzberg, 1966)
- Personalities affect the ability perform assignments and also affect ability to perform specific assignments(Mills et al., 1985)

When software development is applied in phases with step-by-step planning of activities and activity outcomes the “same set of outputs are expected from identical inputs, but people’s reaction to inputs may vary considerably from day-to-day based on a variety of conditions many of them unrelated to the task at hand”(Highsmith, 1995)

Skilled, suitably qualified, sufficient staff or team members are at the center of agile projects (Cockburn & Highsmith, 2001).

2.2.1 The Agile Team

“The best architectures, requirements, and designs emerge from self-organizing teams” Agile Principle

The team is the entity that gets the work done in agile. Not every group of people is a team. A “real team” is a “small number of people with complementary skills who are committed to a common purpose, performance goals and approach for which they hold themselves mutually accountable”(Katzenbach & Smith, 1993). As every group of people is not a team, every team is not a highly performing team.

In their research on teams Katzenbach and Smith identified the traits of the teams as

- Teams achieve more than the sum of the individuals that compose it.
- Challenge is the fuel of team.
- Strong performance ethics is more important than a team promoting environment.
- Teams need to have a clear purpose and their goals must be aligned with their mission.
- Teams are means they are not ends.
- Individuals exist and develop while working in a team.

The main reason of the high performance of a team is having multiple skills which allow it to respond to challenges. Having a clear goal and approach establishes communications which support effective problem solving and initiative. Teams are built by overcoming barriers it encounters.

Teams should not be confused with work groups. In work groups

- The resulting work is the sum of individual effort of the members while teams achieve more than that.
- There is a defined leadership while teams share leadership
- Work is discussed, decided and delegated while in teams discussed, decided and done together.

In traditional project management the party doing the work is told both what to do, when to do and how to do it. For example in waterfall the timeline for analysis of the user requirements is scheduled, the actions necessary for doing it is decided, a budget is created, resources allocated, a deadline for the documents and the output of analysis phase is set by the project management while preparing the project plans. The analysis team is then given the necessary documentation, work schedule, deadlines and held responsible if they are not met.

Contrary to this approach in many agile methods deadline is imposed by the length of the iteration. For example in Scrum the usual iteration length is thirty calendar days and the team selects how much work it believes it can perform within the iteration and the team commits to work.

Agile teams commit to the work which means by the end of the iteration they either deliver or if they understand that delivery is in jeopardy they renegotiate what can be delivered.

In order to the team to "commit" to work it should be doing the commitment. If someone else is delegating the work then there is no commitment and it is a workgroup.

There are ten characteristics associated with high performance team's success(Hanlan, 2004):

1. Participative leadership. Each member is involved and engaged in decision making and working of the team.
2. Effective decision making. Ability to use rational or intuitive methods depending on the nature of the task to make decisions.
3. Open and clear communication. Constructing shared meaning ensures that the efforts of the individuals are aligned. Agile uses face-to-face communication in order to ensure mutual understanding within the team.

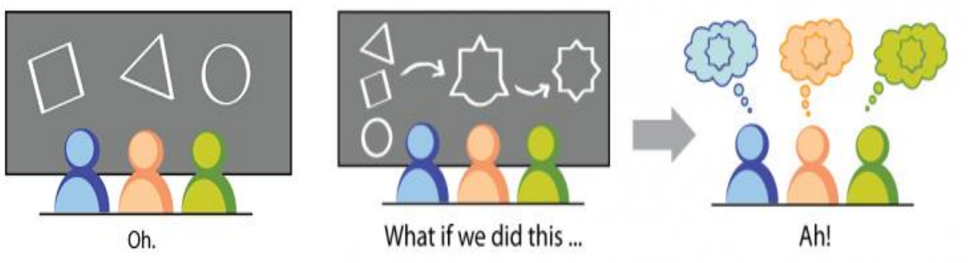

Figure 2.3 Communicating for agreement (Rasmusson, 2010)

4. Valued diversity. Agile teams are cross-functional which means they contain members with a diverse spectrum of skills and experience. When there are many view points better decisions are made and solutions are found. The agile

principle "*the best architectures, requirements, and designs emerge from self-organizing teams*" points this.

5. Mutual trust. Trust is created when team and its members commit to a goal and deliver what they are committed to.
6. Managing conflict. When conflicts are not buried, ignored but dealt with openly and transparently, the team morale is not destroyed.
7. Clear goals. For example in Scrum the role of the product owner is to prioritize the backlog items making it clear for the team the stakeholder and product value thus making clear what the team needs to do.
8. Defined roles and responsibilities. Who does what is clear and members understands which actions show their commitment and support to team and its success.
9. Coordinative relationship. Work members are able to coordinate their work without imposed mechanisms such as coordinating roles.
10. Positive atmosphere. Team culture able to deliver success which is open and positive.

2.2.2 Self-Organization and Managing Self-Organizing Teams

Self-organization is one of the defining aspects of complex adaptive systems and also has an important place in agile software development. It is the CAS property which explains the changes that occur in project teams. Self-organization is the "tendency of a complex system under some circumstances to generate new patterns spontaneously" (Olson & Eoyang, 2001). A system "would be self-organizing if a change is automatically made to the feedback, changing it from positive to negative; then the whole changes from a bad organization to a good one"(Ashby, 2004).

In agile software development the teams are supplied with the environment and resources that makes self-organization possible together with the management practices. Self-organization neither means that the organization is designed and run by workers nor it means letting people do whatever they want to do. It means "that management commits to guiding the evolution of behaviors that emerge from the interaction of independent agent instead of specifying in advance what effective behavior is" (Anderson, 1999).

Self-organizing teams also doesn't mean self-forming teams (Hanoulle, 2007). In his group development model Bruce Tuckman identifies the "necessary and inevitable" phases any team should pass in order to grow, face the challenges, tackle problems, find solutions, plan work and deliver results (Tuckman, 1965). In accordance, facilitating the forming and functioning of the teams is among the responsibilities of the management.

Self-organizing teams are not autonomous, self-managing; they need managers and leaders (Derby, 2011). Self-organization doesn't mean that workers instead of managers engineer on organization design. It does not mean letting employees do whatever they want in whatever way they like. They need clear work goal. They also need the technical skills required to do the work and also the interpersonal skills to work as a team. Also they need tools, information and training.

The role of management is facilitating the formation of self-organizing team and sustaining it. Management chooses what products are going to be built, who will work in which project, the resources (Cohn, 2010). Control still exists but it is exercised by "selecting the right people, creating an open work environment, encouraging feedback from the field, establishing an evaluation and reward system based on group performance, managing the tendency for going off in many directions early on and the need to integrate information and effort later on, tolerating and even anticipating mistakes and encouraging suppliers to become involved early without controlling them" (DeGrace & Stahl, 1990).

An agile team's job is to self-organize around the challenges and within the boundaries and constraints put in place bay management (Cohn, 2010). Management comes up with appropriate challenges and removes impediments to self-organization.

Leaders influence teams in subtle and indirect ways. Changing the teams composition, setting new standards of performance, selection system for team members, and other practices and actions affect the team's performance and how the team will respond to these is impossible to accurately predict by the leader.

Managing teams and self-organizing teams in particular is different from managing traditional organizations or workgroups. Many organizations attempt to create high performance teams but without understanding the underlying dynamic to create, and without having adequate time, resources and managerial skills to develop them, so they fail to do so (Hanlan, 2004).

Managers or leaders may restrain themselves from micro-managing agile teams by (Cohn, 2010):

1. Giving full autonomy to the team for solving the problem they’re given.
2. Helping the team as well as monitoring the progress. For example the Scrum Master is responsible for protecting the team from distractions and impediments from the outside but also from the ones generated within the team.
3. The team as a whole responsible for the outcomes.
4. Learning must be amplified through sharing information. The team must have access to all relevant and necessary information on a timely base.

As it is demonstrated through explaining the agile practices with concepts of management of complex systems, software development fits the domain of complexity since the social aspects and the interconnectedness and interdependency of all actors and elements compose more of a nonlinear emergent system then a linear mechanistic structure.

CAS theory provides a sound theory for software development and management of software projects. Here the CAS and related Cynefin framework are examined as the theory and the foundation of managerial practices of software development projects.

CHAPTER 3

THE PARADIGM OF PROJECT

Paradigm is a term which is generally used in science and epistemology. It means “a pattern or model” (Oxford English Dictionary, 2010). A paradigm is “an outstandingly clear or typical example or archetype”, “a philosophical and theoretical framework of a scientific school or discipline within which theories, laws and generalizations and the experiments performed in support of them are formulated”(Webster, 1981).

Thomas Kuhn defines a paradigm as “universally recognized scientific achievements that for a time provide model problems and solutions for a community of researchers” (Kuhn, 1996). A paradigm is a set of assumptions that decide how the world is perceived. This is valid for perceiving how photography should work(Owen, 2004) to what a project plan should look like.

In this part of the thesis the dominating paradigm of project which is the traditional approach is examined. Then the agile (or Lean) approach to projects which is associated with the increasing success of software projects (GAO, 2012)is examined.

Projects management approaches are the result of management and thinking systems of their ecosystems. For example while the approaches called traditional project management are the results of western thinking of management which puts the emphasis on planning, command and control approaches, planning activities in detail and separating the thinking and doing. Agile project management and lean approaches on the other, hand puts the people, adaptation, harmony, understanding, mutual respect and creating value for the other party to the focus of the project.

The traditional project management is based on the Scientific Management understanding of Frederick Taylor and his colleague Henry Gantt. The thinking in Taylors approach to the management is that faster work can be assured by “standardization of methods, enforced adaptation of the best implements and working conditions and enforced cooperation” and that the workers are incapable of understanding what they are doing(Taylor, 1934). Taylorist model of management is a system of rationalization of production with the focus on separation of the organization and management of work from the operative that is those who are actually doing the job (Lipietz, 1992). This approach to management is also consistent with the "command and control" model of management of Henry Fayol in which he proposed that successful management requires the functions of planning, organizing resources, commanding, coordinating and controlling the work.

The project management advocated by the PMI Institute which codifies the project management practices and one of the most influential organizations in the area project management uses the approaches of Fayol, Gantt and Taylor in which the planning monitoring and controlling are the basic activities of project management.

In traditional project management the requirements are considered to be explicitly definable and documentable at the beginning of the project through detailed analysis of customer demands and behavior.

Agile project management on the other hand focuses on adaptability and flexibility with the focus of the management on interaction of the humans participating in the project (Beck et al., 2001).

The agile approach to software projects can be illustrated as the reversing the project triangle of traditional project management. This approach is closer to a "temporary production system" structuring of the projects in Lean construction (Koskela & Howell, 2002).

The traditional plan oriented, "predict the output, plan the activities" approach is unable to cope with the increased complexity and uncertainty of contemporary projects. This approach to a project is also very different from the actual practice of software development in which a software development team is brought together for a temporary (sometimes indefinite) duration. Many medium to large enterprises have their specialized IT departments and many large companies have in house software development teams which work as employees not as temporary project members.

To obtain coherence between managerial approach and actual practice in using agile software development, here the project is examined as temporary production processes thus projecting the agile approaches over some well-founded product development and production management methods such as Lean and TOC. This is because in production management it is the natural course to focus on the quantity and quality of the product for matching the demand to the supply(Slack et al., 2004) as agile software development focuses on meeting customer satisfaction, quality and sustainably of the processes.

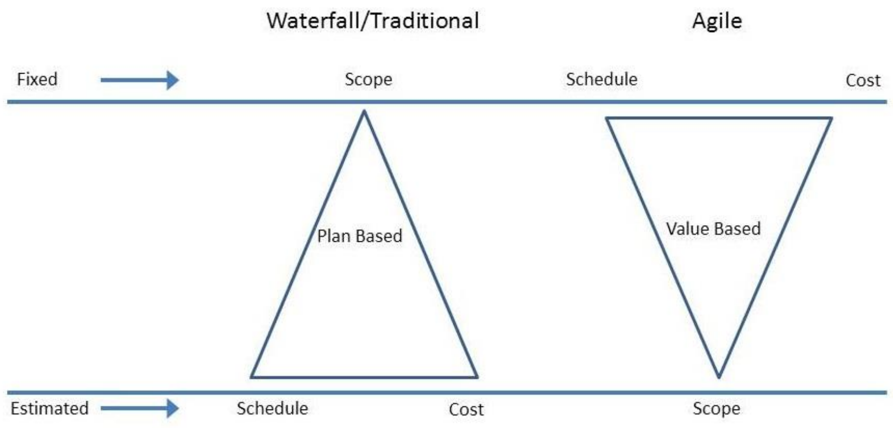

Figure 3.1 Project triangles of waterfall and agile project management (adopted from Leffingwell, 2011)

In agile methods the team is formed for the project. The scope emerges from the interaction of the team and customers. The budget or the schedule, in some cases both of them are fixed from the project initiation. The customer together with the development team decides what item has priority and continuously prune and prioritize the backlog items. The scope is very flexible since the customer is free to demand change in the product at any phase of the project. This way in agile approaches the requirements of the customers are handled in “a more interactive and just-in-time way” (Leffingwell, 2011).

This results in eliminating the iron triangle which symbolizes the dependency of the scope, cost and schedule of the project variables used in traditional project management. In other words whereas in waterfall the scope of the project is fixed at project initiation, the schedule and budget are estimated and planned for; in Agile the resources and schedule or development time-frame is fixed, the scope is estimated and emerges throughout the project duration. This change in perspective and understanding of the concept of project transforms the project to a temporary production.

Perceiving software production as temporary production and grounding this perception upon CAS theory provides an explanation of the behaviors observed in software development. This improves understanding and provides means to predict future course of action. Also it enables adaptation of innovative and useful processes from production settings to software development projects. This way cooperation and sharing of common experiences through a common knowledge framework is enabled among different project teams." Such an understanding provides a basis on which tools for analyzing designing and controlling be built" (Koskela & Vrijhoef, 2001)

Since production systems are of pivotal importance for this thesis and the agile approaches examined in this thesis are production originated here the two most fundamental production methods are reviewed.

3.1 PRODUCTION SYSTEMS

We are going to win and the industrial West is going to lose out: there's nothing much you can do about it, because the reasons for your failure are within yourselves. Your firms are built on the Taylor model; even worse so are your heads. With your bosses doing the thinking while the workers wield the screwdrivers, you're convinced deep down that this is the right way to run a business.

For you, the essence of management is getting the ideas out of the heads of bosses and into the hands of labor.

We are beyond the Taylor model. Business, we know, is now so complex and difficult the survival of firms so hazardous in an environment increasingly unpredictable, competitive and fraught with danger that their continued existence depends on the day-to-day mobilization of every ounce of intelligence. Konosuke Matsushita, 1979

Production systems were the core of business in the past century. Many management and organization theories and practiced have originated from production systems and production economy. Although there are production systems as many as the number of production organizations at the most basic level production methods are separated into two as the American production system (mass production system) and the Toyota production system.

3.1.1 AMERICAN PRODUCTION SYSTEM

American production system is named "American System" from the fact that in the 19th century it was associated with the American companies implemented this production system. The APS has two defining characters (Hounshell, 1984):

- Interchangeable parts.
- High degree of mechanization which results in the more efficient use of labor than making things through skilled hand workers.

The historical development of American Production System begins with the phenomenon of interchangeable parts.

Interchangeable Parts: Until 18th century handguns and rifles were handmade by skilled workers one by one. Parts of one rifle did not fit to another since they were handmade and parts were not interchangeable. This meant when there was a malfunction in one of the parts whole rifle was discarded. In 1785 French gunsmith Honore Blanc proposed to the French army to mass produce rifles. As for demonstration he produced batches of interchangeable parts and in a demonstration quickly assembled a number of rifles.

Thomas Jefferson who at the time of demonstration acting as envoy to France during the American Revolutionary War, sent details of Blanc's methods to America. Eli Whitney duplicated Blanc's methods and demonstration. When other American producers begin to use this method complex merchandise like sewing machines and typewriters with interchangeable parts were produced (Alder, 1997). As United States grew dramatically as an industrial power the new manufacturing system got most of the credit for success.

Mechanization (Interchangeable People): From the early 19th century machines began to take the work and place of craftsman with required technical skills to the job. The craftsman needed a great time and effort to train. Using the machines doing the work required little skill and training.

One of the most famous names in the mechanization era is Frederick Winslow Taylor (1856-1915) whose work would later become the substance of industrial engineering, operations research and manufacturing engineering pioneered the breaking down of work. His approach which he himself called the "scientific management", provides "detailed instruction and supervision of each worker in the performance of that worker's discrete task" to the worker (Taylor, 1903).

Mass Production is made famous by Henry Ford. Henry Ford's Ford Motor Company introduced the economic paradigm in which the APS flourished is little variety, large lots. Henry Ford is famous for saying "they can choose which ever color they want as long as it's black".

Mass production is built around basically an assembly line. It is about making many copies of products, very quickly, using the assembly line in which workers do their individual work on the partially complete product. In mass production the "skill is built into the tool" which makes workers replaceable easily.

Mass Production began to fail when markets changed and customer demand began to diversify. What followed Mass Production was the conventional production method with large variety, small lot (TWI, 2012). Markets are born; they mature, change through gradual or sudden expansions or contractions, and eventually die(Kotler & Keller, 2008). As markets mature customer demands diversify and stimulate the development of large variety small lot production. Because of many restrictions production departments of manufacturing companies cannot make products in accordance to every individual customer demand. Production departments will try to produce goods according to the schedule table in large lots, without stoppages in production line in order to increase the efficiency of production and decrease in the operational usage of the equipment. They will increase stock and inventories against breakdowns, defective products and absenteeism (TWI, 2012).

This creates an inertia which reduces innovation and creativity in order to achieve high levels of efficiency in production. Companies in this situation lose their ability to respond to changes in customer requirements.

3.1.2 TOYOTA PRODUCTION SYSTEM

When Kiichiro Toyoda began manufacturing cars in the post-war Japan at October 1949 it was clear to him that his company could not compete with the mass production model

American manufacturers used (Womack & Jones, 2003). Mass production meant making thousands of identical parts to gain economies of scale but in Japan required raw materials were scarce, orders were not plenty and the demand continuously varied. So economies of scale were impossible. What Kiichiro Toyoda envisioned was that every part arrived to the assembly line “Just-in-Time” for assembly without any waiting and every part is produced on demand. This way the company wouldn’t spend money in stocking materials that was not already sold.

It was Taiichi Ohno who was a machine shop manager working at Toyota developed the Toyota Production System. He studied Ford’s production system and examined the inventory management system in American supermarkets. Adding the insights of the workers he managed and his experience and through years of experimentation (Poppendieck & Poppendieck, 2006) Toyota Production System was developed. Taiichi Ohno explains that the system rests on two pillars: Just-in-Time flow and autonomation which he calls Jidoka (Ohno, 1988).

3.1.2.1 Just-in-Time Flow

Cost is the value of money that has been used up to produce something and hence not available for use anymore (Sullivan & Sheffrin, 2003). The cost of goods can be reduced in two main ways. One is the economies of scale which means the unit cost of a product is reduced as the size of the facility ant the usage levels of other inputs increase (Sullivan & Sheffrin, 2003). This method used by the American System of Manufacturing.

The other method for reducing costs is improving the flow of a service from first receipt of a customer’s demand to the eventual satisfaction of that demand (Seddon, 2008). In trying to manage and reduce costs firms often raise total costs by creating What the Just-in-time flow means eliminating the stockpiles of in-process inventory that was manufactured in the name of economies of scale before they are needed. By eliminating large stockpiles of pre-produced parts which are called inventory the product is produced in small batches so the system is organized to make quick changes in order to manufacture different parts needed (Shingo, 1981).

Lowering the amount of inventory used in production is important because it is like the water level in a river with boats on it. When there is enough water the big rock under the water remains hidden but when the water level decreases the rock needs to be removed. When water is lowered more, smaller rocks are revealed and they need to be removed. Furthermore the inventory is a financial burden to the organization and a waste which can be removed through proper management methods.

The defects in the product and processes that are not useful or efficient are when the inventory levels are high. Other than the cost of keeping inventory these defects are hidden waste that costs money, time and other valuable resources and remain hidden until the inventory levels are dropped.

Since Toyota Production System targets to improve overall production instead of maximum utilization of a few processes revealing these hidden costs is important.

Translated into software development Just-in-time aims deploying after quick iterations thus saving in deployment, integration and training instead of adding many features (user requirements) to the software and taking long time to deploy, deploying after quick iterations thus saving in deployment, integration and training(Poppendieck & Poppendieck, 2006).

Adopting Just-in-Time means stop trying to maximize local efficiencies and improve overall production performance by revealing the hidden problems and costs by lowering the inventory levels.

The term *lean* is first coined by John Krafcik in his 1988 article “Triumph of the lean Production System” (Krafcik, 1988). His research is developed and made popular by “The Machine That Changed the World” book which was written by Jim Womack, Daniel Jones and Daniel Ross (Womack et al., 1990). Toyota Production Method is known as Lean Production since then. Since the thinking behind the Lean Production is fundamentally different from the established methods and habits of management there have been difficulties in applying Lean in many organizations (Womack & Jones, 2003).

The point in Lean is that it “transfers the maximum number of tasks and responsibility to those who are actually doing the value adding and it has a system for detecting defects that quickly traces every problem once discovered to its ultimate source”(Womack et al., 1990).

3.1.2.2 Jidoka: Highlighting/visualization of problems

Sakichi Toyoda was a Japanese inventor and industrialist who is referred to as the father of the Japanese industrial revolution. He is also the founder of Toyota Industries Co. Ltd. His most famous invention was the automatic power loom in which he implemented the principle of Jidoka (autonomous automation, automation without a human touch). The principle of Jidoka which means that the machine stops itself when a problem occurs became later a part of the Toyota Production System. The machines produced by Toyota could operate without any operators because when anything went wrong the machines detected the error and shut down automatically. Autonomation or Jidoka in its Japanese name; means that work is organized so that even smallest abnormalities are detected, work is stopped and the cause of the problem is remedied before the work (Ohno, 1988). The purpose of autonomation is that it makes possible the rapid or immediate address, identification and correction of mistakes that occur in a process.

For example contrary to Mass Production rather than waiting until the end of production line to detect and separate defective products, Jidoka makes it possible that as soon as an error occurred the production is stopped, error detected and the reason of the error is eliminated thus preventing further errors.

Pull System: The pull system defines a technique in which any part is manufactured only at the demand of the customer. In the traditional approach to software project management during the planning phase long-term view of the software application is visualized and planned in detailed. This approach causes to increase the complexity of the project. Also this delays the software and end-users lose interest (Nallasenapthi, 2006).

Work Cells: Operations or work teams in lean production are organized in work cells which better utilize people and improve communication.

Batch Size Reduction: Manufacturing organizations produce large batch sizes in order to maximize machine utilization. Lean development links production of goods to customer demand so the ideal batch size is one. Since batch size of one is not always practical the goal is to practice continuous improvement to lean principles.

3.2 TRADITIONAL PROJECT MANAGEMENT

In this part of the thesis the de-facto project management standard used in many organizations and projects will be examined.

The de-facto reference on project management is Project Management Body of Knowledge (PMBOK) Guide published by Project Management Institute. PMBOK contains the *"sum of knowledge within the profession of project management"*. It contains the "*traditional practices"*(PMBOK, 1996). It identifies and describes the subset of the knowledge body which is "*generally accepted*". By generally accepted the authors mean that the "knowledge and practices described are applicable to most projects most of the time". This book is the reference guide for thousands of project managers around the world.

PMBOK defines a project as "*a temporary endeavor undertaken to create a unique product or service*" (PMBOK, 1996).

Examples of projects include:

- Developing a new product or service
- Effecting a change in structure, staffing or style of organization
- Designing a new transportation vehicle
- Developing or acquiring a new or modified information system
- Constructing a building or facility
- Running a campaign for political office
- Implementing a new business procedure or process.

There are two main factors that make a project; these are being temporary and being unique. Temporary means that every project has a definite beginning and a definite end. The end is reached when the projects objectives have been achieved or when it becomes clear that the project objectives will not or cannot be met so the project is terminated. Being temporary does not necessarily mean that the project duration is short since many projects continue for several years and sometime decades in case of project such as space exploration, large transportation projects. Also temporary does not apply to the product or service created by the project. The uniqueness of the project refers to that something which has not been done before and therefore unique (PMBOK, 1996). In order to meet needs and expectations managing the project involves balancing competing sometimes conflicting demands such as;

- Scope, time ,cost
- Stakeholders with differing needs and expectations
- Identified requirements (needs) and unidentified requirements (expectations)

"Project management is the application of knowledge, skills, tools, and techniques to project activities in order to meet or exceed stakeholder needs and expectations from a project" (PMBOK, 1996). The required knowledge and common practice of project management is separated into knowledge areas called Project Management Knowledge Areas which describe the process components of projects.

The PMBOK Guide shows nine project management knowledge areas:

- Project integration management
- Project scope management
- Project time management
- Project cost management
- Project quality management
- Project human resource management
- Project communications management
- Project risk management
- Project procurement management

3.2.1 Project Phases and The Project Life Cycle

Projects are divided into several project phases to reduce the uncertainty and make the project manageable. This division is made to provide better management control and to establish links between the phase of the project and the ongoing operations of the organization. The sum of all project phases is named as the project life cycle.

Project phases are separated by completion of one or more deliverables which is a "tangible verifiable work product such as a study, a detailed design or a working prototype" (PMBOK, 1996). The deliverables and the phases are defined at the beginning phase of the project as a general sequential logic to ensure that the product of the project is properly defined from the beginning. The phases are named after the primary phase deliverable such as requirements, design, build, text, start-up, turnover etc.

The phases of the project life cycle are sequenced according to the technology transfer or hand-offs. For example the designs of a construction created in one phase are the input of the next phase. Deliverables from the preceding phase are approved before work starts on the next phase. But sometimes a subsequent phase begins prior to approval of the previous phase deliverables when the risks involved are deemed acceptable (PMBOK, 1996).

Project life cycles generally define:

- What is the input and output of each phase?
- What technical work should be done in each phase?
- How the phases should be sequenced?
- Who should be involved in each phase?

For example U.S. Department of Defense directive 5000.2 describes a series of acquisition milestones and phases such as in the following table.

Table 3.1 Project Phases and Milestones

Phase	**Milestone**
Determination of Mission Need	Concept Studies Approval
Concept Exploration and Definition	Concept Demonstration Approval
Demonstration and Validation	Development Approval
Engineering and Manufacturing Development	Production Approval
Production and Deployment	Operations And Support

An example for project lifecycle can be given as in the following construction project phases.

- Feasibility
- Planning and Design
- Production
- Turnover and start-up

A similar life-cycle for a software project would be like:

- Requirements

- Design
- Implementation
- Testing
- Deployment

3.2.2 Project Stakeholders

Project stakeholders are individuals and organizations who are actively involved in the project or whose interests be positively or negatively affected as a result of project execution or successful project completion. It is the duty of the project management team o identify the stakeholders determine what their needs and expectations are, and then manage and influence those expectations to ensure a successful project (PMBOK, 1996).

3.2.3 Project Management Processes

The methodology defined in the PBMOK Guide is process based which means; the work described as being accomplished is done in processes(PMBOK, 1996). Processes consist of

- Inputs (documents, plans, designs etc.)
- Transformation (Tools and Techniques, mechanisms applied to inputs
- Outputs (products, documents etc.)

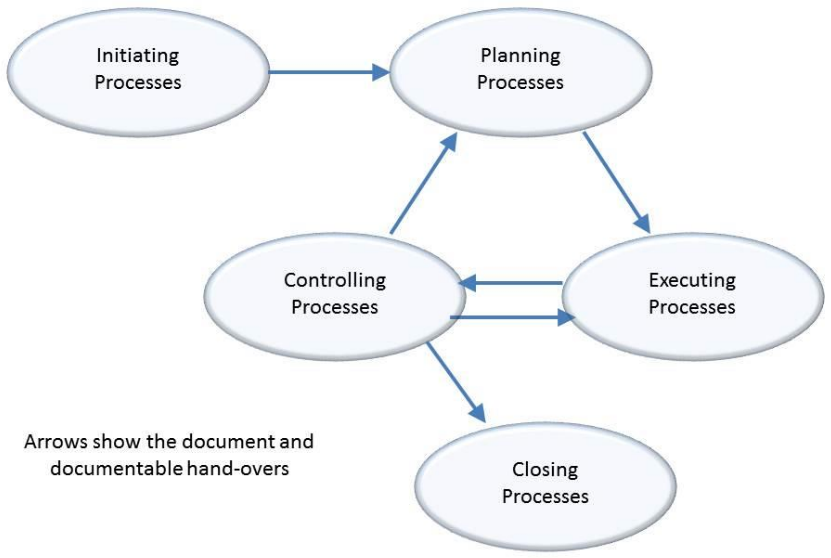

Figure 3.2 Project management phases in traditional project management (Adopted from PMBOK, 1996)

Processes overlap and interact throughout a project or the phases of the project. The processes generally in the guide are divided into five phases (PMBOK, 1996):

1. Initiating
2. Planning: creating and maintaining a workable plan to accomplish the business need to address the business requirement that the project intends to meet.
3. Executing: coordinating the resources including the people to carry out the plan.
4. Monitoring and Controlling: measure progress and take corrective action when necessary in order to ensure the project goals are met.
5. Closing: formal acceptance or rejection of the project output and ending the project.

3.2.3.1 Planning Processes

The traditional approach to project management focuses on the planning phase of the project(Koskela & Howell, 2002). The reason given in the PMBOK for this is, since the project “involves doing something which has not been done before” planning is “of major importance”. “Planning is not an exact science”(PMBOK, 1996). Two different teams could generate very different plans for the same project. If one aspect of the plan (e.g. the completion date) is unacceptable to the stakeholders then project resources, cost or even scope may need to be redefined starting from the beginning because of their dependencies to each other

In traditional approach since the processes are dependent to each other they must be performed in essentially the same order they are planned and sequenced. In this approach of project management activities necessary to accomplish the scope of the project needs to be defined before they can be scheduled or budgeted.

The processes which are called core processes form the essence of planning phase. The core processes are:

- Scope Planning – development of the scope statement
- Scope Definition-
- Activity definition –
- Activity Sequencing-
- Schedule Development
- Resource Planning
- Cost Estimating-
- Cost Budgeting
- Project Plan Development

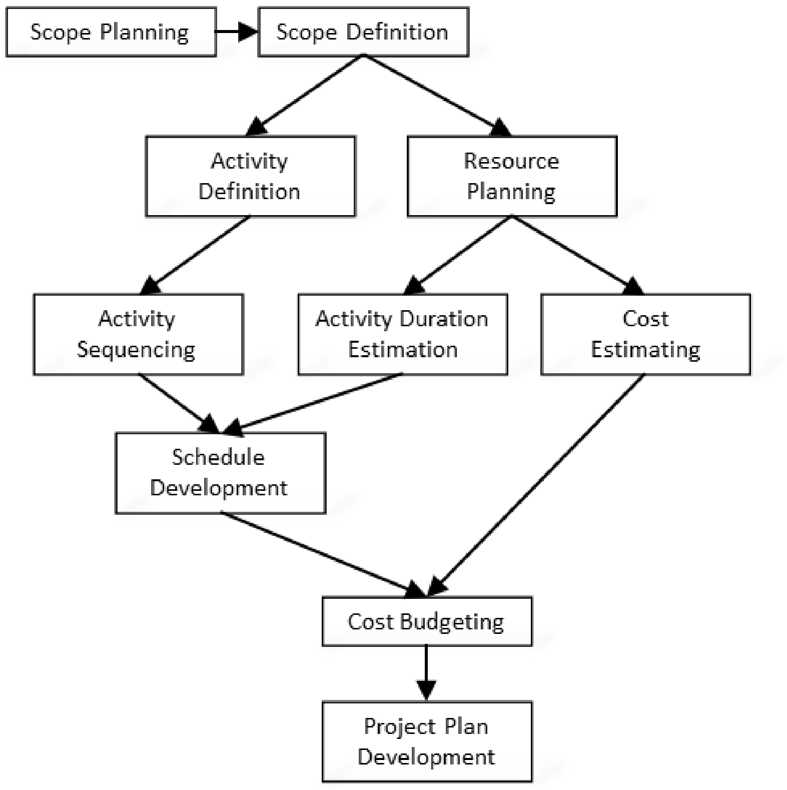

Figure 3.3 Core processes for traditional project management. Adopted from PMBOK, 1996

Besides core processes the project includes some non-optional facilitation processes such as quality planning, organizational planning, communications planning, risk identification, risk quantification, risk response development, procurement planning, solicitation planning. These depend on the nature of the project. While some projects contain high degree of risk before the project begins, some others have low to no initial risk but since the cost and schedule is aggressive become a high risk project (PMBOK, 1996).

3.2.3.2 Executing Processes

The executing of the processes is carrying out the project plan by performing the activities decided and sequenced in the plan (PMBOK, 1996).

3.2.3.3 Controlling Processes

Control is the measuring the project performances against the project plan, identifying variances and taking corrective action. When variances observed adjustments to the plan are made by repeating the appropriate project planning processes(PMBOK, 1996). When a finish date for an activity is missed this may require that budgeting, scheduling for the subsequent activities, staffing plans to be altered.

In both the classic project management and the waterfall methodology of software development people are compartmentalized into stages (business analysts, architects and designers, developers or coders, testers etc.). Software development is approached like a production line conveyor belt. Communication between functional groups and teams is maintained through documentation. Business analysts compile the system specifications and pass the finished requirements specification document to “architects”. Architect plan the software system and create diagrams that show how the code should be written. These design diagrams are passed to “coders” who implement the design (Szalvay V. , 2004).

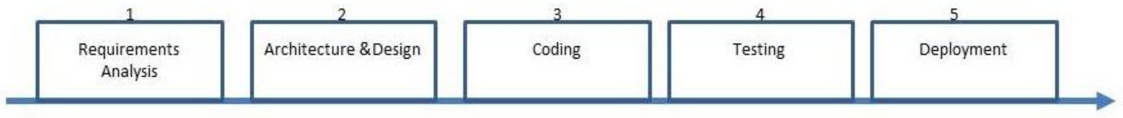

Figure 3.4 Traditional project management phases.

Projects are conceived and completed by people (Howell, Macomber, Koskela, and Draper, 2004). It is the people who apply tacit knowledge, scentific theories, experience for everyday work or problem solving or innovation. The main reason many project fails is, less than useful requirements, lack of experience of project managers or lack of motivation of the staff. Many times people focus on the tools and theories but this alone shows that people are the problem much many times more than they are solutions.

3.2.4 Drawbacks of Waterfall Development

The main disadvantages and shortcomings of waterfall development is software projects can be summarized as:

- Risk mitigation is postponed until late stages
- Document-based verification is postponed until late stages
- Attempting to stipulate unstable requirements too early; change of requirement is perceived as a bad thing in waterfall.
- Operational problems discovered too late in the process at the acceptance testing
- Lengthy modification cycles and rework
- The requirements are functionality focused, quality attributes are not included (Johansen and Gilb, 2005).

3.3 AGILE PROJECT MANAGEMENT AND SOFTWARE DEVELOPMENT

Any prediction of the future ensures a poor outcome (Ackoff et al., 2006). There are two important terms about the future in any design including project planning; forecast and assumption. Forecasts are about probable futures; assumptions are about possible futures. Airplanes carry lifejackets for passengers despite the fact that they do not forecast having a crash to the sea on their trip. But it is assumed that a crash to the sea (and survival) is possible however unlikely it is.

There are two ways to deal with the assumed futures. When there are relatively few and explicitly describable possible futures planners take into account their outcome and plan for each possibility. This process is called contingency planning. When one of the possible future became reality the prepared plans is invoked and used. For example a company can develop contingency plans for their market share to grow, staying the same or fall. When one of the cases becomes reality appropriate plan is ready to be used.

When there are more contingencies than can be planned for or it is non-feasible to do so, or impossible to prepare that many plans. The way to deal with such a case is design the organization, institution, and the project plan in a flexible and responsive way so it can change rapidly and effectively to meet any contingency which it may or may not be prepared for (Ackoff et al., 2006). This is the cornerstone of agile thinking.

Agile project management and software development approaches emerged in response to the inefficiencies and shortfalls of traditional project management methods which are unable to cope with the environmental uncertainty and complexity and fail to deliver the required high quality software thus not meeting the customer requirements.

The inefficiency of traditional software development methods resulted in the emergence of agile software development methods such as Extreme Programming (XP),Dynamic Systems Development Method (DSDM), Feature-Driven Development(FDD), Adaptive Software Development, Scrum, Open Unified Process (Open UP), Agile Unified Process (Agile RUP), Kanban, Lean, Crystal Methods to meet the demands in rapidly changing environments (Highsmith, 2002).

The term agile is used to justify many different sets of methods. Since agile principles are what they are -principles- in practice they have been interpreted in many different ways. When criticised agile principles are presented as the direct opposite of plan, structure and hardcore architecture which are considered the core compenents of traditional waterfall approach (Rakitin, 2001).

Agility "*is the ability to both create and respond to change in order to profit in turbulent business environment*" (Highsmith, 2002). Agility is the ability to adapt to change in a responsive manner (Marquis, 2012).

An agile entity whether it is an organization, workforce or software development team is proficient at change (Kidd, 1997). An agile organization is able to respond to unpredictable changes in its environment in a timely and effective way.

Examination of agility concept across disciplines such as manufacturing, business and management as well as software development reveales that there are several concepts such as flexibilty, iterations, complexity and leanness are interwined (Cockburn and Fitzgerald, 2004). Thus a broad definiton of agility is provided as "the continual readinss of an ettiy to rapidly or inherently, proactively ar reactively, embrace change, through high quality, simplisti, economical components and realtionships with its environment" (Conboy and Fitzgerald, 2004).

Lyytinen and Rose (2006) examined agility in the context of information systems development(ISD). The agility in ISD is about why and how ISD organizations sense and respond swiftly as they develop and maintain information systems applications.

Agile organizations embrace change by creating change or responding to change. They outperform their competitors by sensing and responding to shifts in business environments quickly (Marquis, 2012). They can change directions quickly and they are flexible; they can change what doesn’t work and switch methods.

Agility in business context has four factors(Marquis, 2012):

1. Configurability: Ability to **change** the configuration or design of the system.
2. Responsiveness: The ability to change the system **quickly** in response to situational changes.
3. Employee adaptability: Teams need to be able to **adapt** to the changing environment and configuration.
4. Process-centric view: Critical tasks or processes must be kept on execution with little variation. Processes and **process improvement** is critical for delivering value for the customer.

Software development projects are about software development at first which is solving problems of the customers and then they are projects which are delivering value to the customer in a structured way.

Software development projects are in most cases solution to unique problems or development of first of kind products (Atwood, 2006). Almost in every case the product developed in the software project is unique to specific client or customer (Poppendieck & Poppendieck, 2003).

In 2001 the creators of many of the agile software development methodologies and practitioners came together and created an Agile Manifesto which defines and synthesizes ho core beliefs underlying the movement. This manifesto summarizes their belief that there is a better way to produce software.

Table 3.2 Manifesto for Agile software development

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions	over	processes and tools
Working software	over	comprehensive documentation
Customer collaboration	over	contract negotiation
Responding to change	over	following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck	James Grenning	Robert C.
Mike Beedle	Jim Highsmith	Martin
Arie van Bennekum	Andrew Hunt	Steve Mellor
Alistair Cockburn	Ron Jeffries	Ken Schwaber
Ward Cunningham	Jon Kern	Jeff Sutherland
Martin Fowler	Brian Marick	Dave Thomas

Table 3.3 Principles behind the Agile Manifesto

We follow these principles:

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

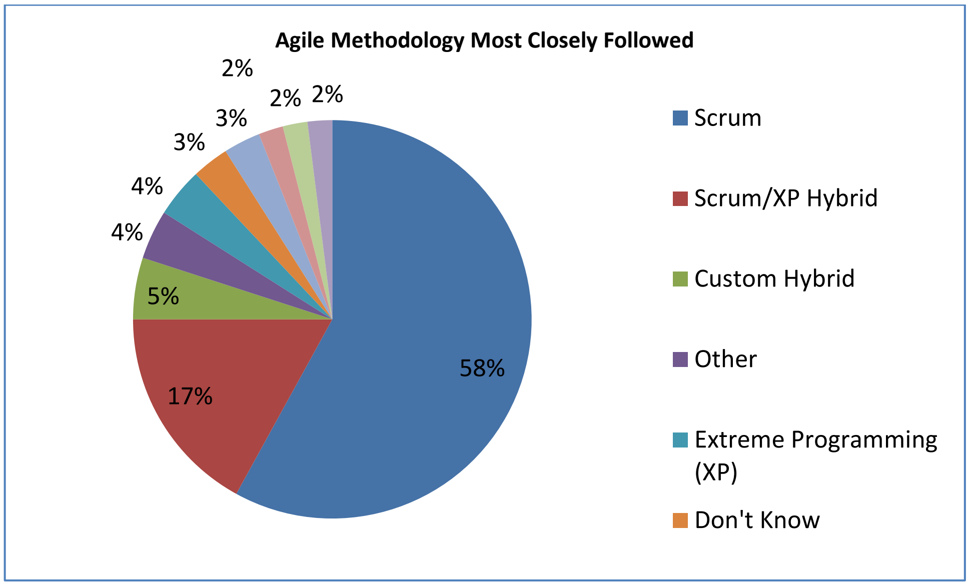

Figure 3.5 Most widely adopted agile methods

The idea behind the agile development method is to develop the software or product through repeated cycles –iterative, and in smaller portions at a time. This way the developers can take advantage of what was learned during the previous development cycles. This learning is the result of both developing and using the emerging system. The product can be modified and new functions can be added in these iterations with higher quality and better insight as a result of learning process.

While at first it may be seem as a collection of different methodologies; agile approach is a shift in paradigm and understanding of what a project is, how projects are managed, how work is done, how value is created in software projects.

Agile development is not a methodology; it is an ecosystem of people, relationships and uncertainty. Agile development is describing a holistic environment that includes a “complexity paradigm”, collaborative values and principles and “barely sufficient methodology“(Highsmith, 1995).

The quick change which sometimes becomes turbulence in the markets creates opportunities but exploring these opportunities requires a different mindset than those in a stable business. In production processes the main purpose is to reduce the amount of change or variations. In exploration what needs to be reduced is the cost of change instead of amount of change because change is unavoidable (Highsmith, 2002).

Unlike plan-driven approach to project management Agile doesn't try to lock down all the requirements before designing and building. In Agile top management (or main customer) establishes the few critical business objectives that serve as planning guideposts (Shriver & Birckhead, 2008). Agile teams use an approach similar to what inventors use. They try something small, test it in use, get customer feedback if it show promise build on it by iterations else try something else. This is how complexity is managed in management science (Snowden & Boone, 2007) and how empiric processes work (Schwaber, 2004).

Agile offers several advantages to business owners and teams such as(Shriver & Birckhead, 2008):

- Getting products to markets faster with reduced development cycles
- Business leaders see the results quickly and have more control over development costs.
- Projects destined to fail can be cancelled early with low risk and losses.
- Priority changes can be mirrored in the project with minimal waste.
- Collaboration of business leaders with development teams builds trust trough communication and delivery of committed product result in high morale.
- Increase in transparency and accountability.

In agile; analysis, testing, design, coding, and anything needs to be done is done:

1. in very short iterations
2. by the self-organizing team
3. Iterations which should result immediately with working software.

When developing software with agile the principle "working software is the primary measure of success" is realized by three facts should be considered (Rasmusson, 2010).

1. It is impossible to gather all the requirements at the beginning of a project.
2. Whatever requirements you do gather will change.
3. There will always be more to do than time and money will allow.

The first fact reveals that the project begins without knowing everything up front. To the contrary requirements are discovered as the project continues. The second means that change is unavoidable so the plans are made when needed.

The third stresses the continuous prioritization of the revealed requirements so the important things get done first and less important ones are saved for the last.

The implicit assumption behind waterfall is that the user requirements can be known and is revealed through analysis. As a result of this assumption it is implied that with detailed requirements specifications, design specification, detailed planning and risk analysis and risk management and the like; it is possible to deliver “the scope” with a fixed schedule and fixed resources (PMBOK, 1996). User requirements are almost always incomplete and inconsistent; while priorities and requirements in every class of users of the software is changing and conflicting (Cook, 2011).

Scope management related to attempting waterfall practices was the single largest contributing factor for failure (Thomas, 2001). Changing business requirements are the highest ranked source of software project failure.

Trying to define the entire requirement set (establishing the scope) followed by a long gap before those requirements are delivered is no longer feasible. It is almost guaranteed that the requirements will change after they have been documented.

Use of agile approaches increases the success rate of projects which means meeting the customer requirements and creating value, fast (StateOne, 2010). Agile approach is used from personal software projects to space station system integration, from construction projects to national agriculture development projects United Nations (Matta & Ashkenas, 2003). It is used for customer product developments and international assistance programs.

- 93% of its clients reported improved productivity as a result of using Agile methods(Fretty, 2005)
- 88% found the quality of the products to be better
- 83% experienced better business satisfaction (Sliger & Broderick, 2008)

CHAPTER 4

SOFTWARE DEVELOPMENT APPROACHES

Our modern world depends on software. Software produced embodies most of the world’s intellectual property (Leffingwell, 2011). Since the introduction of the computers to daily life the calculation power of the computers has increased by 10000 times. What is more important is that the scope and reach of the computers and software have been so widened that the consequences of success or failure has increased exponentially both in terms of economics and human life. Consider the software errors in planes that cause human lives (Dershowitz, 2006) or the economic scale of the damage to economy. By 2002 the estimated cost of software bugs to U.S. economy was $60 billion each year. (Tan, 2009). The security flaws in software to U.S. economy costed $180 billon a year (Rice, 2007).

Many different methodologies-process frameworks used to structure, manage and control work- have been developed and used for software development.

Not just unable to meet user needs but also deliver lower-quality software and more late than needed software development practitioners tried to move to mare “agile” and “leaner” software methodologies. This movement resulted in a migration to more exploratory and lighter-weight processes through the software development history.

No “best practice” fits all projects, teams and situations. Conventional wisdom and approaches fail in the face of rapid change, complexity and uncertainty (Barabba et al., 2002). This creates insecurity because the world is in a period of very rapid change and managers feel that there must be a right answer(Drucker, 1997).   This leads to managers either sticking to the method they know best or they read about a fad (latest management fashion), try it, find that it doesn’t work abandon the effort and move on the next thing (Christensen, 2001).

Different types of projects carried out in different environments are likely to require quite different development processes if they are to be successful (MacCormack & Verganti, 2003). Every project faces risks(Matta & Ashkenas, 2003) and uncertainties which happen to exist as a result of the interaction of forces inside or outside the project organization that have the potential to affect how the project organization gather resources such as materials, skilled employees, information and support from its environment and how it delivers results(Jones, 2010)

Since product development approaches are closely associated with respective production methods that use them they will also be reviewed. For example as Toyota Production System is different from Mass Production Systems the Toyota Product Development system is different from traditional product development.

In October 13 1994 Mosaic Communications Corporation released Mosaic Netscape 0.9 which was subsequently renamed Netscape Navigator. It was advertised as “the web is for everyone” and the stated goal of the product was to “level the playing field among operating systems” by providing a consistent web browsing experience across them. The browser interface was identical on any computer regardless of the operating systems the computer run. This was a direct threat to Microsoft which viewed the commoditization of operating systems as a direct threat to Windows operating system which was its main business. Microsoft released several successive versions of the Internet Explorer its own web browser to catch up with Netscape. In late 1995 analysts thought Microsoft would fail by the disruptive innovation threatening its core business of operating system. Although the first two versions of IE lacked the quality to match Netscape, IE version 3.0 was considered equivalent of or better in some aspects than Netscape Communicator. This achievement relied on the Explorer team’s development process to a great extent (Maccormack, 2001).

Development of the IE3 was a life or death matter for Microsoft. The CEO Bill Gates refused to meet any developer unless it had anything to do with internet and Internet Explorer.

The IE3 teams processes which is common now in Internet-software development was different from the past approaches used in Microsoft. Uncertain environments such as Internet-software development requires interactivity which enables the customer evaluate the design before specifications are stabilized since user requirements and demands cannot be known beforehand. In IE3 development critical parts of the functionality were delivered to customers early in the development and core design is changed to according to customers' feedback (MacCormack, 2001). Even at the early phases the customers tested actual working versions of the product.

The overall architecture was developed in such a way that separate component teams fed their components into the product in different times and order as they were working in parallel. Instead of one big effort to integrate all the nuts and bolts, new component modules are integrated when they included only 30% of the final functionality. This was enough to get meaningful feedback on how the product worked, also provided a base-line product which was handed to development partners. From this point on "daily builds" which integrated new code to the complete product begin. Automated tests which provided rapid feedback enabled the team to add functionality, test the impact of each feature and make adjustments. When the product was 50% ready a beta version is distributed to the customers and 70% to 90% ready when a second beta was used to gather customer feedback on bugs and new feature offers. These feedbacks resulted in major design changes and introduction of features that didn't exist in the initial design specifications.

New feature addition and daily integration continued until the final weeks of the project. The product was not frozen until a week before the release of the product.

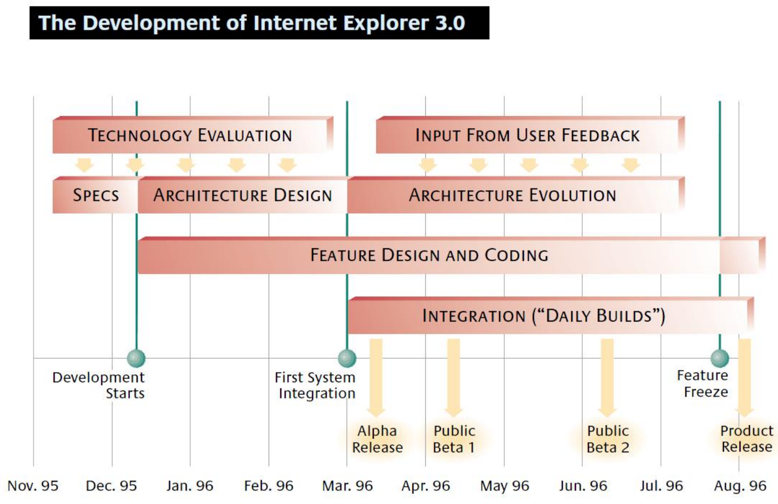

Figure 3.6 Project phases for IE 3.0 Development (MacCormack, 2001)

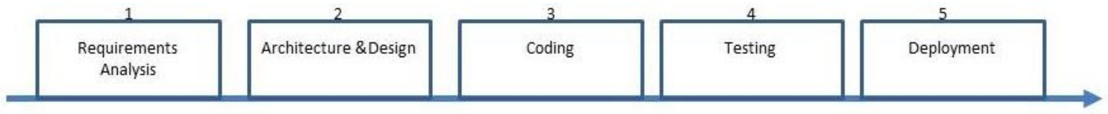

Figure 3.7 Traditional project management phases.

The diagrams above show the contrast between the traditional development model which is highly structured, sequential in which the design decisions made during the development is tracks through documentation and the more "Agile" model which embraces changes to even the core architecture late into development, accepts user feedback and integrates all the components without sacrificing the functionality.

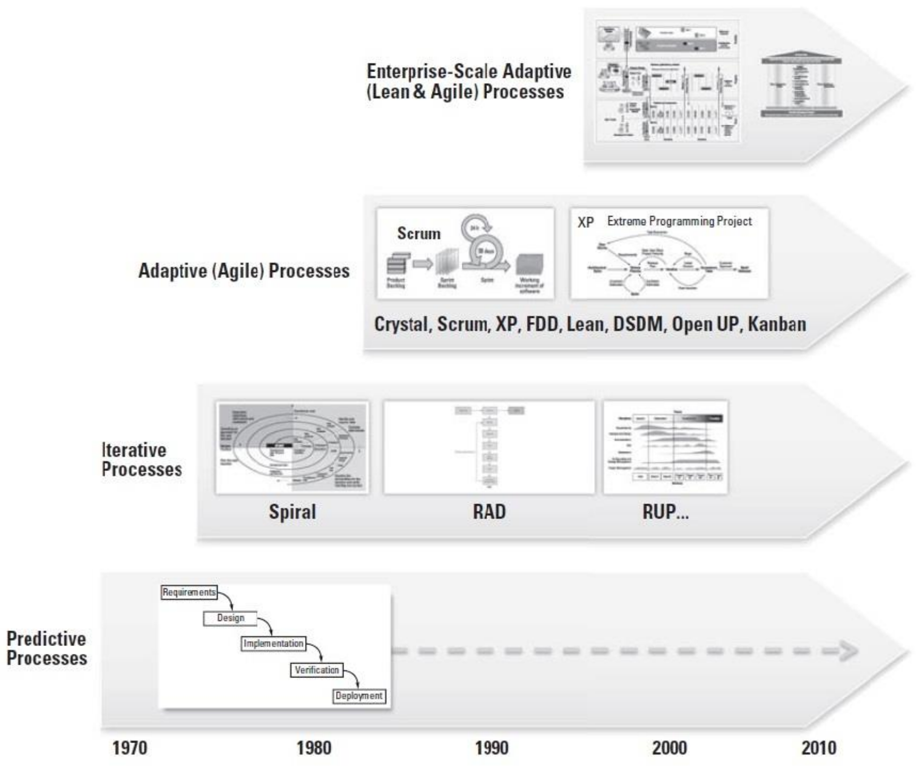
Enterprise-Scale Adaptive (Lean & Agile) Processes
Adaptive (Agile) Processes
Scrum
XP Extreme Programming Project
Crystal, Scrum, XP, FDD, Lean, DSDM, Open UP, Kanban
Iterative Processes
Spiral
RAD
RUP...
Predictive Processes
Requirements
Design
Implementation
Verification
Deployment
1970
1980
1990
2000
2010

Figure 3.8 History of software development methodologies (Adopted from Tail Ridge Consulting LLC).

3.4 WATERFALL OR TRADITIONAL SOFTWARE DEVELOPMENT

As the software industry advanced after its inception in the 1950s and 1960s the need to be able to better predict and control large-scale software projects caused the conventional project management methodology to be adapted to software development (Leffingwell, 2011). This sequential, stage-gated "waterfall" software process model is shown in the Figure 4.4.

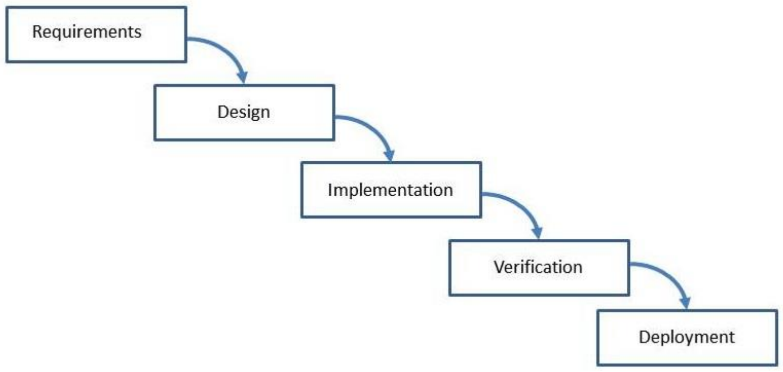

Figure 3.9 Main steps of waterfall methodology for software development.

Winston Royce who is often (wrongly) credited with the creation of the waterfall model of software development actually described it as a model that would not work for large-scale software development (Royce, 1987). Royce points that the implementation of the project phases as they are in the described is “risky and invites failure”. The problems he points out is that in the testing phase which is the first event for which timing, storage, input/output, transfers, etc. are “experienced as distinguished from analyzed. When the system fails to satisfy the external constraints in which it is tested or deployed then a major redesign is required. The design changes in this case will so disruptive that the software requirements will be violated. Since these requirements provide the rational for the whole system either the requirements must be modified or o substantial change in the design. In both cases the project as well begins from zero.

What Royce actually proposed is an enhanced model in which a prototype is built first right after the preliminary program design and this prototype along with the feedback between phases improved and built into the final product. Royce recommended doing the analysis, design and development phases twice.

“If the computer program in question is being developed for the first time , arrange matters so that the version finally delivered to the customer for operational deployment is actually the second version insofar as critical design/operations areas are concerned”(Royce, 1987). Royce suggests that for a development project of 30- months might have a 10-month pilot model and

Although Royce predicted that the pure waterfall model illustrated above isn't suitable for large scale software development; it become the de-facto model in the software development community (Leffingwell, 2011). This model is credited with the lack of success in the majority of the software projects. Standish Group's Chaos report survey (Standish, 2009) notes the following.

- 31% of projects cancelled before completion.
- 53% of the projects cost more than 189% of their estimates.
- Only 16% of projects were completed on time and on budget.
- For the largest companies completed projects delivered only 42% of the planned features and functions.

3.4.5 Scope Management and Business Requirements In Waterfall Model

The basic assumption of waterfall method is that the requirements of the customer can be revealed through detailed analysis and determined "up front", the activities to meet these requirements can be decided upon and planned for, thus the cost and duration of the project is knowable from the beginning. (Leffingwell, 2011) .
Scope of the project consists of the user or client requirements that the product will meet. But historically it is observed that misunderstood, changing or unmet requirements are the most important reasons of the project failure (Chaos, 2009).

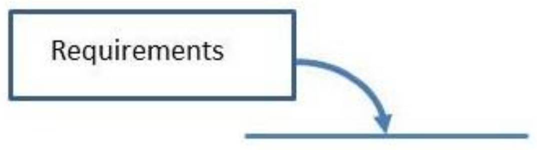

Figure 3.10 Phase handover in waterfall software development.

Assuming that the requirements are complete, stable and will not change until the delivery of the product is the major cause for project failure in waterfall (Taylor, 2000).
However clear, through and correct specifications before development is impractical and almost impossible because (Parnas & Clements, 1986):

- A system's user seldom knows exactly what they want and cannot articulate all they know.
- Even if we could state all requirements there are many details that we can only discover once we are well into implementation.
- Even if we know all these details as humans we can master only so much complexity.
- Even if we could master all this complexity external forces lead to changes in requirements some of which may invalidate earlier decisions.

David Parnas and Paul Clements state that for all the above reasons "the picture of the software designer deriving his design in a rational, error-free way from a statement of requirements is quite unrealistic" (Parnas & Clements, 1986). In 1998 Standish Group issued a report named "CHAOS: Charting the seas of information Technology" which analyzed 23.000 projects to find the causes of software project failures. The three most common factors that caused projects to fail which are associated with waterfall are (Standish Group, 2009):

- Lack of user input: 13% of all projects
- Incomplete requirements and specifications: 12% of all projects
- Changing requirements and specifications: 12% of all projects

In waterfall once the requirements are analyzed and fixed (project scope management) the actions to complete the project are planned (PMBOK, 1996). After fixing the scope with a work break down the actions necessary to complete the project are defined, sequenced, their durations estimated and the project schedule is developed (project time management and schedule development). Then budget for the actions and resources are allocated (project cost management).

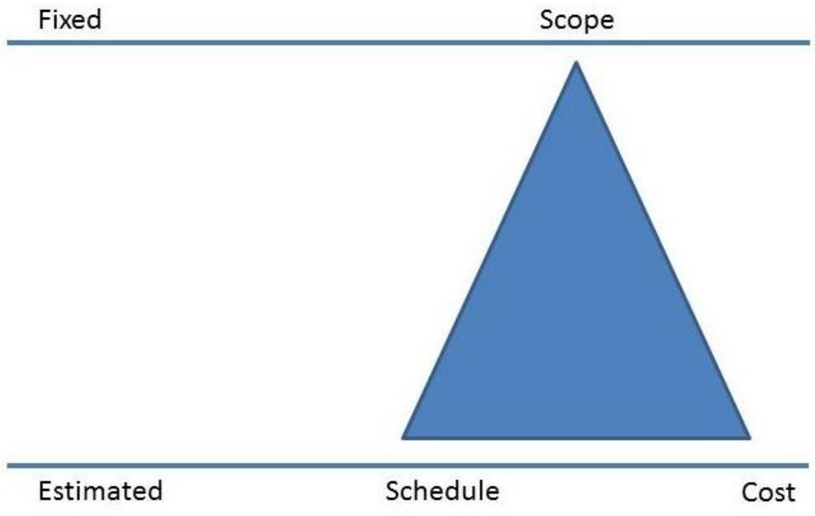

Figure 3.11 Project management triangle in traditional project management

This scope-schedule-cost triangle is called the project management triangle or the iron triangle in project management nomenclature.

The triangle symbolizes the interdependence of the cost-schedule-requirements (scope) of the project. The project management must balance the time, cost and scope constraints of the project. The project triangle illustrates the process of balancing constraints since the three sides of the triangle are connected changing one side of the triangle affects at least one other side.

For example if the project schedule is shortened the project owner must either increase the budget(cost) since more resources must be invested to get the work done or the scope is reduced since with the resources available the planned work cannot be completed in less time (Cahtfield, 2010).

In many projects the business requirements change long before a system meeting the requirements is delivered (Chaos, 2009). This is the case where there are known and well defined requirements.

Also in many cases the products of higher complexity needs to be developed. The development of “cutting edge technology” where products often approaching the limits of current technical systems or the products developed to meet emerging customer needs powerful means of shortening development time (Jensen, Piecko, Henning, Short, and Shiller, 2004).

3.4.6 Quality in Waterfall

Traditional project management addresses the quality issue by using a separate quality process than the core processes. Quality here is defined as “the totality of characteristics of an entity that bear on its ability to satisfy stated or implied needs”(PMBOK, 1996).

One critical aspect of waterfall which is separating accountability and feedback from the work shows itself here. Because “the temporary nature of the project means that investments in product quality improvement, especially defect prevention and appraisal must often be borne by the performing organization since the project may not last long enough to reap the rewards”(PMBOK, 1996)

Quality assurance in waterfall is all the planned and systematic actives implemented within the quality system to provide confidence that the project will satisfy the relevant quality standards.

Being unable to produce quality software with the traditional methods is one of the main reasons why agile is proposed. In year 2000 US Department of Defense replaced the software acquisition standard Mil-Std-498 with a new set of instructions DOD 5000.2 in which more agile acquisition is recommended.

In DOD Instruction 5000.2 the preference of a more agile approach is encouraged. “There are two approaches, evolutionary and single step [waterfall] to full capability. An evolutionary approach is preferred. In this approach the ultimate capability delivered to the user is divided into two or more blocks with increasing increments of capability. Software development shall follow an iterative spiral development process in which continually expanding software versions are based on learning from earlier development”.

The main reasons for the adaptation of waterfall include (Larman & Basili, 2003);

- Being simple to explain and recall. Agile methods are more complex to understand and describe. Winston Royce described the original waterfall with prototyping and two iterations but it devolved into sequential steps as adopted.

- It gives an illusion of an “orderly, accountable and measurable process with simple document driven milestones (e.g. Analysis completed.) “. Although the waterfall model attempted to satisfy management accountability goals it failed to do so (Curtis, 1987).
- It was labeled ideal and appropriate for software development although the scientific evidence and real life experience favoring Iterative and Incremental Development.

Below the situations where waterfall is appropriate and where not are listed(CMS, 2008).

When using waterfall is appropriate:

- When developing a transaction-oriented batch systems
- When the project has clear objectives and a well-defined solution.
- No immediate need for implementation
- Project requirements have been stated unambiguously and in detail.
- Users of the software have full knowledge in the business domain and the application.
- Project manager or project teams members lack experience
- Project cadre is subject to change
- Formal approvals in designated milestones are required
- Projects with low risk of not meeting user requirements but has a high risk of missing budget or schedule targets follow Waterfall.

When using waterfall is not appropriate:

- When requirements are prone to change or not well understood
- Web projects. Web based business projects with continual evaluation of what works and what does not work. In this kind of projects users’ requirements are explored with empirical process so require experienced, flexible cross-functional teams.
- Real-time systems
- Event-driven systems
- Projects implementing cutting-edge technology

3.5 PROTOTYPING

A prototype is a representative model or simulation of the final system (Warfel, 2009). Prototyping is not a standalone, complete methodology for software development. It is an approach to manage the parts of a project structured in a methodology such as waterfall, Spiral or RAD. Prototyping is a natural and necessary part of any design process for products like car, ship or airplane computer system. In most software development projects making a prototype is seen as an extra cost not a regular practice. When the project involves; high risk developments, web sites, systems or applications with both hardware and software components together prototyping is critical for product development success prototyping becomes critical. Cost-to-benefit ratio of prototyping increases as system complexity increases (Warfel, 2009).

When the system has many interactions and transitions such as AJAX and RIAs which leverage state-based interactions writing a requirement becomes difficult. When requirements are identified and defined in documents it is open to different interpretation by different readers. When the requirements become detailed and complex then prototyping is used for clarification and reduction in the number of rework(Warfel, 2009). In prototyping risk is reduced by breaking a project into smaller segments and change in the system made easier. Prototypes may be discarded by they may as well evolve to a working system.

The most critical aspect of prototyping in software projects is it “addresses the inability of many users to specify their needs and the difficulty of systems analysts to understand the user’s environment by providing the user with a tentative system for experimental purposes at the earliest possible time”(Janson & Smith, 1985).

User is involved throughout the prototyping which increases the likelihood of user acceptance of the final implementation. What Winston Royce proposed in his famous paper was a preliminary program design of the complete system with user involvement before any analyze done on the system thus reducing risk.

Risk is the possibility of loss, injury or other adverse or unwelcome circumstance; a change or situation involving such a possibility (Oxford English Dictionary, 2010). Risk is “any uncertainty that if it occurs would affect one more project objectives"(Hillson, 2004). It is the quantified uncertainty.

Risk in projects has:

1. *Execution Risk:* Risk that designated activities won't be carried out properly (in time and order) Project plans, time-lines (schedule), and budget is used to reduce the execution risk in projects.
2. White-space Risk: Some
3. *Integration Risk:* Even if all the right activities have been anticipated the result of these activities may be difficult or impossible to be integrated into a one complete whole once they are completed.

3.6 ITERATIVE AND INCREMENTAL DEVELOPMENT

Iterative development or iterative enhancement is a practical means of top-down, stepwise refinement approach to software development (Basili & Turner, 1975). Iterative development in which a system is built using a well-modularized, top-down approach requires that the problem and its solution be well understood (Basili & Turner, 1975). Even if the implementers have previously undertaken a similar project it is sill "difficult to achieve a good design on the first try" (Basili & Turner, 1975).

Iterative and incremental development was a natural part of software development in the industry shaping cornerstone projects in government projects such as nuclear submarine programs and large army projects (Larman & Basili, 2003). IBM which developed large government projects such as the Project Mercury beginning from the 1958, command and control system for the U.S Navy's first Trident nuclear submarine, US Department of Defense space and avionics systems were the programs developed with iterative development. IBM and its main competitor TRW which developed ballistic missile defense used iterative development with iterations as long as one year. Winston Royce who is credited with waterfall approach, worked at TRW. TRW's chief scientist Barry Boehm is the originator of iterative and incremental spiral model for software development.

NASA's space shuttle software which was developed by IBM from 1977 to 1980 was developed with iterative and incremental development. The development team built the software in 17 iterations over 31 months with an average of eight weeks per iteration (Madden & Rone, 1984). Early in the program it was recognized as a necessity due to the size, complexity and evolutionary (changing requirements) nature of the program. The shuttle project exhibited what is observed today in many agile development efforts such as timeboxed iterations, feedback-driven specification refinements and so on (Larman & Basili, 2003).

Tom Gilb in his IID practice introduced the terms "evolution" and "evolutionary" to the software development. In what he called "evolutionary project management" Gilb discussed that "a complex system will be most successful if it is implemented in small steps and if each step has a clear measure of successful achievement as well as a retreat possibility to a previous successful step upon failure. You have the opportunity of reviving some feedback from the real world before throwing in all resources intended for a system and you can correct possible design errors"(Gilb, 1976).

The first step in iterative development is a simple initial implementation of a skeletal sub problem of the project. This skeletal form of the overall project forms into the final implementation which meets the complete set of project specifications. From the skeletal form a project control list is created that contains all the tasks that need to be performed in order to achieve the desired final implementation. This project control list acts as a measure of the "distance" between the current state of the project and the final implementation (Basili & Turner, 1975). For each iteration step the task which will be done in that step is selected, its designed, coded and debugged (implemented), the overall product is analyzed for fitness to use (analysis). These steps are repeated until the project control list is empty, until product satisfies customer, budget is depleted or time for the project is spent.

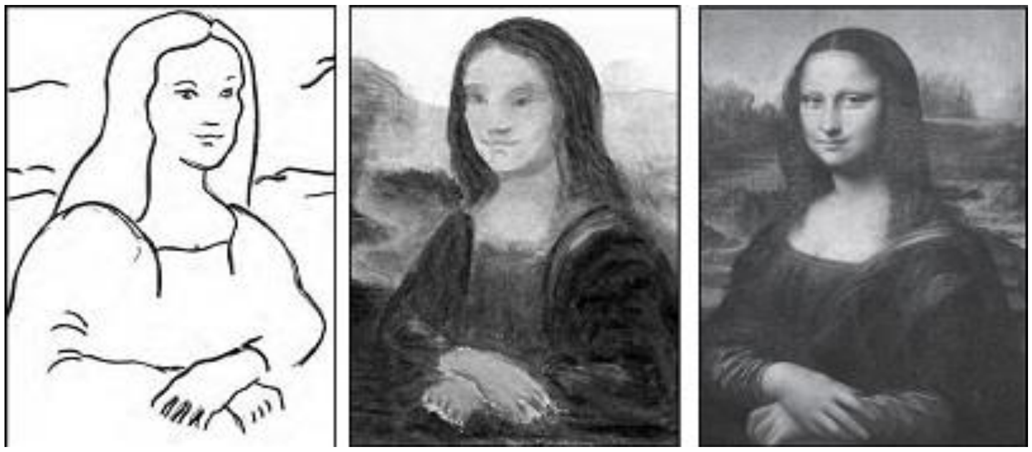

Figure 3.12 Iterative work (Rework-Adapted from Cockburn, 2008).

Developing software in an iterative, evolutionary and incremental way has been in practice for decades (Larman & Basili, 2003).

Incremental development is a "staging and scheduling strategy in which various parts of the system are developed at different times or rates and integrated as they are completed"(Cockburn, 2008). The alternative of incremental development is to develop the entire system with the integration being done at the end.

Iterative development is a "*rework scheduling strategy in which time is set aside to revise and improve parts of the system*" (Cockburn, 2008). The alternative strategy is to plan to get everything right in the first time.

Iterative and incremental strategies do not require, include, exclude or imply each other. Either one can be used with or without the other.

Time is explicitly set aside for the improvement of the software or product developed in iterative development. Iteration means "*rework*". In many projects the most problematic parts is the implementation of user requirements (Standish Group, 2009) and user interfaces (Cockburn, 2008) so these are the parts which require the most revising in the work. Also technology used in the project, architecture and the algorithms in the logic part of the software; are areas that almost always require rework. The iterative development has two different strategies.

1. Developing the system in the best possible way that can be planned for; so the changes will be minimal and they can be incorporated with relative ease.
2. Developing the least possible amount of the system, so wasting less time in prior work and then getting evaluations so less rework for the parts already useful.

The selection of the strategy depends on the context and circumstances. This will be examined in detail.

Iterative and incremental development is in use from as early as 1960s. The roots of the IID can be found on the work of Walter Shewhart who was a quality expert at Bell Labs. Shewhart proposed “plan-do-study-act” cycles (PDSA) to improve the quality(Deming, 2000). W. Edward Deming –the quality guru promoted the use of PDSA and described it in his boot Out of the Crisis in 1982 calling it a “Shewhart cycle”. In Japan where Deming introduced them has been called Deming cycle. In 1973 Thomas Gilb explored their use in software development (Gilb, 1976).Iterative enhancement was examined as a practical technique for software development for creating an easily modifiable product and to facilitate reliability (Basili & Turner, 1975).

In 1950s IID was used in the development of X-15 hypersonic jet and use of IID is considered to be one of the major factors of success of the project. After the success of the X-15 project NASA began to apply it the IID practice in other major projects. In project Mercury –the project in which NASA sent man to space for the first time- IID is used for software development. In Project Mercury the iterations were as short as half a day and the development team applied test-driven development in which tests for the working software is planned and written before the increment in software is coded.

Increasing demand for shortening time-to-market pressures, together with the failures inherent in waterfall model and advances in technology opened the way to more innovative models. A short example of these discovery-based models will be given here.

The main shift of focus in the move from waterfall to iterative methods is from big, up-front design (BUFD) to discovery-based approach. The software requirement specifications, design specifications and similar planning tools which define and govern the implemented product are no more used. Vision documents, use-case models define what is to be built. The iteration is applied over a quickly built base model to discover the “real user requirements” thus reducing the risk of the project (Leffingwell, 2011). Then more traditional process of implementation, testing and deployment is used to build on agreed requirements.

Where IID approach is most appropriate:

- Large projects where requirements are not well known or changing.
- When project budget, resource availability, technology is changing rapidly.
- When cutting-edge technology is used

3.7 THE SPIRAL MODEL

Pioneered by Barry Boehm (Boehm, 1988) the spiral model still has an emphasis on requirements from the beginning. It combines linear and iterative models. In this model an early spiral or iteration validates the requirements before a larger spiral with the traditional design, coding, integration and testing steps is initiated. The focus is on risk assessment and on minimizing project risk by breaking the project into smaller segments and smoothing the change during the development phases. Throughout the project life cycle risks are evaluated and considerations of project continuation are weighed.

In spiral development “each cycle involve a progression through the same sequence off steps for each portion of the product and for each of its levels of elaboration from an overall concept-of-operation document down to the coding of each individual program.(Boehm, 1988)”. The sequences of steps (quadrants) in each trip as the spiral is traversed are:

1. Determining the objectives, alternatives and constraints of the iteration
2. Evaluating alternatives, identifying and resolving risks
3. Developing and verifying deliverables from the iteration
4. Planning for the next iteration.

Spiral model help reducing project risk by selecting the appropriate approach for each iteration separately. For example when technological risk is high the iteration IID, when requirements are fixed but strict schedule must be followed Waterfall may be used.

This model is credited being the starting point for discovery-based iterative and incremental methodologies.

Below the situation where spiral approach is appropriate and where not are listed(CMS, 2008).

When spiral is appropriate

- When risk must be avoided
- Real-time or safety-critical systems
- Resource allocation is not a problem.
- Project manager has high level skills and experience.
- Accuracy in meeting the requirements is essential.

When spiral is least appropriate

- Project risk is low
- High level of accuracy is not essential
- Resource allocation is an issue

3.8 RAPID APPLICATION DEVELOPMENT

Rapid Application Development (RAD) is an iterative approach with the key objective of fast development and delivery of high quality system at a relatively low cost of investment. It centers on prototyping and user involvement where the analysis, design, build and test phases of the development life cycle are compressed into a sequence of short, iterative cycles of development(Berger et al., 2009). It aims to produce high quality systems quickly through the use of iterative prototyping at any stage of development with active user involvement through the use of automated development tools. The computerized tools generally include object oriented programming techniques, Graphical User Interface builders, Computer Aided Software Engineering (CASE) tools, Database Management Systems, code generators. The focus in RAD is fulfilling business needs of the customer with technology and engineering perfection being less important (CMS, 2008).

Rad is characterized by small development teams of typically for to eight persons. Such teams contain both developers and users which have the authority to make design decisions (Beynon-Davies et al., 1999). RAD projects are also relatively small scale and of short duration since if the project takes more than six months to complete it is likely to be overtaken by developments in the business.

Using RAD systems that meet business requirements can be developed with low costs. Business owners and users actively participate during development and the resulting systems can be quickly implemented into the business. The focus in this approach is on the business essentials so esthetic aspects of the software are usually not considered.

The systems developed with RAD can be changed rapidly and provide a fit between user requirements and business policies.

This approach may result in dramatic savings in terms of money, time, human capital and effort.

The down side of using RAD will be the lower overall system quality. Since system architecture is constrained by RAD tools, adding more and more features over time degrades system performance and the system may become unstable. Procedures considered in administration of projects such as detailed documentation, formal reviews and milestones may not be applicable to RAD projects.

For example: A simple salesman reporting application for a sales group developed with IBM WebSphere Lombardi or Microsoft FoxPro.

When RAD is appropriate

- Small-to-medium sized projects with short duration.
- Well defined and narrow business objectives and clearly defined user groups
- Computationally simple applications.
- When the User system reflects the system functionality.
- Technical architecture is clear and components are in place.
- Project consists of mainly analysis or reporting of data.
- Management support and effective project management skills in both socially and business terms exits.

When RAD is least appropriate

- Safety-critical systems
- Real-time systems
- Large projects with distributed information systems
- Computationally complex systems where great amounts of data is analyzed, manipulated, created, transferred.
- Vague business objectives with unclear project scope
- Decision making is not available in a timely basis.
- Large project teams which needs coordination.
- Users are not committed to or timely available for the project.
- Project involves new technologies or the technical requirements are fuzzy.

CHAPTER 4

SELECTING AND TAILORING THE PROCESSES FOR SOFTWARE DEVELOPMENT

The projects approaches and methods mentioned in the previous chapters have different advantages and disadvantages making them suitable or unsuitable to for a project depending on the circumstances of each project. Because of the uniqueness of each projecta projects first stage should be selecting and designing the development process itself (Cusumano et al., 2009). Choosing the right process is important because managers have a wide range of choices such as waterfall, spiral, RAD, Extreme Programming, Evo, Scrum, Crystal, and Scrumban, TOC, Kanban among others. Although each one of these is a complex set of coherent practice set each one has different best case to be used.

For example Hewlett-Packard uses a small number of process “templates” and each of these has defining criteria for when to use each one.

Software development projects have many aspects from the managerial point of view that needs to be taken into consideration when structuring the development project.

Project structure has an impact on the nature of resulting design. Open source products developed by distributed teams are more modular than one which is closed-source and developed by a collocated team (MacCormack & Verganti, 2003). This will have an impact when a collocated team is moved to a distributed team later or vise-versa. Trying to keep the product bug-free by rigidly structuring the development environment and structure, results in losing innovativeness of the product.

When making decisions on the practice strategy for the software project(Cusumano et al., 2009):

1. Projects context should dictate the development strategy.
2. Process capabilities can overcome the disadvantages of work environment
3. Since project structure choice will affect resulting product design alternatives must be evaluated for project effectiveness and performance of the product.

4. Creativity and innovation versus productivity and quality trade-off must be evaluated carefully.

McCormack et al. created a framework to select the appropriate product development method to fit the environment which helps fitting the development method to environmental, customer, technological and organizational situation (Maccormack et al., 2012).

They suggest a four step process to select and define an appropriate development-process style.

1. Defining different development styles to create organizational portfolio of these styles.
2. Defining the criteria for style selection such as the amount of technical and market risks
3. Attacking inertia when shifting modes since managers and employees both tend to keep the habits and they have personal preferred for specific styles.
4. Managing the style portfolios not only in terms of method content but also incentives and culture of the responsible bodies.

The product-planning processes must include an explicit step for selecting or designing appropriate product-development style that fits the corresponding environment or context. The focus here is aligning assumptions or business objectives with the maturity of the market. Any disagreements in this point will cause friction, resistance, confusion and mistakes when changing the development strategy.

After objectives are defined the factors such as overall development process, team structure and organization, processes for developing understanding of customers, processes for technology development, intellectual property strategy, platform and product architecture, measurement, evaluation and reward systems must be designed. None of these are enough to deliver optimal results so they must complement and support each other.

McCormack, Crandall, Henderson and Toft uses the framework developed by and for HP Company to match the product development processes developed to fit the process to the environment (Maccormack et al., 2012).

They argue that applying the same “best-practice” to all situations while ignoring the major differences between the projects, results in failure and missed opportunities.

Organizations invest heavily to define and standardize the way they develop products and services. Studies show that despite these efforts new product success rates little or no improvement (Griffin, 1997).

In business context one critical factor that defines the nature of the product development is the market needs which in software development coined as “requirements”. Early in the life of an industry when there is a great deal of uncertainty there is a variety of offerings (Tushman & Anderson, 1986) since it is not clear what attributes the customer values in the product and how much he is willing to pay for those (Maccormack et al., 2012).

The example given is HP company in matching the unique organizational processes and structures to manage in different contexts (Maccormack et al., 2012). In order to match process with business context HP first looks at the business demands which are generally divided into three segments as start-up, growth and maturity. In these segments the customer requirements, market size, technology maturity, available solutions and knowledge are determined.

For example in the emergent sector of the spectrum HP uses a series of “iterative prototypes to create unique, first-of-a-kind services, which were farther evolved as information gathered in previous iteration allowed the firm to understand the opportunity more fully”(Maccormack et al., 2012). The firms sought a market and a customer which allowed the researchers of the company to explore the landscape quickly through rapid, evolutionary prototypes and with great amount of interaction with the customer. A series of prototypes are developed in conjunction with lead-user customers which allowed rapid exploration of customer requirements, the necessary product features and emerging technologies. Simple rules are used to focus on the overall strategic vision together with lightweight processes (Sull & Eisenhardt, 2001). Lead-user customers and the production teams constant interacted constantly in a highly responsive mode (Hippel, 1988). Rework (iteration) was the standard mode of work and learning was the center of the process with prototypes being discardable. The result of this effort is the company resolves many uncertainties and gains a great amount of knowledge thus being in a better position to exploit the market opportunities. All this is done while the market is still emerging and with a relatively small amount of R&D budget.

Table 4.1 HP's three development styles (MacCormack, 2012)

	Emergent	Agile	Efficient
Development Process	Lightweight process, Rapid information exchange, Continuous customer interaction to identify customer value	Evolutionary iterative process, milestone releases, beta versions to actual customers, continually re-prioritization of features	Well-defined staged, gated process with clear phases, explicit tasks and deliverables, monitoring by plan
Product Specifications	High-level sketches, Specifications are the output of emergent projects, input to agile and efficient projects	Established in general up front, updated on feedback on performance and customer needs	Established in detail up front, well-understood customer requirements and technical solutions
Customer Understanding	Zero distance between developers and lead users, High-bandwidth, low-latency channels communication, frequent prototypes	Mechanisms to work with real customers that represent market, interaction through early beta versions for feedback on features and performance	Traditional market research. Developers and customers may have distance between
Technology, Invention	Get functional prototypes to customers early and often, help establish a dominant design	Adapt technical choices and features through early customer releases, responding to feedback by incrementally evolving current dominant design	Dominant design "+1" features, Cost reduction, adding new functionality
Team Structure	Small, physically collocated, cross functional team of smart, highly skilled people who are comfortable with high levels of ambiguity	Program manager with direct responsibility for functional staff integrating information and evolve the design in rapid and controlled way	Functional structure with cost reduction and using expertise of different departments
Platform and Product Portfolio	Create one product and iteratively improve it. Don't commit to one platform, rework platform design after opportunity is clarified.	Define platform, enhance portfolio with derivative products to explore possible adjacent customer sub-segments, minimize white space	Create few, long-lived platforms that support differentiated products with common components.

Measurement, Evaluation and Reward	validate opportunities, check assumptions, define specifications	increase in revenues and profits, addition of new functionally per release, customer satisfaction, responsiveness to changing demands, winning competitive product reviews	cost/unit, reduction in costs, contribution to organization profits, customer loyalty, support costs

The questions in the following figure are used by the managers to develop or tailor the appropriate development strategy starting by understanding demands of different business contexts.

Table 4.2 Selecting and defining an appropriate development-process style (MacCormack, 2012)

Steps	Key Questions
Step 1: Define the business context	Who are the customers? How well known are their problems? How feasible and proven are technical solutions? How big and fast is the market growing? How well are we positioned versus competitors? Given this, what factors should the strategy optimize? Deliverable: criteria to be optimized by chosen style
Step 2: Select appropriate development style	How well is the strategy delivering this today? Is a change in style required? Can we apply an existing style of development or should we define a new style? Deliverable: Selected predefined style or characteristics for new style
Step 3: Define and implement style	How to create a set of practices that will produce the desired result using development process, product specs, customer understanding, technology, IP strategy, organization, team leadership, architecture, measurements. Deliverables: Style implementation road map
Step 4: Monitor and review over time	How can the style performance review embedded in the life cycle for each project? What are the criteria showing that performance not adequate and change required? Deliverable: Measures of style performance

Monitoring and changing the development process style important because in each of the context shown above the objective for the activities are different. In emergent markets managers must decide if there is a market opportunity that can be profitably served. If such a market exists than the nature of the products and services should be unearthed.

In evolving business context it is known that a market exists but rapidly changing customers' needs must be understood and business must be scaled to meet the demand.
In stable businesses the organization must defend its market share and position fighting lower-cost competitors on well-known customer demand. MacCormack et al concludes "that one size does not fit all".
Cockburn created a framework to select the appropriate software development method to fit the environment and requirements using 3 dimensions (Cockburn, 2001):

1. Project size in terms of people count and communication
2. Criticality of the project
3. Project priorities

4.1 TEAM GROUP SIZE AND COMMUNICATION

The main purpose for using a methodology is to coordinate the people so when the project is larger the methodology gets "heavier".
Projects have roles and people with different and particular skills and characters attend to these roles and work within different types of teams. Thus any methodology must have implicit or explicit attention to people and people work characteristics to begin with.
The methodology growth is directly proportional with the number of roles in the project rather than the number of people (Harrison & Coplien, 1996). Roles in an organization are basic units of abstraction for organizational studies (Cain & Coplien, 1993). Typical roles in software development are designer, system tester, project manager, business analyst, and coder. While several people can play the same role, one person may play different roles at different times or phases.
For an organization with n roles the number of possible links between roles is n(n-1)/2(Jones, 2012). As the number of roles increase the number of communication paths increase as the square of the number of roles but the number of actual links increase linearly (Cain & Coplien, 1993). Thus the communication saturation which is the ratio of actual collaborations to the possible collaborations decreases. For a small team the communication saturation is complete which means everybody talks to everybody else.

When the team size increase the communication saturation rate drops quickly down to ten to thirty percent.(Cain & Coplien, 1993). This means that the roles are not communicating with one another which increase risk that critical information does not reach to the right person or reach in a timely manner.

The communication modes in software development are based on Media Richness Theory which is developed by Richard L. Daft and Robert H. Lengel and is used to rank and evaluate the richness of a certain communication medium. Information richness is "the ability of information to change understanding within a time interval" (Daft & Lengel, 1986).

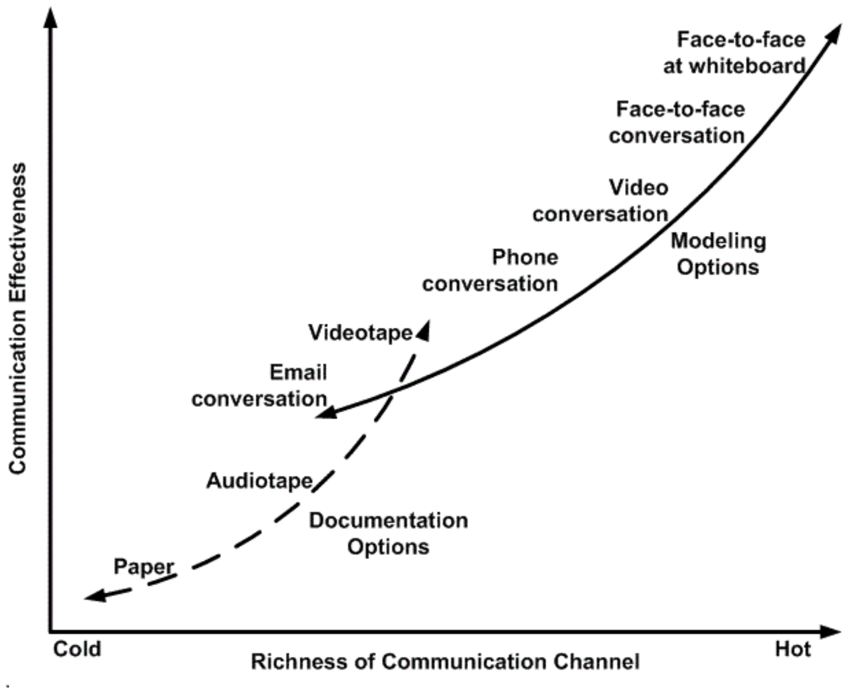

Figure 4.1 Efficiency of communication (Cockburn, 2002).

When people talk face-to-face at a whiteboard, then interact in real-time with question and answer together with body gestures, drawings with the whiteboard creating an environment for materializing the abstract(Cockburn, 2003).

4.2 SYSTEM CRITICALITY AND RISK LEVEL

System criticality defines the amount of damage in terms of loss of comfort, loss of money, loss of life if the system doesn't function in a proper way (Cockburn, 2000).

Risk in software development is defined as "a problem that could cause some loss or threaten the success of the project but which hasn't happened yet. These potential problems might have an adverse impact on the cost, schedule or technical success of the project, the quality of the software products or project team role" (Wiegers, 1998). Risk is "a combination of an abnormal event or failure and the consequences of that event or failure to a systems operators, users or environment" (Glutch, 1994).

Failure in aircraft software is more serious than a failure in software for keeping sports-match results. When developing software for aircraft the team will likely to use pre-determined forms to fill in the necessary fields for creating use cases, using a standardized software tool than relying on the hastily written specs or verbally relayed use-cases.

4.3 PROJECT PRIORITIES

The project stakeholders and sponsors might want to have the software as soon as possible, to be defect free or want very high visibility on the processes. Each priority requires a different approach. While some methods or processes ensure the support of software in later stages of its life-cycle some processes focus on reduction of bugs and fix requirements, some other focus on fast product delivery with responsive processes.

Cockburn draws on the seven principles for designing a methodology called the Crystal Methods. These are:

1. "Interactive, face-to-face communication is the cheapest and fastest channel for exchanging information."
2. Excess methodology weight is costly
3. Larger teams need heavier methodologies
4. Greater ceremony(formality and completeness of the artifacts of production)

5. Increasing feedback and communication reduces the need for intermediate deliverables.
6. Discipline, skills and understanding counter process, formality and documentation.
7. Efficiency is expendable in non-bottleneck activities.

The framework is considered objective since the people on the project are countable and criticality and priorities are easy to assess. It is used for:

- What sorts of methodologies fit what sorts of projects
- Indices for selecting the appropriate methodology category for a project

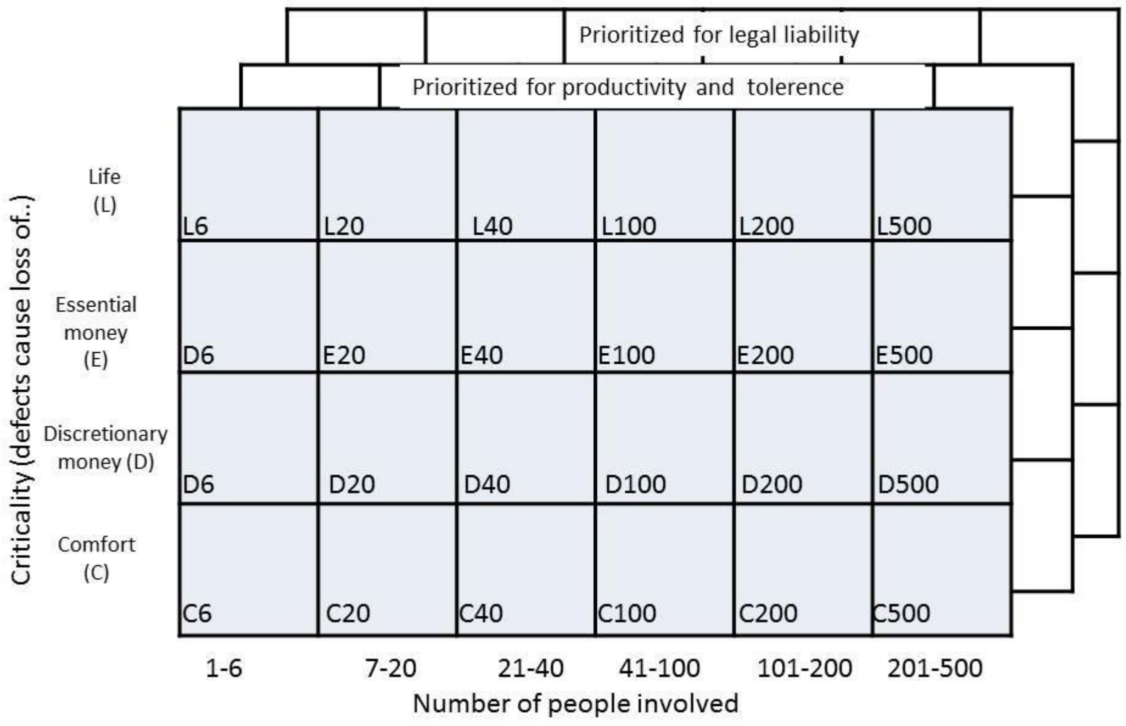

Figure 4.2 Alistair Cockburn's Scale for Project Classification (Cockburn, 2000).

Cockburn suggests the main dimensions for selecting a methodology are the staff size and system criticality (Cockburn, 2000). By methodology the complete elements of "people, roles, skills, teams, tools, techniques, processes, activities, milestones, work products, standards, quality measures and team values" is intended. The individual approaches to design of software or the standards rarely define the success of the project.

For example a five-person project to produce a prototype of a mission-critical business system using object-oriented component architecture was run as a C5 category project with 5 people working with "good people, small team, in one room, receiving adequate training and without any distractions "(Cockburn, 2000).

What must managers be careful about is that projects may change dimensions over their life cycle. An example Cockburn gives about a project that began as a D4 project with the aim of "get it done and go home" becomes an E5 project when it was understood that:

- The estimations for timeframe was erroneous
- The complexity was greater because of need of interfacing to other software

And the project becomes E15 when design problems became apparent and people have to live the project and others joined.

Cockburn emphasis that (A.R.Cockburn, 1999)

- Almost any methodology can be made to work on some projects
- Any methodology can manage to fail on some project
- Heavy processes can be successful
- Light processes are more often successful and the people on such projects credit the success to the lightness of the methodology.
- Managers must find ways to "tailor a methodology to the idiosyncrasies of any particular project to benefit from the tailoring"

Little et al. developed an approach to categorize projects in terms of complexity and uncertainty (Little et al., 2005). The complexity drivers of the project are:

- Team size
- Mission criticality
- Team location
- Team maturity
- Domain knowledge gaps
- Dependencies

Table 4.3 Complexity attributes (left to right, simple to highly complex) (Little et al.)

Attribute	1	3	5	7	10
Team Size	1	5	15	40	100
Mission Critical	Speculative	Small User base	Established market	Mission critical with large user base	Safety critical with significant exposure
Team Location	Same Room	Same Building	Within Driving Distance	Same Time Zone +/- 2	Multi-site, World Wide
Team Capacity	Established Team of experts	New team of experts	Mixed team of experts and novices	Team with limited experience and a few experts	New team of mostly novices
Domain knowledge gaps	Developers know the domain as well as expert users	Developers know the domain fairly well	Developers require some domain assistance	Developers have exposure to the domain	Developers have no idea about the domain
Dependencies	No dependencies	Limited and/or well insulated	Moderate	Significant dependencies	Tight integration with several projects

Uncertainty in decision making means situation where the current state of knowledge such as the nature of things is unknown, the consequences, magnitude of circumstances, conditions or events is unpredictable and credible probabilities to possible outcomes cannot be assigned. Information theory defines uncertainty as the degree to which available choices or the outcomes of possible alternatives are free from constraints. In statistics it means situation where neither the probability distribution of a variable nor its mode of occurrence is known. The two aspects of uncertainty are; variability and ambiguity (Hillson & Murray-Webster, 2005).

Uncertainty has three dimensions (Jones, 2010).

1. *Complexity:* Complexity is a function of the strength, number and interconnectedness of the forces the project is facing. The project environment can be from *simple* to highly *complex*.

The number and different power levels of stakeholders, communication channels, and power plays, interactions between the project team, project manager and stakeholders are factors that increase the complexity in any project.

2. *Dynamism:* Dynamism is the result of how much and how quickly the forces affecting the project are changing. The project environment can be from *stable* to highly

dynamic. When there is continuous high level management support, stable project team and well defined customer requirements projects are more stable.

3. *Richness:* Richness is a function of the amount of resources available to support the project. Project environment can be from *poor*to*rich* in resources. Skilled and experienced project team members and managers, time, budget, customer feedback, priority information about project deliverables, knowledge regarding risk, stakeholder support are some of the resources for the project.

For example user feedback, priority information about the project deliverables, access to skilled team members, management and user support, supporting environment so on are necessary resources for the project. Teams which have access to these resources have greater chance of success. Valid and timely user feedback is one the most critical resources affecting the software products success especially in a quickly changing environment such as internet. Lack of access to resources results in a product with no user base.

The level of uncertainty in a project affects the impact of development practices (MacCormack & Verganti, 2003).

The uncertainty of the project in HP framework is investigated by the marked conditions and project constraints. The indicators of project uncertainty are:

- Market uncertainty
- Technical uncertainty
- Project duration
- Scope flexibility and dependencies of other projects to that particular project.

Table 4.4 Uncertainty attributes (left to right, simple to highly complex)

Attribute	1	3	5	7	10
Market uncertainty	Known deliverable, possibly defined contractual obligation	Minor changes in market target expected	Initial guess of market target is likely to require steering	Significant market uncertainty	New market that is unknown and untested
Technical uncertainty	Enhancements to existing architecture	We think we know how to	We're not quite sure if	Some research is required	New technology,

		build it	we know how to build it		new architecture. Research is required
Project duration	1-4 week	6 months	12 months	18 months	24 months
Dependents/Scope Flexibility	Well defined contractual obligations or infrastructure	Scope is not very flexible	Scope has some flexibility	Scope is highly flexible	Independent

When projects are examined in terms of the given attributes for complexity and uncertainty they are cross plotted as in the figure. Since the projects in each quadrant are similar they require similar action.

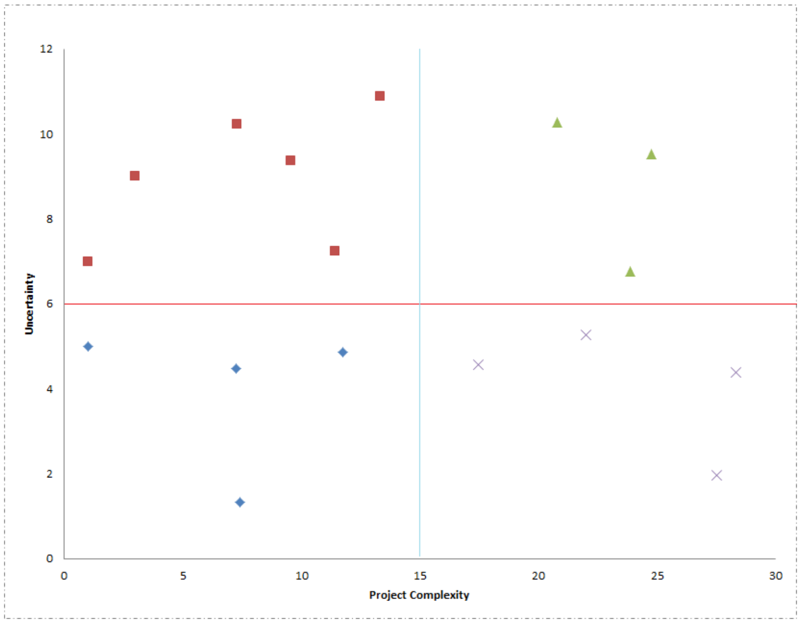

Figure 4.3 Rapid Quadrant Assessment for project processes evaluation (Little et al, 2005)

Projects on the lower left quadrant (called dogs) require very little process and documentation since although they have some uncertainty their duration is short to limit the impact of the uncertainty.

Projects in the upper left quadrant (called colts) are simple, young projects requiring high agility with self-organizing teams because these projects are getting started and have high degree of uncertainty.

Projects on the lower right quadrant (called cows) don’t have much uncertainty but require more process because of their complexity. Managerial processes such as requirements tools, functional specifications, detailed project plans are required for coordination.

Projects on the upper right quadrant (called bulls) are large and complex projects with high levels of uncertainty. They require both heavier processes to deal with complexity as in the cows and experienced project managers to deal with uncertainty.

Boehm and Turner analyzed organizations experience with agile and disciplined methods and characterized the “home grounds” in which agile and disciplined methods have been most successful (Boehm & Turner, 2003). Through this analysis they determined five critical decision factors that “organizations and project can use to determine whether they are in agile or disciplined home grounds or somewhere in between“. These five decision factors are (Boehm & Turner, 2003)

1. Project size
2. Criticality
3. Personnel
4. Dynamism
5. Culture

Table 4.5 Five critical agility/discipline decision factors

Factor	Agile Considerations	Discipline Considerations
Size	Small products and teams. Reliance on tacit knowledge limits scalability	Methods evolved to handle large products and teams. Hard to tailor down to small projects
Criticality	Untested on safety-critical products. Potential difficulties with simple design and lack of documentation	Methods evolved to handle critical products. Hard to tailor low-criticality products

Dynamism	Simple design and continuous refactoring. Rework is expensive in highly stable environments	Detailed plans and Big Design Up Front excellent for highly stable environment but results in expensive rework in dynamic environments.
Personnel	Highly skilled personnel with profound knowledge. Risky to use unskilled personnel	Needs highly skilled and experienced experts during project definition but work can be done with relatively unskilled personnel afterwards.
Culture	Thrives in dynamic culture with empowered, free thinking personnel	Thrives in a culture where people feel comfortable and empowered by having their roles defined by clear policies and procedures.

Based on the five critical decision factors are used to assess the risk in using one of the approaches. The more a particular project conditions differ from the home ground defined for an approach the more risk in using the approach in its original form. In this case blending the approach with complementary practices from the opposite method reduces the project risks.

Boehm and Turner developed a similar sorting of people according their skill and understanding for performing various method-related capabilities such as tailoring, adapting or using a method (Cockburn, 2008).

Table 4.6 Agile and Disciplined method home grounds (Boehm and Turner, 2005)

Characteristics	**Agile**	**Disciplined**
Application		
Primary Goals	Rapid value; responding to change	Predictability, stability, high assurance
Size	Smaller teams and projects	Larger teams and projects
Environment	Turbulent; high change; project-focused	Stable; low-change; project/organization focused
Management		
Customer Relations	Dedicated on-site customers; focused on prioritized increments	As-needed customer interactions; focused on contract provisions
Planning and Control	Internalized plans; qualitative control	Documented plans, quantitative control
Communications	Tacit interpersonal knowledge	Explicit documented knowledge
Technical		
Requirements	Prioritized informal stories and test cases; undergoing unforseeable change	Formalized project, capability, interface, quality, forseeable evolution requirements
Development	Simple design; short increment; refactoring assumed inexpensive	Extensive design; longer increments; refactoring assumed expensive
Test	Executable test cases define requirements, testing	Documented test plans and procedures
Personnel		
Customers	Dedicated, collocated CRACK* performers	CRACK* performers, not always collocated * Collaborative, Representative, Authorized, Committed, Knowledgable
Developers	At least 30% full-time Cockburn level 2 and 3 experts; no Level 1B or -1 personnel	50% Cockburn Level 2 and 3s early; 10% throughout; 30% Level 1B's workable; no Level -1s**
Culture	Comfort and empowerment via many degrees of freedom (thriving on chaos)	Comfort and empowerment via framework of policies and procedures (thriving on order)

One of the biggest impacts of the agile software development is on developers. Agile software development puts people above all other factors for success. Developers must be amicable, talented skilled and able to communicate well (Highsmith & Cockburn, 2001). Building upon Cockburn's levels of developers Boehm and Turner identified the developer levels shown below do determine the quality of people skills available for the projects (Boehm & Turner, 2003).

Table 4.7 Levels of Software Method Understanding and Use

Level	Characteristics
3	Able to revise a method (break its rules) to fit an unprecedented situation
2	Able to tailor a method to fit a new situation
1A	With training, able to perform discretionary method steps. With experience becomes Level 2
1B	With training able to perform method steps (coding a simple method, refactor, following coding standards etc.)
-1	May have technical skills but unable or unwilling to collaborate or follow shared methods.

Boehm and Turner suggests managers use the following dimension map to assess where the project currently is by consulting key stakeholders.

For the organization the main key future trends to consider are:

- Pace of change and agility need
- Dependability on software and need for discipline
- Ability to meet stakeholder’s evolving needs, keeping up with competitors
- Increasing demand for highly skilled people
- Ability to keep up with existing and emerging technical challenges

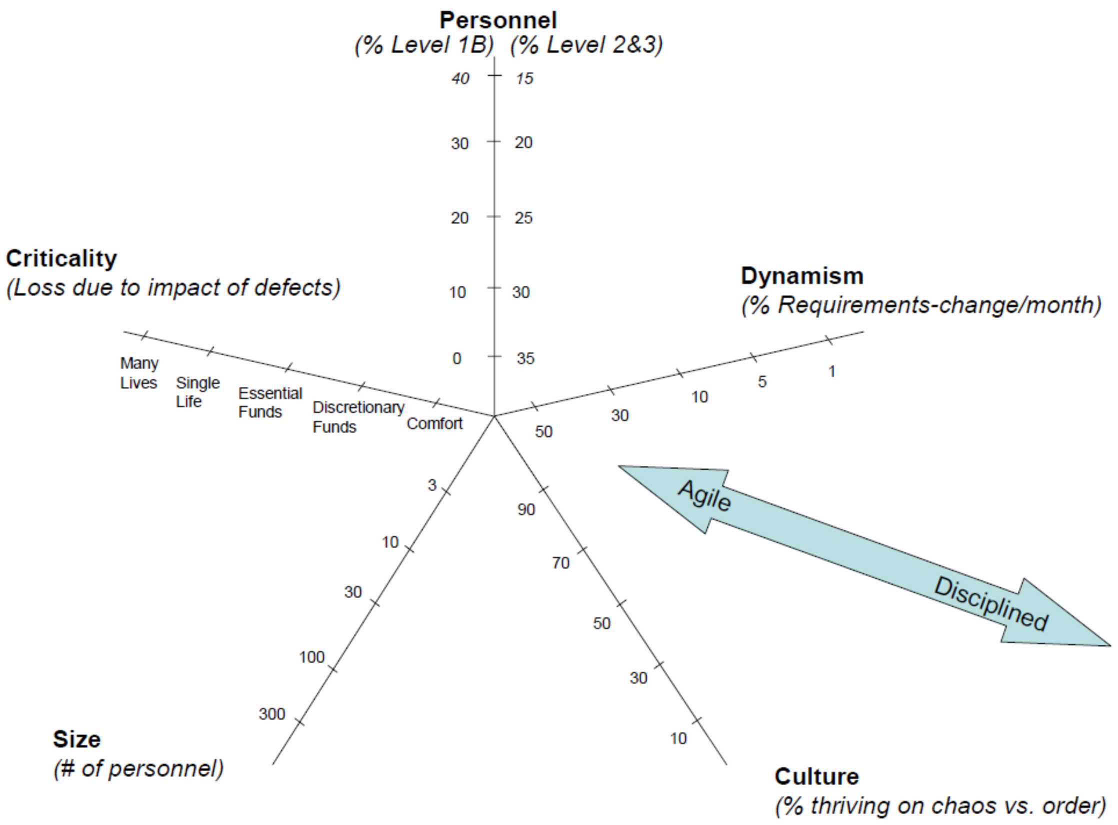

Figure 4.4 Dimensions affecting software projects (Boehm and Turner, 2003.

Each methodology has a different approach to certain risks which it gives a greater importance or tries to avoid. For example if programmers are likely to make coding errors that code reviews have priority. If project designers are prone to leave during the project, they are required to write extensive design documentation.

CHAPTER 5

AGILE METHODS WITH PROJECT AND PROCESS MANAGEMENT ORIGINS

In this part of the thesis project and process management methods with origins in the industry will be examined.

While Scrum is a new product development approach Lean, TOC and Kanban are methods arise from Toyota Production System's flow view of production which is different from the batch process and mass production understanding of production in western management thinking.

5.1 SCRUM

A scrum which is an abbreviated for of scrummage, which is a method of re-starting play in a rugby game. A scrum is formed by the players who are designated forwards binding together in three rows. The scrum then "engages" with the opposition team so that the players' heads are interlocked with those of the other side's front row. Both teams may then try to compete for the ball by trying to hook the ball backwards with their feet (Rugby_Union, 2009).

Figure 5.1 Scrum in rugby game (Rugby_Union, 2009)

In 1986 Takeuchi and Nonaka in the whitepaper the published “The New New Product Development Game” suggested that “the rules of the game in product development are changing. Many companies have discovered that it takes more than the accepted basics of high quality, low cost and differentiation to excel in today’s competitive market. It also takes speed and flexibility”.

A “rugby approach ” to product development in which dedicated , self-organizing teams, the teams of which like actual rugby scrum teams who work together to gain control of a ball and move it up to the field work together to deliver product(Takeuchi & Nonaka, 1986). In the sequential relay-like system “crucial problems tend to occur at the points where one group passes the project to the next”. In Scrum dedicated teams, by maintaining continuity across phases “smooth out” this problem.

Scrum is a holistic approach with six characteristics (Takeuchi & Nonaka, 1986):

1. Built-in stability
2. Self-organizing project teams
3. Overlapping development phases

4. Multilearning
5. Subtle control
6. Organizational transfer of learning

According to Nonaka these are pieces that fit like a jigsaw puzzle to forma a fast flexible process for new product development process which in turn is a vehicle for introducing creative, market-driven ideas and processes into an old, rigid organization.

Table 5.1 Differences of traditional product development and the Scrum (Pichler, 2012)

Old School	**New School**
Different roles: product manager, product marketing, project manager, functional managers, quality manager	Only the product owner is in charge of the product and leading the project.
Product managers are decoupled from the development teams by processes, departments and physical space	Product owner is a member of the Scrum team and works with the Scrum Master and development team
Up-front market research, product planning, business analysis, design.	Just enough up-front work to create vision to roughly describe what the product will be like.
Up-front requirement analysis and freeze.	Product and requirement discovery is an ongoing process in which the requirements emerge. Contents of the project backlog evolve from customer and user evaluation and feedback.
Feedback comes after product implementation and launch.	Feedback is taken through early and frequent customer and user interaction to help create a product that meets requirements.

5.1.1 The Philosophy of Scrum

Scrum project management and product development is designed to enable empirical process control for software development (Schwaber, 2004).

In any organization as the complexity increases the central control and dispatching systems break down. The way to cope with complexity is to increase the independency of the system agents and set appropriate set of rules for them to follow (Snowden & Boone, 2007). The more complex the system, the more likely it is that central control systems will not work.

What Scrum does is to move control from central scheduling and dispatching authority to the individual teams doing the work. The more complex the project the more decision making power is delegated to those closer to the work (Schwaber, 2004). Empirical process which is the core of Scrum; control has three elements: *transparency, inspection* and *adaptation*. Transparency ensures that every element in the process is observable to every member of the project. Inspection is the activity of critically evaluating haw work flows through the process. Adaptation is applying the lessons learned during the inspection on the next iteration of the process (Hawkins, 2012).

This is done because Scrum methodology of software development has been implemented in situations where people are desperate for the product and their regular defined approach is not working (Highsmith, 2002).

Scrum is an agile project mangement and product framework, a way for people to work, a style for building products (Jeffries, 2011). Scrum as a method framework for software development was created by Ken Schwaber and Jeff Sutherland in the 1990s. Scrum is a project management method used on urgent projects that are critical to the organization and is preferred when the requirements are unknown, unknowable or changing (Schwaber, 2004).

Scrum which is the most widespread of all the agile methods (StateOne, 2010) is a popular Agile project management method which uses the concept of empirical process control for managing complex, changing software projects (Szalvay V. , 2004).

In the Scrum process project plans are continuously reviewed and adapted to the changing conditions based on the empirical reality of the project. Agile project management has four variables in software development. These variables are interconnected and a change in one of them creates a change in at least one other.

- Cost: Available resources define the effort put into the system
- Schedule: As the project timeline changes the project is affected.
- Requirements: The scope of the work which needs to be done.
- Quality: Quality in development and production is basically a communication problem(Petzinger, 1999)

5.1.2 The Methodology of Scrum Software Development

When a team starts with Scrum they'll see the things that are in their way and be in a position to improve. When there are "impediments" outside the control of the teams direct control management and the organization is responsible for removing these impediments (Jeffries, 2011). The Scrum set up consists of roles, meetings and artifacts.

The roles are

- Product Owner
- Team
- Scrum Master

Meetings

- Sprint planning
- Sprint demo
- Sprint retrospective
- Daily scrum

Artifacts

- Product backlog
- Sprint backlog sprint.
- Increment
- Burn Down Chart

5.1.3 Sprint

The most critical element of Scrum is Sprint. In Scrum the work gets done in a series of Sprints which are basically a certain amount of time (time-boxed) which is usually two to four weeks(Jeffries, 2011).

A sprint consists of the Sprint Planning Meeting, Daily Scrums, the work of development, the Sprint Review and the Sprint Retrospective.

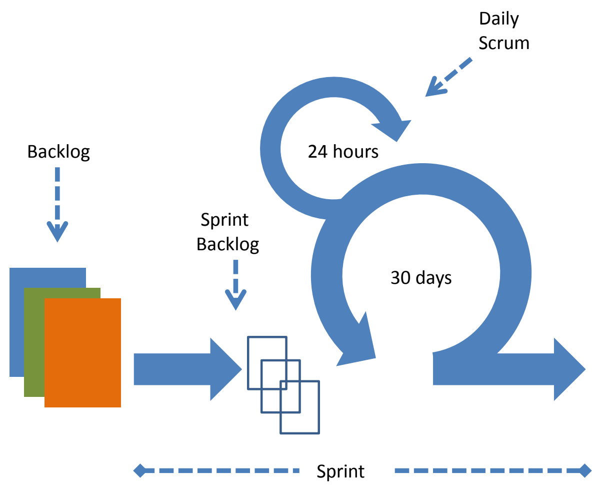

Figure 5.2 Scheme of Scum software development

At the end of every Sprint a usable, potentially releasable product increment is created(Schwaber, 2004). Immediately after the conclusion of the previous Sprint a new one begins.

During the Sprint:

- No changes, that can change the Sprint Goal decided in the Sprint Planning Meeting are allowed.
- The composition of the team is kept constant.
- Goals tha will cause a decrease in quality are not allowed.
- If more information becomes avaliable; then the scope of the Sprint may be negotiated between the Product Owner and the Team.

Sprint is a smaller version of the all over development project with the maximum of one-month time limit. As in every project the sprints delivers a working useful product.

Sprint enables predictability through inspection and adaptation of progress towards the goal in two to four weeks' time. This way also the risk is reduced to and limited with the Sprint time (Schwaber, 2004).

5.1.4 Meetings

The meetings in Scrum are prescribed events so they create regularity and minimize the unplanned meetings to interrupt the working of the team(Schwaber & Sutherland, 2011). This way waste is minimized in the planning process and appropriate amount of time is spent planning. Every event in Scrum beginning from the Sprint is a formal opportunity for inspection and adaptation. Transparency is critical during all the events.

5.1.4.1 Sprint Planning Meeting

Whatever will be getting done in the Sprint is planned at the sprint planning meeting with the involvement and collaboration of the entire Scrum Team. Sprint planning meetings are time-boxed to four hours for two-week and eight hours for one-month Sprints(Schwaber, 2004). The scrum planning meeting must result with the answers to two questions:

1. What will be delivered by the end of this Sprint?
2. How will the work be achieved?

During the sprint planning meeting Product owner prioritizes the Product Backlog items.

5.1.4.2 Daily Scrum Meeting

Each day during the sprint a project status meeting which is called a *daily scrum* or *daily standup* occurs. This meeting has a special place in Scrum and it has specific guidelines:

- All Development Team members come prepared with updates.
- Meeting starts on time even members are missing
- Meeting happens at the same time, same place every time.
- Meeting is timeboxed to 15 minutes.
- Everybody can attend but only the developers speak

During daily standup each member answers these three questions:

- What have you done since yesterday?
- What are you planning to do today?
- Any impediments, problems prevent progress?

If any these are noted by the Scrum Master and resolution is tried outside the meeting. Details are not discussed at the meeting.
Daily scrum meetings are the most critical feature of Scrum method. The purpose is not to meet but to communicate and to improve (Ely, 2003). Every developer is declaring his/her commitment for the day. Daily Scrum is not a status reporting meeting, it is where the team starts to self-organize.

5.1.4.3 Sprint Retrospective Meeting

Retrospective means looking back at what has been done, learning from in and deciding on solid action steps that future performance is improved. Retrospective and the culture of learning and continuous improvement are among the greatest strengths of Toyota (Morgan & Liker, 2006). Agile embraces change which means it is mainly used in projects that need high flexibility in the processes. The flexible processes are inherently emergent, which means they emerge as the nature of the work to be done unrevealed. The process used in every project is different which results in continuous need for learning (Smith, 2007). Trying to learn what went wrong after the project failed makes the learning effort a postmortem. In agile retrospectives are learning exercises where the objective is to learn what went well, what not so well and make sure the process is improved.

5.1.5 Roles

The roles of people working through a Scrum process are detailed here.

5.1.5.1 Product Owner

Schwaber states that the product owner is responsible for the product backlog management and the value of the team performance (Schwaber, 2004)
Product owner in Scrum is the counterpart for the role of project manager in traditional project management. The Product Owner is responsible for maintaining a product backlog that describes the product and continues to with the requirement of the business. As more information becomes available about the product, about the customer or any changes in the

market the product needs to change to meet the new requirements. The product owner in this case adjusts and reprioritizes the backlog to fit the changes direct the project. Having one person in changer across releases ensures continuity and reduces handoffs and encourages long-term thinking (Pichler, 2012). Several roles such as product manager, project manager, and product marketer share the responsibility for bringing in traditional project management. This creates an ambiguity in product ownership which Agile tries to prevent.

5.1.5.2 Development Team

Scrum teams are self-organizing and cross-functional. Rather than being directed by others self-organizing teams choose how best to accomplish their work themselves. Cross-functional means that the team has all competencies needed to accomplish the work without dependence to people outside the team.

5.1.5.3 Scrum Master

The Scrum Master is responsible for the implementation of Scrum principles(Schwaber & Sutherland, 2011).He does so by facilitating interactions that maximize the value created by the Scrum Team. The Scrum Master helps the team by ensuring the Scrum Team applies Scrum theory, practices and established rules. Also helps those outside the Scream Team understand which interactions with the Scrum Team are helpful and which are not.

The Scrum Master’s role is that of a coach and facilitator which fits between the project and the customer. The Scrum Master is responsible for the correct and continuous implementation of the Scrum process. Coaching the product owner and supporting the project processes is the role of the Scrum Master. He helps about how to manage the work of the team using the product backlog, the sprints and review meetings (Hunton, 2012)

The Scrum Master has roles related both to Product Owner, the Development Team and to the organization (Schwaber & Sutherland, 2011).

Scrum Master and the Product Owner

The Scrum Master helps to the Product Owner with

- Product Backlog management
- Communicating the vision, goals, and Product Backlog items with the team
- Development of the Product Backlog items
- Long-term product planning in an empirical environment
- Understanding agility and its practice

The Scrum Master helps to Team with:

- Coaching the Team in self-organization and cross-functionality
- Coaching in the interactions with a non-agile organization
- Facilitating team development
- Leading the team for high-value products.

Scrum Master and the Organization

The Scrum Master helps the Organization in

- Adopting Scrum
- Planning the implementation of Scrum
- Change to increase the Teams productivity

5.1.6 Artifacts

Here the artifacts used during Scrum software development are explained.

5.1.6.1 Product Backlog

The Product Backlog is the single source of requirements and changes to be made to the product (Schwaber & Sutherland, 2011).It is the prioritized wish list created by the Product Owner (Pichler, 2012). Even though it is a list of requirements it is never complete. In the earliest phases of product development it contains the initially known and best laid out requirements. As the product and the environment of use evolves the Product Backlog also evolves. Since Scrum as with any agile methodology embraces change the Product Backlog is dynamic meaning it can constantly change to identify what the product needs to be competitive and useful.

The items in the Product Backlog are all the features, functions, requirements, enhancements and fixes for the future releases.

The Product Backlog is a prioritized list so the high ordered items in it are more precisely estimated with greater clarity and increased detail. Less urgent items have less detail. When items have enough information about them and are decomposed into items which can be “done” within the Sprint time-box they are “ready” for selection in a Sprint Planning Meeting (Schwaber & Sutherland, 2011).
As more information about the items and feedback about the product become available the teams add detail, estimates and order to the items in the Product Backlog. The Development Team estimates the implementation time for each. In estimation process the Product Owner may help in the Development Team‘s understanding and selecting trade-offs but the Development Team is the only responsible part in estimation.

5.1.6.2 Sprint Backlog

A small part of product backlog selected by the team to implement in one sprint.

5.1.6.3 Increment

The increment is all the items combined in the product backlog completed during the most recent sprint and all previous sprints.

5.1.6.4 Burn Down Chart

A popular metric used in agile software development is the burndown chart which measures work remaining versus time or iteration. This chart is displayed for everyone to see and snows the remaining work in the sprint backlog. It is updated every day and gives the progress of the sprint. It clarifies how the team is doing against plan.

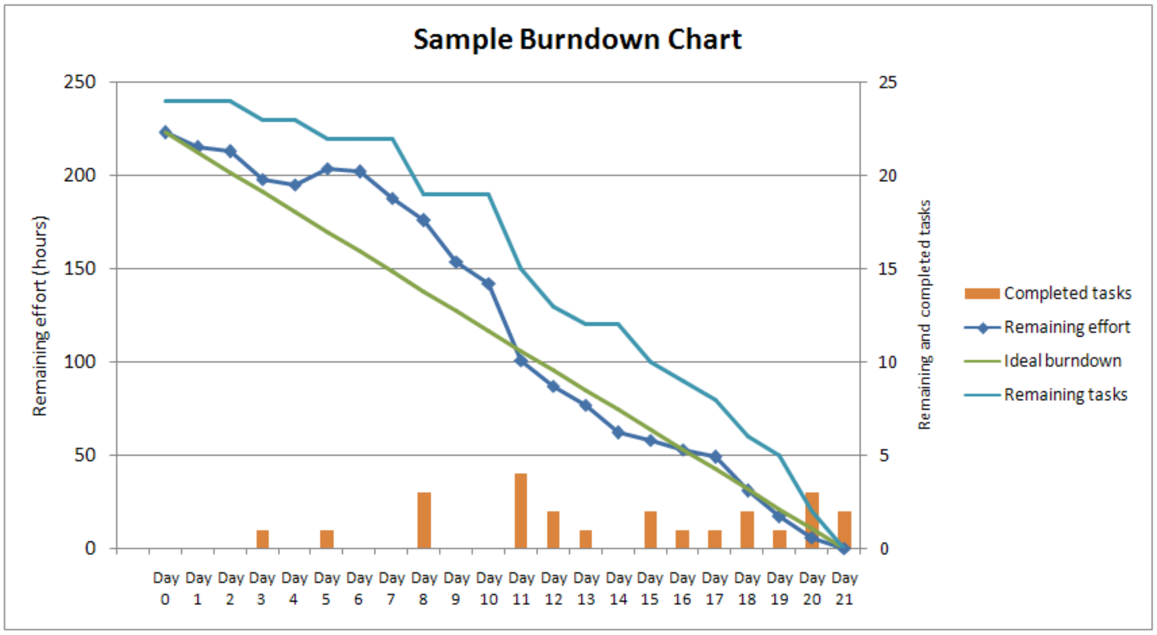

Figure 5.3 Burndown chart in Scrum.

Key Scrum practices are:

- Work is done in "sprints" which are timeboxed iterations of a fixed 30 days or less duration.
- Work within a sprint is fixed. Once the scope of a sprint is committed, no additional functionality can be added, except by the development team.
- All work to be done is characterized as product backlog, which includes new requirement to be delivered, the defect workload an infrastructure and design activities.
- A Scrum Master mentors the empowered, self-organizing, and self-accountable teams which are responsible for delivery of potentially shippable product with increment at each sprint.
- A product owner plays the role of the customer proxy.
- A daily stand-up meeting is a primary communication method.
- A heavy focus is placed on timeboxing. Sprints, stand-up meetings, release review meetings, and the like are all completed in prescribed times.

- Typical Scrum guidance calls for fixed 30-day sprints, with approximately 3 sprints per release, thus supporting incremental market releases on a 90 day time frame.

5.2 LEAN SOFTWARE DEVELOPMENT

Toyota has long been viewed as the most successful car manufacturer in the world (Womack & Jones, 2003). The Toyota Production (TPS) System has been pointed as the source of Toyota's source of success as a manufacturer. The mindset and approaches used in TPS such as the kanban method, quality circles, and lean methods have been introduced all over the world in many organizations to imitate the success of Toyota Company (Spear and Bowen, 1999).

The TPS can be thought as The Thinking Production System because when truly implemented the company has to think differently and everyone in the company must engage in thinking for the company all the time (Minoura, 2003).

Toyota creates a community of scientists. Whenever a specification is defined this means a set of hypotheses that can be tested which is the basis of the scientific method (Spear and Bowen, 1999). For any kind of change a rigorous problem-solving process is used. In the process the current state is assessed in detail and an experimental test for proposed changes which if successful will be the improvement itself is prepared. Without scientific rigor change efforts become random trial and errors.

What distinguishes Toyota system from the Mass Production that it helps the actual work owners to engage in experimentation and design production processes (Spear and Bowen, 1999).

This part of the thesis is about Lean Methodology. Lean Production in which Lean Software Development has its roots is the other name of Toyota Production System.

Here a comparison of Lean Manufacturing and lean software development features are given.

Table 5.2 Lean manufacturing and Lean software development features

Lean Manufacturing	Lean Development
Frequent changes in production	Frequent changes in product (software releases)
Short throughput in manufacturing	Short development cycles
Reduced inventory waiting for manufacturing	Reduced information inventory between development steps
Frequent transfer of small batches of parts between manufacturing steps	Frequent transfer of preliminary information between development steps
Adaptability to volume, product mix and design changes.	Adaptability to design, schedule and cost changes
Workers have high productivity result of broad task assignments.	Highly productive developers as a result of broad task assignment
Priority of quick problem solving and continuous process improvement	Focus on continuous product and process improvement and incremental innovation.
Quality, delivery time and productivity improved simultaneously	Quality, delivery time and productivity improved simultaneously

5.2.1 Applying Lean to Software Development

The target of the Lean and Lean software development in particular is to "manage complexity by accepting that complexity and uncertainty are natural in social systems and knowledge work"(Lean Systems Society, 2012).

The term Lean Software Development was used for the first time as the title for a conference organized by the ESPRIT initiative of the European Union in Stuttgart Germany, October 1991. Independently the following year, Robert Charette in 1992 suggested the concept of Lean Software Development as part of his work exploring better ways of managing risk in software projects.

Jidoka means that the organization has reflexes in place that will respond instantly and correctly to events without going through all the management chain (Poppendieck & Poppendieck, 2006).

The national Institute of Standards and Technology Manufacturing Extensions Partnership's Lean Network defines Lean Development is "A Systematic approach to identifying and eliminating waste through continuous improvement, flowing the product at the pull of the customer in pursuit of perfection". (Jerry Kilpatrick, 2001)

Lean Software Development "reduces defects and cycle times while delivering a steady stream of incremental business value" (Windholtz, 2003)
Lean thinking is based on a set of economic and mathematical principles that describe the flow of product information within the enterprise but which are also applied to supplier and customer elements of the larger business value chain.
Larman and Vodde described a framework for lean software thinking which maps core principles and practices to a manageable software context (Larman & Vodde, 2009). The graphical representation of the framework which is called "house of lean thinking" is illustrated in figure 6.4.

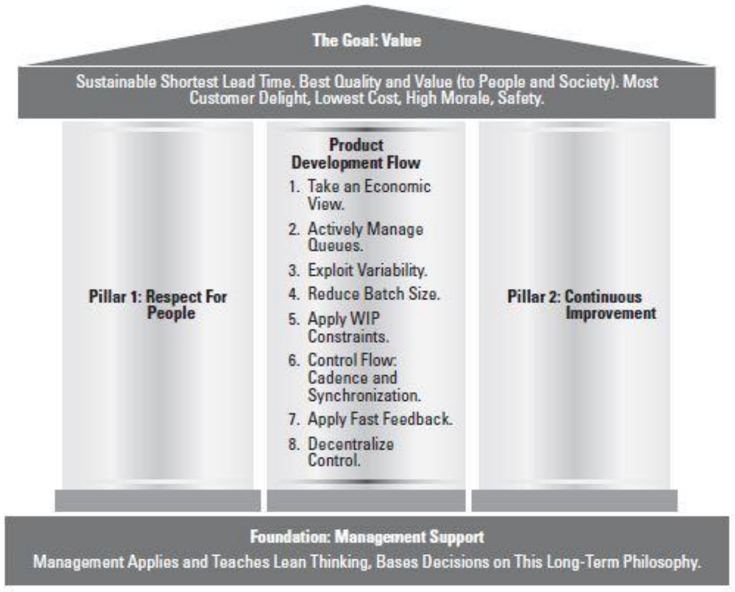
The Goal: Value
Sustainable Shortest Lead Time. Best Quality and Value (to People and Society). Most Customer Delight, Lowest Cost, High Morale, Safety.
Pillar 1: Respect For People
Product Development Flow
1. Take an Economic View.
2. Actively Manage Queues.
3. Exploit Variability.
4. Reduce Batch Size.
5. Apply WIP Constraints.
6. Control Flow: Cadence and Synchronization.
7. Apply Fast Feedback.
8. Decentralize Control.
Pillar 2: Continuous Improvement
Foundation: Management Support
Management Applies and Teaches Lean Thinking, Bases Decisions on This Long-Term Philosophy.

Figure 5.4 The house of Lean (Larman and Vodde, 2009).

The framework has five elements:

1. The Goal: Delivering value fast and in a sustainable way
2. Respect for people: First of the pillars.
3. Continuous improvement: Other pillar.
4. Management support: the foundation of the house.
5. Product development flow: the contents of the house

The goal of lean is : to deliver the maximum amount of value to the customer in shortest possible time(Leffingwell, 2011).

Lean is “looking at the timeline, from the moment the customer gives an order to the point where the cash is collected. And reducing the timeline by reducing the non-value-added wastes” (Ohno, 1988). It is “a way to deliver software as fast that the customers don’t have time to change their minds” (Poppendieck & Poppendieck, 2003).

Lean production tries to solve the problems of customers by upholding the following fundamental concerns of the customers (Womack & Jones, 2003):

- Solve customer problems completely
- Don’t waste customers time
- Provide exactly what the customer wants
- Deliver value exactly where customer wants it
- Supply value exactly when customer wants it
- Reduce the number of decisions the customer must make to solve his problems

5.2.1.1 Seven Principles Of Lean Software Development

Lean practices from manufacturing and management don’t translate easily to software development because software and development are individually quite different than operations and logistics.

In lean software development the main focus is on the people and communication. If people who produce the software are respected and they communicate efficiently it is more likely that they will deliver good product and the final customer will be satisfied.

The seven principles of lean software development are (Poppendieck and Poppendieck, 2006);

- Eliminate waste
- Build Quality In
- Create Knowledge
- Defer Commitment
- Optimize the Whole

- Deliver Fast
- Respect People

5.2.1.1.1 Principle 1: Eliminate Waste

Toyota Production System is a management system for “the absolute elimination of waste” (Ohno, 1988). Ohno explains how this is done as; “looking at the timeline from the moment a customer gives us an order to the point when we collect the cash. And we are reducing that timeline by removing the non-value adding wastes”.

The first step then is revealing what the value is. Value in software is generally captured through requirements.

Identifying and defining value in software development may be problematic because it generally shifts or changes mainly because of the customer’s lack the understanding of *what they want. It is difficult because* (Parnas & Clements, 1986)*:*

- The customer may not know or articulate what exactly they want. Even if all requirements are identified there are many details that cannot be discovered until after well into implementation.
- Even if they are discovered changes in environment result in invalidation of the earlier progress.

This makes it imperative that business goals shape the software and technology used note the other way around (McAfee, 2004). The developers, IT people and business people should really understand each other’s world.

The next step is beginning to unearth the waste. Value must be delivered to the customer at the time and place where it will provide most value and waste is whatever prevents this. What is done during software development and doesn’t add value it is a waste.

Waste (muda in Japanese) in organizations can be found in seven forms as Shigeo Shingo defined them (Shingo, 1981):

Table 5.3 Waste in Lean Manufacturing and their mapping to Lean development

Manufacturing	**Software Development**
In-process Inventory	Partially Done Work
Over-production	Extra Features
Extra Processing	Relearning
Transportation	Handoffs
Motion	Task Switching
Waiting	Delays
Defective Product	Defects (bugs)

Partially Done Work: The biggest source of waste in production systems is inventory (Goldratt, 2004). Inventory is not only handled, moved, stored, tracked and retrieved it also takes time and effort (Womack & Jones, 2003).

In the software development the inventory is "partially done work" (Poppendieck & Poppendieck, 2006). Ideas for software, requirements defined but not implemented, functionality implemented but not tested, software tested but not deployed, buggy functions waiting to be fixed, code needs to be documented are all inventories which are not adding value to the customer and considered waste in Lean Software Development.

Over-production: One of the biggest and most costly wastes in software is the functions or features which the customer rarely uses (Standish Group, 2009). Every feature which is not adding value to the customer and added to software is waste and also a "recipe for disaster" (Poppendieck & Poppendieck, 2003). Every feature of the software coded before business justification is not only a waste of developer time and organization resources but also by adding complexity to software life-cycle because every line of code costs money to write and more money to support.

Relearning: The problem with documenting all the design decisions as they are made is; that these documents are never looked at again. Trying to capture the knowledge gained from trying things that do not work is most relevant but is a challenge. This challenge is addressed in Lean software development by the principle "Create Knowledge".

Handoffs: In traditional project management documents are the medium that carries the knowledge gained and created before and during the project phases. The problem with documents is they are mainly left unread (Basili & Turner, 1975). Handing off information to the “other party” or to those responsible for the next phase of the software development life-cycle causes the about 50% of the knowledge to be lost (Poppendieck & Poppendieck, 2006). This means after four handoffs only 6 percent of the knowledge available at the beginning is left. This is the nature of the tacit knowledge. The issue in project teams is retaining this knowledge. The agile principle “*business people and developers must work together daily throughout the project”* and “t*he most efficient and effective method of conveying information to and within a development team is face-to-face conversation*” addresses this issue and aims to keep knowledge live and valid. Some methods of reducing waste of knowledge in handoffs is reducing their numbers, using cross-functional teams, using high bandwidth communication such as face-to-face discussion, direct observation, simulation and prototyping.

Task Switching: Knowledge work requires large chunks of knowledge for deep concentration and thinking (Drucker, 1967).Human brain doesn’t have the ability of multitasking. What it is doing when trying to multitask is switching from task to task rapidly. Trying to multitask reduces productivity and effectiveness of the brain work (Buser & Peter, 2011). It results in significant working memory disruptions (Clapp et al., 2011). Distraction results in reduced creativity and application of knowledge to different problems (Foerde et al., 2006).

For example a developer needs to finish three tasks called Task A, Task B and Task C by three weeks and he is multitasking. Since multitasking is actually switching between tasks what he actually does is doing small slices of work from each task. He divides each task into eight small pieces and begins to do the work by switching between them. As it is demonstrated in the following figure none of the tasks get done by the end of the third week (Poppendieck & Poppendieck, 2006).

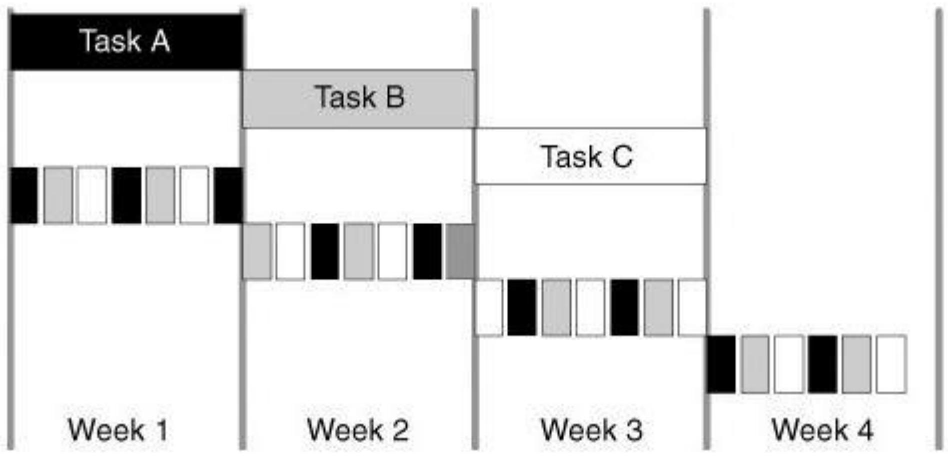

Figure 5.5 Loss of value through multitasking

This is the best case scenario because time lost while switching, time for adaptation and most importantly the potential value lost (at least two weeks for Task A more that one week for Task B) is not considered.

In order to reduce task switching these methods can be used:

1. Rotate off teams or part of the team for each iteration, to handle different tasks such as maintenance and support.
2. Every day; time is set assigned in which the team jointly solves issues of the last 24 hours. New development begins after it.

Delays: One of the benefits of having a collocated and cross-functional team is that with regular feedbacks and easy access to tacit knowledge delays can be decreased while increasing the quality of the decisions made about the product. Even if colocation is no possible making sure that knowledge is available to every developer where ever they need when they need.

Other points of delay in the development projects are (Poppendieck & Poppendieck, 2006):

- Waiting for project approval
- Waiting for people assignment
- Waiting for people to be available for work
- Waiting for change approvals

- Waiting for tests to complete
- Waiting for system integration

Every delay in the working software is value lost for the customer of the product which Lean tries to minimize.

Defects (bugs): Unit testing and acceptance testing is an important part of agile practices which aim to reduce defective code. Writing tests before writing the code forms the design of the code that the developer will write. Writing unit tests before writing code aids in design, reduces complexity and helps the developers to “tackle hard problems step by step” (Osherove, 2009). Test driven development also supplies the best practical documentation about the system because shows how the code is supposed to work.

5.2.1.1.2 Principle 2: Build Quality In

Drucker states that quality is “not what the supplier puts in. It is what the customer gets out of the product or service and is willing to pay for” (Drucker, 1993).

According to Joseph Juran quality is “fitness for use” (Juran, 1951). According to the Quality Glossary of the American Society for Quality “the characteristics of a product or service that bear on its ability to satisfy stated or implied needs”(ASQ, 2010).

The most important point in software development is meeting customer requirements. There are two methods for ensuring quality or meeting customer requirements: one inspection of the product after it has been produced as in testing the software in the test phase and the other one inspect to prevent defects (Shingo, 1989). The best way to ensure quality is to prevent defects in the first place. If this is not possible the product is inspected after each step in the production so the defects can be detected.

Inspection of the defects after the product is produced of software code is written requires that the detected defects (or bugs in software) mean creating a queue for these defects and this contradicts the first principle of Lean Software Development which aims to eliminate all waste and software code waiting for fix is waste. This means quality must be ensured through. In agile methodology as a common practice this is done by using test-driven-development in which unit tests and acceptance tests are written before the associated code(Schooenderwoert & Morsicato, 2004). In the test-driven development first the developer writes an automated test case which defines a desired improved or a new function. The test initially fails because no code has been written. Then he proceeds to write the code which does what is intended and passes the test.

5.2.1.1.3 Principle 3: Create Knowledge

In waterfall method for project management or software development the implicit idea is that the stakeholders or the team undertaking the project has all the knowledge and this knowledge is readily available for those who need it in the beginning of the project and is separate from the coding. In fact every project mainly the software development projects are knowledge-creating projects (Poppendieck & Poppendieck, 2006). A big upfront design cannot anticipate the complexity of the emerging product nor calculate the continuous feedback from development of the actual software. Creating knowledge in Lean Software Development means expect an evaluating design and not to waste time with freezing design specifications prematurely.

McCormack identifies four practices for the success of software development (MacCormack, 2001)

1. Early release of a minimum feature set to customers for evaluation and feedback
2. Daily builds and rapid feedback from integration tests.
3. A team and leader with the experience and instincts to make good decisions
4. A modular architecture that supports the ability to easily add new features

Companies with a good record of product development continuously generate information and increase the knowledge level in the organization (Nonaka, 2007).

5.2.1.1.4 Principle 4: Defer Commitment

Although planning is indispensable in many situations plans become useless fast(Babik, 2005). Plans change since “worldly affairs do not always go according to plan and orders have to change rapidly in response to change in circumstances” (Ohno, 1988) and trying to stick with the plans once they are set; endangers the business.

In many cases the developers of stakeholders of any project have less than necessary information to make any critical decisions. Waiting before making critical decisions, committing to action and using the time for learning about the situation is a way to deal with uncertainty.

Uncertainties are situations or environments which are dominated by the unknown or unknowable variation in the variables. The "solutions" in uncertain situations has to be contingent on future events making for multiple possible solutions rather than one best solution. The problem solving effort has to be drawn out into the future so the solution can be modified as the events and requirements unfold. In this case the problem solving team is not dispersed, it is kept intact so as circumstances revealed the solution is adjusted.

Uncertainty, especially when it is the result of complexity arising from the complexity of the situation requires that various approaches are tried to tackle tough problems. In these cases an experimental environment is created and experiments are made in order to allow patterns of solutions to emerge (Snowden & Boone, 2007). Critical decisions are delayed until more information regarding the situation becomes available and approaches that encourage interaction are used so patterns emerge.

5.2.1.1.5 Principle 5: Deliver Fast

Competitive advantage is the ability of one company to outperform another because its able to create more value from the resources at their disposal. The sources of competitive advantage are skills and abilities in the organization in value creation activities such as R&D, managing new technology or being able to evolve with the changing environment (Jones, 2012). These skills and abilities of the company are called "core competencies".

Core competency is a concept in management theory. Core competency is a specific factor that a business sees as being central to the way it or its employees works.

It has three key criteria (Hamel & Prahalad, 1990);

1. It is not easy for competitors to imitate
2. It can be re-used widely for many products and markets.
3. It must contribute to the end consumers experienced benefits.

Prahalad and Hamel who coined and elaborated the term "core competency" explains that in the short run a company's competitiveness derives from the price/performance attributes of the current products. But in the long run the competitiveness "*derives from the ability to build at lower cost and more speedily than competitors*". The real sources of advantage are in the management's ability to consolidate corporate wide technologies and production skills into competencies that empower individual businesses to adapt quickly adapt changing opportunities (Hamel & Prahalad, 1990). For example iPhone changed how cellular phones designed and used almost overnight. Apple's creating innovation that disrupts the market as it did with iPhone in cellular market, or with iPod in personal media players is the company's core competency.

In today's business and learning environment, time is one of the most critical factors of any organization. Many competitors release their product within weeks of the market leader's release. The faster almost always beats the slower. Being fast doesn't cut it. The product must also be of high quality.

Nokia was the market leader in cellular phones before iPhone. With its introduction iPhone changed how phones are perceived almost overnight and customers lost their interest in the products which Nokia was world leader. Unable to develop and market "smartphones" with competitive to iPhone Nokia lost most of its market share to Apple, Samsung and HTC. At the same time despite being the first to develop smartphone concept Microsoft lost its share of smartphone operating system market to Apple's iOS and Google's Android.

This means software development teams must be able to release software and features fast, respond to fixes and change quickly and they must do so in a sustainable pace or they'll either burnout or slip on the quality.

Unimplemented product ideas are the inventory of the software development teams. Turning an idea to a product quickly is a core competency for organizations. These organizations are able to compete on time basis. This means they can deliver products faster since they have both cost advantage and premium market advantage over their competitors (Hamel & Prahalad, 1990). They have cost advantage because delivering fast means eliminating huge amounts of waste and waste means cost. They have also premium market price advantage since the early products get both premium prices because they have no competitors and they have the advantage of networking effect. Repeatable and reliable speed is impossible without quality which means having a good understanding of the customer. With sustainable speed and high quality such firms are able to experiment with new ideas and products and learn what works.

Sustainable quality is inseparable from speed of delivery because without one the other is worthless.

5.2.1.1.6 Principle 6: Respect People

There are four cornerstones of the Toyota Product Development System. Three of these are about the people involved in the product development system.

One of the principles behind the Agile manifesto is “At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly”. Searching and minimizing delays, handoffs and other non-value-added activities is integral to the software development activities in lean (Leffingwell, 2011).

Development teams are empowered to evolve their practices and improvements. They are responsible for problem-solving and reflection skills and decide for themselves for making appropriate improvements.

5.2.1.1.7 Principle 7: Optimize the Whole

Lean thinking requires a “systemic approach to management of the operations throughout and across all components of the enterprise”(Leffingwell, 2011). What is needed is “to learn to work in the system which means every team, every, division, every component of the system is there not for individual competitive profit or recognition but for contribution to the system as a whole on o win-win basis” (Deming, 2000)

The main tendency in managing projects of developing products is the work break down and completing the parts in the planned manner (PMBOK, 1996). A lean organization optimizes the whole value stream which consists of all the work done from customer’s order to delivery of the value.

Any organization is a system and dealing with systems requires systemic thinking (Ackoff, 2004). Systemic thinking is holistic and synthetic which is different than reductionist and analytic thinking. Reductionist and analytic thinking derives properties of wholes from the properties of their parts. Holistic and synthetic thinking derive properties of parts from properties of the whole that contains them. This can be exemplified by the architecture of a house. When an architect designs a house he first sketches the house as a whole and then puts rooms into it. The principal criterion he employs in evaluating a room is what effect it has on the whole. If the house will be better by making a room worse than that would be an acceptable trade-off (Ackoff, 2004).

Unlike the Mass Production System which focuses on the productivity of the parts and processes to increase efficiency; Toyota Production System or Lean focuses on the optimization of the whole(Ohno, 1988). While management from the Mass Production school would never like to allow employees doing nothing; a manager using the Lean would rather have them do nothing instead of letting them do anything for the sake of doing something (Jacob et al., 2010). Decisions made slowly by consensus thoroughly considering all options and the decisions are implemented rapidly. The organizational knowledge base is protected by developing stable personnel. After the key milestones and project ends all shortcomings and errors in the project are reflected by the teams and management (Leffingwell, 2011).

5.3 THEORY OF CONSTRAINTS

Theory of constraints (TOC) is a continuous improvement methodology developed by Eliyahu Goldratt and introduced in the book “The Goal” (Goldratt, 2004).It was originally applied to management of production and manufacturing operations. In the later books the methodology was extended to project management, supply chain management (Goldratt , 1997), sales, marketing and distribution channels (Goldratt, 1994). Theory of Constraints is a prescriptive theory. Prescriptive theories supply system level guidelines to the subject. How to apply these guidelines to the individual processes or problems are left to the practitioner.

TOC enables managers to answer three fundamental questions about the change needed for the improvement of the system in order to better meet organizational goals. The questions are:

- WHAT to change?
- What to change TO?
- HOW to cause the change?

These questions are designed to focus the managerial effort to the improvement of the overall system. Even though they will have impact on individual processes, these processes will be modified for the overall system improvement not for optimization of the processes (Murauskaite & Adomauskas, 2008).

The core idea of the TOC is that for any given system and at any time of its working there will be at least one constraint in the system determining the production output of the system. A production system is only as fast as the slowest process in the chain of production. The capacity of this slowest link is the current system constraint (Anderson, 2003)TOC does not optimize the individual steps in the system. The focus is on maximizing the throughput of the system. The goal is to increase the production capabilities of the system while reducing the cost of production and shortening the overall production time.

The five basic steps of TOC to achieve this goal are (Goldratt, 2004):

5.3.1 Step 1 Identify the constraint(s)

Find the process which by being the weakest link in the chain of system at a given time affects the overall performance of the whole system. This may be a physical or policy.

5.3.2 Step 2 Exploit the constraint to maximize productivity

When the bottleneck is found it is essential to ensure that it works 100%. This may be done by either pruning non-value adding activities at the bottleneck or by increasing the capacity of it. Making the bottleneck “lean” by removing non-value adding activities the capacity of the bottleneck can be increased with no additional resources.

5.3.3 Step 3 Subordinate all other steps or processes to the speed or capacity of the constraint.

After performing the second step all of the system must be adjusted to enable the bottleneck to operate with maximum effectiveness. This is done by changing policies, rules, shifting responsibilities of the bottleneck resource to other processes etc.

5.3.4 Step 4 Elevate the constraint.

Increasing the throughput in the current constraint increases the throughput of the overall system. Elevating the constraint means taking necessary measures to break the bottleneck. This means making necessary resource (money, energy, time etc.) investment after doing everything possible in the third step.

5.3.5 Step 5 Go to step 1 and start over.

If the constraint in step 1 is removed and is not a bottleneck for the system anymore, this means the system has a new bottleneck somewhere else. Thus it is time to start from the step 1 all over again. This is the process of continuous improvement and doesn’t have an end. Continuous change in the system and the environment may result in a bottleneck which was removed earlier becoming a bottleneck again which means it has to be revised.
The system has three basic measurements in TOC.

1. **Throughput** which is the rate at which the system generates money through sales
2. **Inventory** is all the money that the system has invested in purchasing things which it intends to sell
3. **Operational Expenses** is all the money the system spends in order to turn inventory into throughput.

5.3.6 Applying Theory of Constraints to Software Development

Depending on the methodology used for software development there are processes where ideas needs to go through before they are delivered to the customer as working software. Some of these are(Spolsky, 2012),

1. Decision making points –is the idea going to be implemented?
2. Design processes
3. Implementation of the idea
4. Testing process
5. Debugging process
6. Deployment of the software

The code implemented and waiting to be tested is the inventory of the software development process. The cost of code inventory can be very high. The items not delivered to the customer as working software and there is high competition in the customer market the product can become obsolete in a very short time.

There are three places where most of the inventories in software development accumulate (Spolsky, 2012):

1. Feature backlogs. Time is wasted for designing more items when there are items to be implemented in the feature backlog
2. The bug database. Trying to resolve every bug report without prioritizing or allowing more time spent for bug fixes than product development is time wasted.
3. Undeployed features. When there are customer requirements implemented in the product but not deployed they are not creating value for the customer. Undeployed software is one of the biggest wastes in software development.

5.4 KANBAN

Kanban is a word of Japan origin which literally means “signboard” (Gross and Mcinnis, 2003). In a manufacturing environment kanban cards are used as a signal to tell an upstream process to produce more. The workers in each step of the process can only produce when they are signaled from the downstream step with a kanban. Taiichi Ohno- the architect of Toyota Production System- developed kanban to implement Just-in-Time (JIT) and control the processes (Ohno, 1988).

5.4.7 How is the Kanban used?

Kanban in software development is used for "incremental evolutionary improvement" through the use of TOC and reducing the work-in-progress of software development team through batch-size reduction and variability reduction variability. Before implementing the kanban, the system's capacity is agreed upon. This capacity known as Work in Progress is the amount of work the system can perform with quality and without waste. A number of kanban (cards) equivalent to the agreed capacity of the system (WIP) are placed in circulation. A work can be done only when there is a card available. This free card is attached to the work and follows it through the system. When there are no cards then no work can be done which means the system is working on the maximum number of works previously agreed upon. New work waits for a card to become available so it can start. When the work is finished then the card is detached from it and recycled to waiting queue so new work may begin.

Toyota uses kanban not only to manage cost and flow but also to identify the problems in the value flow and to pinpoint the opportunities for continuous improvement. In traditional work operations a push based system is used for either manufacturing or delegation of work. Contrary to the push system kanban is a pull production control system which controls the amount of flow of material through a system (Hopp & Spearman, 2004).

With a push system work is initiated at the beginning of the production system driven by a customer or production order and its production schedule. When first process in the production line completes work it is *pushed* to the next process in the line regardless of whether the next process is ready for work. This goes on until the output is delivered to the customer. If everything works as planned and scheduled the product should be completed on time and delivered. But conditions change continuously and high levels of WIP began to accumulate. Any process which for some reason didn't finish the process as scheduled begins to accumulate inventory before it. This in turn results in long lead times, late delivery resulting in waste in the system (Marsh & Conard, 2008).

In a pull system the work in progress (WIP) is pulled through the operation based on the downstream (the following process) demand rather than the traditional approach to push WIP.

The idea governing the pull system is that the customer should drive the demand not the schedule. When the customer for a process (the customer of every processes is the following process) purchases a good work is pulled from the immediately preceding process in the chain and this is repeated successively all the way back to raw materials.

Pull systems as a distinction from the push systems; by using kanban limit the level of WIP(Hopp & Spearman, 2004). Lower levels of WIP reduce response time to process demands through the operation by Little's Law.

Past research on pull systems showed to be effective at reducing waste in both WIP and response time when applied to operations with repetitive outputs for either physical or non-physical tasks(Marsh & Conard, 2008)(Hopp & Spearman, 2004). This means kanban can be used for knowledge work.

The operators through using kanban use visual signals to understand how much to produce and when to stop. The principles of kanban also tell the operators what to do in case of a problem and who to consult. Use of kanban also visualizes the schedule status for the managers and supervisors (Gross and Mcinnis, 2003).

5.4.1 Applying Kanban to Software Development

The Principles behind the Agile Manifesto states that "Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely."

It is a common practice that the software development and project teams are not in the decision loop on the amount of work they are required to do. Most teams are required to commit to unknown deliverables with unrealistic project schedules and without enough resources (Anderson, 2010). The stress which is the result of workloads on software developers many times makes it impossible to sustain a development phase even for a moderate length of time. The result is teams losing their productivity at best.

Every development team is unique in terms of skills, capabilities, experience and form. This uniqueness includes the situation they are in. Each project is different in terms of budget, schedule, scope, requirements, quality demands and risk profile. Also organizations are different from each other in their culture, operations, supply chain; management practices (Anderson, 2010).

Use of kanban is software development setting is synthesizes of some ideas from Lean with Theory of Constraints of Eliyahu Goldratt.

Goldratt's Theory of Constraints (TOC) improves a system by eliminating bottlenecks that constrain the performance of the system one by one. In using TOC a bottleneck is identified and alleviated until it is no longer a constraint. When this is accomplished a new bottleneck emerges and the cycle is repeated.

When the workflow of a software development lifecycle is modeled as a value stream and then kanban is used as a tracking and visualization system to track state changes of emerging work as it flows through the system the bottlenecks in the development lifecycle becomes apparent. In the pull systems the work-in-progress is limited to an agreed-upon quantity. This way the workers are prevented from becoming overloaded which is one of the critical wastes in the Lean.

In Agile software development card wall is a popular mechanism for visual control. This may be in form of a cork board with index cards pinned to it or whiteboard with sticky notes to track the work-in-progress.

In some agile development environments and teams Kanban is used as a substitute for some of the common practices of agile. For example teams can use kanban boards instead of iterative development and burn down charts (Polk, 2011)

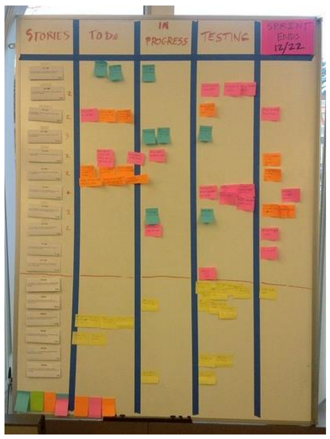

Figure 5.6 A Kanban Board

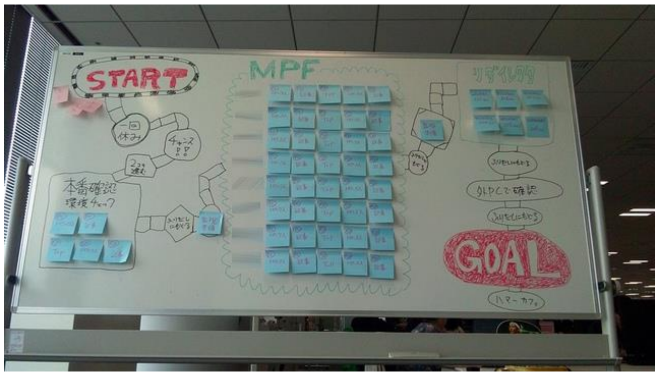

Figure 5.7 Another Kanban Board

In software development a kanban system is used to limit the work-in-progress (Anderson, 2010). Kanban in software development is used as a way to help the development team to control the environment and manage the work (Polk, 2011).

In order to be defined as kanban the work control system must have two properties.

It must have

1. Explicit work-in-progress limit
2. A signaling to pull new work through the system

Used this way Kanban makes it possible to balance the demand on the team against the throughput. This enables a sustainable pace of development so all developers can achieve work with balance (Anderson, 2010). This in turn makes high quality and high performance possible. Combining improved flow with better quality enables shortening lead times thus improve predictability and due-date performance.

Visualizing and exposing problems Kanban facilitates resolving and eliminating the problems and their effects thus improve collaboration and continuous improvement. Kanban uses five core properties to create an emergent set of Lean behaviors in the organization. The properties are(Anderson, 2010);Visualize workflow, limit work-in-progress, measure and manage flow, make process policies explicit, improve collaboratively using models such as Theory of Constraints, Systems Thinking, Edward Deming’s System of Profound Knowledge, and Lean.

Since each team and each software development environment is unique, the Kanban used by these teams are unique.

Kanban is neither a software development lifecycle methodology nor a project management method. In order for Kanban to be used some methodology or process should already be in place. With the use of Kanban the process is visualized and incrementally improved.

The primer goal of applying Kanban in software development is to improve the processes used in development and the environment. To enable continuous improvement in the organization needs change. The six steps to overcome the barriers of change in the organization used in Kanban are (Anderson, 2010):

1. Focus on quality
2. Reduce Work-in-Progress
3. Deliver Often
4. Balance demand against throughput
5. Prioritize
6. Remove sources of variability to improve predictability

5.4.1.1 Focus on Quality

The principles behind the Agile manifesto mentions craftsmanship which focuses implicitly on quality (Manifesto, 2001). The quality of the software can be improved by both traditional and agile approaches (Anderson, 2010). Also code inspection, collaborative analysis and design, use of design patterns, use of modern development tools improves quality.

5.4.1.2 Reduce Work-in-Progress

Reducing the WIP increases the delivery rate and the quality (Kotter & Cohen, 2002). C17 Globemaster III is a large military transport aircraft which is used for rapid strategic airlift of troops and cargo. From its beginning the C-17 program met with development difficulties, quality issues, late deliveries and cost overruns. From year 1992 to 1993 deliveries were behind schedule. When it became apparent that meeting the delivery schedule with the current production tempo is impossible; the program management temporarily took airplanes out of workflow thus reducing WIP so that cycle times improved. At the time aircraft moved through various stations on the assembly line with specific set of tasks conducted at each station. Even if the tasks at one station were not completed the aircraft were moved to the next station where the personnel from the previous station tried to catch-up with their work. This was done to meet the schedule (Converge Consulting Group, 2005). The project manager made quality of work priority rather than the schedule. He decided that no plane would move forward in the production cycle until all tasks in that station were completed well. This meant that the planes in the upstream stations are temporarily taken out of workflow. When the plane holding up the work is moved ahead than all others would proceed with work.

As a result while plane number 12 was late as others before it with the introduction of the new practice plane number 13 was delivered ahead of the schedule. Quality and cycle-time improved and all other planes were delivered ahead of the schedule until the program termination.

5.4.1.3 Deliver Often

Delivering small frequent high-quality releases builds trust between the stakeholders (Anderson, 2010). Delivering often is the result of optimization of the processes, creating agility and is established through reduction of waste and variability in the development.

5.4.1.4 Balance Demand against Throughput

Balancing demand against throughput means the rate which new development requests for requirements made correspond with the rate of software development team’s rate of delivery. The teams pull new work from the development queue (backlog) at the rate they finish work.

5.4.1.5 Prioritization

When the output rate of the development team is predictable and quality is built in the software management will be in a better position to focus on prioritizing the items in the production queue and they will be delivered in the order they are requested.

5.4.1.6 Removing Sources of Variability

Variability in industrial processes has been studied since 1920s and pioneered by Walter Shewhart and his followers Joseph Juran and Edwards Deming; founders of the quality movement in Japan and in the world. With the reduction of variability the cost of development and evolution of the software product is minimized (Svahnberg et al., 2001).

5.4.2 Implementing Kanban

Kanban drives change and improvement by optimizing the processes which are already in use (Anderson, 2010). At the beginning nothing is changed in the processes; workflow, job titles, roles, and responsibilities and practices. The first focus is the amount of WIP and the interface to an interaction with upstream and downstream parts of your business. The team which is doing the work is to map the value stream as it exists. When the process in use is different from the official process sanctioned by the organization the process which is actually in use should be mapped. Since team members can use the card wall as a process-visualization tool only if it reflects what they actually do; mapping it is elemental for Kanban.

5.4.2.1 Start and End points of Control

It is important to map the political sphere of control and limiting WIP within it and visualize the workflow within it. For example when implementing Kanban in a software development function and the team is responsible for analysis, design, coding and testing then this value stream is visualized and the interactions with the partners in the upstream and downstream who provide requirements, prioritization and scope are negotiated for the new style of interaction. This way the pull system which is implemented is limited to the environment of influence of the team (Anderson, 2010).

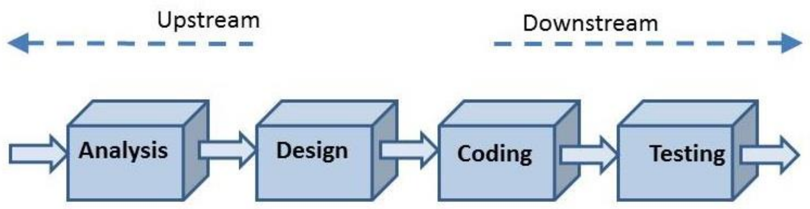

Figure 5.8 Work stream in a software process

5.4.2.2 Work Item Types

Once the value stream which the team is responsible for is selected the types of work that arriving to this workflow is defined such as incoming work, outgoing work and work that exists within this workflow. While incoming work may be such as user story, use case, functional requirement or feature, outgoing work may be functions, features implemented, bug fixes, refactored code, code changes, improvements etc.

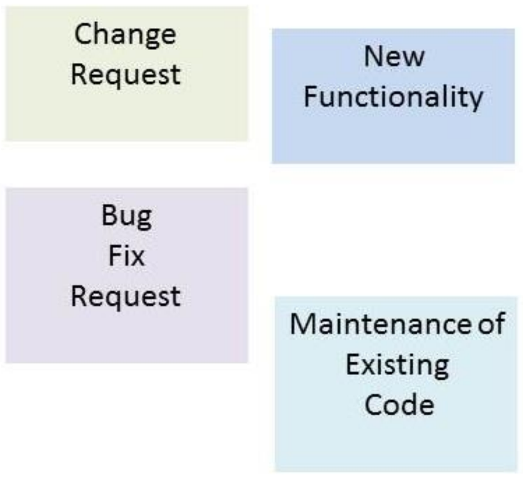

Figure 5.9 Work items in Kanban

5.4.2.3 Drawing the Card Wall

Typically card walls show the work done within the workflow not the functions of job descriptions of the team members.

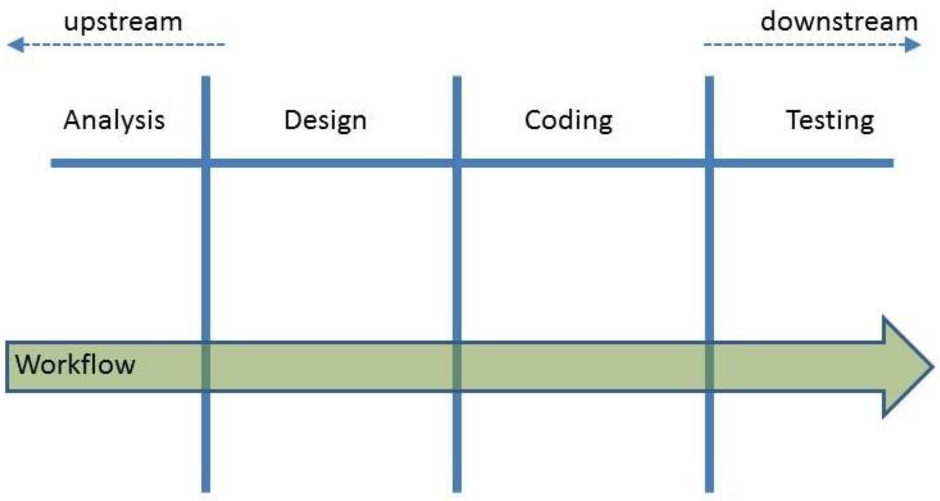

Figure 5.10 Workflow in Kanban

The common practice in drawing card wall is separating the work types into separate swim lanes in which ever work type is tracked. For example if the team is prioritizing maintenance over implementation of new functionality and change requests then these are visualized on the board with widths appropriate to their percentage in the overall work.

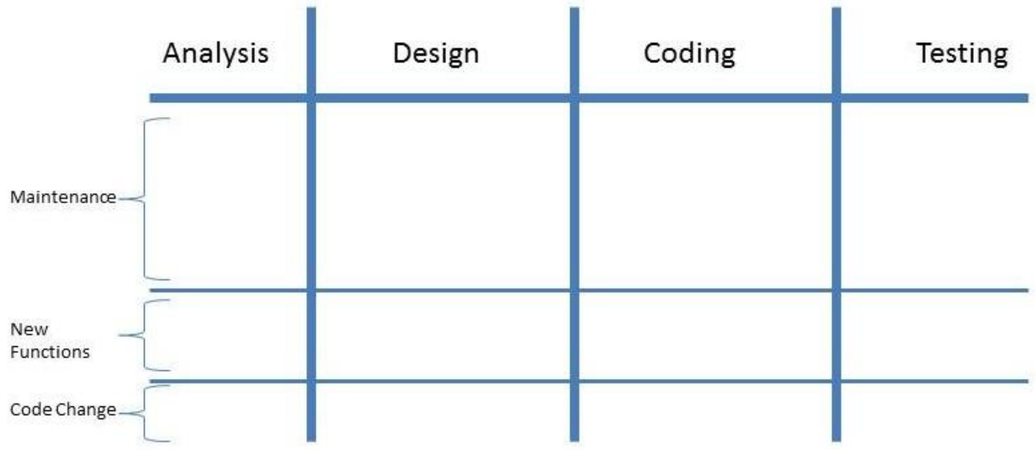

Figure 5.11 Kanban swim lanes for different types of work

5.4.2.4 Work Item Cards

Work item cards are designed to meet the requirements of the team and facilitate the pull system. Each card contains several pieces of information and they are instrumental for individuals to make pull decisions. An example for item card is illustrated here. It contains:

- Unique id number for work item
- Title of the item
- The date of entry to the system. This is used for FIFO (first in first out) queuing and can be used for the lead time of the work item.
- A token signaling that the item is late than specified lead times.
- Name or avatar of the assigned developer.

When designing the work item cards the focus is that it should contain sufficient information to the team members necessary for facilitating project-management decisions and should empower team members with transparency of process, objectives and risks (Anderson, 2010). When the team is clear on project goals and objectives, has up to date risk information and is empowered with the transparent processes Kanban facilitates a self-organizing risk-management system.

Respecting individuals is one of the seven principles of the lean software management. By trusting the team for making their own scheduling and prioritization decisions the team may enable a Lean organization.

5.4.2.5 Limiting Work for Tasks

Multitasking in work environment kills productivity, dampens creativity and makes workers anxious (Dean & Webb, 2011). Reserving chunks of time for reflection on the work being done and the lack of availability of proper time management in organizations is a critical problem for management. Peter Drucker emphasizes that “most of the tasks of the executive require for minimum effectiveness a fairly large quantum of time”. When people have highly fragmented days with many activities, meetings etc. their creative thinking decrease significantly (Amabile, 2002). A majority of managers in a survey of Reuters; state that information overload lessens job satisfaction and damage personal relationships. It goes as far as damaging physical health (Bawden & Robinson, 2005).
Limiting the amount of work on each developer and preventing multitasking is a critical step for improving knowledge productivity (Converge Consulting Group, 2005).
When implementing Kanban; if a software development team has three testers; the practice is to limit the WIP in testing by three which is one per person (Anderson, 2010). When there is organizational resistance for limiting to one item per developer for it being too restrictive two items per person is considered the upper limit. Since every development environment is unique this limits can be found with empirical methods.

5.4.2.6 Limits for Queues

Queues are where the work item which is done in the upstream process waits to be “pulled” by the next process. Since waiting is considered a waste (muda) in Lean the queues should be as small as possible. The queues exist to absorb the variations in the workflow. If the work is “flowing” through the teams workflow; than the queues don’t need to be big. But if a stop-go behavior is observed which causes some team members to be idle because of the variability in the work completion times of the upstream tasks then queues may be enlarged but the priority must be finding the source of variability and removing it.

Imposing WIP limit across the value stream is about unearthing and revealing organizational and process problems and dysfunctions as they are for developer productivity. The policies and processes which result in sub-optimal productivity, long lead-times and low quality become visible when there is WIP limit. Continuous improvement culture emerges by discussion of these constraints in transparent and open manner and collaboration of colleagues in overcoming them (Anderson, 2010).

CHAPTER 7

FINDINGS

Software development is much more than creating algorithms and writing computer code. Computer science has a theory of computation which deals with whether and how efficiently problems can be solved through a model of computation using algorithms. It has sub branches, automata theory, computability theory and computational complexity theory. Software Engineering and software development on the other hand doesn’t have a theory(Jacobson & Spence, 2009) to define the concepts, ideas and show how software should be developed, what works and what not works.

A theory is a “coherent group of general propositions used to explain a given class or phenomenon”(Webster, 1981). The goal of any theory is “to clarify concepts and ideas that have become confused and entangled” (Clausewitz, 1984). Any question or problem can be examined clearly only after defining the terms and concepts in the question. The theory should clear what are the terms and concepts and explain how one thing is related to another also help to keep the important and unimportant separate. Theory “doesn't provide rules and regulations for action. It develops a way of thinking rather than prescribe the rules of action” (Pellegrini, 1997).

A theory is required “to distinguish between cause and effect, trivial and important, and peripheral and central” (Eccles, 1965). Even the theory is “imperfect or incomplete it can clarify many obscure matters” (Eccles, 1969).

First and above all software is about solving problems for people thus software development is a social undertaking. No matter how small the total number of people included in or affected by it, software development is an organized social activity to meet people’s requirements and solve their problems.

In the preceding parts of this thesis it is shown that this activity of social undertaking may be called a "temporary production system" (Koskela & Howell, 2002) or a project. Furthermore this production system can be perceived as a complex adaptive system (CAS).

In software projects there exists stakeholders, customers, development teams, managers, organizations, requirements, specifications, knowledge created, errors made, lessons learned accompanying a great amount of interaction and communication. Every part is connected to every other part in some way or other. This connectedness and dependencies which is called complexity (Jones, 2012) are in most cases many times beyond the comprehension ability of managers.

In the preceding chapters CAS as a theoretical foundation for software development is examined. Using the CAS as the management theory in software development has advantages in multiple levels.

Instead of focusing the separate actions and events management can make sense of both the whole and the parts with using the CAS theory. This is especially useful in software projects in which the rapidly changing requirements and fast paced environment requires high levels of adoptability and agility.

Furthermore software projects require high level of human interaction with multi layered roles, stakeholders, risks and uncertainties. Although there is extensive literature on the technical aspects of project management the most successful project managers are the ones with greater human skills and experience besides their technical skills. Founding project management upon CAS helps project managers by making individual motivations and actions comprehendible together with the project constraints, the goal of the project and the system as a whole.

Complexity is a concept of systems which is a set of interacting elements or parts which have different kind of relationships within the system than with other elements outside the system. The opposite of complexity is independency.

A complex system is a system with interdependent interacting elements that show properties emerges from these interactions which are not found in any of its parts. Complexity theory is "investigating the properties and behavior of the dynamics of nonlinear systems" (Alberts & Czerwinski, 1996).

A software development project is system. Managing a system is not done by managing individuals or parts of the system but by managing the interactions of the parts that make the system. As Russell Ackoff puts it “You have to understand how the interactions of the parts and the parts with the whole and its environment, create the properties of the whole. Cause-effect is about actions, not interactions.”(Ackoff & Allion, 2003). He states that many managers focus on managing the separate actions of the organizations or projects parts. This course action is based on assumption that improving the performance of the parts separately improves the performance of the whole. Ackoff states “Contrary to this belief improving individual parts can cause total failure of the whole” (Ackoff, 1986).

The most critical shortcoming of waterfall and traditional project management in this context is that in these approaches a product is planned as the outcome of the project and then the activities that are expected to produce that outcome are planned, sequenced, scheduled, and budgeted. If any problem arises it is dealt with the improvement of the responsible part. And in this train of thought the project manager is separate from the development ecosystem and he can make choices from outside the interactions (Stacey, 2000).

This is contrary to the nature of software because software is complex, abstract its requirements are incomplete and software development is an ecosystem of research and design with a highly complex nature. Project management team is part of this ecosystem, thus its actions and the way it interacts with other parts directly and non-linearly affect the outcome. Also since it is a system improving parts or phases individually improves neither the whole nor the outcome. One important result of complexity inherent in the system is that it makes predicting outcomes very hard even when detailed information about the parts and their interaction is available.

This is because understanding the parts and interactions doesn't increase the knowledge about resulting emergent behavior. It is like a double pendulum which is a very simple mechanical device and all its physical properties are known, but for some energy level its motion is completely chaotic thus unpredictable.

Development team, customers, stakeholders, requirements, specifications, technical structure organization, are parts of the software development in which even all the properties have been known the outcome would still not be predicted.

When software development is perceived as a complex adaptive system, the managerial focus shifts from planning activities to interactions. The interaction within the team and between the team and the customers and stakeholder becomes more important since emergent product is the result of this interaction.

Managing interactions begins with the job design of the individuals in the project to interpersonal communications and communication with customer and stakeholders.

Appelo offers a six faceted management model in complex situations which is compliant with agile development and with a focus on agents of the system and their interactions (Appelo, 2011):

1. Energize People: Managers must focus on keeping people motivated creative and participative in the software development because they are the most important elements of the project.
2. Empower team: When the teams are empowered, authorized and able to build trust then they self-organize for the best possible outcome
3. Manage Self-organization: Since complexity results in the unpredictability of the outcome, people must have clear purpose and shared vision.
4. Develop Competence: The quality of the team members must be developed for the team to be able to achieve the goal.
5. Grow Structure: Structures (methodologies/processes in software development) are for the coordination of the individual and team efforts thus processes must enhance communication.
6. Optimize the whole: People, teams and organizations must continuously improve in order to keep up with change and success.

A social system has its own goal as do its parts and the subsystems it contains. Managing a social system is managing complexity because the social system has multiple goals in each of its multiple levels with some goals being incompatible or conflicting within or between different levels.

Accepting the software development as a social system shifts the managerial focus from managing individual activities to managing interactions of the agents and elements in the system.

Complex systems cannot be controlled (Rzevski, 2011) but can be managed by simple rules (Eisenhardt & Sull, 2001), through facilitation, using interactions (Ackoff, 1986).

As a way of understanding and managing complex systems Snowden offers opening up discussions, setting barriers, stimulating attractors, encouraging dissent and diversity and managing starting conditions and monitoring for emergence as methods for managing complexity (Snowden & Boone, 2007).

An application of CAS theory to software development can be given in Scrum methods. In Scrum the basic interaction of the development team is through the daily scrum meetings in which every team member answers 3 questions

1. What I did yesterday?
2. What I am going to do today?
3. What impedes progress?

This is encapsulation of communication which is an object oriented programming term and gives a common interface for team interaction. These simple rules of interaction enable the development team to organize itself for the best outcome and to overcome the problems.

The scrum master facilitates self-organization within the team using subtle management methods (Takeuchi & Nonaka, 1986) respecting the people in the team but also directing the team in the right direction. In Scrum management's main task becomes to remove obstacles.

In CAS self-organization and the emergence of outcome can be managed through the containers-differences-exchanges (CDE) framework by Eoyang which identifies three conditions that influence the speed, aim and outcome of self-organization in human systems(Eoyang, 2001). These are:

- The containers that bound the system such as time frame for development cycles, software development team itself, people sharing the same title or responsibility. For example the development team is a container in software development so are the customers.

- The differences within the container that articulate patterns that affect the behavior of the group or individuals and is affected by them. For example in waterfall the development team contains only the similar minded/tasked developers while analysis team contains the analysts, testing team testers so on. On the other hand in Scrum development team is cross-functional, multi-disciplined, meaning analysts, developers, testers, coders, architects are together in all the steps of the development.
- The exchanges of information, resources, power, and money are essential for the complex adaptive system to change and emergence. People talk, learn, exchange, act together, share experiences and stories. All these exchanges result in the patterns of being and behavior to emerge. For example in Lean software development "create knowledge" principle results in fast learning and diffusion of experience results in early detection of problems thus increase success rates.

Every project is unique in its own way thus require a different, specifically tailored and managed process or processes. This thesis offers the use of project and process management methods explored and reviewed in this thesis, as a baseline approach to be selected and tailored for each project with regard to the dimensions compiled in Table 9.2

Table 6.1 Dimensions of software development projects

MacCormack	**Cockburn**	**Rapid Quadrant Assessment**	**Boehm and Turner**
Nature of requirements Communication Technical Maturity People and Team Structure	System criticality Group size and communication Process needs and organizational necessities	Complexity: Team capacity, size, location People quality Dependencies of systems Uncertainty: Requirements Technical research need Project duration Scope flexibility	Team size and nature of the knowledge Criticality Dynamism of environment and requirements Quality of the personnel Organization culture

Also a landscape model for software projects, complex systems domains adopted from David Snowden is used.

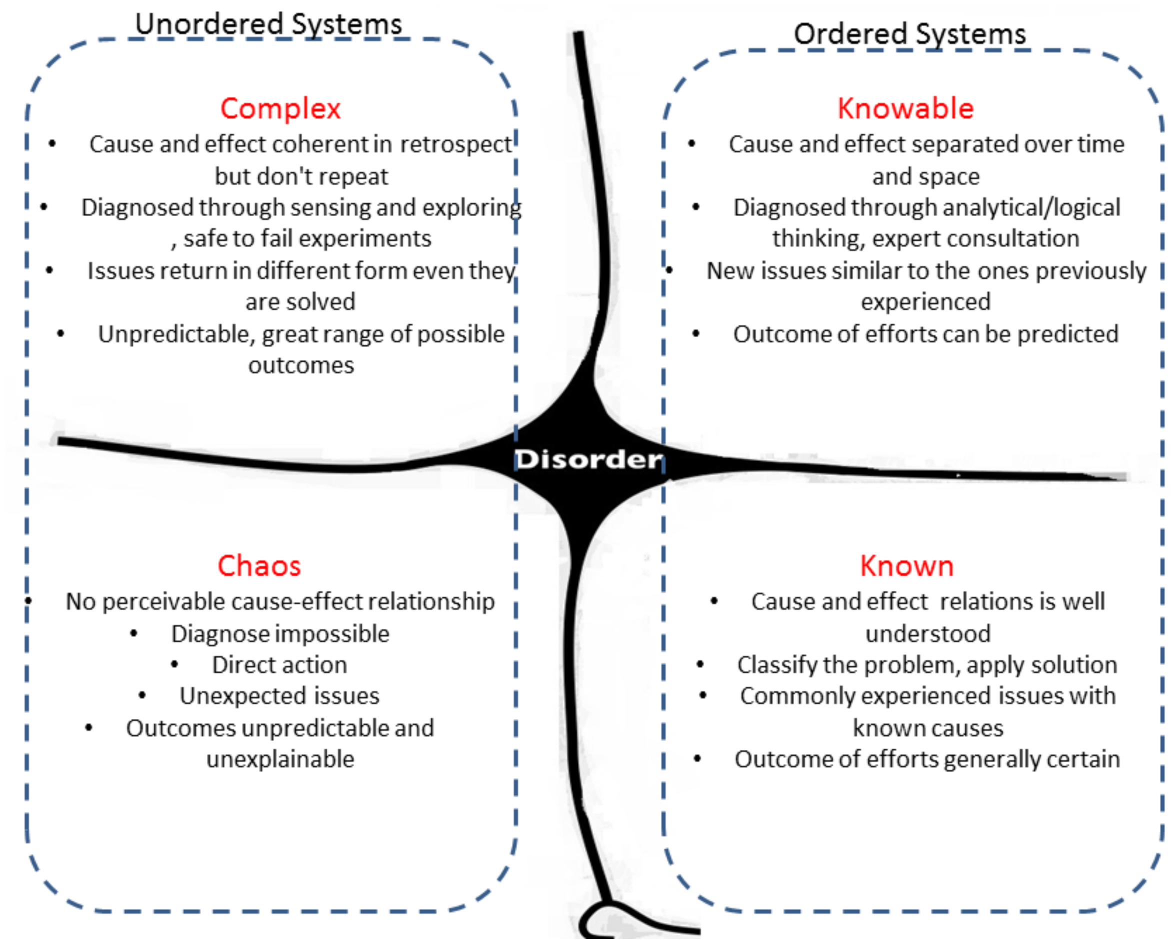
Unordered Systems
Ordered Systems
Complex
• Cause and effect coherent in retrospect but don't repeat
• Diagnosed through sensing and exploring , safe to fail experiments
• Issues return in different form even they are solved
• Unpredictable, great range of possible outcomes
Knowable
• Cause and effect separated over time and space
• Diagnosed through analytical/logical thinking, expert consultation
• New issues similar to the ones previously experienced
• Outcome of efforts can be predicted
Disorder
Chaos
• No perceivable cause-effect relationship
• Diagnose impossible
• Direct action
• Unexpected issues
• Outcomes unpredictable and unexplainable
Known
• Cause and effect relations is well understood
• Classify the problem, apply solution
• Commonly experienced issues with known causes
• Outcome of efforts generally certain

Figure 6.1 Characteristics of domains in Cynefin framework (adopted from Snowden, 2007).

Learning happens more in failure than in success (Snowden, 2010). Best practices work in simple domains where effect and cause relationships hold true. These best practices and many experiences fail in unpredictable complex domains such as human systems. This is especially critical in software development projects where earlier experiences and patterns are tried to be applied to the unique setting of the new project.

One other point that requires attention in using the proposed project and process management methods discussed which are Scrum, Kanban, Lean Development and TOC is that they are not mutually exclusive, inclusive or require each other. This means one can use Scrum together with Kanban, TOC with Lean etc. What we have done is to map the optimal domains each one of these is used.

Scrum:

Scrum is especially useful when requirements are emergent and the environment is highly uncertain. While by having working software at the end of each iteration the customer is better able to understand and express their requirements, continuous prioritization of backlog and high bandwidth communication with the team they are able to direct the emergence of the software towards their goal.

The quick cycle of learning and reflection enables the team to both improve skills and leverage experience and enables to self-organize for better performance.

When to use Scrum

- Requirements are emergent or unknown by the project team and difficult for the customer to express.
- Development team is highly qualified, collocated and self-organizing.
- When there is high uncertainty in the project.

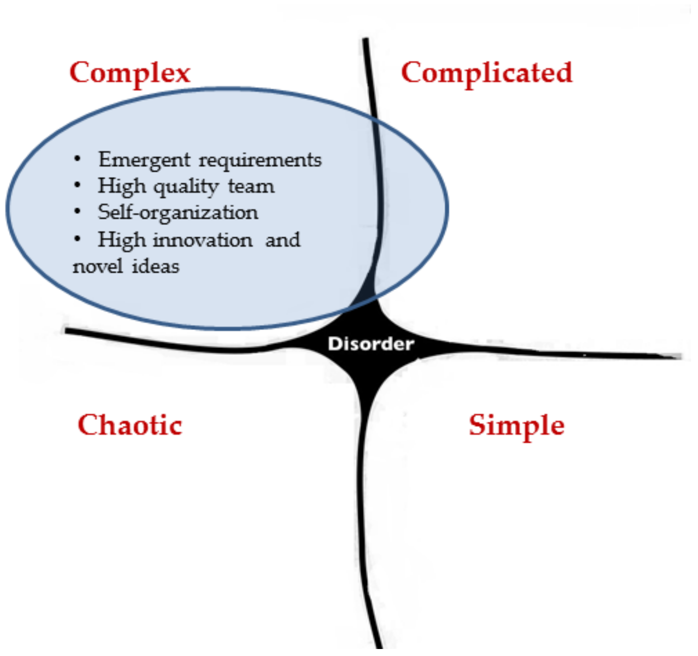

Figure 6.2 Scrum fitness in Cynefin domains

Lean software development accepts the unpredictability in processes, outcomes, bugs, errors and embraces the change in purpose, goals and the scope during project lifecycle. As a result lean upholds the ability of reacting to unfolding events with a system highly adaptive to changing environment and situation. This is done while “accepting the human conditions”(Lean Systems Society, 2012). Processes and interactions in the project must take human conditions such as being complex and logical thinks while being led by emotions and changing psychology and needs. Social needs of the people in the process are at least as important for the project as the project requirements.

Lean processes are descriptive so they can be overlaid to the existing processes.

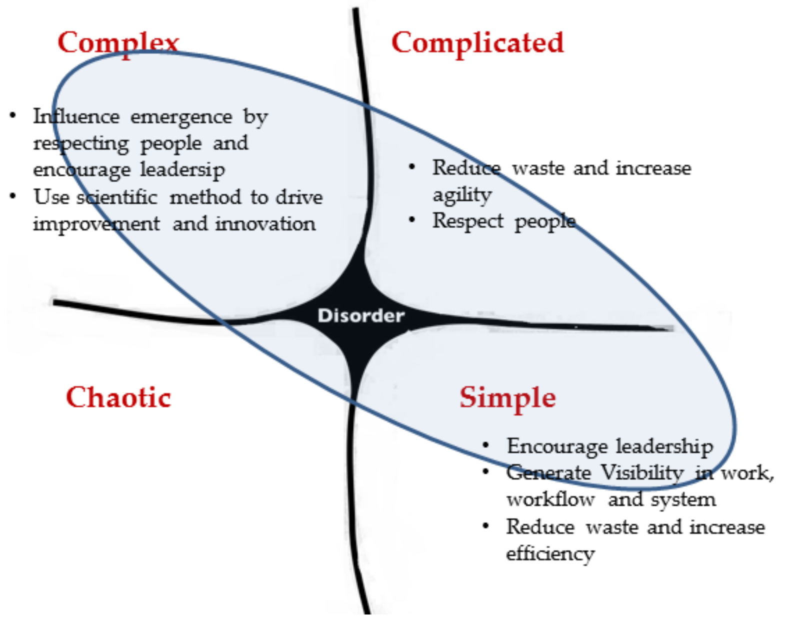
Complex
Complicated
• Influence emergence by respecting people and encourage leadersip
• Use scientific method to drive improvement and innovation
• Reduce waste and increase agility
• Respect people
Disorder
Chaotic
Simple
• Encourage leadership
• Generate Visibility in work, workflow and system
• Reduce waste and increase efficiency

Figure 6.3 Lean software development fitness in Cynefin domains

Kanban

Kanban is pull system for "achieving a sustainable pace of work and improve processes with minimum resistance"(Anderson, 2010). Kanban is implemented without changing any process or existing system at first. The system is improved with an incremental evolutionary approach with collaboration of the stakeholders.

When to use Kanban

When demand exceeds capability and flow is uneven and irregular. When improvement is necessary but existing processes must be kept or difficult to change. When more lean development is aimed to reduce waste, increase sustainability of development effort, create visibility in the project.

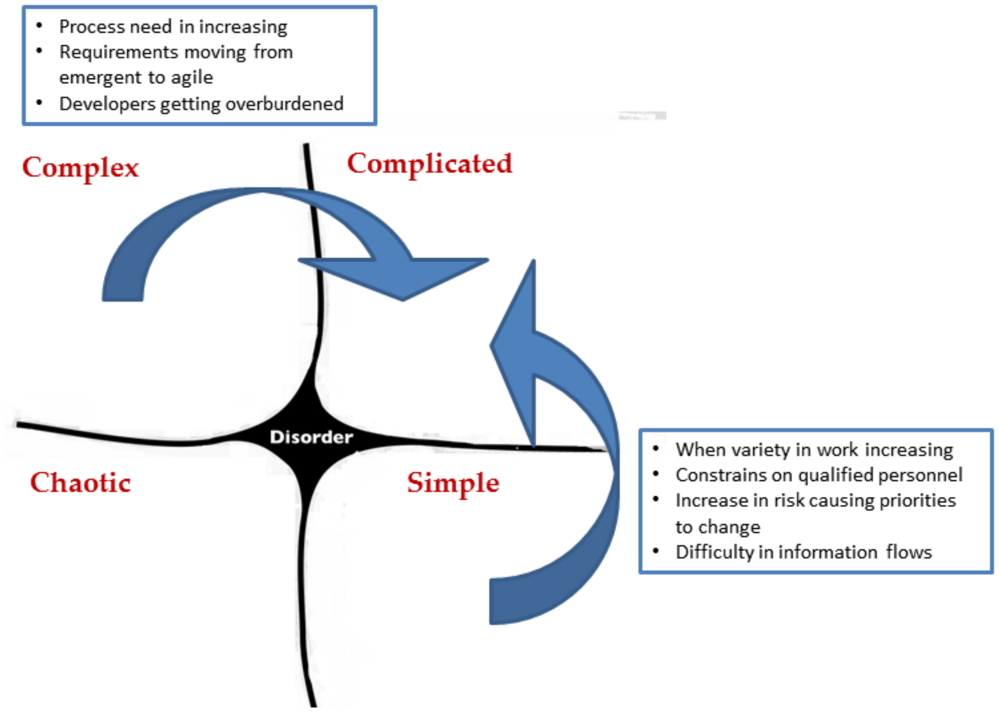
• Process need in increasing
• Requirements moving from emergent to agile
• Developers getting overburdened
Complex
Complicated
Disorder
Chaotic
Simple
• When variety in work increasing
• Constrains on qualified personnel
• Increase in risk causing priorities to change
• Difficulty in information flows

Figure 6.4 Kanban fitness in Cynefin domains.

CHAPTER 8

CONCLUSION

7.1 THE GOAL OF THE THESIS

The goal of this thesis was to develop and apply a framework that would enable to transfer the experiences and knowledge gained in production, project and process management methods to software development. The methods chosen are known to apply and increase success in complex environments that contain high risk, high uncertainty. Since software development is a social undertaking with similar conditions the transfer of the knowledge was considered to profit development project managers.

7.2 THE METHOD

When the literature concerning software project management, knowledge work and organizational environments with high uncertainty it was revealed that project management and software development didn't have a theory upon which the proposed practices and knowledge transfer can be judged and measured for validity and fitness. Without a theory of what works and why it works the success and failure of software development projects has no means to transfer the lessons to other projects. Since "practice without sound doesn't scale" a theory for software development is compiled and applied as complex adaptive systems.

Secondly it was observed in the literature review that the project management approach in agile software development has a completely different paradigm of project. Paradigms are patterns of thoughts that are used to make sense of the world around. Paradigms are the biggest drivers of actions and processes used in management and projects.

Agile development is a huge deviation from the traditional project management in terms of both how people and also how projects are managed. The "temporary production method" paradigm of agile is compared with “*a temporary endeavor undertaken to create a unique product or service*” understanding of traditional project management.

Thirdly agile software development methods with project and process management origins are examined. Lean software development, KANBAN, Scrum and TOC which are widely used in successful production and service companies with very complex environments are reviewed.

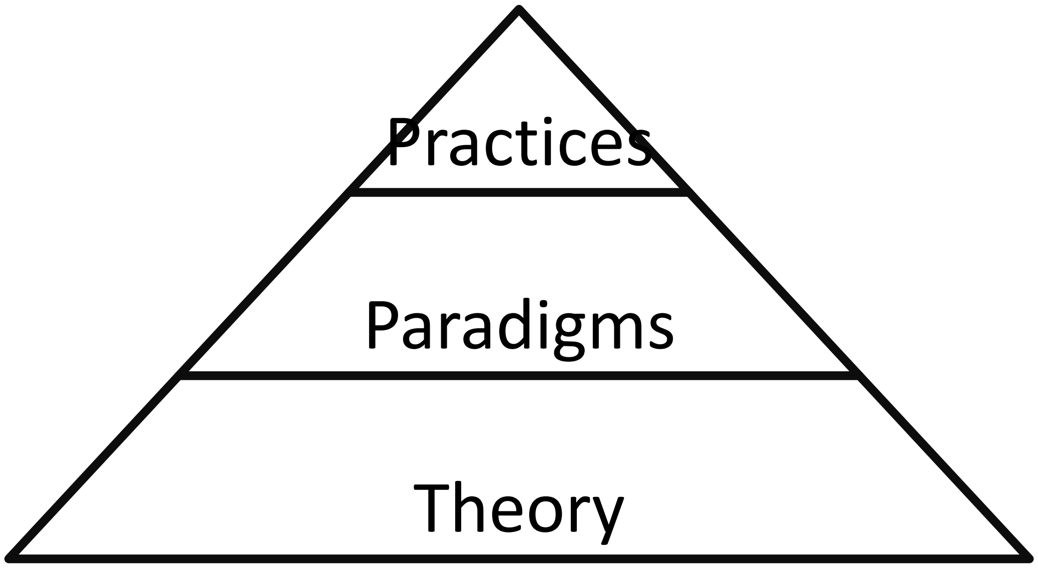

Figure 7.1 Conceptual levels of software development discipline

The behavior of human endeavor can be examined by the pyramid above. Every behavior and practice is the result of the assumptions and perceptions that form the paradigm. The scholars the writers and the practitioners of the field hold them usually subconsciously (Drucker, 1999).

Whereas the paradigms in social sciences has no effect on the natural universe they study, paradigms of a social discipline such as project management effect the behavior of its practitioners because it deals directly with people.

For this reason this thesis primarily focused on building the theory and paradigms underlying agile project management. Without this foundation application of different methods or practices would not have measurable and sustainable success.

For the practices layer of the concept pyramid for software development the project and process management methods are examined together with the CAS domains they are best suited for.

7.3 CONSTRAINTS OF THE STUDY

Since this thesis focuses primarily on software project management practices and methods, extensive field work is required to test the validity of this theory. Although there are extensive resources for project management practitioners, lack of literature on the underlying theory of project management together with lack of previous study on the application of studied theory constraints comparative study.
Lack of project teams and organizational resources required to apply the theory and observe the outcomes, restrict the scope of the study.

7.4 SHORTCOMINGS AND FUTURE WORK

The theoretical foundation examined in this thesis needs to be applied by multiple project and management settings.
The social, organizational and economic effects of applying the theory together with the paradigm and practices proposed require multidisciplinary and extensive research.
Software projects with different levels of complexity, risks, criticality and developer quality must be used test to validate the proper implementation of the theory.

REFERENCES

A.R.Cockburn, A., 1999. Characterizing people as non-linear, first-order components in software development. In *4th International Multi-Conference on Systems, Cybernetics and Informatics*. Florida, 1999. HeT Technical Report.

Ackermann, C. & Lindvall, M., 2006. Understanding Change Requests to Predict Software Impact. In *Software Engineering Workshop*. Florida, 2006. IEEE/NASA.

Ackoff, R.L., 1986. *Management in Small Doses*. New Jersey: Wiley.

Ackoff, R.L., 2004. Transforming the Systems Movement. In *Interantional Conference on Systems Thinking in Management 04*. Philadelphia, 2004. ICSTM.

Ackoff, R.L. & Allion, R.J., 2003. Iconoclastic management authority advocates a systemic approach to innovation. *Strategy and Leadership*, 31(3), pp.19-26.

Ackoff, R., Emery, F. & Ruben, B., 2005. *On Purposeful Systems: An Interdisciplinary Analysis of Individual and Social Behavior as a System of Purposeful Events*. New Jersey: Aldine Transaction.

Ackoff, R.L., Magidson, J. & Addison, H.J., 2006. *Idealized Design*. New Jersey: Wharton School Publishing.

Alberts, D.S. & Czerwinski, T.J., 1996. Complexity, Global Politics and National Security. In *Proceedings of Complexity Global Politics and National Security Conference*. Washington, 1996. National Defense University.

Alder, K., 1997. Innovation and Amnesia: Engineering rationality and the fate of interchangeable parts manufacturing in France. *Technology and Culture*, 38(2), pp.273-311.

Amabile, T.M., 2002. Time pressure and creativity in organizations: A longitudinal field study. *Harvard Business Review*.

Anderson, P., 1999. Seven Levers for Guiding the Evolving Enterprise. In III, J.H.C. *The Biology of Business*. New York: Jossey-Bass Books. pp.113-214.

Anderson, D.J., 2003. *Agile management for Software Engineering*. New Jersey: Pearson Education.

Anderson, D.J., 2010. *Kanban: Successful Evolutionary Change for Your Technology Business*. Washington: Blue Hole Press.

Appelo, J., 2011. *Management 3.0*. Boston: Pearson Education.

Ashby, W.R., 1958. Requisite variety and its implications for the control of complex systems. In Cybernetics, I.A.f. *Cybernetica*. Namur, Belgium: International Association for Cybernetics. pp.83-99.

Ashby, W.R., 2004. Principes of the self-organizing system. *ECO*, pp.102-26.

ASQ, 2010. *Quality Glossary*. [Online] Available at: http://asq.org/glossary/q.html [Accessed 25 September 2012].

Atwood, J., 2006. *Is Software Development Like Manufacturing?* [Online] Available at: http://www.codinghorror.com/blog/2006/10/is-software-development-like-manufacturing.html [Accessed 10 September 2012].

Babik, L., 2005. Plans are useless but planning is indispensable. In *PMI Global Congress Proceedings*. Panama City, 2005. PMI International.

Barabba, V., Pourdehnad, J. & Ackoff, R.L., 2002. On misdirecting management. *Emerald*, pp.5-9.

Basili, V.R. & Turner, A.J., 1975. Iterative Enhancement: A practical Technique for software Development. *IEEE Transactions on Software Engineering*, pp.390-96.

Bawden, D. & Robinson, L., 2005. The dark side of information: Overload, anxiety and other paradoxes and pathologies. *Journal of Information Science*, 20, pp.1-12.

Beck, K. et al., 2001. *Manifesto for Agile Software Development*. [Online] Agile Manfesto Authors Available at: http://agilemanifesto.org/.

Berger, H., Beynon-Davies, P. & Cleary, P., 2009. The utility of a rapid application development approach for a large complex information system development. *Information Systems Journal (USA)*, pp.549-71.

Beynon-Davies, P., Carne, C., Mackay, H. & Tudhope, D., 1999. Rapid application development: an empirical review. *Operational Research Society*, pp.211-25.

Boehm, B., 1988. A Spiral Model of Software Development and Enhancement. *IEEE Computer*, May. pp.61-72.

Boehm, B. & Turner, R., 2003. *Balancing Agility and Discipline: A Guide for the Perplexed*. Boston: Pearson Education.

Boehm, B. & Turner, R., 2003. Rebalancing Your Organization's Agility and Discipline. In Maurer, F. & Wells, D. *Extreme Programming and Agile Methods - XP/Agile Universe 2003*. New Orleans: Springer Berlin Heidelberg. pp.1-8.

Bohner, S.A., 1996. Impact Analysis in the software change process: a year 2000 perspective. In *Software Maintenance 1996 Proceedings*. Monterey, 1996. IEEE.

Burns, T. & G.M.Stalker, 1961. *The Management of Innovation*. London: Oxford University Press.

Buser, T. & Peter, N., 2011. *Multitasking: Productivity Effects and Gender Differences*. Amsterdam: Tinbergen Institute.

Cahtfield, C., 2010. *Project 2010 Step by Step*. Washington: Microsoft Press.

Cain, B.G. & Coplien, J.O., 1993. A Role-Based Process Modeling Environment. *Proceeedings of the Second International Conference on the Sofware Process*, pp.125-33.

Cash, J.I., McFarlan, F.W. & McKenney, J.L., 1992. *Corporate Information Systems Management: Issues Facing Senior Executives*. Illionis: McGraw-Hill Inc.

Christensen, C., 2001. Christensen Interview "the innovateors educator looks at why great companies fail and why theory trumps data". *Strategy & Business*, (fourth quarter), p.37.

Chu, D., 2011. Complexity: Against Systems. *Theory in Biosciences*, pp.61-78.

Clapp, W.C., Rubens, M.T., Sabharwal, J. & Gazzaley, A., 2011. Deficit in switching between functional brain networks underlies the impact of multitasking on working memory in older adults. *Proceedings of the National Academy of Sciences of the USA*, pp.7213-17.

Clausewitz, C.v., 1984. *On War*. Princeton: Princeto University Press.

CMS, C.f.M.a.M.S., 2008. *Selecting a Development Approach*. Washington: Department of Health and Human Services.

Cockburn, A.A.R., 1999. Characterizing people as non-linear, first-order components in software development. In *International Multi-Conference on Systems, Cybernetics and Informatics Proceedings*. Florida, 1999. Humans and Technology.

Cockburn, A., 2000. Selecting a Project's Methodology. *IEEE Software*, pp.64-71.

Cockburn, A., 2001. *Agile Software Development*. New Jersey: Addison-Wesley Professional.

Cockburn, A., 2003. People and Methodologies in Software Development. *PhD Thesis*.

Cockburn, A., 2008. *Shu Ha Ri*. [Online] Available at: http://alistair.cockburn.us/Shu+Ha+Ri.

Cockburn, A., 2008. Using both incremental and iterative Development. *CrossTalk: The Journal of Defense Software Engineering*, pp.27-30.

Cockburn, A. & Fitzgerald, B., 2004. Toward a Conceptual Framework of Agile Methods. In *XP/Agile Universe*. Berlin, 2004. Springer-Verlag.

Cockburn, A. & Highsmith, J., 2001. Agile Software Development: The People Factor. *IEEE Computer*.

cohen, D., Lindvall, M. & Costa, P., 2004. An introduction to Agile Methods. *Advances in Computers*, 62, pp.1-66.

Cohn, M., 2010. *Succeeding with Agile*. New Jersey: Addison Wesley.

Conboy, K. & Fitzgerald, B., 2004. Toward a conceptual framework of agile methods: a study of agility in different disciplines. In *WISER '04 Proceedings of the 2004 ACM workshop on Interdisciplinary software engineering research*. New York, 2004. ACM Association for Computing Machinery.

Converge Consulting Group, 2005. *Master of cycle time: littles law*. [Online] Available at: http://www.converge-group.net/283/ [Accessed 18 October 2012].

Convis, G., 2001. *Role of Management in a Lean Manufacturing Environment*. [Online] Available at: http://www.sae.org/manufacturing/lean/column/leanjul01.htm [Accessed 16 September 2012].

Cook, D.A., 2011. On becoming a software engineer. *Crosstalk Journal of Defense Software Engineering*, p.39.

Curtis, W., 1987. On building software process models under the lamppost. In *International Conference of Software Engineering proceeedings*. Washington, 1987. IEEE CS Press.

Cusumano, M.A., MacCormack, A., Kemerer, C.F. & Crandall, W., 2009. Critical Decisions in Software Development: Updating the state of the practice. *IEEE Software*, pp.84-87.

Daft, R.L. & Lengel, R.H., 1986. Organizational Information Requirements, Media Richness and Structural Desing. *Management Size*, 32(5), pp.554-71.

Davenport, T.H., 2005. *Thinking for a Living: How to Get Better Performances And Results from Knowledge Workers*. Boston: Harvard Business Review Press.

Dean, D. & Webb, C., 2011. Recovering from information overload. *Organization Practice*, pp.23-35.

DeGrace, P. & Stahl, L.H., 1990. *Wicked Problems, Righteous Solutions a catalog of modern engineering paradigms*. New York: Prentice Hall.

Deming, W.E., 2000. *Out of the Crisis*. Massachusetts: Massachusetts Institute of Techology.

Derby, E., 2011. *Misconceptions about Self-Organizing Team*. [Online] Available at: http://agile.dzone.com/news/misconceptions-about-self [Accessed 31 October 20012].

Derek, C. & Dahlman, C., 2005. *The Knowledge Economy, the KAM Methodology and World Bank Operations*. Washington: The World Bank.

Dershowitz, N., 2006. *Software Horror Stories*. [Online] Available at: http://www.cs.tau.ac.il/~nachumd/horror.html [Accessed 07 September 2012].

Diment, K., Yu, P. & Garrety, K., 2009. Complex Adaptive Systems as a Model for Evaluating Organisational Change Caused by the Introduction of Health Information Systems. In *Proceedings of the Healt Informatics Society of Australia Conference*. Canberra, 2009. UOW.

Drobka, J., Noltz, D. & Raghu, R., 2004. Piloting XP on Four Mission-Critical Projects. *IEEE Software*, pp.70-75.

Drucker, P.F., 1967. *The Effective Executive*. New York: Harper Collins Publishers.

Drucker, P.F., 1993. *Innovation and Entrepreneurship*. New York: First Harper Business.

Drucker, P., 1997. Drucker Interview "Peter F. Drucker - a meeting of the minds". *CIO Magazine*, p.2.

Drucker, P.F., 1999. *Management Challenges for the 21st Century*. Massachussets: Butterworth-Heinemann.

Dubakov, M., 2011. *The Future of Agile Software Development*. [Online] Available at: http://www.targetprocess.com/rightthing.html [Accessed 10 September 2012].

Eccles, H.E., 1965. *Military Concepts and Philosophy*. New Jersey: Rutgers University Press.

Eccles, H.E., 1969. Military Theory and Education: The need for and nature of. *Naval War College Review*, p.72.

Eisenhardt, K.M. & Sull, D., 2001. Strategy as Simple Rules. *Harvard Business Review*, pp.106-16.

Eisenhardt, K.M. & Sull, D.N., 2001. Strategy as Simple Rules. *Harvard Business Review*, pp.107-16.

Ely, J., 2003. *More on daily Start-up meeting*. [Online] Available at: http://joeelylean.blogspot.com/2003/01/more-on-daily-start-up-meetings-few.html [Accessed 6 November 2012].

Eoyang, G., 2001. Conditions for Self-organizing in Human Systems. *Unpublished Doctoral Dissertation*, pp.2-9.

Esque, T.J., 1999. *No Suprises Project Management*. California: ACT Publishing.

Foerde, K., Knowlton, B.J. & Poldrack, R.A., 2006. Modulation of competing memory systems by distraction. *Proceedings of the National Academy of Sciences of the USA*, pp.11778-83.

Fretty, P., 2005. Reconciling Differences. *PM Network*, pp.13-25.

GAO, 2012. *Effective Practices and Federal Challanges in Applying Agile Methods*. Washington: United States Government Accountability Office.

Garvin, D.A., 1984. What does Product Quality Really Mean? *Sloan Management Review*, pp.25-44.

Gilb, T., 1976. *Software Metrics*. Little, Brown and Co.

Gilb, T., 1976. *Software Metrics*. London: Little, Brown and Co.

Gilberth, F.B., 1922. Process Charts First Steps in Finding the One Best Way to Do Work. *ASME transactions*, p.1029.

Glutch, D.P., 1994. *A construct for describing software development risks*. Pnnsylvania: SEI Carnegie Mellon University.

Goldenfeld, N. & Kadanoff, L.P., 1999. Simple Lessons from Complexity. *Science*, 284, pp.87-89.

Goldratt, E.M., 1994. *Its Not Luck*. Massachusetts: The North River Press.

Goldratt, E., 1997. *Critical Chain*. Massachusetts: The North River Press.

Goldratt, E.M., 2004. *The Goal: A Process of Ongoing Improvement*. Massachusets: North River Press.

Gordon, J.S., 2007. *10 Moments That Made American Business*. [Online] Available at: http://www.americanheritage.com/articles/magazine/ah/2007/1/2007_1_23.shtml [Accessed 4 November 2012].

Griffin, A., 1997. PDMA research on new product development practices: Updating trends and benchmarking best practices. *Journal of product innovation management*, pp.429-58.

Gross, J.M. & Mcinnis, K., 2003. *Kanban made simple*. New York, Broadway, USA: AMACOM American Management Association.

Hamel, G. & Prahalad, C.K., 1990. The core competence of the corporation. *Harvard Business Review*, pp.78-90.

Hanlan, M., 2004. *High Performance Teams: How to make them work*. Connecticut: Praeger Press.

Hanoulle, Y., 2007. *How to grow a self-organizing team*. [Online] Available at: http://www.hanoulle.be/2007/09/how-to-grow-a-self-organizing-team/ [Accessed 31 October 2012].

Harrison, N.B. & Coplien, J.O., 1996. Patterns of productive software organization. *Bell Labs Technical Journal*, pp.138-46.

Hawkins, B., 2012. *Emprical Process Control: Why Scrum Works*. [Online] Available at: http://barryhawkins.com/blog/2012/04/13/empirical-process-control-why-scrum-works/ [Accessed 18 September 2012].

Herzberg, F., 1966. *Work and nature of man*. Cleveland: World Publishing.

Highsmith, J.A., 1995. *Adaptive Software Development*. New York: Dorset House.

Highsmith, J., 2002. *Agile Software Development Ecosystems*. Indianapolis: Addison Wesley.

Highsmith, J.A., 2002. *Agile Software Development Ecosystems*. Massachusetts: Pearson Education Inc.

Highsmith, J. & Cockburn, A., 2001. Agile Software Development: The Business of Innovation. *IEEE Computer*, 34(9), pp.120-22.

Hillson, D., 2004. *Effictive Opportunity Management For Projects-Exploiting Positive Risk*. New York: Marcel Dekker.

Hillson, D. & Murray-Webster, R., 2005. *Understanding and managing risk attitude*. Hants: Gower Publishing Ltd.

Hippel, E.v., 1988. *The sources of Innovation*. Oxford: Oxford University Press.

Holland, J.H., 2006. Studying Complex Adaptive Systems. *Journal of Systems Science and Complexity*, pp.1-8.

Hopp, W.J. & Spearman, M.L., 2004. To Pull or Not to Pull: What Is the Question? *Manucturing and Service Operations Management*, pp.133-48.

Hounshell, D.A., 1984. *From the American System to Mass Production, 1800-1932: The Development of Manufacturing Technology in the United States*. Maryland: Johns Hopkins University Press.

Howell, G.A., Macomber, H., Koskela, L. & Draper, J., 2004. Leadership and Project Management: Time for a shift from Fayol to Flores. *Proceedings of the 12th Annual Meeting of the International group for Lean Construction*, pp.22-29.

Hunton, S., 2012. *ScrumMaster is not a project manager by another name*. [Online] Available at: http://www.scrumalliance.org/articles/436-a-scrum-master-is-not-a-project-manager-by-another-name [Accessed 5 September 2012].

Jacob, D., Bergland, S. & Cox, J., 2010. *Velocity:Six Sigma and the Theory of Constraints to Achieve Breakthrough Performance*. New York: Simon & Schuster.

Jacobson, I. & Spence, I., 2009. *Why We need a theory for software engineering*. [Online] Available at: http://www.drdobbs.com/architecture-and-design/why-we-need-a-theory-for-software-engine/220300840?pgno=2 [Accessed 19 december 2012].

Jangir, S.K., Gupta, N. & Agrawal, S., 2012. Evolution and Melioration of Software Management Processes. *International Journal of Software Engineering and Applictaions*, pp.61-83.

Janson, M.A. & Smith, L.D., 1985. Prototyping for systems development: A critical appraisal. *MIS quarterly*, 9(4), pp.305-16.

Jeffries, R., 2011. *xprogramming: an agile software development resource*. [Online] Available at: http://xprogramming.com/articles/kate-oneal-what-is-scrum/ [Accessed 30 July 2012].

Jensen, R.L. et al., 2004. Online Collaboration in Virtual Product Development Results From the IMS RPD 2001 Project. In *Computing Solutions in Manufacturing Engineering*. Brasov, 2004. Cosme04.

Johansen, T. & Gilb, T., 2005. From Waterfall to Evolutionary Development(Evo): How we rapidly created faster , more user-friendly and more productive software products for a competitive multi-national market. In *INCOSE*., 2005. INCOSE.

Jones, G., 2010. *Organizational Theory Design and Challenges*. New Jersey: Pearson.

Jones, G., 2012. *Organizational Theory Design and Change*. New Jersey : Pearson Education.

Juran, J., 1951. *Quality Control Handbook*. New York: McGraw-Hill.

Kajko-Mattsson et al., 2001. Taxonomy of Problem Management Activities. In *Proceedings of the Fifth European Conference on Software Maintenance and Reengineering*. Washington, 2001. IEEE Computer Society.

Kakar, A.K., 2012. A Theory of Software Development Methodologies. In *Southern Associaciation for Information Systems Conference*. Atlanta, 2012. Southern Association for Information Systems.

Katzenbach, J. & Smith, D., 1993. *The wisdom of Teams*. Boston: Harvard Business School Press.

Kidd, P.T., 1997. Agileenterprisestrategy:anextgenerationmanufacturing concepts. In *Proceedings of the IEEE Coloquium on Agile Manufacturing*. Washington, 1997. IEEE.

Koskela, L. & Howell, G., 2002. The Theory of Project Management Explanation to Novel Methods. In *Proceedings of Annual International Group for Lean Construction Conference-IGLC-10*. Gramado, 2002. IGLC.

Koskela, L. & Howell, G., 2002. The underlying theory of project management is obsolete. In *PLI Research Conference*. Seattle, 2002. PMI.

Koskela, L. & Vrijhoef, R., 2001. Is the current theory of construction a hindrance to innovation. *Buliding Research and Information*, pp.40-46.

Kotler, P. & Keller, K., 2008. *Marketing Management*. New Jersey: Prentice Hall.

Kotter, J.P. & Cohen, D.S., 2002. *The Heart of Change*. New York: Harvard Business School Publishing.

Krafcik, J.F., 1988. Triumph of the Lean Production System. *Sloan Management Review*, pp.41-52.

Kuhn, T.S., 1996. *The Structure of Scientific Revolutions*. Chicago: University of Chicago Press.

Larman, C. & Basili, V.R., 2003. Iterative and Icremental Development: A Brief History. *IEEE Computer*, 36(6), pp.47-56.

Larman, C. & Vodde, B., 2009. *Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum*. Boston: Pearson Education.

Lawrence, P.R. & Lorsch, J.W., 1967. Differentiation and Integration in Complex Organizations. *Admistrative Science Quarterly*, 12(1), pp.1-48.

Lean Systems Society, 2012. *Lean Systems Society Credo*. [Online] Available at: http://leansystemssociety.org/credo/.

Leffingwell, D., 2011. *Agile Software Requirements*. Boston: Pearson Education Inc.

Lipietz, A., 1992. *Towards a New Economic Order*. New York: Oxford University Press.

Little, J.D.C., 1961. A Proof for the Queuing Formula L= λW. *Operations Research*, 9, pp.383-420.

Little, T. et al., 2005. Context-Adaptive Agility: Managing Complexity and Uncertainty. *IEEE Software*, pp.28-35.

MacCormack, A., 2001. Product Development Practices That Work: How Intarnet Companies Build Software. *MIT Sloan Management Review*, 40(2), pp.75-84.

Maccormack, A., 2001. Product Development Practices That Work: How Internet Companies Build Software. *MIT Sloan Management Review*, (Winter), pp.75-84.

Maccormack, A., Crandall, w., Henderson, P. & Toft, P., 2012. Do you need a new product-development strategy. *Research-Technology Management*, pp.34-43.

MacCormack, A. & Verganti, R., 2003. Managing the Sources of Uncertainty: Matching Process and Context in Software Development. *The Journal of Product Innovation Management*, (20), pp.217-32.

Macomber, H., 2007. *Notes on The Underlying Theory of Project Management Is Obsolete*. [Online] Available at: http://www.reformingprojectmanagement.com/lenses/project-management-theory/notes-on-the-underlying-theory-of-project-management-is-obsolete/ [Accessed 2 September 2012].

Madden, W. & Rone, K., 1984. Design, Development, Integration: Space Shuttle Flight Software System. *Computer ACM*, pp.914-25.

Manifesto, A., 2001. *Manifesto for Agile Software Development*. [Online] Available at: http://agilemanifesto.org/ [Accessed 18 October 2012].

Marquis, H., 2012. *Organizational Agility - The Real Benefit of Cloud Computing*. [Online] Available at: http://www.pmhut.com/organizational-agility-the-real-benefit-of-cloud-computing [Accessed 8 October 2012].

Marsh, R.F. & Conard, M.A., 2008. A pull system for delegatino knowledge work. *Operations Management Research*, (1), pp.61-68.

Matta, N.F. & Ashkenas, R.N., 2003. Why Good Projects Fail Anyway. *Harvard Business Review*, pp.109-14.

McAfee, A., 2004. Do you have too much IT. *MIT-Sloan Management Review*, pp.29-42.

McCarthy, I.P., Tsinopoulos, C., Allen, P. & Rose-Anderssen, C., 2006. New Product Development as a Complex Adaptive System of Decisions. *Journal of Product Innovation Management*, pp.437-56.

McManus, J., 2004. *Risk Management in Software Development Projects*. Rochester: Elsevier.

Mills, J., Robey, D. & Smith, L., 1985. Conflict-handling and personality dimnsions of project-management personnel. *Psychological Reports*, pp.1135-43.

Minoura, T., 2003. *The 'Thinking' Production System*. [Online] Available at: http://www.toyotageorgetown.com/tps.asp [Accessed 07 August 2012].

Mintzberg, H., 1992. *Structure in Fives: Designing Effective Organizations*. New York: Prentice Hall.

Mnkandla, E. & Dwolatzky, B., 2007. Agile Methodologies Selection Toolbox. In *International Conference on Software Engineering Advances*. Cap Esterel, 2007. ICSEA.

Morgan, P., 1997. *The Design and Use of Capacity Development Indicators*. Quebec: CIDA.

Morgan, G., 2007. *Images of Organization*. London: SAGE.

Morgan, J.M. & Liker, J.K., 2006. *The Toyota Product Development System;Integrating People, Process and Technology*. New York: Productivity Press.

Morien, R., 2005. Agile Management and the Toyota Way for Software Project Management. In *IEEE 3rd Intarnational Conference on Industrial Informatics INDIN'05*. Perth, 2005.

Murauskaite, A. & Adomauskas, V., 2008. Bottlenecks in Agile Software Development Identified Using Theory of Constraints Principles.

Nonaka, I., 2007. The Knowledge-Creating Company. *Harvard Business Review*, pp.162-71.

Ogunnaike, B.A. & Ray, W.H., 1994. *Process Dynamics, Modeling and Control*. New York: Oxford University Press.

Ohno, T., 1988. *Toyota Production System: Beyond Large-Scale Production*. Portland: Productivity Inc.

Olson, E.E. & Eoyang, G.H., 2001. Using Complexity Science to Facilitate Self-organization Process in Teams. *OD Practitioner*, pp.1-15.

Olsson, R., 2006. *Managing Project Uncertainty by Using an Enhanced Risk Management Process*. Vasteras: Arkitektkopia.

Osherove, R., 2009. *Unit Testing*. Connecticut: Manning Publications Co.

Owen, D., 2004. *Copies in Seconds*. Boston: Simon&Schuster.

Oxford English Dictionary, 2010. *Oxford English Dictionary*. New York.

Parnas, D.L. & Clements, P.C., 1986. A Rational Design Process: How and Why to Fake It. *IEEE Transactions Software Engineering*, pp.251-57.

Pellegrini, R.P., 1997. *The links between Science, Philosophy and Military Science*. Alabama: Air University Press.

Pelrine, J., 2011. On Undarstanding Softwar Agility: A Social Complexity Point of View. *E:CO*, 13(1-2), pp.26-37.

Petzinger, T., 1999. *The New Pioneers: The Men and Women Who Are Transforming the Workplace and Marketplace*. New York: Simon and Schuster Inc.

Pichler, R., 2012. *Agile Product Management with Scrum*. Boston: Pearson Education Inc.

PMBOK, P.S.C., 1996. *A Guide to the Project Management Body of Knowledge*. North Carolina: PMI.

Polk, R., 2011. Agile and Kanban in Coordination. *IEEE Computer*, pp.263-68.

Poppendieck, M. & Poppendieck, T., 2003. *Lean Software Development An Agile Toolkit*. Boston: Addison-Wesley Professional.

Poppendieck, M. & Poppendieck, T., 2006. *Implementing Lean Software Development*. New York: Addison-Wesley Professional.

Rakitin, S., 2001. Manifesto Elicits Cynicism. *IEEE Computer*, pp.4-6.

Rasmusson, J., 2010. *The Agile Samurai How Agile Masters Deliver Great Software*. Dallas: The Pragmatic Bookshelf.

Ravilious, K., 2006. *Ancient Greek Computer's Inner Workings Deciphered*. [Online] Available at: http://news.nationalgeographic.com/news/2006/11/061129-ancient-greece.html [Accessed 26 september 2012].

Rice, D., 2007. *Geekonomics The Real Cost of Insecure Software*. Boston: Addison-Wesley Professional.

Royce, W.W., 1987. Managing the development of large software systems: concepts and techniques. In *ICSE'87 Proceedings of the 9th international conference on Software Engineering*. Los Alamitos, 1987. IEEE Computer Society Press.

Rugby_Union, 2009. *law 20: Scrum*. [Online] Available at: http://www.irblaws.com/2012/index.php?amendment=21 [Accessed 5 November 2012].

Rzevski, G., 2011. A practical Methodology for Managing complexity. *Emergence: complexity and Organization*, pp.50-62.

Schooenderwoert, N.V. & Morsicato, R., 2004. Taming the Embedded Tiger – Agile Test Techniques for Embedded Software. In *Proceedings for Agile Development Conference*. Utah, 2004. IEEE Computer.

Schwaber, K., 2004. *Agile Project Management with Scrum*. Washington: Microsoft Press.

Schwaber, K. & Sutherland, J., 2011. *The Scrum Guide*. California: Scrum Org.

Scott, W.R., 1981. *Organzations: Rational, Natural and open systems*. New Jersey: Prentice Hall Inc.

Seddon, J., 2008. *Systems Thinking in the Public Sector*. Axminster: Triarchy Press. Available at: http://www.thesystemsthinkingreview.co.uk/index.php?pg=18&utwkstoryid=266&backto=15&keyword=Economy%20of%20scale#_ftn8 [accessed 20 September 2012].

Shingo, S., 1981. *Study of Toyoda Production System from an Industrial Engineering Viewpoint*. New York: Productivity Press.

Shingo, S., 1981. *Study of Toyota Praduction System*. Boston: Productivity Press.

Shingo, S., 1989. *Study of Toyota Production System*. New York: Productivity press.

Shriver, R. & Birckhead, D., 2008. *Delivering Business Value with Lean and Agile*. Vayoming: Dominion Digital, Inc.

Slack, N., Chambers, S. & Johnston, D., 2004. *Operations Management*. London: Pearson Education.

Sliger, M. & Broderick, S., 2008. *The Software Project managers bridge to agility*. Boston: Pearson Education.

Smith, P.G., 1990. Fast-cycle Product Development. *Engineering Management*, pp.11-16.

Smith, P.G., 2007. *Flexible Product Devlelopment:Building Agility for Changing Markets*. California: John Wiley & Sons Inc.

Snowden, D., 2010. *Falcons and white feathered pigeons*. [Online] Available at: http://cognitive-edge.com/blog/entry/3474/falcons-and-white-feathered-pigeons/.

Snowden, D.J. & Boone, M.E., 2007. A Leader's Framework for Decision Making. *Harvard Business Review*, pp.27-36.

Sobek, D.K., C.Ward, A. & Liker, J.K., 1990. Toyota's Principles of Set-Based Concurrent Engineering. *Sloan Management Review*, pp.67-83.

Solera, J., 2012. *PROJECT MANAGEMENT LEADERSHIP*. [Online] Available at: http://pmlead.org/2012/05/25/why-does-cbpm-work [Accessed 31 Augustus 2012].

Spear, S. & Bowen, H.K., 1999. Decoding the DNA of the Toyota Production System. *Harvard Business Review*, pp.97-106.

Spolsky, J., 2012. *Software Inventory*. [Online] Available at: http://www.joelonsoftware.com/items/2012/07/09.html [Accessed 19 September 2012].

Stacey, R.D., 2000. *Complexity and Management*. New York: Routledge.

Standish Group, 2009. *The Chaos Report*. Massachusetts: Standish Group.

StateOne, 2010. *State of Agile Survey*. Atlanta: Versionone Inc.

Sull, D. & Eisenhardt, K., 2001. Strategy as simple rules. *Harvard Business Review*, pp.107-16.

Sullivan, A. & Sheffrin, S., 2003. *Economics:Principles in Action*. New Jersey: Pearson Prentice Hall.

Svahnberg, M., Gurp, J.v. & Bosch, J., 2001. *On the Notion of Variabiity in Software Product Lines*. Karlskrona: Blekinge Tekniska Hogskola.

Szalvay, V., 2004. *An Introduction to Agile Software Development*. Washington: Danube Technologies.

Takeuchi, H. & Nonaka, I., 1986. The New New Product Development Game. *Harvard Business Review*, pp.2-11.

Tan, G., 2009. *A Collection of Well-Known Software Failures*. [Online] Available at: http://www.cse.lehigh.edu/~gtan/bug/softwarebug.html#yorktown [Accessed 07 September 2012].

Taylor, F.W., 1903. *Shop Management*. New York: Harper & Brothers.

Taylor, F.W., 1934. *The Principles of Scientific Management*. New York: Harper and Brothers.

Taylor, A., 2000. *IT projects: sink or swim*. British Computer Society.

Thomas, M., 2001. IT Projects Sink or Swim. *British Computer Society Review*.

Tuckman, B.W., 1965. Devemopmental Sequence in Small Groups. *Psychological Bulletin*, 63, pp.384-99.

Tushman, M.L. & Anderson, P., 1986. Technological discontinuities and organizational environments. *Administrative Science Quarterly*, pp.439-65.

TWI, T.P.S.D.O., 2012. *Introduction to Production Systems*. [Online] Available at: http://www.twinetwork.com/files/upload/articles/Introduction%20to%20Production%20Systems.pdf [Accessed 3 November 2012].

Vego, M., 2011. On Military Theory. *JF Quarterly*, pp.59-67.

Vidgen, R. & Wang, X., 2006. Organizing for Agility: A Complex Adaptive Systems Perspective on Agile Software Development Process. In *14th European Conference on Information Systems*. Goteborg, 2006.

Volberda, H.W. & Lewin, A.Y., 2003. co-evolutionary Dynamics Within and Between Firms: From Evolution to Co-evolution. *Journal of Management Studies*, pp.2111-36.

Wang, X. & Conboy, K., 2009. Undarstanding Agility in Software Develpoment From a Complex Adaptive Systems Perspective. In *European Conference on Information Systems*. Verona, 2009. ECIS.

Warfel, T.Z., 2009. *Prototyping: A Practitioners Guide*. New York: Rosenfeld Media.

Webster, 1981. *Webster's Third New International Dictionary*. Springfield: Merriam Webster's Inc.

Weick, K.E., 1995. *Sensemaking in Organizations*. California: Sage.

Whitesides, G.M. & Ismagilov, R.F., 1999. Complexity in Chemistry. *Scince*, 284(5411), pp.89-92.

Wiegers, K.E., 1998. Know Your Enemy: Software Risk Management. *Software Development*, pp.50-62.

Williams, L. & Cockburn, A., 2003. Agile Software Development: Its about Feedback and Change. *IEEE Computer*, pp.39-44.

Windholtz, M., 2003. *Lean Software Development*. [Online] Available at: http://www.objectwind.com/papers/LeanSoftwareDevelopment.html [Accessed 21 December 2011].

Womack, J.P. & Jones, D.T., 2003. *Lean Thinking: Banish Waste and Create Wealth in your Corporation*. New York: Free Press.

Womack, J., Jones, D. & Roos, D., 1990. *The Machine That Changed the World*. New York: Rawson Associates.